UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Enhancing Decision-Making in Software Development: An Automated System for Software Requirements Change Impact Analysis

by Kareshna Zamani

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy, in the School of Computer Science

Under the supervision of Dr. Fahimeh (Danna) Ramezani

University of Technology Sydney
Faculty of Engineering and Information Technology

July 2025

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Kareshna Zamani, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy in Computer Science, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

Date: **30/07/2025**

TABLE OF CONTENTS

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my two supervisors, Dr. Fahimeh(Danna) Ramezani and Dr. Mohsen Naderpour. Your guidance, wisdom, and unwavering support have been instrumental in shaping this research and bringing it to fruition. Your expertise and insightful feedback have not only enriched my academic journey but also inspired me to reach new heights. I am profoundly grateful for the time, effort, and encouragement you both provided throughout this challenging and rewarding journey.

I am also grateful to Professor Didar Zowghi, for her support and encouragement at the beginning of this journey. Her belief in my work and her early guidance helped lay the foundation for this research.

I would also like to extend my sincere thanks to my husband, Hamidreza, whose patience, understanding, and love have been my anchor during this demanding process. Your constant support and belief in me gave me the strength to persevere, even during the most challenging times.

To my beloved daughter, Artemis, thank you for your patience and bringing joy and light to my life. You have been a source of endless inspiration and motivation. This achievement would not have been possible without the sacrifices you both made.

A special thanks goes to my sisters, Mona and Panteha, who, despite being overseas, continuously encouraged me to progress further and pushed me to strive for excellence. Your words of encouragement were a constant source of motivation.

To my mother, thank you for your unwavering support from afar and for always believing in me. Your trust and faith in my abilities have been a tremendous source of strength throughout this journey.

I also wish to express my heartfelt gratitude to my late father, who passed away when I was in high school. His inspiration to study and his memory has been guiding forces in my life. Though he is not here to see this accomplishment, his spirit has been with me every step of the way.

Finally, I would like to thank everyone else who has contributed to my PhD journey, whether through academic support, friendship, or encouragement. This work is as much a testament to your support as it is to my efforts.

List of Publications

Here is the list of the author's papers that overlap with this thesis:

1. An In-depth Exploration of Machine Learning Algorithms for Software Requirements Change Impact Prediction, Kareshna Zamani, Mohsen Naderpour, Fahimeh Ramezani, Mukesh Prasad, 2024, Neural Computing and Applications journal, under review

2. Machine Learning in Requirements Engineering: A Mapping Study, Kareshna Zamani, Didar Zowghi, Chetan Arora, (2021), pp. 116-125, REW 2021: Proceedings of the 2021 IEEE 29th International Requirements Engineering Conference Workshops, Notre Dame, Ind., E1

3. A Prediction Model for Software Requirements Change Impact. Kareshna Zamani, 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
CIA	Change Impact Analysis
NLP	Natural Language Processing
IR	Information Retrieval
RAG	Retrieval-Augmented Generation
BM25	Best-Matching 25
ML	Machine Learning
SVM	Support Vector Machine
NB	Naive Bayes
LLM	Large Language Model
FAISS	Facebook AI Similarity Search
MRR	Mean Reciprocal Rank
F1	F_1-Score
API	Application Programming Interface
VRAM	Video Random-Access Memory
ANN	Approximate Nearest Neighbor

Abstract

Change Impact Analysis (CIA) is a critical task in software requirements engineering, aiming to predict the effects of requirement changes on related artifacts and systems. Traditional CIA methods often rely on manual inspection and heuristic-based reasoning, which are time-consuming and error-prone. This research addresses these limitations by proposing an automated framework for Software Requirements Change Impact Analysis (SRCIA), leveraging advances in Machine Learning (ML), Natural Language Processing (NLP), and Artificial Intelligence (AI).

The framework integrates a range of approaches, including traditional ML models, NLP-based techniques, BEIR-based retrieval methods, and a Retrieval-Augmented Generation (RAG) system, to assess their effectiveness across multiple datasets of varying complexity. Evaluation metrics such as precision, recall, F1 score, BLEU, and ROUGE are used to benchmark performance.

A central contribution is the development of a RAG-based solution that combines Large Language Models (LLMs) with modern information retrieval techniques. By incorporating vector database tools like LanceDB and FAISS, along with prompt engineering strategies, the framework achieves accurate and context-aware impact predictions. This enables robust adaptation to real-world, unstructured, and evolving requirements. The research provides a practical, scalable, and extensible solution to support automated CIA in complex software projects.

Chapter 1.

Introduction

1.1. Background

Requirements engineering (RE) plays an essential role in capturing correct and complete requirements and is considered one of the most critical and challenging stages of developing software. Errors in the requirements can be expensive in terms of lost time, revenue, reputation, and project sustainability (Beecham, Hall & Rainer 2005). When a single requirement statement changes within software requirements specification (SRS), it may trigger multiple changes throughout the SRS. A manual analysis of how these requirement changes affect other requirements is time and effort-intensive and error-prone. Requirements changes can potentially lead to inconsistencies in SRS, particularly in large systems(Arora et al. 2015a; Nejati et al. 2016). Therefore, analyzing the impact of requirements changes is essential to ensure accuracy, reliability, and consistency. It is also necessary to assess the effects of changes on downstream artifacts, such as software design and source code (Bjarnason et al. 2014).

This thesis introduces a novel requirements engineering change impact analysis (CIA) framework designed for application during the software development phase. This framework leverages requirement artifacts as the primary source for Enhancing Decision-Making in Software Development by conducting impact analysis.

The rapid evolution of software systems has brought unprecedented complexity to their development and maintenance processes. Modern software applications span diverse domains, including healthcare, finance, and education, demanding scalable and efficient engineering methodologies. Software requirements are a crucial stage in the software development lifecycle, serving as the foundation for understanding the software's purpose, functionality, and boundaries. This stage involves the meticulous process of identifying, analyzing, and defining what the software is expected to accomplish (Zowghi & Paryani 2003).

Among the various facets of software engineering, requirements engineering serves as the cornerstone, defining the specifications that guide development teams in creating functional and reliable systems. So, it plays a critical role in ensuring that the needs and preferences of all stakeholders are adequately captured and documented (Li & Huang 2018). However, the dynamic nature of software projects often necessitates frequent modifications to requirements, leading to the need for robust mechanisms to manage and assess the impact of such changes effectively.

Change Impact Analysis (CIA) has emerged as a critical process within RE, addressing the challenge of identifying and understanding the ramifications of altering software requirements. Changes may arise due to evolving customer needs, technological advancements, or regulatory updates, and their ripple effects can span multiple components of a software system. Without systematic CIA, these changes can lead to defects, delays, and increased costs, and jeopardizing project outcomes.

Traditional approaches to CIA rely heavily on manual techniques and rule-based methods, which, while effective in certain scenarios, struggle to cope with the growing complexity and scale of modern software systems. The advent of advanced computational models, particularly in the domains of machine learning (ML) and natural language processing (NLP), has provided new opportunities to automate and enhance the accuracy of CIA. These models enable engineers to analyze relationships and dependencies among requirements with greater precision, reducing the likelihood of overlooked impacts and facilitating proactive decision-making.

Recent advancements in ML, NLP, and information retrieval have revolutionized the way software requirements are analyzed. Techniques such as Retrieval-Augmented Generation (RAG) and frameworks like BEIR combine powerful retrievers with generative models, enabling contextually rich analyses of requirements. These models utilize structured and unstructured data, capturing syntactic and semantic nuances that traditional approaches may miss. By employing transformer-based architecture and embedding techniques, these advanced systems align textual descriptions with potential impact areas, offering significant improvements in both precision and recall.

This chapter outlines the research gaps /motivations, research questions, objectives, contributions, methodology and the structure of the thesis.

1.2. Research Gap

Despite considerable research efforts in CIA, particularly in software maintenance and evolution, there is a substantial gap regarding the use of predictive models for CIA in RE. Current approaches focus on specific aspects like traceability or dependency analysis without leveraging ML's predictive capabilities. For example, (Arora et al. 2015a) relied on correlation rates to evaluate change impacts without early-stage dependency definition, which lacks the predictive power needed for proactive impact analysis. Similarly, (Hassine, Rilling & Hewitt 2005) applied slicing and dependency analysis at the use case map level, limited by its dependency on predefined structures. The use of semantic role labeling (SLR) by (Baumer, White & Tomlinson 2010) improved relationship identification but did not extend to predictive modeling.

The motivation for this research stems from the identified gap in current methodologies. By integrating ML into CIA, this research aims to develop a predictive model that enhances the ability to manage requirements changes proactively. This model will be evaluated using real-world datasets to ensure its practical applicability and effectiveness, providing a robust tool for software project managers to make informed decisions regarding change requests.

ML offers significant advantages for CIA, particularly in terms of predictive accuracy. ML algorithms can analyze historical data to identify patterns and predict future changes with high precision, which is essential for proactive CIA. Furthermore, ML is adept at handling the complexity and dynamic nature of software requirements and their interdependencies, outperforming traditional rule-based methods. The scalability of ML models allows them to process large datasets efficiently, making them suitable for large-scale projects where traditional methods fall short. Additionally, ML models have the capability to continuously learn and adapt from new data, thereby improving their predictive capabilities over time.

1.3. Research Questions

Software requirements constantly evolve, and new requirements often emerge (Brucker & Julliand 2014). Changeability has continued to be one of the critical software development challenges since Brooks identified it in his landmark paper (Brooks, F. 1987). Requirement changes present many challenges that hinder completing a project that precisely fulfills the client's demands.

CIA is a crucial task in RE as changes to the requirements are the main reason for software evolution (Bjarnason et al. 2014). As discussed in Section 1, performing requirements CIA manually might lead to additional complexity, extra cost, and time. Due to the growing and dynamic nature of requirements and various variables such as change type, requirement interdependencies, and impact of change, managing requirements change is highly complex and challenging (Morkos, Shankar & Summers 2012). Another challenge that imposes a restriction is generalizability. Prior studies explicitly mentioned the need for further experiments in other domains, especially with the help of domain experts, to determine whether their approaches and tools can be generalized, although this leads to high costs (e.g., (Arora et al. 2019; Hein, Voris & Morkos 2018)). An automated solution is thus required to perform CIA, as changes happen iteratively. For instance, an accurate model to predict new changes and their impact on the system can benefit requirements analysts in deciding if a change request should be accepted or rejected.

The ability to anticipate and analyze a change in requirements, predict its progression, and determine the effect early in the requirements engineering stages would enable requirements analysts to make better decisions about implementing change, especially in large-scale projects (Hein, Voris & Morkos 2018; Morkos & Summers 2010). This may be used to estimate the value of implementing requirement changes (Morkos & Summers 2010). This research was motivated by the need to present an automated approach for CIA using neural information retrieval approaches. This research attempts to address the following research questions:

RQ1: How can AI techniques, specifically NLP and ML, be applied to analyze the impact of requirement changes on other requirements and software artifacts?

RQ2: How can information retrieval techniques enhance the assessment of requirement changes on software artifacts?

RQ3: Which AI techniques or combinations of techniques are best suited for accurately predicting the impacts of requirement changes?

RQ4: How can these techniques maintain accuracy and precision across different application domains with distinct requirements specifications?

RQ5: How can requirements CIA be automated using the insights gained from AI and IR techniques?

1.4. Research Objectives

The primary aim of this research is to develop a predictive model capable of forecasting which software requirements will be impacted by a given change. The goal is to support project managers in making informed decisions regarding the acceptance or rejection of specific requirement changes, ultimately enhancing the efficiency and accuracy of the software development process.

This research introduces an algorithmic-based prediction model that leverages ML and NLP techniques to forecast the impact of requirement changes. The model aims to automate CIA and improve upon traditional, manual methods, which are often time-

consuming and error-prone, especially in large-scale software projects. The key objectives of the research are as follows:

Objective 1: Develop methodologies for comprehensive data preparation and feature engineering to support the proposed CIA models.

Objective 2: Develop a framework for CIA using the capabilities of NLP and ML.

Objective 3: Implement information retrieval techniques to enhance the assessment of requirement change impacts on software artifacts.

Objective 4: Embed AI and IR techniques to determine the most effective methods for accurate prediction of requirement change impacts.

Objective 5: Analyse and evaluate the robustness and applicability of these techniques across different application domains with distinct requirements specifications.

Objective 6: Design and implement an automated framework that integrates AI and IR techniques to predict the impacts of requirement changes on software artifacts and other requirements.

The significance of this research lies in its potential to advance the field of software requirements management by offering a scalable, automated solution to CIA. When dealing with smaller projects or a limited number of changes, manual impact analysis, while time-consuming, remains feasible. However, as complexity and volume of changes increase, manual methods become inefficient and error-prone. Automating CIA for large-scale, rapidly evolving software specifications can drastically reduce human error while enhancing both the speed and accuracy of analysis.

This research focuses on automating the prediction of future requirement changes by utilizing historical change requests accumulated over periods of one to three months. ML techniques are employed to predict the likely impact of new changes, providing valuable insights into how requirements will evolve throughout the software lifecycle (Basri et al. 2016).

Precise CIA is crucial for informed decision-making during software development, particularly when planning and prioritizing requirements in both traditional and agile

methodologies. By integrating existing manual approaches with an advanced predictive model, this research aims to improve the effectiveness of software development by providing accurate, real-time predictions on the impact of requirement changes.

1.5. Research Contributions

This research makes several significant contributions to the domain of software requirements engineering, focusing on developing innovative solutions for predicting the impacts of requirement changes. Each contribution corresponds to the outcomes of the research objectives, as outlined below:

1. **Enhanced Dataset Preparation for Comprehensive Evaluation**

A key contribution of this research lies in the comprehensive preparation of datasets used for training and evaluating the proposed models. Three distinct datasets, Dataset-I, Dataset-W, and Dataset-O, were curated from real-world sources, encompassing a total of 891 requirements and 77 change requests. These datasets were carefully selected and preprocessed to ensure they represent varying levels of complexity, domain-specific terminologies, and linguistic diversity, providing a robust testbed for the proposed frameworks. The rationale for selecting these datasets is further elaborated in Chapter 3, where their complexity, linguistic characteristics, and representativeness of various application domains are discussed in detail. This justifies their suitability for evaluating the adaptability and robustness of the proposed models.

The preparation process involved extensive normalization of the data to standardize terminologies and linguistic structures across datasets, coupled with tokenization to break down requirements and change requests into manageable components. Semantic relationships within the requirements were preserved and enhanced through the generation of sentence embeddings using the all-MiniLM-L6-v2 model. Additionally, addressing data imbalance posed by diverse requirements was a crucial focus; techniques such as oversampling for minority classes and augmenting underrepresented datasets were employed to reduce bias and improve model performance.

Furthermore, this research introduced a benchmark dataset tailored specifically for requirement change impact analysis, categorized by complexity, length, and domain specificity. These benchmark datasets provide a valuable resource for future studies and facilitate cross-comparison of methods in this domain. To ensure high-quality and reliable evaluation, domain experts manually annotated change requests to establish relationships with impacted requirements, thereby creating a strong ground truth. The datasets also emphasize generalizability, representing a wide array of application domains such as Web Service, Telecommunications and Satellite, enabling the evaluation of the proposed models' adaptability across diverse contexts.

This contribution underscores the importance of high-quality dataset preparation in advancing the field of requirements change impact analysis. The carefully curated datasets serve as a foundation for training, evaluation, and future research, enhancing the scalability, precision, and robustness of automated solutions in software requirements engineering.

2. Development of AI-Based framework for Impact Analysis:

Based on the second research objective, this work contributes an AI-based approach that leverages Natural Language Processing (NLP) techniques such as dependency parsing, named entity recognition (NER), and term frequency-inverse document frequency (TF-IDF) for feature extraction. Additionally, it incorporates ML models, including Random Forest, Support Vector Machines (SVM), and Decision Trees, to analyze the impact of requirement changes on other requirements and software artifacts. This contribution demonstrates the capability of combining syntactic, semantic, and contextual analysis to improve precision and recall in impact analysis, offering a novel perspective on dependency analysis in software engineering.

3. Designing of an Information Retrieval Framework:

Addressing the third research objective, this research designs an IR-based framework for assessing the impacts of requirement changes. The framework incorporates state-of-the-art retrieval techniques, including BM25 for lexical matching, Bi-Encoders for dense vector similarity, and Cross-Encoders for re-ranking. These techniques collectively

enhance the retrieval and ranking of relevant requirements in response to change requests, ensuring both lexical and semantic alignment with the query. By leveraging these advanced retrieval methods, the framework achieves a balance between precision and recall, making it a robust tool for impact analysis in dynamic software engineering contexts

4. Integration of AI and IR Techniques in a Hybrid Framework:

In line with the fourth research objective, a hybrid framework that integrates AI and IR techniques is proposed and implemented. This hybrid framework integrates NLP-based and BEIR-based approaches for predicting requirement change impacts. The NLP-based approach leverages CoreNLP and SpaCy for linguistic feature extraction, including syntactic parsing and named entity recognition while the BEIR-based approach combines lexical retrieval, dense retrieval, and re-ranking. By integrating these approaches, the framework enhances the precision of semantic similarity measurements and the recall of relevant impacted requirements. The results demonstrate that combining these techniques improves the accuracy and robustness of impact predictions, particularly in handling datasets with diverse linguistic structures. This contribution sets a foundation for future hybrid approaches in requirements engineering, offering a balanced and adaptive solution for complex software development scenarios.

5. Introduced a systematic approach to develop a domain-specific framework by evaluating various AI techniques across different datasets.

Following the fifth research objective, the research evaluates the proposed solutions across three real-world datasets (prepared from contribution 1), covering 891 requirements and 77 change requests. The results highlight the generalizability of the approaches and their adaptability across varying application domains. This evaluation provides empirical evidence of the frameworks' effectiveness, contributing valuable insights for practitioners and researchers working with diverse datasets.

The proposed domain-specific framework refers not to a one-off, statically tailored solution, but to a dynamically adaptable architecture that continuously learns from new,

domain-specific data. Rather than hard-coding rules or manually tuning parameters for each application area, our framework employs transfer learning, initializing models on a broad corpus of software-engineering documents, and then fine-tunes on smaller, project-level datasets. This two-stage approach ensures the core model captures general change-impact patterns (e.g. traceability relations, dependency structures) while adapting dynamically to the terminology, style, and process nuances of each target domain.

6. Design an automated/domain-specific system for Requirement Change Impact Prediction using the integrated AI/IR technique suitable different datasets

The sixth and final contribution is the development of an automated framework leveraging the Retrieval-Augmented Generation (RAG) system, which integrates AI and IR techniques for predicting requirement change impacts. This framework employs advanced vector retrieval methods, including LanceDB and FAISS, for efficient context retrieval and combines them with generative capabilities of LLMs, such as Phi 3.5, to deliver accurate and context-aware predictions.

Although RAG has become popular in open-domain QA and chatbots, our implementation distinguishes itself in several important ways. First, we employ a structured vector index built on LanceDB, which organizes traceability-annotated artifacts, such as requirements, design documents and issue logs, so that retrieval emphasises semantically and procedurally relevant passages rather than mere surface-level similarity. Second, we use hybrid prompt engineering: rather than providing the model with raw text snippets alone, our prompts incorporate contextual signals like requirement IDs and change-request metadata alongside the retrieved content, guiding Phi 3.5 to generate domain-specific, accurate responses. Finally, we introduce an iterative retrieval–generation loop, in which initial candidate impacts are re-scored against the index and the top results are fed back into the model for a second synthesis pass. This two-pass cycle significantly enhances both precision and explainability compared with one-shot RAG approaches.

By dynamically adapting retrieval and generation processes to the characteristics of different datasets, the system ensures relevance and precision across diverse

requirements. This automated framework significantly reduces manual effort and enhances decision-making processes in software requirements engineering, offering scalability, adaptability, and efficiency in handling complex and evolving requirements landscapes.

1.6. Research Methodology

In conducting research, several methodologies can be employed depending on the nature of the research questions and the objectives of the study. The most commonly used methodologies include qualitative research, quantitative research, mixed methods, design science research, and empirical studies.

- Qualitative Research is primarily exploratory and is used to gain an understanding of underlying reasons, opinions, and motivations. It provides insights into the problem and helps to develop ideas or hypotheses for potential quantitative research. Methods such as interviews, focus groups, and case studies are typically used in qualitative research.
- Quantitative Research involves systematic investigation of phenomena by gathering quantifiable data and performing statistical, mathematical, or computational techniques. This method is often used to test hypotheses or measure variables and relationships. Surveys, experiments, and observational studies are common methods used in quantitative research.
- Mixed Methods Research combines both qualitative and quantitative approaches, allowing for a more comprehensive analysis by leveraging the strengths of both methodologies. This approach is particularly useful when the research question requires both the depth of qualitative insights and the generalizability of quantitative findings.
- Design Science Research Methodology (DSRM) focuses on the creation and evaluation of artifacts designed to solve identified problems or achieve specific goals. It is especially prevalent in fields like information systems and software

engineering, where the development of new tools, methods, or frameworks is necessary.

- Empirical Studies involve the collection and analysis of data from real-world observations or experiments. This methodology is particularly useful for testing hypotheses, validating models, or evaluating the practical effectiveness of solutions. Empirical research can provide robust evidence about the behavior of a system or the impact of specific interventions.

The research methodology employed in this study is grounded in the DSRM, a structured approach commonly used to develop knowledge through the creation of artifacts that serve as solutions to defined problems (Hevner et al. 2004); (Peffers et al. 2007). DSRM is particularly well-suited for this research, as it allows for the systematic design, development, and evaluation of new methods and models to address specific challenges in software requirements engineering and ML.

Our research follows the key stages of DSRM, as depicted in Figure 1.1, beginning with the identification of the problem. This initial phase involved conducting a comprehensive systematic literature review (SLR) focusing on the application of ML in requirements engineering. The goal of this review was to assess the effectiveness of ML in improving the requirements engineering process and its associated artifacts, as well as to identify gaps in the current literature (Kitchenham & Charters 2007).

The literature review was conducted following the Evidence-Based Software Engineering (EBSE) paradigm, as described by Kitchenham et al. (2004). This approach involved defining specific research questions, implementing a robust search strategy, compiling a list of related studies, and applying strict inclusion and exclusion criteria. Additionally, we employed backward snowballing and manual searches to ensure that all relevant studies were captured. The review was limited to papers published between 2010 and 2020, a period marked by a significant increase in publications on the intersection of ML and RE. This timeframe was chosen to focus on the most recent advancements in the field. This thesis limits its primary literature review to works published between 2010 and 2020. During this period, the foundations of requirements-change impact analysis such as supervised classification models, rule-based traceability

techniques, and early NLP integrations—were established. While significant advances have occurred since 2020 (notably the application of transformer-based models to traceability), these are reviewed comprehensively in Chapter 2 to highlight how they extend the pre-2021 methodologies examined.

From the selected papers, we extracted data on various aspects, including the ML techniques employed, the specific problems and challenges addressed, the datasets utilized, and the evaluation metrics used to assess the performance of ML techniques in RE. The analysis of 65 relevant papers revealed that ML is a powerful tool for automating RE tasks, addressing complexity, and reducing costs and development time. These insights were instrumental in refining the research objectives and aims, guiding the subsequent stages of our methodology.

With a clear understanding of the research problem and objectives, the next phase involved selecting appropriate datasets and defining the design cycles for the study. This step included a literature review on public datasets, followed by the collection of industry datasets where necessary, to ensure that the data used in our research was both relevant and comprehensive.

The design and development phase was then initiated, focusing on two primary models: the ML model and the NLP model. These models were developed iteratively, leveraging information retrieval techniques to enhance their accuracy and effectiveness. The design process was informed by the research questions identified earlier, ensuring that the developed models addressed the key challenges in software requirements CIA.

Once the models were developed, they were subjected to a rigorous demonstration phase, where their practical applicability was tested in real-world scenarios. This was followed by an evaluation phase, in which the models were assessed based on known performance parameters, such as accuracy, efficiency, and scalability. The evaluation provided critical feedback, which was used to refine the models further.

Finally, the results and findings from the research were communicated, contributing both to the academic body of knowledge and to practical applications in the field of software engineering. The iterative nature of DSRM ensured that each stage of the

research process was interconnected, with continuous feedback loops facilitating the refinement of the research outcomes.

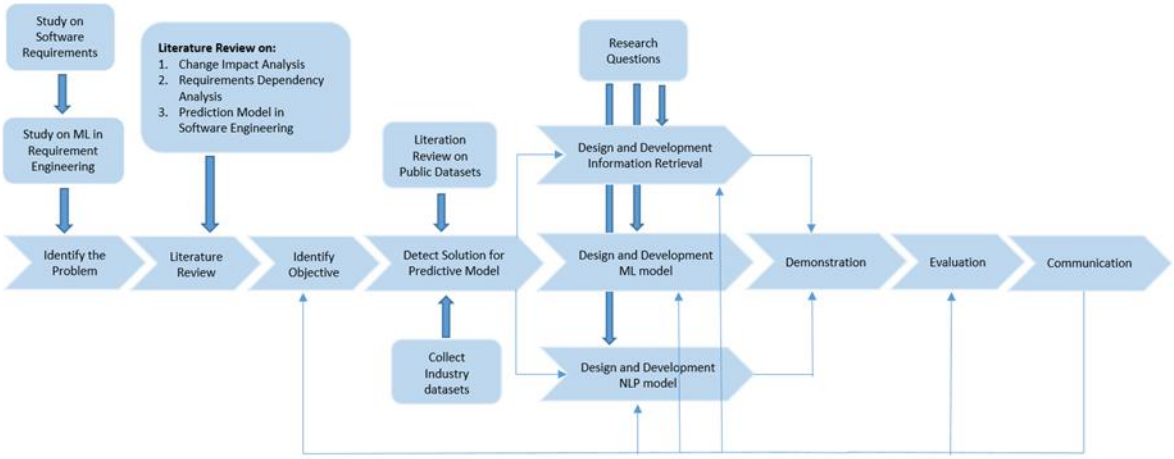

Figure 1.1.The Design Science Methodology Process Model [9]

1.7. Thesis Structure

This thesis is organized into the following chapters:

Chapter 1: Research Background and Objectives

This chapter introduces the research problem, providing background information on software requirements and the challenges of managing changes in requirements. It outlines the research aims, objectives, and contributions, setting the foundation for the study by addressing gaps in the literature and defining the research problem.

Chapter 2: Literature Review

Chapter 2 presents a comprehensive review of related works, focusing on software requirements change impact analysis, ML techniques, and NLP. The chapter also outlines the research questions and methodology for the mapping study that guided the systematic literature review.

Chapter 3: Research Framework

This chapter details the core framework for CIA developed in this research. It provides a structured view of the research stages, explaining the datasets used, the data

collection process, and the implementation of the proposed solutions. This chapter establishes the groundwork for understanding the methodologies applied throughout the thesis.

Chapter 4: Implementation of ML Algorithms

This chapter discusses the application of ML techniques for predicting the impact of requirement changes. It provides a detailed description of the technical approach, the implemented ML models, and the results obtained from the analysis. The chapter also includes a comparative evaluation of state-of-the-art algorithms.

Chapter 5: Implementation of the Dual-Model Framework

Chapter 5 introduces a dual-model framework that integrates NLP-based solutions with BEIR benchmark-based retrieval techniques. It explains the structure of the framework, including data collection, verification, preparation, and methodology branches. The chapter also presents the evaluation metrics and results of the implemented solutions, offering a comprehensive analysis of the effectiveness of the framework.

Chapter 6: Implementation of Retrieval-Augmented Generation (RAG) Model

Chapter 6 focuses on the use of the RAG model to enhance the predictive accuracy of CIA. It provides an overview of the RAG model, its implementation, and a discussion on how it compares with other methods used in this research. The chapter concludes with insights into future research directions and opportunities for improvement.

Chapter 7: Evaluation of the proposed models

This chapter compares the performance of ML, NLP-based, BEIR-based, and RAG models across structured, semi-structured, and unstructured datasets. Using metrics like precision, recall, and F1-score, along with visualizations such as radar charts, it highlights each model’s strengths and limitations. The chapter concludes by discussing the practical implications for CIA and proposing a hybrid framework to address varying dataset complexities.

Chapter 8: Conclusions and future work

Chapter 8 serves as the concluding chapter of this thesis, summarizing the key findings and contributions of the research while reflecting on how the objectives and research questions outlined in Chapter 1 were addressed. It synthesizes insights from the evaluations and analyses presented in previous chapters, emphasizing the significance of the developed frameworks and techniques in advancing the field of requirements change impact analysis. This chapter delves into the broader implications of the research findings, outlining how they contribute to the field and offering recommendations for potential advancements and enhancements to the proposed solutions. By integrating the outcomes of the study with its broader implications, Chapter 8 provides a cohesive and forward-looking conclusion to this research.

Chapter 2.

Literature Review

2.1. Introduction

This chapter provides a comprehensive review of the existing literature in software requirements engineering and CIA. It establishes the foundational concepts necessary to understand the research context, including the evolution of software requirements, the nature and challenges of managing requirements changes, and the role of CIA in software engineering.

The chapter begins by introducing key concepts in software requirements engineering, including the processes of requirements elicitation, analysis, validation, and documentation. It explores the evolution of requirements engineering practices, highlighting the shift from early informal approaches to structured, iterative, and agile methodologies. The dynamic nature of software requirements and their susceptibility to change are examined, along with the classification and implications of different types of requirement changes, corrective, adaptive, perfective, and preventive.

Next, the chapter delves into the background and significance of CIA in software engineering, tracing its origins and development. It discusses various CIA approaches, including dependency analysis and traceability analysis, and explores how CIA addresses the challenges posed by evolving requirements in complex software systems. Special emphasis is placed on the role of automated techniques, such as NLP and ML, in enhancing the efficiency and precision of CIA.

The chapter also presents a systematic mapping study that surveys and categorizes the existing body of research on requirements engineering and CIA up to 2020. This mapping study identifies key contributions, methodologies, and gaps in literature, serving as a basis for positioning the current research within the broader academic landscape. However, as the mapping study focuses on works published until 2020, recent advancements such as Retrieval-Augmented Generation (RAG) systems and Beir-based approaches are not included. These modern techniques, although promising, were introduced after the timeframe of this study and are beyond its scope.

By synthesizing insights from prior studies, this chapter highlights the need for a novel CIA framework tailored to address the challenges of managing requirements changes during the software development phase.

2.2. Software Requirements

The primary objective of requirements engineering is to facilitate a consensus among stakeholders—such as product managers, product owners, business analysts, customers, and developers—by clearly articulating their needs and aligning them with the project's goals. This process is vital because errors or oversights in the requirements phase can have far-reaching consequences, leading to costly delays, revenue loss, damage to reputation, and potentially jeopardizing the project's sustainability (Beecham, Hall & Rainer 2005).

One of the first steps in this process is requirements elicitation, which involves gathering business requirements through interactions with key stakeholders. This stage is essential for understanding the demands and expectations that will shape the software's development. Elicitation transforms a set of informal ideas into formal, structured expressions that can guide subsequent stages of development (del Águila & del Sagrado 2016a).

Following elicitation, these requirements undergo rigorous analysis to validate their feasibility and ensure that they can be realistically implemented within the system. This step involves not only technical assessments but also considerations of how the

requirements align with the overall business objectives. The culmination of this stage is the creation of requirements documents, which are then validated with the stakeholders to ensure completeness and accuracy.

A well-crafted and comprehensive requirements specification is one of the most critical artifacts in the requirements engineering process. It serves as the blueprint for the entire software development process, guiding the project from inception through to completion. This document is not merely a technical manual but a living document that reflects the negotiated compromises and agreed-upon features that will drive the project forward.

Overall, requirements engineering can be viewed as the systematic process of identifying, documenting, and managing the features and services that the software must provide, along with the constraints that govern its development and operation. This process is integral to the success of any software project, as it lays the groundwork for all subsequent development activities.

Figure 2.1 illustrates the workflow of software requirements engineering. The process begins with a Feasibility Study, resulting in a Feasibility Report that informs the Requirements Elicitation and Analysis stage. During this phase, System Models are developed, and User and System Requirements are articulated. These inputs feed into the Requirements Specification, which is a formal document outlining the system's functionalities and constraints. This specification is then subjected to Requirements Validation to ensure accuracy and completeness before it is finalized as the Requirements Document. Each step in this workflow is interconnected, reflecting the iterative nature of software development, where validation and feedback loops are critical to refining and ensuring the quality of the requirements.

2.3. Evolution of Software Requirements

The evolution of software requirements is a reflection of the broader changes in software engineering practices and the increasing complexity of software systems. Over the past several decades, the process of defining and managing software requirements

has undergone significant transformations, driven by advancements in technology, methodologies, and the growing demand for more complex and adaptive software systems.

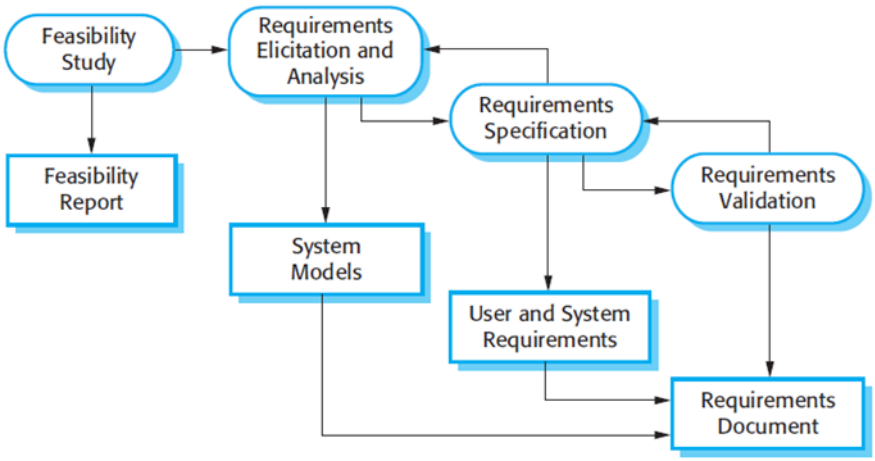

Figure 2.1.Software Requirement Process (Sommerville, 2004)

A. Early Approaches to Requirements Engineering

In the early days of software development, requirements engineering was a relatively informal process. Requirements were often captured through ad-hoc discussions and documented in unstructured formats, such as text-based specifications or simple diagrams. These early approaches were adequate for small-scale projects where the scope of the software was limited, and the development team was small. However, as software systems grew in size and complexity, the limitations of these informal methods became apparent. Requirements were frequently ambiguous, incomplete, or inconsistent, leading to costly rework and project delays.

B. The Advent of Structured Requirements Engineering

The 1970s and 1980s marked a significant shift in requirements engineering with the introduction of structured methodologies. The Waterfall model, one of the earliest formalized software development methodologies, emphasized a linear approach to software development where requirements were defined upfront and served as the foundation for all subsequent stages of development (Royce, 1970). This model

necessitated a more rigorous approach to requirements specification, leading to the development of structured techniques for requirements elicitation, analysis, and documentation.

During this period, the notion of "correctness" in requirements became a central focus. Requirements needed to be clear, precise, and verifiable to ensure that the final software product met the intended goals. Techniques such as data flow diagrams (DFDs) and entity-relationship diagrams (ERDs) were introduced to model requirements in a more structured and systematic way (Yourdon, 1989).

C. The Emergence of Iterative and Agile Approaches

The late 1980s and 1990s saw the emergence of iterative and incremental development methodologies, such as the Spiral model (Boehm, 1988), which introduced the concept of revisiting and refining requirements throughout the software development lifecycle. This approach acknowledged the reality that requirements often change as stakeholders gain a better understanding of their needs and as the market environment evolves. Iterative methodologies allowed for more flexibility in handling these changes, reducing the risks associated with rigid, upfront requirements specification.

The turn of the century brought about the widespread adoption of Agile methodologies, which revolutionized requirements engineering by promoting a more collaborative and adaptive approach. In Agile frameworks, such as Scrum and Extreme Programming (XP), requirements are captured in the form of user stories and are continuously refined through iterative cycles known as sprints (Beck et al., 2001). This approach emphasizes direct communication between developers and stakeholders, fostering a dynamic environment where requirements can evolve in response to feedback and changing business priorities.

2.4. Understanding Requirements Change

A. Nature of Requirements Change

Software requirements are inherently dynamic and subject to change throughout the development process. The nature of these changes' stems from a variety of factors that are often interrelated and context dependent. One primary reason for changes in software requirements is the evolving business environment in which organizations operate. As market conditions, customer needs, and competitive pressures shift, the software must adapt accordingly, necessitating changes to its requirements. Additionally, stakeholders often gain a clearer understanding of their needs as the project progresses, leading to refinement and modifications in the initial requirements. Technological advancements can also drive changes, as new tools, platforms, or methodologies become available that could enhance the software's functionality or performance.

Moreover, regulatory and compliance requirements can impose changes, especially in industries that are heavily regulated, such as healthcare, finance, and aerospace. As new laws or standards emerge, software systems must be updated to remain compliant, resulting in adjustments to their requirements. These factors underscore the fluid nature of software requirements, making change management a critical aspect of the software development process.

B. Types of Requirements Changes

Requirements changes can be broadly categorized into four types: corrective, adaptive, perfective, and preventive.

- **Corrective Changes:** These changes are initiated to fix defects or issues identified in the requirements after they have been initially defined. Corrective changes ensure that the software meets the intended functionality and performance standards by addressing errors, inconsistencies, or omissions in the original requirements.
- **Adaptive Changes:** Adaptive changes occur when the software needs to be modified to work in a new or changed environment. These changes are often driven by shifts in the business environment, new customer demands, or changes in the external system that the software interacts with. Adaptive changes are essential for ensuring that the software remains relevant and functional in a changing context.

- **Perfective Changes:** Perfective changes involve the enhancement of existing software functionalities to improve performance, maintainability, or user experience. These changes are typically driven by user feedback or the desire to optimize the software's operations. While the software may be fully functional, perfect changes aim to make it more efficient or user-friendly.
- **Preventive Changes:** Preventive changes are proactive modifications made to software requirements to avoid potential issues in the future. These changes often involve refactoring or restructuring the software's architecture to improve its scalability, security, or robustness, thereby reducing the likelihood of defects or failures as the software evolves.

Each type of requirement change has its own set of implications for the software development process, requiring careful consideration and planning to ensure that the changes are effectively integrated without disrupting the project's overall timeline and objectives.

2.5. Challenges in Managing Requirements Change

Managing requirements change is one of the most complex and challenging aspects of software development. One of the primary difficulties lies in maintaining traceability and consistency across the various artifacts that constitute the software's documentation. When a requirement changes, it can have a cascading effect on related requirements, design documents, test cases, and even code. Ensuring that all related components are updated accordingly is crucial to maintaining the integrity of the software system.

Another significant challenge is stakeholder alignment. Different stakeholders may have conflicting priorities or interests, making it difficult to achieve consensus on the nature and scope of changes. This can lead to delays, increased costs, or scope creep if not managed effectively. Additionally, the iterative nature of modern software development methodologies, such as Agile, means that requirements are continually evolving. This constant state of flux can be difficult to manage, particularly in large-scale

projects where multiple teams are working concurrently on different aspects of the software.

Resource allocation is another critical issue. Implementing changes often requires additional time, budget, and human resources, which may not have been accounted for in the original project plan. This can strain the project's resources, leading to potential delays or compromises in quality. Furthermore, the introduction of new requirements can increase the complexity of the software system, making it more difficult to test and validate. This, in turn, can increase the risk of defects or failures in the final product.

Lastly, the impact of changes on project timelines and delivery schedules can be significant. Unplanned changes can disrupt carefully coordinated schedules, leading to delays and increased pressure on the development team. Effective change management requires a delicate balance between accommodating necessary changes and maintaining the project's overall momentum and focus.

In summary, while changes in software requirements are inevitable, managing these changes effectively is crucial to the success of the project. This involves not only technical considerations but also strategic planning, stakeholder management, and resource allocation to ensure that changes are implemented smoothly and do not adversely affect the project's outcome.

2.6. Change Impact Analysis in Software Engineering

CIA is a critical aspect of software engineering, particularly in managing the effects of changes in software requirements. As software systems evolve, requirements often change, leading to engineering changes (ECs) that can have significant implications for the development process. An engineering change is typically defined as a modification to a system component—whether in design, functionality, or other aspects—after it has been released (Shankar et al. in press). These changes can vary in scale and complexity and may affect multiple stakeholders over an extended period.

The process of managing these changes begins with an Engineering Change Request (ECR), a document that outlines the details of the proposed change and is circulated

among relevant stakeholders for review and approval. If the ECR is approved, it is followed by the release of an Engineering Change Note (ECN), which formalizes the change and notifies all stakeholders of its implementation. The final stage involves archiving the change, documenting the reasons, outcomes, and impacts for future reference (Chen, Shir & Shen 2002; Morkos & Summers 2010).

Effective CIA is essential in this process, as it involves assessing how a proposed change will propagate through the system and what other components will be affected. By analyzing ECNs and other related documents, researchers can develop models to predict the impact of changes and manage the risks associated with them (Morkos, Shankar & Summers 2012). This predictive capability is particularly important in large-scale systems, where the interdependencies between components can make the effects of changes difficult to anticipate.

This thesis introduces a novel requirements engineering CIA framework designed for application during the software development phase. This framework leverages requirement artifacts as the primary source for enhancing decision-making in Software Development by conducting impact analysis.

This chapter outlines the research background, objectives, motivation, contributions, and the structure of the thesis.

2.7. Change Impact Analysis Background

CIA was first introduced and studied in 1993 by Arnold and Bohner (Arnold 1996; Arnold & Bohner 1993). They indicated that impact analysis involves identifying the possible effects of a change or predicting what needs to be modified to implement a change. The need to forecast and manage the impact of software changes increases as software systems become extremely large and complicated. Software CIA gathers the current data of the software system to identify which components will be affected by the proposed change or how the components will affect each other. Based on Arnold and Bohner (Arnold & Bohner 1993), there are two main perspectives for CIA, including software dependency analysis and traceability analysis (Arnold 1996). Arnold et al.

described three main steps to analyze the change impacts in a system (Arnold & Bohner 1993):

- Analyze change specifications and software artifacts.
- Trace potential impacts
- Implement the requested changes

Changes initiated by a change request involve the change specification and classification process, which finishes with identifying the change type (Jayatilleke & Lai 2013). Whenever addressing a change, many requirements cannot be considered independent of other requirements in the SRS, as different types of relationships can exist between them. As a result, an action performed on one requirement may have unexpected impacts on another. Therefore, there is a need to identify requirement interdependencies (Jayatilleke, Lai & Reed 2018). It is essential to investigate how requirements are dependent when there is no semantic or syntactic similarity between them. The investigated dependencies between requirements can be used to develop a predictive model to forecast CIA.

In recent decades, considerable research has focused on reviewing the CIA, especially in software maintenance and evolution (Alkaf et al. 2019; Jayatilleke & Lai 2018; Lehnert 2011a, 2011b). CIA has been applied to source codes (Brucker & Julliand 2014) and requirements traceability (Goknil, Kurtev & Berg 2016; Li et al. 2008; Zhang et al. 2014). Some researchers have reported the effects of requirements changes on data design, architecture design, and software design (Von Knethen 2002; Yazdanshenas & Moonen 2012). Few researchers have studied the challenges of change verification and validation (Bjarnason et al. 2014) and co-changing artifacts to gather more information about software artifact evolution (Antoniol, Rollo & Venturi 2005). Some approaches have attempted to automate the process of analyzing the change impacts. (Alkaf et al. 2019) performed an automated CIA approach for User Requirements Notation models. Arora et al. (Arora et al. 2015a) proposed a strategy based on NLP for analyzing the impact of change in natural language requirements. (Nejati et al. 2016) proposed an approach to automatically identify the impact of requirements changes on system design when the requirements and design elements are expressed using models. Jayatilleke (Jayatilleke,

Lai & Reed 2018) presented a technique for requirements change analysis that relied on changes arising at higher levels. (Bano et al. 2012) performed a systematic literature review on the causes of requirement change, categorizing them into necessary and accidental causes. (Aryani et al. 2009) proposed a methodology for analyzing change propagation in software using the domain-level behavioral model of a system.

In requirement specification, one solution to assess the effect of the change is to look for the precise accordance of the terms contained in the change and its potential definitions and expressions in other specifications. Change can progress across semantically related terms that are not exact matches or relevant syntactic diversities. In this situation, it is appropriate to apply a relatedness measure that considers phrases (Arora et al. 2015a). Besides, dependencies in requirements play an essential role in the analysis of change propagation (Zhang et al. 2014).

Many dependency or interdependency models have been developed to define and distinguish relationships based on requirements' structural and semantic properties to find the relationships between requirements (Zhang et al. 2014). However, there has been no empirical assessment of these dependency forms regarding usefulness and applicability (Zhang et al. 2014). To define the possible effect of requirement changes on the overall system, Hassine (Hassine, Rilling & Hewitt 2005) applied both slicing and dependency analysis at the level of the use case map (rather than between requirements in natural languages). Baumer (Baumer, White & Tomlinson 2010) showed that semantic role labeling could improve computational metaphor identification and more effectively identify relationships with semantic import than typed dependency parsing.

(Arora et al. 2015a) showed that in their approach, there is no need to define requirements dependencies in the early stage because the propagation condition can determine if there is a correlation between a changed requirement and the others. Since all potential conditions cannot be enumerated, constructing an explicit dependency graph is difficult. Rather than utilizing typed dependencies, they used correlation rates to evaluate the impact of changes. Typed dependencies focus on syntactic structure and

grammatical relations, while semantic roles emphasize conceptual and semantic structure (Baumer, White & Tomlinson 2010; de Marneffe & Manning 2008).

Alsalemi (Alsalemi & Yeoh 2017) performed a systematic literature review focused on predicting requirements volatility. According to their research, only a few papers have been published on predicting volatility requirements, and the majority of papers worked on the causes of requirements change and its effect on project performance. Their work underlines that more empirical studies need to be carried out to address the practical aspect of requirements volatility better.

Dhamija (Dhamija & Sikka 2019) presented a systematic study on the advancement of CIA techniques. The study's findings exposed a scope of research investigating the hidden dependency between software requirements that are not clearly visible. Techniques for identifying hidden dependencies among software objects, such as specifications, design, and code, need to be proposed. The existing literature focusing on CIA in RE showed that very few studies presented a prediction model for requirement change impact, and it is an under-explored area (Yang et al. 2020).

Anjali (Anjali, Dhas & Singh 2022) evaluated various CIA techniques focused on requirement defects in software development. The study categorizes these methods, assessing their effectiveness in identifying and mitigating defects. Highlighting the need for automated CIA tools, the research emphasizes improving accuracy and efficiency in defect management to maintain software quality and reliability.

Elapolu (Elapolu et al. 2024) proposed a blockchain-based framework for requirement traceability, integrating a data acquisition template and graph-based visualization for dual-level traceability (artifact and object levels). By leveraging blockchain, the framework ensures security, immutability, and enhanced collaboration among distributed stakeholders. This approach demonstrates significant improvements in managing dynamic requirements and securing traceability data.

Zhang (Zhang, Tan & Yang 2021) analyzed the impact of requirement changes on product development progress using system dynamics. The study divides the development process into three phases—concept development, detail design, and pilot

production, and examines how requirement changes cause reworks, affecting the overall development duration. The authors highlight that requirement changes, especially in the later phases, significantly increase development time and introduce uncertainty. By modeling the development process and simulating requirement changes, the study provides insights into managing these changes to minimize delays and improve project management.

Akbar (Akbar et al. 2020) investigate the challenges of requirements change management (RCM) in global software development (GSD) projects. By conducting a systematic literature review (SLR) and validating the findings through a questionnaire survey, they identify 25 RCM challenges. These challenges are categorized based on organization type (client and vendor) and size (small, medium, large), highlighting their significance in different GSD contexts. The study emphasizes the need for tailored strategies to manage RCM effectively across diverse organizational settings, underscoring the complexity and importance of RCM in maintaining software quality and project success in GSD environments.

Arif (Arif, Mohammad & Sadiq 2023) proposed a method combining UML and the NFR framework to analyze both functional and non-functional requirements of information systems. The technique uses UML diagrams (use-case, class, and activity diagrams) for modeling functional requirements (FRs), while the NFR framework is employed to handle non-functional requirements (NFRs) using a fuzzy-based approach to deal with vagueness in soft goal interdependencies. The applicability of this method is demonstrated through a library information system case study, showcasing how the integration of these techniques can enhance the precision and comprehensiveness of requirements analysis.

Anwer (Anwer et al. 2024) introduced BECIA, a behavior engineering-based approach for CIA. BECIA employs Integrated Behavior Trees (IBT) and Integrated Composition Trees (ICT) to model system requirements and their dependencies. The approach includes a Requirements Components Dependency Network (RCDN) and a Change Impact Indicator (CII) to quantify change impacts using Kolmogorov Complexity. By automating the transformation of IBTs to ICTs and subsequently to RCDNs, BECIA

enhances the efficiency and accuracy of CIA. The study demonstrates the approach's applicability through evaluations of student projects, highlighting its potential to improve change management in software development.

While traditional CIA methods such as rule-based dependency analysis, traceability matrices, and manual heuristics have proven effective in limited and controlled environments, they face notable limitations in handling modern software development complexities. Scalability remains a core challenge, manual and semi-automated techniques struggle to scale across large and continuously evolving software systems where thousands of interdependent requirements exist. Moreover, these methods often rely on structured formats or predefined relationships, making them less effective when dealing with unstructured or ambiguous textual data, which is common in real-world requirements specifications. These limitations directly motivate the adoption of advanced NLP and ML techniques. By leveraging language models and learning-based approaches, NLP/ML systems can process large volumes of unstructured requirements, identify latent dependencies, and offer predictive insights that traditional methods cannot. This transition addresses the need for automation, precision, and adaptability in CIA, especially in domains characterised by linguistic variability and high change frequency.

2.8. Machine Learning in CIA

The integration of ML techniques into Requirements Engineering has emerged as a significant advancement in the software development lifecycle, particularly in optimizing the extraction, analysis, and prediction of requirements-relevant knowledge. Classification and regression, two key tasks in supervised learning, are foundational to this integration. Classification involves predicting discrete labels, such as identifying whether a requirement is likely to change or remain stable, while regression predicts continuous values, such as estimating the magnitude of a change’s impact. These tasks are pivotal in automating decision-making processes in CIA.

In the early stages of software development, system or business analysts must meticulously capture and document software requirements, which serve as critical input

to the Software Requirements Specification (SRS) document. Given the importance of generating a comprehensive and accurate SRS, optimizing the knowledge extraction process is paramount (Sandhu et al. 2015). ML techniques have been instrumental in enhancing the efficiency and accuracy of this process, particularly in dealing with the vast amount of data contained within requirements documents, which are often written in natural language (NL). The challenge of transforming these NL requirements into structured formats amenable to automated analysis has led to the development of various NLP techniques (Arora et al. 2019). When combined with ML, these techniques enable automation in requirements analysis, significantly reducing manual effort and improving the precision of outcomes (Li et al. 2018).

The application of ML in RE encompasses a range of tasks, including requirements traceability, ambiguity management, and the generation of test cases (Holzinger et al. 2018). By applying learning algorithms to datasets derived from previous projects, ML models can be trained to recognize patterns and predict outcomes, thereby supporting requirements analysts in their decision-making processes. This automation is particularly valuable in large-scale projects where the sheer volume of requirements can overwhelm traditional manual analysis methods (Lwakatare et al. 2019).

To provide a solid foundation for the algorithms applied in this research, the following subsections present the mathematical background of key ML methods, including Decision Trees, Random Forests, Support Vector Machines (SVMs), and Neural Networks. These methods represent widely adopted approaches to classification and regression tasks in CIA.

2.8.1. Decision Trees

Decision Trees are supervised learning algorithms that classify data by splitting it into subsets based on feature values (Boutaba et al. 2018). They use a tree-like structure where each internal node represents a decision (split), and each leaf node represents a class label or outcome (Tufail et al. 2023).

$$IG(D, A) = H(D) - \sum_{v \in A} \frac{|D_V|}{|D|} H(D_V) \qquad (1)$$

Where $\mathrm{H(D)} = -\sum_{\{i=1\}}^{k} p_{i\backslash \log_{2(p_I)}}$ (2) is the entropy of dataset D, and p_I is the probability of class i.

- Gini Index:

Alternatively, the Gini Index measures impurity(Quinlan 1986):

$$\mathrm{Gini(D)} = 1 - \sum_{\{i=1\}}^{k} p_i^2. \qquad (3)$$

The algorithm selects the split with the highest Information Gain or lowest Gini Index to grow the tree.

2.8.2. Random Forests

Random Forests are ensemble learning methods that improve the robustness of Decision Trees by using Bootstrap Aggregation (Bagging) to train multiple trees and aggregate their predictions (Breiman 2001).

- Bagging:

  Random samples D_i are drawn with replacement to create diverse training sets. The final prediction is the aggregate of individual trees:

$$f_{\{ensemble\}(x)} = \frac{1}{T}\sum_{\{t=1\}}^{T} f_{t(x)} \qquad (4)$$

where T is the number of trees.

- Feature Selection:

At each split, a random subset of features is chosen to reduce correlation between trees, improving generalization and reducing overfitting.

2.8.3. Support Vector Machines (SVMs)

Support Vector Machines are powerful algorithms that find the optimal hyperplane to separate classes in a high-dimensional space (Christopher J.C. Burges 1998).

- Objective Function:

For linearly separable data, SVMs maximize the margin between classes:

$$\text{minimize } \left(\frac{1}{2}\right)\left|\left|\mathrm{w}\right|\right|^2 \text{subject to } \mathrm{y_i}(\mathrm{w}\cdot\mathrm{x_i}+\mathrm{b})\geq 1 \qquad (5)$$

where w is the weight vector, b is the bias term, and y_i is the label.

- Kernel Trick:

For non-linearly separable data, kernels map inputs to higher-dimensional spaces. A common kernel is the Radial Basis Function (RBF):

$$\mathrm{K}(\mathrm{x_i},\mathrm{x_j})=\exp\left(-\gamma\left|\left|\mathrm{x_i}-\mathrm{x_j}\right|\right|^2\right) \qquad (6)$$

2.8.4. Logistic Regression

Logistic Regression is a supervised learning algorithm used for binary and multi-class classification tasks. Unlike linear regression, which predicts continuous values, logistic regression predicts probabilities, transforming the output using the logistic function.

- **Logistic Function:**

  The logistic function is defined as:

$$P(y=1|x)=\frac{1}{1+e^{-z}} \qquad (7)$$

where z=w·x+b , w is the weight vector, x is the input features, and b is the bias term.

- **Log Loss Function:**

  Logistic Regression optimizes the log-loss (cross-entropy) function to find the best weights w and bias b:

$$L=-\frac{1}{N}\sum_{i=1}^{N}\left[y_i\log\left(P(y=1|x_i)\right)+(1-y_i)\log\left(1-P(y=1|x_i)\right)\right] \qquad (8)$$

where y_i is the actual class label for sample i, and $P(y=1|x_i)$ is the predicted probability for the positive class.

Logistic Regression works well for linearly separable data but may struggle with non-linear relationships unless extended using techniques like polynomial feature transformations or kernel methods (Tufail et al. 2023).

2.8.5. Gaussian Naive Bayes (NB)

NB is a probabilistic classifier based on Bayes' Theorem. It assumes that features are conditionally independent given the class label and follow a Gaussian (normal) distribution.

- **Bayes' Theorem:**

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)} \qquad (9)$$

where $P(C_k|x)$ is the posterior probability of class, $P(x|C_k)$ is the likelihood, $P(C_k)$ is the prior probability of class $P(x)$ is the evidence.

- **Likelihood with Gaussian Distribution:**

  For Gaussian NB, the likelihood $P(x|C_k)$ is modeled as:

$$P(x|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}} \qquad (10)$$

where μ_k and σ_k^2 are the Mean and variance of feature x for class C_k.

- **Decision Rule:**

  The class prediction is made by selecting the class with the highest posterior probability:

$$C_{\text{predicted}} = \arg\max_{C_k} P\,(C_k|x) \qquad (11)$$

Gaussian NB is particularly effective for datasets where the features follow a normal distribution. Its simplicity and efficiency make it a popular choice for text classification, spam detection, and other real-world problems(Boutaba et al. 2018; Tufail et al. 2023) .

2.9. Natural Language Processing in CIA

NLP has become an increasingly important tool in the domain of CIA, particularly given the challenges associated with managing and analyzing software requirements, which are often documented in natural language. Software requirements are typically expressed in natural language due to its flexibility and ease of use, making them

accessible to both technical and non-technical stakeholders. However, this flexibility also introduces variability, ambiguity, and potential inconsistencies into the requirements, complicating the process of CIA.

The inherent ambiguity and complexity of natural language pose significant challenges for automated analysis, making NLP a critical component in enhancing the precision and effectiveness of CIA processes. NLP techniques are specifically designed to address these challenges by enabling the automated extraction, interpretation, and processing of natural language requirements.

NLP can be employed to parse and analyze the textual content of requirements documents to identify key entities, relationships, and dependencies. This capability is particularly valuable in CIA, where understanding the relationships between different requirements is crucial for predicting the impact of changes. By using techniques such as part-of-speech tagging, named entity recognition, and dependency parsing, NLP helps structure and clarify the relationships within the requirements, making them more amenable to further analysis. This structured analysis is essential for ensuring that changes are accurately assessed and that their impacts are fully understood before implementation.

This research employs two widely used NLP libraries, SpaCy and CoreNLP, which are instrumental in implementing robust solutions for analyzing software requirements. The following subsections explore the technical details and specific functionalities of SpaCy and CoreNLP and their relevance to the tasks undertaken in this study.

2.9.1. CoreNLP

Stanford CoreNLP is a comprehensive NLP toolkit developed by the Stanford NLP Group. It offers a wide range of linguistic analysis tools, including tokenization, sentence splitting, part-of-speech tagging (POS), named entity recognition (NER), lemmatization, dependency parsing, and coreference resolution. CoreNLP's strength lies in its ability to handle complex syntactic and semantic analysis, making it highly suitable for

understanding the grammatical structures present in software requirements (Manning et al. 2014). Key features of CoreNLP include:

- **Tokenization:** CoreNLP provides robust tokenization capabilities that handle a wide variety of text inputs, including multi-word expressions and special symbols.
- **Part-of-Speech Tagging:** CoreNLP's POS tagging module uses sophisticated models to ensure high accuracy across diverse datasets.
- **Dependency Parsing:** CoreNLP employs advanced algorithms, including universal dependency representations, to analyze syntactic structures. It uses graph-based dependency parsing. The parser constructs a dependency tree T by maximizing the sum of scores for all edges (Pipit Muliyah, Dyah Aminatun, Sukma Septian Nasution, Tommy Hastomo, Setiana Sri Wahyuni Sitepu 2020):

$$T* = \ argmax\ (T\ \in\ T)\Sigma\ s(h,m) \qquad (12)$$

  - T: The set of all valid dependency trees.
  - (h,m): An edge from the head hhh to the modifier mmm.
  - $s(h,m)$: A scoring function for each edge.
- **Sentiment Analysis and Coreference Resolution:** Beyond basic NLP tasks, CoreNLP offers features such as sentiment analysis and coreference resolution, enabling more nuanced analysis of text. Coreference resolution in CoreNLP often uses a probabilistic model to determine whether two mentions $m1$ and , $m2$ refer to the same entity (Lee et al. 2013). This can be represented as:

$$P(coref \mid m1,m2)\ =\ \sigma(w\ \cdot\ \varphi(m1,m2)) \qquad (13)$$

  - σ: The sigmoid function.
  - w: The weight vector learned during training.
  - $\varphi(m1,m2)$: The feature vector encoding attributes of $m1$ and $m2$.

2.9.2. SpaCy

SpaCy is an open-source NLP library designed for fast, efficient processing of large volumes of text. Its pipeline architecture allows easy customization, enabling users to add components as needed. SpaCy excels in named entity recognition (NER) and

provides pre-trained models for multiple languages, making it ideal for global NLP applications. Its high processing speed and flexibility make it an excellent choice for text preprocessing and feature extraction. Key features of SpaCy include:

- **Tokenization:** SpaCy employs a rule-based tokenizer to segment text into tokens, accounting for language-specific nuances like abbreviations and contractions.
- **Part-of-Speech Tagging:** Using state-of-the-art statistical models, SpaCy assigns grammatical roles to each token, facilitating syntactic analysis.
- **Dependency Parsing:** SpaCy builds dependency trees to represent grammatical relationships between words in a sentence, enabling a deeper understanding of sentence structure (Pipit Muliyah, Dyah Aminatun, Sukma Septian Nasution, Tommy Hastomo, Setiana Sri Wahyuni Sitepu 2020). It uses transition-based dependency parsing algorithms, which can be represented mathematically as follows:

The parser operates in a state-transition system:

$$T = (C, A, t0, Tf) \qquad (13)$$

- C: The set of all possible configurations.
- A: The set of actions (e.g., Shift, Reduce, Left-Arc, Right-Arc).
- $t0$: The initial configuration of the parser.
- Tf : The set of terminal configurations.

The algorithm transitions between states using a learned scoring function $s(c, a)$, where $c \in C$ and $a \in A$ .The parser selects actions aaa to maximize the score:

$$a* = argmax\ (a \in A)s(c, a) \qquad (14)$$

- **Named Entity Recognition (NER):** SpaCy uses pre-trained models to extract named entities such as dates, quantities, and system components from text. NER in SpaCy relies on sequence labeling tasks modeled using Conditional Random Fields (CRFs) (Song, Zhang & Huang 2019). A CRF assigns a probability to a sequence of labels Y given a sequence of tokens X:

$$P(Y \mid X) = \exp\ (\Sigma\ \varphi(y\_(i-1), y\_i, X)) \ / \ \Sigma\_Y' \exp\ (\Sigma\ \varphi(y'\_(i-1), y'\_i, X)) \quad (15)$$

Here, $\varphi(y_{i-1}, y_i, X)$is the feature function that scores the compatibility of the label sequence with the input sequence.

One of the major challenges in CIA is identifying hidden dependencies, relationships between requirements that are not explicitly stated through keywords or syntactic structure. Traditional approaches struggle with these implicit links, particularly when requirements are phrased differently but convey semantically related intentions. NLP techniques directly address this limitation. For example, dependency parsing enables the construction of grammatical trees that expose subject–verb–object relations, helping analysts detect when two requirements act upon the same concept in different forms. Likewise, NER supports terminology alignment by extracting and normalising domain-specific terms across diverse requirement expressions. Semantic Role Labeling further enhances this by framing requirements around action agent object structures, revealing deep semantic similarities even when vocabulary differs. Together, these NLP techniques contribute to effective dependency mapping, allowing the framework to uncover latent links between requirements that would be missed by surface-level analysis alone. This capability is especially critical in large-scale, heterogeneous systems where implicit dependencies are common and costly to overlook.

By combining these techniques, our framework goes beyond surface-level text matching. Dependency parsing uncovers grammatical links, SRL reveals deeper semantic connections, and NER aligns domain-specific terms, together forming a robust basis for mapping both explicit and hidden requirement dependencies.

2.10. BEIR: Benchmarking Information Retrieval

The Benchmarking Information Retrieval (BEIR) framework is a comprehensive platform designed to evaluate information retrieval (IR) systems across diverse datasets and tasks. BEIR incorporates various retrieval approaches, including lexical retrieval, dense retrieval, and hybrid methods, enabling the assessment of IR models' performance in handling a wide range of scenarios. It is particularly relevant in NLP for evaluating semantic search and similarity-based applications (Thakur et al. 2021).

This section outlines the mathematical foundation of the BEIR framework and its relevance to CIA. The following concepts are integral to BEIR's methodologies: lexical retrieval with BM25, dense retrieval with vector embeddings, and hybrid models combining the two approaches.

2.10.1. Lexical Retrieval with BM25

BM25 is a probabilistic scoring function widely used for lexical retrieval. It ranks documents based on the frequency of query terms, adjusting for document length and term saturation (Robertson & Zaragoza 2009; Thakur et al. 2021). The BM25 scoring function is defined as:

$$Score(q,d) = \Sigma \left[IDF(t) * \frac{\left(f(t,d) * (k1+1)\right)}{\left(f(t,d) + k1 * \left(1 - b + b * \left(\frac{|d|}{avgdl}\right)\right)\right)} \right] \quad (16)$$

Where:

- q: Query terms.
- d: Document.
- t: A term in the query q.
- $f(t,d)$: Frequency of term t in document d.
- $f(t,d)$: Length of document d.
- $avgdl$: Average document length across the corpus.
- $k1$: Hyperparameter controlling term frequency saturation (typically $k1$ =1.2 or $k1$ =2.0).
- b: Hyperparameter controlling the impact of document length normalization (commonly b =0.75).

The Inverse Document Frequency (IDF) measures the importance of a term and is given by:

$$IDF(t) = \log\left[\frac{(N - n(t) + 0.5)}{(n(t) + 0.5)}\right] \quad (17)$$

Where:

- N: Total number of documents in the corpus.
- $n(t)$: Number of documents containing the term t.

BM25 excels in capturing exact matches between query terms and documents while adjusting for variations in term frequency and document length.

2.10.2. Dense Retrieval Using Bi-Encoders

Bi-Encoders are used for efficient dense retrieval by encoding the query and documents independently into a shared embedding space. This method allows for rapid computation of similarities between queries and a large corpus of documents using vector operations (Thakur et al. 2021).

A neural network f encodes the query q and document d into dense vectors q and d , respectively:

$$q = f(q), d = f(d) \qquad (18)$$

Where:

- $f(q)$: Embedding of the query.
- $f(d)$:Embedding of the document.

The similarity between the query and document embeddings is computed using cosine similarity:

$$Sim(q,d) = \frac{(q \bullet d)}{(||q|| * ||d||)} \qquad (19)$$

- $q \bullet d$: Dot product of the query and document embeddings.
- $||q||$ and $||d||$: Magnitudes (norms) of the respective vectors.

The documents are ranked based on their similarity scores:

$$Rank(q) = argsort\big(-Sim(q,d_i)\big) \qquad (20)$$

Where $argsort$ sorts the documents d_i in descending order of similarity (Karpukhin et al. 2020).

2.10.3. Re-Ranking with Cross-Encoders

After retrieving a subset of candidate documents using a Bi-Encoder, Cross-Encoders refine the rankings by jointly encoding the query and each candidate document. This method captures more nuanced interactions between query and document terms (Thakur et al. 2021).

A Cross-Encoder takes the concatenation of the query q and a document d as input and produces a relevance score $s(q, d)$:

$$s(q, d) = f_{cross([q;\, d])} \quad (21)$$

- f_{cross}: A neural network (e.g., BERT or RoBERTa) trained for relevance scoring.
- $[q;\ d]$: Concatenation of the query and document as input to the model.

The Cross-Encoder computes a scalar score indicating the relevance of the document to the query. The final ranking is determined by sorting the candidate documents based on their relevance scores:

$$Rank_{cross(q)} = argsort\big(-s(q, d_i)\big) \quad (22)$$

Where:

- $argsort$ sorts the documents $d_i$in descending order of their relevance scores.

2.11. Large Language Models

Large Language Models (LLMs) have revolutionized the field of NLP by enabling systems to perform complex language tasks with remarkable accuracy and contextual understanding. These sophisticated AI systems, such as GPT, BERT, and Phi 3.5, are designed to process and generate human-like text based on user prompts, demonstrating capabilities in reasoning, question answering, summarization, and creative writing (White et al., 2023). LLMs are particularly impactful in tasks requiring high degrees of linguistic sensitivity and contextual awareness, such as requirements CIA.

At their core, LLMs operate by modeling the probability distribution of language. Given an input sequence $x = (x_1\ x_2, \dots, x_n)$, an LLM estimates the likelihood of the next token x_{n+1} based on the conditional probability:

$$P(x\_\{n+1\} \mid x\_1, x\_2, \dots, x\_n) \qquad (23)$$

This probability is learned through large-scale training on diverse text corpora, allowing the model to capture both syntactic and semantic relationships in language. The model's goal is to minimize the cross-entropy loss during training, defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log P\left(x_i \mid x_{<i}\right) \qquad (24)$$

where N is the total number of tokens in the dataset, and $P\left(x_i \mid x_{\{<i\}}\right)$ is the predicted probability of the iii-th token given its preceding context.

2.11.1. Transformer Architecture

LLMs are underpinned by transformer-based architectures, which rely on self-attention mechanisms to process sequential data effectively. Unlike traditional models like RNNs, transformers process sequences in parallel, enabling greater scalability and precision. Key components include:

- **Self-Attention Mechanism**: Each token attends to all other tokens in the sequence to compute contextual relevance. The attention scores are calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \qquad (25)$$

where Q (query), K (key), and V (value) are projections of the input embeddings, and d_k is the dimensionality of the key vectors. This mechanism enables the model to capture long-range dependencies (Vaswani et al. 2017).

- **Multi-Head Attention:** By employing multiple attention heads, the model can focus on different aspects of the input simultaneously, enhancing its ability to understand complex patterns (Vaswani et al. 2017).

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O \qquad (26)$$

- **Feedforward Layers:** Position-wise feedforward networks apply non-linear transformations to each token independently:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \quad (27)$$

These layers enhance the model's ability to learn complex features from the input data.

- **Positional Encoding:** Since transformers lack inherent sequence ordering, positional encodings are added to input embeddings to inject positional information (Vaswani et al. 2017).

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \quad (28)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \quad (29)$$

2.12. RAG Model

The RAG model represents a significant advancement in NLP by integrating retrieval and generation components. This approach allows models to draw upon both pre-trained knowledge and real-time information to respond more accurately to user queries. Unlike standalone generative models, RAG systems can access external knowledge bases, making their responses more contextually relevant and precise(Gao et al. 2023).

A. Information Retrieval in RAG Systems

Information retrieval is a key component of the RAG model, and it is responsible for locating documents that provide context and support for the input query. Classical IR techniques, such as BM25, rank documents based on their term frequency-inverse document frequency (TF-IDF) scores. These techniques are effective for quickly identifying the most relevant documents within a large corpus. More modern approaches use vector embeddings and similarity searches to enhance retrieval precision.

In RAG systems, the retriever acts as a bridge between static pre-trained model knowledge and dynamic, real-time information. By leveraging indexed data, the retriever provides the generative model with the most relevant pieces of information, enriching the generated output (Gao et al. 2023).

The retrieval component identifies the most relevant documents D from a large corpus c for a given input query q. The goal is to maximize the conditional probability of retrieving relevant documents given the query:

$$P(D \mid q) = \prod P(d\_i \mid q) \qquad (30)$$

Where:

- D = {d_1, d_2, ..., d_k}: Set of retrieved documents.
- $P(d\_i \mid q)$: Relevance score for document d_i, computed using similarity metrics (e.g., cosine similarity of embeddings).

Document embeddings d_i and query embeddings q are generated using dense retrieval models like Bi-Encoders:

$$P(d \mid q) \propto Sim(q, d) = (q \bullet d) / (||q|| * ||d||) \qquad (31)$$

Where:

- q: Query embedding.
- d: Document embedding.
- $||q||$and $||d||$: Magnitudes of the embeddings.

B. Generative Language Modeling

Generative language models, such as Phi-3.5, are based on transformer architectures that use attention mechanisms to understand and generate text. These models are pre-trained on extensive corpora to learn linguistic patterns, enabling them to produce human-like text based on input prompts (White et al. 2023). However, without real-time data integration, they are limited by their training cut-off and lack of specific domain knowledge.

By incorporating retrieved documents into the input, the generator can dynamically access external knowledge, improving the output's specificity and contextual accuracy. This integration is critical for applications like CIA, where changes in requirements need to be analyzed with up-to-date contextual understanding.

The generation component uses a conditional language model to generate output y based on the query q and the retrieved documents D. The generative process maximizes the likelihood of the output sequence y = { y _1, y _2, ..., y _n}:

$$P(y \mid q, D) = \prod P(y\_t \mid y\_ < t, q, D) \quad (32)$$

Where:

- y_t : The t token in the generated sequence.
- $y_{<t}$: The sequence of tokens generated before y_t .
- D: Retrieved documents conditioning the generation.

By incorporating retrieved documents into the input, the generator produces responses that reflect both pre-trained knowledge and real-time contextual information.

C. Joint Objective

The RAG model combines the retrieval and generation components to optimize the joint probability of the output y and the retrieved documents D given the query q:

$$P(y, D \mid q) = P(D \mid q) * P(y \mid q, D) \quad (33)$$

The final objective is to maximize this joint probability.

During training, RAG optimizes the following loss function using Maximum Likelihood Estimation (MLE):

$$\boldsymbol{L = -\sum \log \sum P(D \mid q) * P(y \mid q, D)} \quad (34)$$

D. Fine-Tuning

In fine-tuning, the retriever and generator are trained jointly or sequentially:

- Retriever Fine-Tuning: Adjusts $P(D \mid q)$ to improve retrieval quality.

- Generator Fine-Tuning: Updates $P(y \mid q, D)$ to better synthesize responses based on retrieved documents.

2.13. Vector Databases

Vector databases are specialized data management systems designed to store, retrieve, and query high-dimensional vectors efficiently. Unlike traditional relational databases focusing on structured data, vector databases are optimized for managing embedding vectors derived from text, images, or other data types. These embeddings represent data in a mathematical form that captures semantic relationships, making vector databases ideal for applications requiring similarity searches, such as RAG systems.

2.14. Mathematical Heuristics for Optimizing Similarity-Based Analysis

Mathematical heuristics play a significant role in computational systems, particularly in scenarios where optimizing performance and resource utilization is critical. These heuristics are especially relevant for tasks involving similarity-based analysis, such as those in NLP and BEIR-based frameworks. The approaches in this research compute similarity scores between sentence pairs based on various linguistic features and contextual information, often leading to computational overhead when processing large datasets. By applying mathematical heuristics, the research streamlines this process, ensuring both efficiency and accuracy.

Heuristics operate through approximate methods and predefined rules that guide decision-making without requiring exhaustive computation. While they do not guarantee a globally optimal solution, they provide a practical and effective way to identify high-priority elements from a dataset. For similarity-based analysis, heuristics enable prioritization and filtering of sentences based on their relevance, reducing the need to process less significant data points.

Three commonly applied heuristics in this research are score thresholding, significant drop detection, and relative score proportionality.

- **Score Thresholding** involves setting a baseline threshold to exclude sentences with similarity scores below a certain level. This baseline is calculated based on the score distribution across the dataset to ensure only sentences with meaningful similarity are retained for further analysis.
- **Significant Drop Detection** identifies substantial drops in similarity scores among ranked sentences. A sharp decline often indicates the boundary where semantic alignment diminishes, helping to distinguish between sentences closely aligned in content and those with reduced relevance.
- **Relative Score** Proportionality compares each sentence's similarity score to the highest score in the dataset. Sentences with scores below a predefined proportion (e.g., 50%) of the highest score are excluded to focus on the most semantically relevant results.

Prior research has demonstrated the effectiveness of such heuristics in various domains. For instance, thresholding techniques have been used in information retrieval systems to improve relevance ranking, while significant drop detection has been applied to enhance clustering methods by identifying natural boundaries in data. Similarly, proportionality-based heuristics have been employed in ranking systems to ensure high-priority results are emphasized.

This research builds on these established techniques to address the challenges of sentence similarity analysis in the NLP and BEIR-based frameworks. By integrating these heuristics, the study not only improves computational efficiency but also ensures the semantic integrity of the results, providing a robust foundation for further applications in software requirements engineering.

2.15. Applied Evaluation Metrics

In this research, the performance of different models for CIA is evaluated using a comprehensive set of metrics, each providing insights into specific aspects of model performance. These metrics are crucial for assessing the effectiveness and reliability of the proposed solutions in identifying impacted requirements. This section describes the evaluation metrics employed in this study, along with their mathematical formulations.

2.15.1. Precision

Precision measures the proportion of correctly identified impacted requirements to the total number of predicted impacted requirements. It evaluates the accuracy of the model in minimizing false positives. Precision is mathematically defined as (Powers 2020):

$$Precision = \frac{TP}{TP+FP} \quad (35)$$

Where:

- True Positives (TP): Correctly predicted impacted requirements.
- False Positives (FP): Incorrectly predicted impacted requirements.

2.15.2. Recall

Recall, also known as sensitivity, quantifies the model's ability to identify all relevant impacted requirements. It assesses the completeness of the model by minimizing false negatives. Recall is expressed as (Powers 2020):

$$Recall = \frac{TP}{TP+FN} \quad (36)$$

Where:

- False Negatives (FN): Relevant impacted requirements that were not predicted by the model.

2.15.3. F1 Score

The F1 Score combines precision and recall into a single metric, providing a harmonic mean. It balances the trade-off between precision and recall, particularly useful when both metrics are equally important. The F1 Score is given by (Powers 2020; Takahashi et al. 2022):

$$F1\ Score = 2 * \frac{Precision\ .Recall}{recision+\ Recall} \quad (37)$$

2.15.4. Mean Reciprocal Rank (MRR)

MRR evaluates the ranking quality of the first correct impacted requirement. It measures how quickly the most relevant impacted requirement is surfaced. MRR is calculated as (Yacouby & Axman 2020):

$$MMR = \frac{1}{N}\sum_{\{i=1\}}^{N}\frac{1}{Rank_i} \quad (38)$$

Where:

- $Rank_i$: Rank position of the first correct prediction for the i query.

2.15.5. Normalized Discounted Cumulative Gain (NDCG)

The NDCG measures the ranking quality of predicted impacted requirements, giving higher weights to items ranked at the top. It is expressed as (Wang et al. 2013):

$$NDCG = \frac{DGG}{IDCG} \quad (39)$$

$$DGG = \sum_{i=1}^{p}\frac{2^{rel_i-1}}{log_2(i+1)} \quad (40)$$

Where:

- rel_i: Relevance score of the i item.
- p: Number of predicted items.

2.15.6. Partial Credit

The Partial Credit metric assigns a score to predictions that are partially correct or closely related to the ground truth. It provides a nuanced evaluation of the model's performance, especially in scenarios with linguistic variability or context-dependent predictions. A common approach is to assign a score between 0 and 1 based on the degree of similarity or relevance between the predicted answer P and the ground truth G (Persson 2023). One such formula is :

$$\text{Partial Credit (PC)} = \frac{\text{Similarity}(P,G)}{\text{Maximum Possible Similarity}} \quad (41)$$

2.15.7. Precision@K and Recall@K

Precision@K and Recall@K evaluate the model's performance within the top K results. They are particularly useful for ranking-based evaluations in scenarios with a high number of potential matches (Liu et al. 2016; Patel, Tolias & Matas 2022). These metrics are defined as:

$$\text{Precision@K} = \frac{\text{TP@K}}{K} \tag{42}$$

$$\text{Recall@K} = \frac{\text{TP@K}}{\text{Total Relevant Items}} \tag{43}$$

2.15.8. Mean Average Precision (MAP)

The Mean Average Precision (MAP) provides an aggregated measure of precision across all relevant results. It is computed as (Henderson & Ferrari 2017):

$$\text{MAP} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|\text{R}_i|}\sum_{k=1}^{|\text{R}_i|}\text{Precision@k} \tag{44}$$

Where:

- N: Total number of queries.
- $|\text{R}_i|$: Number of relevant items for the iii-th query.

2.16. Related Literature Reviews

In this section, we present a sample from the existing SLRs on RE that specifically focus on ML and NLP in RE. Some of the findings in these studies are not specific to RE and have covered studies in software engineering (e.g. Haq et al. (Haq et al. 2019)).

Sufian et al. (Sufian et al. 2019) conducted an SLR on software requirements prioritization techniques. They reviewed 33 studies from 2009 until 2017. They have covered 40 different requirement prioritizations techniques, among these, one tool uses ML classification to identify requirements. Dermeval et al. (Dermeval et al. 2016) performed a systematic review on the applications of ontologies in RE. They reviewed 67 papers from 2007 to 2013. They concluded that ontologies can potentially be used to deal with several RE issues (e.g. integration between requirements and software

architecture, and requirements communication). Their findings revealed that most studies focused on textual requirements analysis, which involve the use of ML and NLP techniques. Binkhonain et al. (Binkhonain & Zhao 2019) reviewed the literature in ML algorithms for the identification and classification of non-functional requirements. The study considered 24 papers from 2008 to 2019. Águila et al. (del Águila & del Sagrado 2016a) performed a literature review to describe the state of the art in Bayesian networks for enhancement of RE. The authors reviewed 20 papers from 1999 to 2013. Haq et al. (Haq et al. 2019) conducted an SLR that identified the use of the expert system in RE process. They reviewed 22 papers from 1986 to 2019. They concluded that ML showed significant results in supporting RE activities. At the same time with our conduct of this mapping study, Zhao et al. (Zhao et al. 2020) have performed a systematic mapping study on NLP for RE. The authors identified 404 relevant primary studies from 1983 to 2019 concerning the NLP technologies used in RE.

2.17. Mapping Study

Although systematic literature reviews (SLRs) of many aspects of RE have been published in the last decade (e.g. (Ghozali et al. 2019) (Alsanoosy, Spichkova & Harland 2019) (Horkoff et al. 2019)), none of them focuses specifically on reviewing empirical studies of ML applications in RE. We are thus motivated to conduct a mapping study to identify, analyze and summarize the advances in the applications of ML in RE and to identify the current state of the art. This review also allows us to identify areas that still need more research and determine the trends of future studies.

The mapping study presented in this chapter provides a systematic and structured review of the literature related to software requirements engineering and CIA up to the year 2020. The primary purpose of this study is to gain a comprehensive understanding of the existing body of work in these domains. By systematically identifying, categorising, and analysing key research contributions, methodologies, and challenges, the study highlights research trends, strengths, and gaps. This ensures that the current research aligns with established knowledge while addressing overlooked challenges, laying the foundation for the proposed research framework.

2.17.1. Scope and Limitations

The mapping study focuses on literature published up to 2020, covering works that explore various approaches to managing requirements changes and conducting CIA. This includes methods involving dependency analysis, traceability techniques, and automation through NLP and ML. However, it is important to note that newer advancements, such as Retrieval-Augmented Generation (RAG) systems and Beir-based approaches, are excluded as they were introduced after the defined timeframe. While these emerging techniques offer valuable solutions, they fall beyond the scope of this study.

2.17.2. Mapping Study Planning and Execution

To conduct our mapping study, we followed the guidelines and the procedures of the evidence-based software engineering paradigm (Kitchenham, B., Budgen, D., & Brereton 2016). The structure of this review process included defining research questions, conducting a search strategy, making a list of related studies, applying inclusion and exclusion criteria, developing snowball and manual search for additional relevant studies, executing quality assessment, data extraction, data synthesis, and analysis.

2.17.3. Search Strategy and Data Sources

Our strategy is composed of two different iterations: primary and secondary search strategy. In the primary phase, we identified the main search terms based on our research questions. After applying the alternative spelling and synonyms, developed search terms were formulated by using Boolean operators (AND, OR, etc.) with search keywords to define inclusion and exclusion criteria at the title and abstract. We investigated the following two major terms to execute against the title and abstract for our searching process: (1) Machine Learning, (2) Software Requirements. The below query shows our identified alternative terms and their concatenation to make our search string which we applied to the title and abstract.

ON TITLE/ ABSTRACT: (("Machine Learning*" OR "ML*") AND ("Software Requirements" OR "Requirements engineering" OR "Requirements elicitation" OR "Requirements analysis" OR "Requirements specification" OR "Requirements modeling" OR "Requirements documentation" OR "Requirements validation" OR "Requirements Management"))

The search string was modified to fit the format of different databases. The modified search strings applied in the form of automatic searches of selected electronic databases and conference proceedings including IEEE Xplore, ACM, Science Direct, SpringerLink, ProQuest, and Scopus. To make sure that we did not miss any important and relevant papers, we also executed our search query manually in reputable and relevant conference proceedings, journals, and workshops websites one by one due to their importance in the respective communities. We prepared a replication package including the protocol and the details of search strategy and results. A list of all customized search strings can be found online in our published replication package: https://zenodo.org/records/5036218

2.17.4. Study Selection Criteria

In the primary stage, these inclusion criteria were applied:

- Articles that are related to our research questions
- Papers that are based on the empirical research method
- Conference, Journal and Workshop papers

The papers with the following criteria were excluded:

- Articles that were not published in English
- Articles that are not in full text
- Articles that are reviews or secondary studies
- Reports, books, book chapters, thesis, general articles, dissertations, editorials, and position papers
- Duplicate results with the same or similar contents from the same authors
- Articles published before 2010

The main reason for the starting date of 2010 is that there has been a surge of publications on ML in RE in the last decade and we were interested in more recent work. We also reviewed random samples of relevant papers published before 2010. This review did not have any significant impact on our findings.

In this step, we read the titles and abstracts of all studies selected in the primary search. We excluded some papers based on the criteria. So, from 5158 papers, only 905 papers went through the secondary search step. Table 2.1 shows the number of selected articles in the primary study before and after applying the exclusion criteria. Following the search strategy, we conducted a manual search against the top journals of REJ (Requirements engineering journal), ESEM (Empirical Software Engineering), TOSEM (Software Engineering and Methodology), TSE (Transactions on Software Engineering), ASE (Automated Software Engineering) and IST (Information and Software Technology); as well as conferences and workshops including International requirements engineering conference (RE), Requirements Engineering: Foundation for Software Quality (REFSQ), International conference on software engineering (ICSE), International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), Workshop on NLP for Requirements Engineering & NLP tool Showcase (NLP4RE) and IEEE International Workshop on Artificial Intelligence for Requirements Engineering (AIRE) from 2010 to April 2020. As a result, 10 new references were added to our list from these venues.

Table 2.1.The Number of Resulted Articles

Database	Weblink	#After Applying Selection criteria	#Final Results
ACM Digital Library	http://dl.acm.org	72	11
IEEE Explore	http://ieeexplore.ieee.org/Xplore	258	10
Science Direct	http://www.sciencedirect.com	3	1
SpringerLink	https://link.springer.com	398	3
ProQuest	http://www.proquest.com	83	20

Scopus	https://www.scopus.com	78	15
Total			60

In the secondary search phase, we reviewed all the papers identified from the primary search. If a paper was found to be relevant, the mentioned inclusion and exclusion criteria were applied to filter out irrelevant ones. We then read complete papers to make a final decision on their inclusion or exclusion in our mapping study. To complete the selection task, we performed snowballing procedure developed by Wohlin (Wohlin 2014). Based on the Wohin's guideline we applied backward snowballing iteratively. The main purpose of this iteration procedure was to find more relevant studies to include. To do that we explored the reference list of the selected studies in backward snowballing and examined the title, abstract, publication venue, author information, and full text in order to exclude papers that do not fulfill our criteria. This iteration continued until no new studies were found. Finally, we collected 12 new studies from reference snowballing.

2.17.5. Quality Assessment criteria

All selected articles (82 studies: 60 papers from primary search, 12 new studies from snowballing, and 10 new papers from a manual search of journals) were assessed for their quality to ensure that all outcomes will add a valuable contribution to our mapping study. We assessed the quality of selected studies by following steps (Kitchenham, B., Budgen, D., & Brereton 2016):

Step 1: Evaluate article quality - The quality assessment checklist developed by Kitchenham (Kitchenham, B., Budgen, D., & Brereton 2016), was independently applied to all 82 primary studies. By applying the criteria, four articles did not pass the minimum score of 50%, so this step resulted in 78 studies.

Step 2: Evaluate publisher quality - The quality of each publisher was assessed by ERA (Excellence of Research in Australia) ranking of 2018. This evaluation framework is meant to give government, industry, business, and the wider community assurance of the excellence of research conducted in Australian higher education institutions. The

goal of this assessment step was to generate an extensive overview of the kind and quality of the resulting papers.

Total results: After the quality assessment, a total number of 65 papers were selected for this mapping study. Figure 2.2 illustrates the overview of the primary studies selection process. The full bibliography of these 65 studies can be found in our replication package.

2.17.6. Data Extraction

To manage citations and references of outcomes, we used Mendeley as a reference manager. The information below was collected from the results:

- Study type (journal, conference, workshop)
- Name of journal, conference, or workshop
- ERA rank of conference, journal, or workshop
- Study aims and objectives
- Title of the article
- Authors and Publisher details
- Publication year
- Full citation
- Location (the country where it is situated)

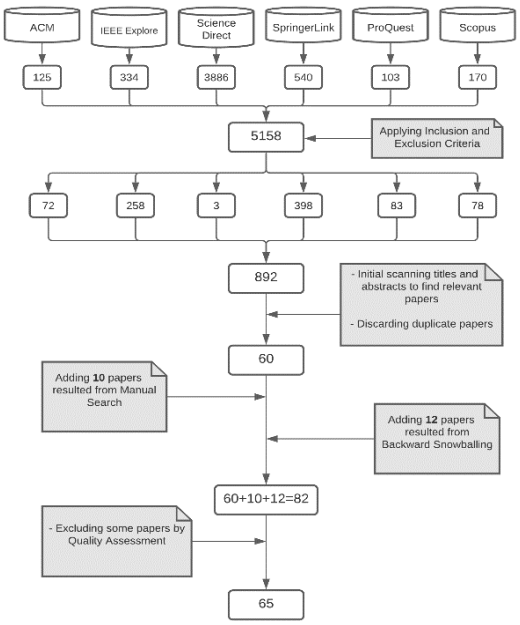

Figure 2.2.Selection of The Primary Studies

- Research method
- How data was collected and analyzed
- The study quality assessment
- Relevance to RQ1, RQ2, RQ3 or RQ4

2.17.7. Data Synthesis and Analysis

The data synthesis is based on answering our four research questions. We conducted thematic coding and analysis to answer our RQs (Klaus Krippendorff 2018)(Cruzes & Dyb 2011). While reading the full text of papers, the coding technique was utilized manually to find the relevant text in 65 studies. To answer RQ1 we analyzed the findings of all selected studies to extract their outcome of used ML algorithms. Based on the functionality of algorithms we categorized them into different groups coming from the relevant coded text by performing thematic coding and analysis. We extracted the list of challenges of using ML in RE to answer RQ2 and we analyzed them based on the groups of perspectives that we provided by the included studies. To answer RQ3 we reviewed the selected papers to investigate the most popular dataset used in RE tasks.

For RQ4 we extracted the information about evaluation metrics and analyzed them according to ML tasks in RE.

2.17.8. Findings

In this section, we describe the characteristics of our 65 included studies.

Publication sources- Among the 65 included studies, 40 (61.5%) are published in conference proceedings, 8 papers (12.5%) are published in workshop proceedings, and the remaining 17 (26%) are journal articles. The majority of these studies are from highly reputable outlets. All the papers included in our review were those that provided enough info about the research method and hence rated above 50% in the quality assessment checklist.

Publication year and study focus- Figure 2.3 presents the number of publications per year from 2010 to April 2020.

Some of the studies did not explicitly or clearly mention their specific focus. Others claimed that their study is useful for a special task, but we have deduced 16 different categories for the selected studies according to their mentioned tasks and this is presented in Table 2.2.

From Table 2.2 it can be seen that the task to which ML has been applied the most is classification with 12 studies. It shows extra needs and attention to automate the classification of requirements written in NL that is not straightforward in the process of RE (Abad et al. 2017b). ML has been utilized for many classification tasks like differentiating between users' requirements and software requirements. Requirements are often classified as functional (FR) and non-functional (NFR). Hence, separating and identifying them manually in SRS documents is a time-consuming task, finding an automated and effective approach to distinguish them has been the focus of several studies (e.g., (Kurtanovic & Maalej 2017), (Haque, Rahman & Siddik 2019)). Out of 12 studies focused on classification tasks, classifiers used for identifying both FR and NFR ((Kurtanovic & Maalej 2017), (Deocadez, Harrison & Rodriguez 2017), (Dalpiaz, Dell'Anna, et al. 2019), (Haque, Rahman & Siddik 2019), (Abad et al. 2017a)); to

automate the classification of NFRs into sub-categories of usability, availability, or performance and to pre-process requirements that standardize and normalizes requirements before applying classification algorithms (Abad et al. 2017a); to classify NFRs into maintainability, operability, performance, security and usability ((De Bortoli Fávero, Casanova & Pimentel 2019); to investigate specific and relevant terms in the text (De Bortoli Fávero, Casanova & Pimentel 2019); to divide specification content elements into requirements and non-requirements ((Winkler, Gronberg & Vogelsang 2019), (Winkler & Vogelsang 2017)); To automate classification task using tools (Hayes, Li & Rahimi 2014) to automate user requests in crowdsourcing RE (Li et al. 2018), and finally boosting text classification by combining text classification algorithms with semantic roles ((Rago, Marcos & Diaz-Pace 2018).

Our results indicate that a fair share of studies (eight studies) have been proposed to address ambiguity. Ambiguity has often been considered a potentially harmful attribute of requirements that leads to challenging the projects, so the primary objective of reducing the ambiguity is having requirements with only one possible interpretation (Boyd, Farroukh & Didar Zowghi 2005). Eight studies focused on improving the requirements extraction task in order to develop an automated solution for requirement analysis. Part of the works focused on the identification of efficiently and dynamically extract and classify requirements-related knowledge properly ((Shakeri et al. 2019), (Memon & Xiaoling 2019)), to extract requirements dependencies (Deshpande, Arora & Ruhe 2019), domain model extraction ((Arora et al. 2019), to extract relevant non-functional requirements (Slankas & Williams 2013), and analyzing the characteristics of requirement expressions to divide them into system-level requirements and instance level in pre-processing step (Chen et al. 2010).

The objective of validation was addressed in four studies by automation of fault-consolidation step (Singh et al. 2018)) and proposing a framework to overcome inconsistencies for the optimal definition of software development sprints (Belsis, Koutoumanos & Sgouropoulou 2014)). The main goal of validation is to ensure that all the documented requirements are correct, complete, and consistent, the designed

solution meets the requirements, and a real-world solution to be built and tested to prove that it meets the requirements (Maalem & Zarour 2016).

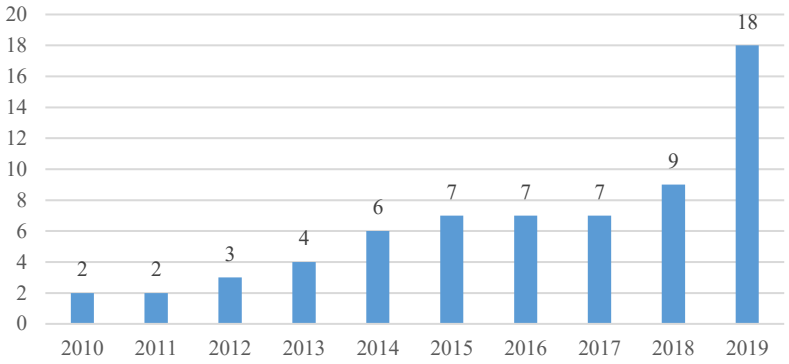

Figure 2.3.Number of Resulted Articles Published Per Year

Five studies focused on quality assessment from different perspectives. (Ferrari, Gnesi & Tolomei 2013) analyzed the structure of the document in the way it is perceived by the reader, while (Parra et al. 2015) assessed the quality of requirements automatically according to the quality criteria posed by the domain expert. (Tamai & Anzai 2018) automated the process of filtering out QR statements from an SRS and classifying them into the quality characteristic attributes as defined in the ISO/IEC 25000 quality model. (Dargan, Wasek & Campos-Nanez 2016) defined quality factors to assess, while (Hayes et al. 2015) addressed requirement testability for understandability and quality. Only one study focused on each of these topics: verification, model transformation, predict vulnerabilities, specification, and identifying business requirements while two studies focused on security and change requests.

According to the data extraction from our set of 65 papers, in this section, we describe our findings to answer the RQs.

Table 2.2. Study Categories

Study focus	**studies**	**frequency**
Classification	(Abad et al. 2017a), (Baker et al. 2019), (De Bortoli Fávero, Casanova & Pimentel 2019), (Dalpiaz, Dell'Anna, et al. 2019), (Deocadez, Harrison & Rodriguez 2017), (Haque, Rahman & Siddik 2019), (Hayes, Li &	12

	Rahimi 2014), (Kurtanovic & Maalej 2017), (Li et al. 2018), (Rago, Marcos & Diaz-Pace 2018), (Winkler & Vogelsang 2017), (Winkler, Gronberg & Vogelsang 2019)	
Requirements Extraction	(Shakeri et al. 2019), (Deshpande, Arora & Ruhe 2019), (Chen et al. 2010), (Memon & Xiaoling 2019), (Slankas & Williams 2013), (Vogelsang & Borg 2019), (Wang 2015), (Arora et al. 2019)	8
Ambiguity	(Osman & Zaharin 2018), (Richa Sharma, Bhatia & Biswas 2014), (Sharma, Sharma & Biswas 2016), (Yang et al. 2011), (Yang et al. 2010), (Dalpiaz, van der Schalk, et al. 2019), (Pal, Sandhu & Pal 2015), (Ferrari & Esuli 2019)	8
Analysis/ Management	(Wang 2016), (Knauss et al. 2015), (Abualhaija et al. 2019), (Osman et al. 2019), (Wang & Zhang 2016), (Misra, Sengupta & Podder 2016)	6
Traceability	(Sultanov & Hayes 2013), (Li et al. 2017), (Wang, Li & Yang 2019), (Li & Huang 2018), (Mezghani & Florence 2019), (Hayes, Payne & Leppelmeier 2019)	6
Quality	(Ferrari, Gnesi & Tolomei 2013), (Parra et al. 2015), (Tamai & Anzai 2018), (Dargan, Wasek & Campos-Nanez 2016), (Hayes et al. 2015)	5
Validation	(Nardini et al. 2012), (Singh 2018), (Singh et al. 2018), (Baker et al. 2019)	4
Prioritization	(Dhingra et al. 2017), (Singh & Sharma 2014), (Perini, Susi & Avesani 2013), (McZara et al. 2015)	4
Risk Management	(Avesani et al. 2015), (del Águila & del Sagrado 2016b), (Yang et al. 2012)	3
Change requests	(Khelifa, Haoues & Sellami 2018), (Arora et al. 2015a)	2
Security	(Malhotra et al. 2016), (Riaz et al. 2014)	2
Verification	(Winkler, Gronberg & Vogelsang 2019)	1
Model Transformation	(Chioaşcă 2012)	1

Predict vulnerabilities	(Imtiaz & Bhowmik 2018)	1
Specification	(van Rooijen et al. 2017)	1
Identifying business	(R. Sharma, Bhatia & Biswas 2014)	1
In Total		65

- **ML techniques/algorithms have been used in RE**

In our analysis, we note that the words 'technique' and 'algorithm' are used interchangeably. A total of 48 different ML algorithms were identified in our selected studies. We classified the algorithms into eight principal groups based on their functionality: Distance-based Methods, Regression, Decision Tree, Bayesian, Kernel Methods, Associated Role Learning, Ensemble Methods, and Artificial Neural Networks. We do not claim to cover all the existing methods exhaustively, rather we present those that are more frequently utilized. Figure 2.4 shows the distribution of algorithm types based on functional similarity. Analysis of results revealed that Kernel Methods, Bayesian and Distance-based are the most popular categories of algorithms, as they were used within 29 and 25 studies, Ensemble Methods is the second in the list with 23, Decision Tree with 21 and finally Artificial Neural Networks, Regression and Associated Role Learning with 12, 9 and 3 respectively. The distributions of each category type are available in our replication package. Figure 2.5 presents a visualization of the data regarding reported ML algorithms used in different studies. Support Vector Machine (SVM) is the most frequently used algorithm that has been employed in 17 studies. The second most used is Naive Bayes (NB), investigated in 14 papers, followed by K-Nearest Neighbors (KNN) in 11 studies, Decision Tree, and Random Forest in 10 and 8 papers respectively.

At least 4 studies have used the combination of different algorithms to improve the accuracy of the results, algorithms' strengths and overcome their limitations (e.g., (McZara et al. 2015), (Rago, Marcos & Diaz-Pace 2018), (Riaz et al. 2014), (Wang, Li &

Yang 2019)). (McZara et al. 2015) presented a semi-automated approach for challenging task of requirements prioritization in large scale projects by using NLP tools and an SMT (Satisfiability Modulo Theories) solver. They mitigate the challenges of variation outputs by updating the input of the SMT solver with iterative pairwise comparisons. In (Rago, Marcos & Diaz-Pace 2018), the researchers improved the accuracy of their classifier by combining the binary relevance and SVM. (Riaz et al. 2014) presented a tool-assisted process, Security Discoverer (SD) by combining K-NN classifier, Sequential Minimum Optimizer (SMO), and Naïve Bayes classifiers after comparing the accuracy of other potential classifiers. (Wang, Li & Yang 2019) proposed a hybrid approach of ML and Logical Reasoning to improve the feature-engineering process to recover requirements traceability recovery. 15 studies have employed several algorithms to just compare them to determine which one outperforms the others based on the specific Dataset-Or different datasets in their domain ((Abad et al. 2017a), (Baker et al. 2019), (Dargan, Wasek & Campos-Nanez 2016), (Deshpande, Arora & Ruhe 2019), (Haque, Rahman & Siddik 2019). (Imtiaz & Bhowmik 2018), (Osman & Zaharin 2018), (Parra et al. 2015), (Riaz et al. 2014), (R. Sharma, Bhatia & Biswas 2014), (Sharma, Sharma & Biswas 2016), (Singh 2018), (Singh et al. 2018), (Slankas & Williams 2013), (Wang & Zhang 2016)). 4 studies proposed methods or techniques by modifying either one or a mix of algorithms to improve the accuracy of results or enhance and optimize the automated models ((Arora et al. 2019), (Li et al. 2018), (Perini, Susi & Avesani 2013), (Ferrari, Gnesi & Tolomei 2013)).

As for classification, clustering, and regression approaches, 48 studies used just classification, 8 studies used clustering, 3 of these 8 studies used only clustering ((Baker et al. 2019), (Ferrari, Gnesi & Tolomei 2013), (Misra, Sengupta & Podder 2016)) while the other 5 studies ((Richa Sharma, Bhatia & Biswas 2014), (Sharma, Sharma & Biswas 2016), (Mezghani & Florence 2019), (Abad et al. 2017a), (Winkler & Vogelsang 2017)) used the combination of clustering and classification. Five studies did not mention a specific algorithm, the authors mentioned that they used ML techniques in some steps of their methodology ((Chen et al. 2010), (Memon & Xiaoling 2019), (Osman et al. 2019), (Dalpiaz, van der Schalk, et al. 2019), (Pal, Sandhu & Pal 2015)). All the 8 studies that used Regression categories of BLR, Logistic Regression, and SGD ((Imtiaz & Bhowmik

2018), (Osman & Zaharin 2018), (Singh et al. 2018), (Abualhaija et al. 2019), (Singh 2018), (Arora et al. 2019), (Winkler & Vogelsang 2017), (Dargan, Wasek & Campos-Nanez 2016)), have employed them as classifiers.

From the ML perspective, the essential steps required to apply ML techniques include (1) data collection, (2) data pre-processing, (3) building an ML model, (4) training and testing the model, and (5) evaluation. Figure 2.6 shows the distribution of the algorithms applied in step 3 building ML models.

In terms of using NLP algorithms in selected studies, we retrieved 40 studies that used one or more NLP algorithms in a total of 31 different algorithms across reported studies. The most commonly investigated NLP technique is the tokenization with part of speech (POS) tagging with 23 studies, followed by chunking and TF- IDF with both in 7 studies. Two papers did not mention the name of the used algorithms; only reported text-mining techniques ((Osman & Zaharin 2018), (Deocadez, Harrison & Rodriguez 2017)).

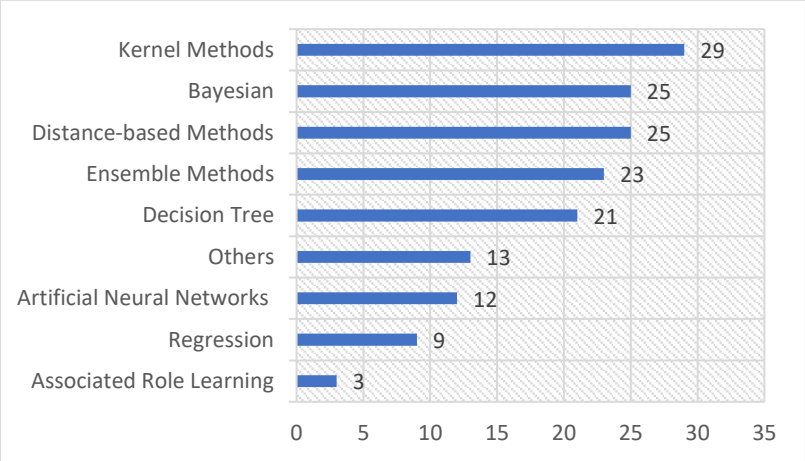

Figure 2.4.Distribution of Algorithm Types Based on Function Similarity

- **Challenges of using ML approaches in RE**

For effective use of ML capabilities in software and applications, it is very important to identify the challenges faced in the process of designing suitable ML solutions. Knowing the possible issues and challenges and how to address them can help the

researchers and analysts to benefit from the usefulness of ML. As a key finding of our analysis, the challenges extracted from selected studies were divided into six main categories: data related, task related, algorithm related, project related, language related, and other challenges. Figure 2.6 demonstrates the frequency of ML challenges in the included articles.

A. Data-related:

Since many ML strategies are focused on gaining from large datasets, the success of ML based research projects strongly relies upon data accessibility, quality, and management (Durelli et al. 2019) (Polyzotis et al. 2017). Data related problems were faced by 35 studies. In fact, this result was expected, since ML is a method that almost always requires data. One of the most important barriers in this category is the **lack of data**. Unfortunately, data is not free or always relevant. The availability of large datasets and possibly, the annotated Dataset-Is reported to be a major problem ((Singh 2018). An ML algorithm needs a large amount of data to train (Singh 2018). Specifically, deep learning algorithms need to be trained on large amounts of data to draw meaningful insights ((De Bortoli Fávero, Casanova & Pimentel 2019)). In a more complex project, more data is required to achieve trustable results. So, when **limited datasets** do not represent all possible situations, the results are not trustable (Shakeri et al. 2019). (Tamai & Anzai 2018) explained that the authors faced difficulty in collecting more SRS from a variety of areas that are large enough to use deep learning. Similarly, (Osman & Zaharin 2018) reported that the result of their study cannot be generalized to all systems because the data used were gathered from just four SRSs that only represent several system domains and limited patterns on requirement specification formation. The incomplete nature of the source corpus is also the outcome of limited data which will affect the accuracy of results (Yang et al. 2012). **Imbalanced classes** were mentioned in the data scope in ML especially in the classification. It occurs in datasets with a disproportionate ratio of observations in each class. 12 studies mentioned this problem.

The other challenges that we identified in the data category were overfitting/ underfitting (5 studies), followed by labeling issues (3 studies), dependency issues, missing datasets, size of data set, data quality issue, all with 2 studies and the issue of

selecting training set with 1 study. Concerning labeling issues, especially for some techniques like neural networks that need a large amount of data to train, it is not possible to manually check the dataset to determine labels are correct. When different people work on an SRS, samples in the dataset may be labeled differently (Winkler, Grönberg & Vogelsang 2019) and the important distinction between quality requirements and constraints is not properly reflected in the labeling (Abad et al. 2017a). The issue of the missing data may arise when the data is collected from users' feedbacks or questionnaires since some of the questions may not have been answered. Although there are some approaches to overcome this such as approximating the null values or calculating the maximum likelihood to minimize the error, all of them need time and effort (Baker et al. 2019).

The main purpose of a reliable ML model is to generalize well to different domains and new data that is evaluated for its performance over time as it is learning from training data. An ideal model should not suffer from **overfitting** or **underfitting**. Three studies, (Winkler, Grönberg & Vogelsang 2019), (Yang et al. 2012), and (Winkler & Vogelsang 2017) reported overfitting issues due to using a relatively small dataset. Overfitting is the case when the model produces excellent results on the training data set but cannot be employed on any unseen data at an acceptable accuracy level. (Rago, Marcos & Diaz-Pace 2018), (Kurtanovic & Maalej 2017) mentioned an underfitting problem that occurs where the model is too simplistic and has not learned enough from the training data. For example, a model trained on fewer or unrepresentative features.

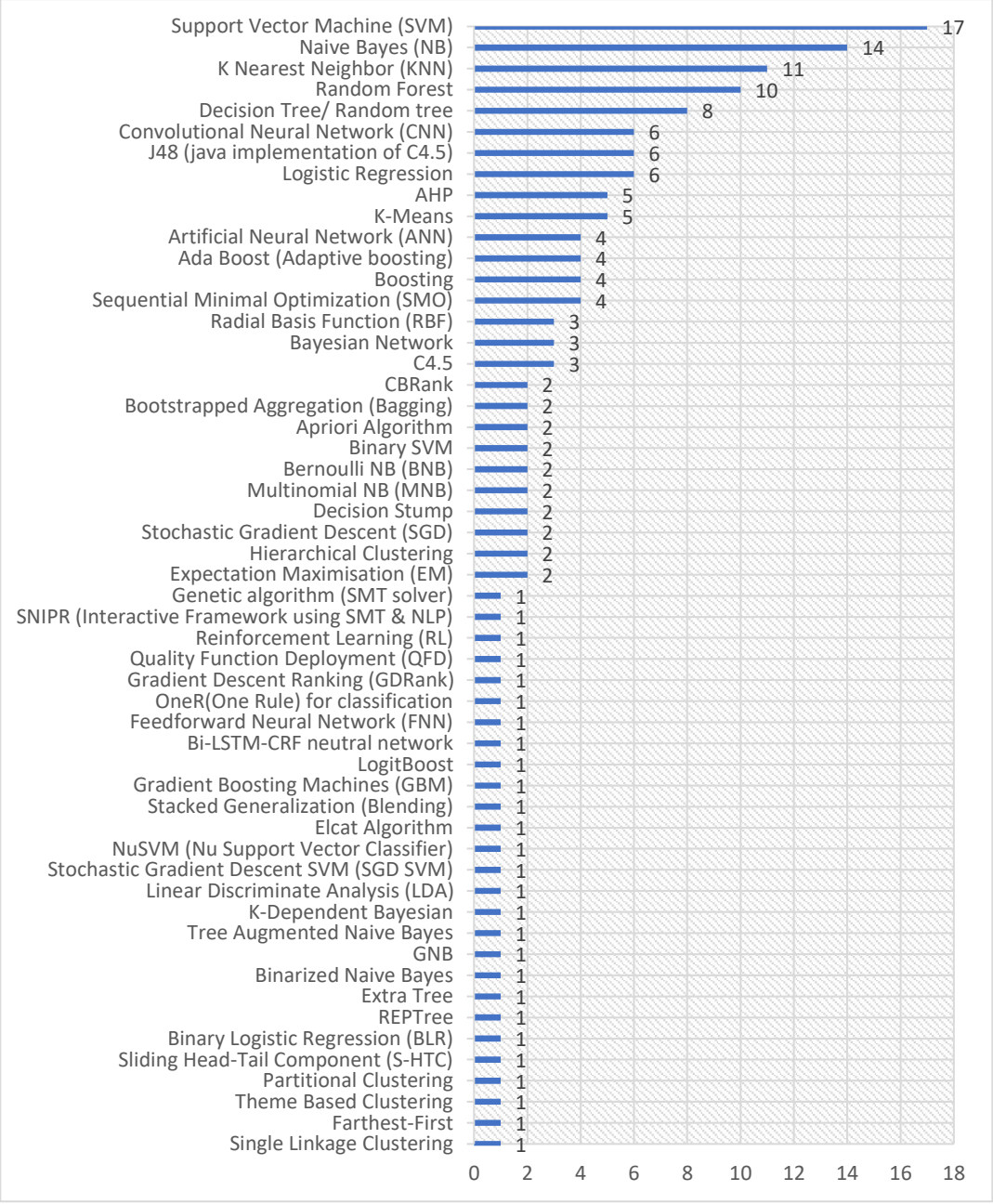

Figure 2.5.The Number of Used Algorithms in Selected Studies

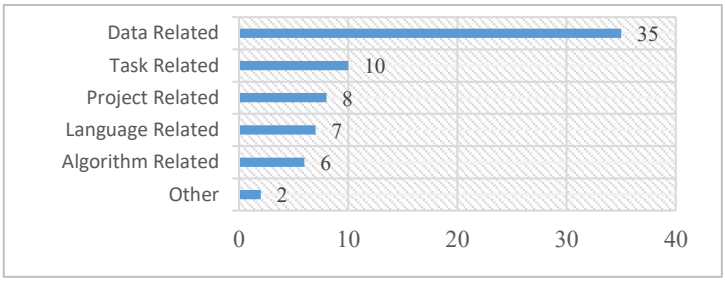

Figure 2.6.The Frequency of Machine Learning Challenges

B. Task-related

In this category, we present the challenges that are related to specific ML based tasks such as classification or regression that were reported by 10 papers. Five studies reported classification problems and misclassification errors for both binary and multi-label classification ((Abualhaija et al. 2019), (Dalpiaz, Dell'Anna, et al. 2019), (del Águila & del Sagrado 2016b), (Wang, Li & Yang 2019), (Shakeri et al. 2019)) that involve predicting a class label for a given set of inputs. Although solutions have been offered to address these challenges, not all studies have utilized these solutions. Generally, the main goal is to train a model with accuracy more than humanly possible. Since a wrong prediction during classification (such as true-fault being classified as false-positive), would lead to fault slippage that will propagate to later phases. It is expected that by using ML algorithm, the highest rate of accuracy can be achieved (Singh 2018). Misclassification may be caused by errors in the classification process of the requirements by the experts because the classifier can learn incorrect classification and replicate the error in the classification of new requirements ((Parra et al. 2015), (Riaz et al. 2014)). Misclassification can also manifest itself during classifying specification elements into requirements and non-requirements (Winkler, Gronberg & Vogelsang 2019). Even different ideas on grouping classes and naming them may cause misclassification as noticed in (Li et al. 2018). The authors worked on the types of user requests as classification targets. They classified user requests manually and they mentioned that labeling a large set of data is cumbersome, so using active learning techniques might be better. They also reported that investigating the appropriate features to represent document items and ML algorithms to train the classifier was a big challenge for their approach.

C. Algorithm- related

In terms of algorithm related challenges, two studies reported the black box nature of ML classifiers like SVM or Neural networks making these algorithms difficult to understand. To overcome this challenge (Dalpiaz, Dell'Anna, et al. 2019) employed two interpretable tools called RuleMatrix and SkopeRules to facilitate the interpretation of ML classifiers by extracting logical rules. RuleMatrix shows which rules are applied to

the data by visualizing them so, it helps to understand, explore, and validate predictive models. (van Rooijen et al. 2017) noted that although their selected method had the black-box problem, in their case due to the lack of methods to learn more about a given problem instance, there was no issue to select a black-box optimizer. The authors of (Slankas & Williams 2013) reported that their proposed approach, NFR Locator, is suitable to extract information from text documents. It is not able to extract information from images or tables.

The structure of text documents might be classified into the NFR category, so they had to parse the files in their native format to distinguish the structural parts such as titles, section lists, etc. The other challenge investigated in (Ferrari, Gnesi & Tolomei 2013) is the necessity of tuning algorithm behavior. They aimed to identify the hidden structure of requirements documents in terms of requirements relatedness and section independence. Sometimes, their algorithm reported dependency among sections that were not related according to the perception of the readers. Moreover, (Chen et al. 2010) reported that although their approach in pre-processing text-based requirements is suitable for goal-oriented requirements, it cannot be used to extract business rules. The authors claimed that the sentence patterns that are describing business rules are more complex than the domain sentence pattern. Regarding using the semantic role labeling method, (Wang 2016) mentioned that the corpus for SLR tasks in SE domain is very few, so they need to use the other domain knowledge as rules to improve the results. (Dalpiaz, van der Schalk, et al. 2019) reported that to reach better results and higher precision in their approach, they need to go beyond domain-independent corpora and use domain-specific information.

D. Project-related

In this category, we present the project-specific challenges that impose a limit or restriction or that prevent approaches from generalizability. Four studies explicitly mentioned the need for further experiments in other domains, especially with the help of domain experts to determine whether their approaches and tools can be generalized ((Ferrari, Gnesi & Tolomei 2013), , (Dalpiaz, van der Schalk, et al. 2019), (Knauss et al. 2015)). Moreover, proposed approaches need to be applied to different scenarios and

multiple industry-scale projects (Baker et al. 2019). Although this leads to significant costs, it helps to reach a full evaluation (Knauss et al. 2015).

One study (Dalpiaz, van der Schalk, et al. 2019) that focused on the expectation of users and stakeholders mentioned that even accurate algorithms and tools need a sufficient level of maturity. This is because any proposed tool that exhibits low usability and contains bugs may decrease the interest in applying them in real projects.

E. Language-related

The last challenge is related to the writing of the requirements including spelling mistakes, structural effectiveness, terminology and vocabulary, and language that have been addressed in six articles. Automated classification of requirements into functional requirements and non-functional requirements remains a challenge (Ernst & Mylopoulos 2010). Stakeholders, as well as requirements engineers, use different terminologies and sentence structures to describe the same kind of requirements. The high level of inconsistency in documenting requirements makes automated classification more complicated and therefore error-prone (Abad et al. 2017b). Furthermore, the requirements reviews are written in NL that inherits the scope to spelling mistakes (Singh et al. 2018). NL understanding relies on the specification readers and writers using the same words for the same concept. This leads to misunderstandings because of the ambiguity of NL that is often not discovered until later phases of the software process and may then be very expensive to resolve. To overcome these problems, writing effective and high-quality requirements will lead to an accurate ML result (Singh 2018). Requirements originating from different documents may be quite different in terms of language and terminology. In other words, documents may contain domain-specific words which are exclusively used in that particular document (Winkler, Grönberg & Vogelsang 2019). Moreover, variations may exist between the security requirements of software systems, even in the same domain. Thus, the selection of documents may influence the type and frequency of identified security-relevant sentences (Riaz et al. 2014). Most of the ML research have used requirements that are written in English and so there is a bias about generalizing the results to the requirements written in other languages (Deocadez, Harrison & Rodriguez 2017). There

are also issues related to NLP-models being more accurate and readily available for English, as compared to other languages. In addition, investigating if a requirement is speculative or not is not an easy task, which is due to the peripheral nature of uncertainty language (Yang et al. 2012).

F. Other challenges

There were other challenges that we extracted from the included studies that did not fit under the above main five categories. For example, since requirements often change over time, another challenge is the stability of requirements. Clients might modify requirements, so fluidity in software requirements becomes a major problem (McZara et al. 2015). Although this is a common problem in RE, when the ML technique is used in this process, it becomes a more significant challenge because the model needs to be trained again when the requirements change. Besides, this will be an issue if the labeling changes with changes in requirements.

Negotiation barriers between the client and business analyst or developer on different grounds such as language, not using consistent terms, and making assumptions about ambiguous requirements is another major challenge that exists in RE tasks (Parra et al. 2015).

- **Identification of datasets used for ML in RE**

Investigating the applied datasets and their associated properties allows us to determine to what extent we can rely on the performance results, and it can provide new insights into why some ML techniques may outperform others.

As discussed above, the top challenge in implementing ML techniques in RE is related to datasets. Since ML algorithms are quantitative, the success of ML related research projects strongly depends on having a large enough dataset (Ferrari, Spagnolo & Gnesi 2017). Many different datasets have been used in the included studies. The terminology used by the authors to describe the type of documents involved in the research were of

varying degrees of abstraction and level of details, e.g., requirements documents, requirements specification, textual requirements, operational requirements, SRSs documents, system-related documents, user requirements, high-level requirements, and low-level requirements. Some studies used real-world datasets while others used sample data that are shared in open-source repositories to use by researchers.

To answer this research question, we looked at the frequency of datasets, the type of data, their organization, and the number of requirements statements in each study.

According to the results, 10 studies did not report any information about their dataset. The number of used datasets varied from 1 to 22. Out of the remaining 54 studies, 22 studies used just one unique dataset, while the biggest number of datasets belongs to (Hayes, Li & Rahimi 2014) by 22 different datasets with a total of 2067 user stories. The reason why they selected 22 datasets was that their research was based on their previous paper that used a single dataset, so they were motivated to increase the number of datasets to get more in-depth results and to increase generalizability. The second largest is for (Tamai & Anzai 2018) by 13 datasets from local governments or other public institutions of industry, medical information, education, library, etc. in Japan totaling 11,538 requirements sentences. The trend of frequency was followed by nine and eight datasets that were used in (De Bortoli Fávero, Casanova & Pimentel 2019) and (Dalpiaz, Dell'Anna, et al. 2019) respectively. The textual requirements in (De Bortoli Fávero, Casanova & Pimentel 2019) were collected from 16 large open-source projects in repositories that contained 23,313 user stories. In (Dalpiaz, Dell'Anna, et al. 2019), data was gathered from eight datasets of PROMISE, ESA Euclid, Dronology, ReqView, and Leeds University's Library online management system, Web Architectures for Services Platforms (WASP) application and two private datasets of Helpdesk system and bespoke user account request and management application (User mgmt.). A considerable number of datasets in this review were reported by (Deocadez, Harrison & Rodriguez 2017) about user reviews. The authors collected data from the App Store in 2015. Since they considered the top paid and free apps from different categories of books, education, games, health, lifestyle, navigation, news, productivity, travel, and

utilities, they reached 40 apps with a total of 932,388 reviews. The remaining pairs of studies (16 studies) considered three datasets or fewer.

As for the frequency, we observed that the most frequently used Dataset-Is PROMISE1 that is an open-source Software Engineering Repository that includes a collection of publicly available datasets and tools for researchers ((Shakeri et al. 2019), (Slankas & Williams 2013), (Malhotra et al. 2016), (Khelifa, Haoues & Sellami 2018), (Abad et al. 2017a), (Baker et al. 2019), (Dalpiaz, Dell'Anna, et al. 2019), (Haque, Rahman & Siddik 2019)). It was inspired by the UCI Machine Learning Repository, which has been extensively used by researchers in that field. The second frequently utilized Dataset-Is Pine by 3 studies ((Sultanov & Hayes 2013), (Li et al. 2017), (Li & Huang 2018)) followed with NASA CM-1 by 2 studies ((Sultanov & Hayes 2013), (del Águila & del Sagrado 2016b)). Pine is a text-based email system developed by the University of Washington that includes true links, high-level and low-level requirements (Sultanov & Hayes 2013). NASA MDP repository includes different datasets which CM1SUB project that concerns a scientific instrument to be carried on-board a satellite was addressed in our selected studies.

When extracting data about the types of documents that were used as a data source for studies, we observed that they include functional requirements, non-functional requirements, high level, and low-level requirements, Operational Test Reports, user stories, Wikipedia Pages, design documents, textual use cases, code modules (classes), correct links, user comments (reviews), user requests and change requests.

Regarding the domain of datasets, healthcare and medical data were used as a data source by 10 studies. Some of these datasets are open source while the others are private. (Slankas & Williams 2013) reported the use of OpenEMR2 that is one of the popular open-source electronic health records and medical practice management solutions. The other healthcare repository is iTrust that was used by (Slankas & Williams

[1] http://promise.site.uottawa.ca/SERepository/

[2] https://www.open-emr.org/

2013), is a medical application that maintains patient medical history and records and permits communication with doctors. It consists of 59 use cases and 11 code modules. It was written in Java.

Industrial data was utilized by 10 studies. Nine studies built their requirements corpus by collecting data from academics and educational domains. Our results indicate that twenty of the studies concerned external corpus as an external reference for the English language for NLP. The most frequently used external corpus is Wordnet3 by 11 studies. The next most used corpus is Wikipedia pages that were utilized in six studies. BNC164 (British National Corpus) was used by two while VerbNet was used by one study. Concerning the language of datasets, all are in English except for (Wang, Li & Yang 2019) and (Tamai & Anzai 2018). eTour dataset that was used by (Wang, Li & Yang 2019) is in Italian. It is an electronic touristic guide developed by students in Italy that contain 58 use cases, 174 classes, and 366 correct links. In (Tamai & Anzai 2018) all 13 SRSs that contained 11,538 requirements sentences were in Japanese.

Since the size of datasets was not reported in all studies, we must categorize them according to the number of documents or sentences as small, medium, and large size. We considered datasets more than 20,000 samples as large, between 1000 and 20,000 as medium, and less than 1000 as small. The largest dataset used by (Deocadez, Harrison & Rodriguez 2017) is 932,388 user reviews carried out of 40 different apps from the app store. The other significant size belongs to (Winkler, Gronberg & Vogelsang 2019) dataset that includes 35000 pre-labeled content elements (20000 requirements and 15000 non-requirements). The second largest utilized one is for (Winkler, Grönberg & Vogelsang 2019) and (De Bortoli Fávero, Casanova & Pimentel 2019) by 27,000 requirements from an automotive domain and 23,313 user stories from 16 large open-source projects in 9 repositories respectively. Out of the 54 datasets, 11 datasets were of unknown size, 22 (about 50 percent) were small and 4 datasets were considered large.

[3] https://dumps.wikimedia.org/enwiki/

[4] http://www.natcorp.ox.ac.uk/

2.17.9. Evaluation metrics for ML approaches in RE

Evaluation metrics play a critical role in achieving the optimal ML model by qualifying its performance (B.Hossin & Sulaiman 2015). Since the performance of ML algorithms may be affected by tasks and domains, the evaluation metric has been employed to decide on which technique is the best match by comparing different techniques offline. It should be noted that selecting an incorrect evaluation metric can lead to select an unmatched algorithm so, the selection of a suitable metric is an essential part of any project to discover whether or not the performance is effective. (Gunawardana 2009). Performance evaluation is tricky for many NLP tasks since there is not easily agreeable "ground truth" or "gold standard". Proper performance evaluation is the subject of much community discussion and even a research topic in its own right. Understanding the proper performance evaluation and performance metrics is very important to make informed business decisions.

Out of 65 selected studies, 42 articles used evaluation metrics to determine the performance of the used algorithms or to investigate which algorithm outperformed the others in terms of accuracy. We categorized the evaluation metrics employed by the selected studies into three categories of use for classification, clustering, and regression. Concerning classification tasks, precision and recall are the most used metrics employed by 38 studies followed by F1-measure by 29 and Accuracy by 15 studies. Requirements engineering has adopted information retrieval metrics including precision, recall, and the F-measure, to assess the effectiveness of any techniques or tools as well as using them to develop applications for RE tasks (Berry et al. 2017). For imbalanced classification when recall and precision are not equally important, a weighted F-measure called Fβ-measure can be used. The result shows that only one study (Winkler, Gronberg & Vogelsang 2019) used both of them to evaluate and optimize their classification tools. The authors tried to carry out a reasonable value for β and to tune the tool by this value. In NLP tools, recall is going to be more important than precision so tool assistance in the RE should be evaluated by a weighted F-measure (Berry et al. 2017).

In terms of validation, some authors, such as (Osman & Zaharin 2018), (Yang et al. 2010), (Abad et al. 2017a), (Winkler, Gronberg & Vogelsang 2019), (Deshpande, Arora &

Ruhe 2019), (Sharma, Sharma & Biswas 2016), (Abualhaija et al. 2019), (Dalpiaz, Dell'Anna, et al. 2019), (R. Sharma, Bhatia & Biswas 2014), (Li et al. 2017) (10 studies) employed K-fold cross-validation to assess how the classifier will generalize to an independent data set that is used to determine the performance of the predictive model to check whether a model is overfitting. The main goal of validation in classification tasks is to determine how well the classifier will perform on unseen data (Williams, Zander & Armitage 2006). Even though 37 studies used precision and 15 used accuracy to discriminate the optimal solution especially for their classifiers, prior studies were concerned about using popular metrics. Hossin et al. (B.Hossin & Sulaiman 2015) explained that accuracy still has several instabilities which are less distinctiveness, less discriminability, less informative, and bias to majority class data. Menzies et al. (Menzies et al. 2008) argued that accuracy and precision are poor indicators of performance for data where the target class is so rare. Figure 2.7 shows the distribution of evaluation metrics for the classification task. Only 10 different metrics were reported to evaluate the quality of the clustering. Figure 2.8 illustrates the distribution of evaluation metrics for the clustering task.

2.17.10. Discussion

The results presented from this mapping study reveal that many different ML algorithms have been applied to RE tasks to improve accuracy and to automate, among other goals.

Our analysis shows that we currently do not have any standards or guidelines to help analysts select the most suitable ML and NLP techniques. Furthermore, it appears that most studies combine various ML techniques in their research to tackle the existing challenges. However, hardly any of them explain the reason for choosing their selected ML algorithms. We believe that it is not clear what kinds of selection criteria have been applied or need to be considered. In addition, two papers discussed steps such as hyperparameter optimization, and feature engineering. This also shows that we do not have a standard template for applying ML on RE problems.

By taking a closer look at the reported datasets and repositories, our concern is how do we decide which Dataset-Is the best match for any ML model? Does the size of the dataset matter? It is not clear when selecting a Dataset-What sort of criteria should be applied. We could not find any quantitative or qualitative checklist to assess the selected datasets. There is no consensus on standard guidelines in the literature for deciding on the choice of dataset. Some studies reported that their results cannot be generalized because their approach needs to be tested on larger scenarios and need to be applied in other domains to improve the results. It is not clear if there is any relationship between the size of the datasets and the specific domain for their application. We believe that these concerns and questions need further research.

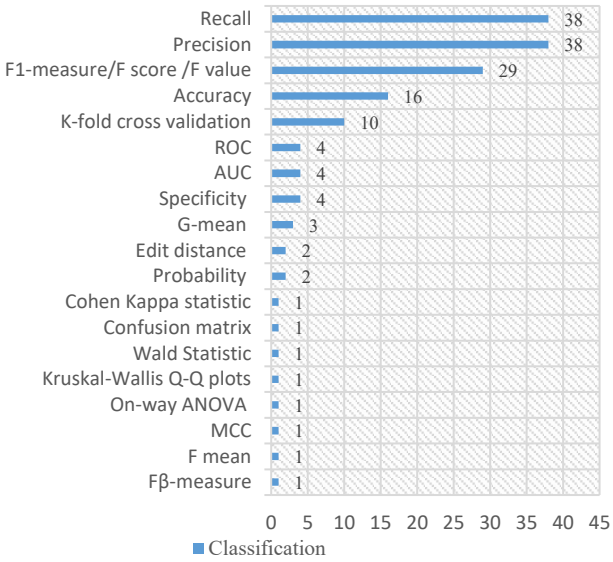

Figure 2.7.Evaluation Metrics for the Classification Task

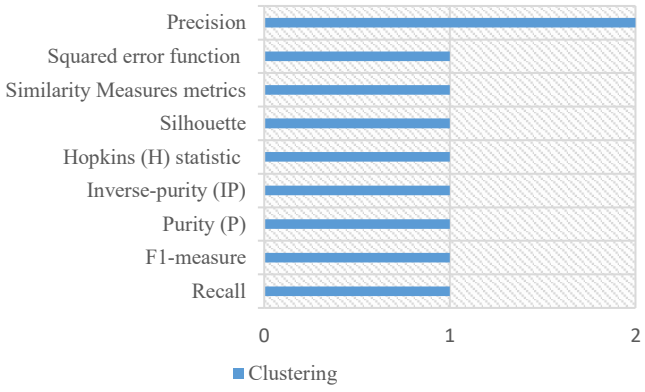

Figure 2.8.Evaluation Metrics for the Clustering Task

2.17.11. Emerging Trends and Future Directions

While our systematic mapping study focuses on literature up to 2020, the past few years have seen rapid advances in large language models (LLMs) that open new opportunities for CIA. State-of-the-art models such as GPT-4, LLaMA, and PaLM demonstrate remarkable capabilities in understanding complex, domain-specific text and synthesizing context-rich responses. Future research should explore fine-tuning these LLMs on requirements-engineering corpora, as well as hybridizing them with knowledge-graph and ontology embedding techniques to capture both procedural and semantic dependencies. Incorporating such recent LLMs could yield more accurate impact predictions and enable zero- or few-shot adaptation to novel project artifacts, thereby extending the dynamic adaptability of our framework beyond the 2010–2020 window.

Moreover, retrieval-augmented generation (RAG) offers a powerful synergy for CIA tasks. By first retrieving semantically relevant passages, anchored in a structured index of requirements, design documents, and change-request metadata, and then conditioning an LLM on these contextual snippets, RAG dynamically adapts its generation to the precise project context. This two-stage loop not only grounds predictions in verifiable sources (improving explainability) but also enables the model to update its “knowledge” in real time as new artefacts are added. Empirically, we anticipate this will reduce false positives in impact candidates and enhance resilience to evolving terminology and process changes—key challenges identified in our mapping study.

2.17.12. Threats to validity

The main validity threat for this mapping study is data collection. Although we chose our search string carefully and carried out a structured and detailed review of each of the selected studies, there is a chance that our collection is not complete because of the risk of not including all relevant studies. Some of the papers may have been written in another language or maybe not be available online. Therefore, we might have missed some significant research papers. The other validity threat is related to data extraction

because some primary studies did not report a precise explanation of their focus, their used methods, and data sources. In terms of ML challenges, we observed a lack of a clear definition of how they tackled the existing challenges. Consequently, this imitation might affect our outcomes. A possible limitation of our mapping study is related to the starting date of 2010. The main reason was to investigate the most recent ML for RE methods and algorithms. To ensure that all relevant studies were located, we manually applied our search string to some of the data sources before 2010 randomly and the number of papers found was insignificant.

2.18. Summary

This mapping study has provided an overview of the existing approaches in ML used for tasks in the RE process. We have presented the results from the analysis of 65 empirical studies published from 2010 to April 2020. The key findings of this mapping study indicate that there are at least two main gaps in literature, one is about selection criteria for ML techniques and the other is that more research is needed to investigate the relevance and appropriateness of datasets for the ML models. Another possibility is an online repository of ML features engineered in different classification approaches. This will probably be saturated at some point if the classification task is directly on the requirements statements or attributes related to SRS. Having such a feature repository will also guide future research on ML for RE.

In order to attain a detailed overview of the current state of using the proper dataset to obtain a reliable result, complete criteria need to be developed to assess which Dataset-Is the best match for which models. It would be beneficial to investigate how we can decide on the relevance of the dataset to our ML algorithms. Typically, today's software applications work in a competitive environment where business priorities frequently change. Therefore, software requirements are constantly evolving, and new requirements often emerge. The ability to analyze a change in requirements, predict its progression, and determine the effect early in the design process would enable engineers to make better decisions about the implementation of changes, especially in large scale projects. What is important to note here is that CIA has not been the direct

focus of any ML for RE studies and it is the missing piece in the research literature. Besides, not enough research has been carried out in prediction modeling in RE and it is an area that is under-explored. In parallel with our mapping study, another study has been conducted as a systematic mapping study by L. Zhao et al. (Zhao et al. 2020) about NLP for RE that surveys the landscape of NLP for RE research to understand the state of the art and identify open problems. This study strengthens some of our findings but differs from our review on one point. Their mapping study only focuses on NLP while our work is about ML in RE. Since ML is a generic term that may also include NLP and deep learning techniques, there are clearly some overlaps between their selected studies and ours, but we compared, we noted that 46 studies that were included in our mapping study were not on their list of selected studies (Zhao et al. 2020).

Chapter 3.

Software Requirements Change Impact Analysis (SRCIA) Framework

3.1. Introduction

In this chapter, the SRCIA (Software Requirement Change Impact Analysis) framework developed for this research is outlined, focusing on the systematic integration of traditional ML, NLP, Beir and RAG models. These models work together to address the challenges associated with CIA in software requirements engineering (SRE). The challenges are specified in chapter 2 section 2.5.

The data sets used in this research are also introduced in this chapter, as they form the foundation upon which all models and approaches are applied. By consolidating the core framework and the data in one place, this chapter provides the essential groundwork that will be referenced throughout the rest of this thesis.

The research is structured in a way that builds upon the lessons learned from earlier stages, gradually advancing toward more sophisticated solutions. This incremental approach ensures that each solution is tested and validated against real-world datasets, ultimately leading to a comprehensive, adaptable framework for CIA.

3.2. SRCIA Framework

The framework proposed in this research is designed to automate and improve the prediction of software requirements changes' impact. This architecture integrates multiple stages of the CIA process, enabling a flexible and modular approach to its implementation.

Figure 3.1 illustrates the visual representation of the comprehensive workflow of the SRCIA framework, detailing the process from initial data collection and preprocessing of requirements documents and change requests, through the preparation of datasets, to the ultimate evaluation of impacted requirements within the SRCIA process, culminating in an approval or disapproval decision

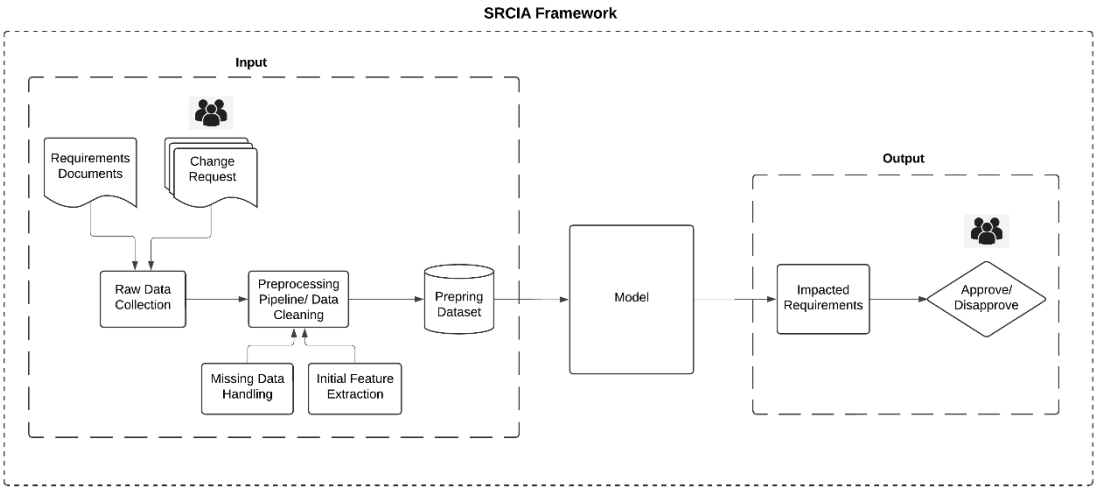

Figure 3.1.Software Requirements Change Impact Analysis (SRCIA) Framework

Figure 3.2 illustrates a more granular look into the internal structure of the proposed framework, from data input to model output, highlighting the interplay between the different components of the framework corresponding to each CIA stage. This flowchart provides a visual guide to the system's architecture and how each stage contributes to overall functionality. At its core, the SRCIA framework employs multiple stages of analysis, incorporating traditional ML models, advanced NLP techniques, and the latest developments in LLMs to ensure the system can handle a wide range of use cases, from basic change impact predictions to more complex, context-aware scenarios.

The data flows through the following components:

1. **Data Collection**

The first phase of the framework involves gathering data from diverse sources, including project specifications, requirements documents, change logs, and historical project data.

2. **Preprocessing Pipeline**

The collected data is then preprocessed using NLP techniques to standardize and cleanse the text, ensuring that it is ready for further analysis.

3. **Change Impact Analysis**

The framework incorporates ML model, a dual-model of NLP and Beir benchmark-based solution and a RAG model (that uses the advanced text generation abilities of LLMs) to predict changes.

4. **Output**

The core function of the framework is to predict which requirements will be impacted by a given change. It returns the most likely affected requirements. These results are then ranked based on the predicted severity of impact.

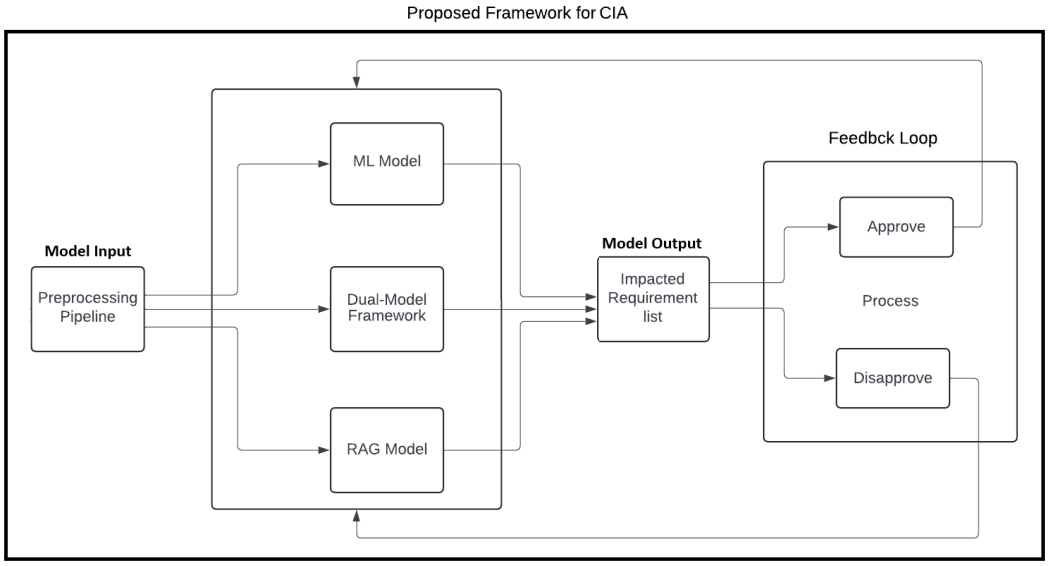

Figure 3.2.Detailed View of SRCIA Framework

5. Feedback Loop and Continuous Learning

A key feature of the framework is its feedback mechanism, which allows users (such as software engineers) to provide input on the framework's output. This feedback is used to improve the accuracy of future analysis, ensuring that the system remains relevant as requirements and project contexts evolve. This modular architecture ensures that the framework is adaptable and scalable, allowing for the integration of new techniques as they emerge in the field.

3.3. The AI models incorporated in the proposed SRCIA Framework

The research outlined in this thesis followed a structured, multi-stage approach, each stage building upon the previous one to develop a comprehensive framework for effective CIA in software requirements engineering.

In this research, a comparative evaluation of all the implemented solutions was conducted. The traditional ML methods, the integrated NLP and BEIR-based solutions, and the RAG model were all assessed based on their effectiveness, precision, and overall performance in predicting change impacts. The results of these evaluations provided valuable insights into the relative strengths and weaknesses of each approach, ultimately informing recommendations for best practices in CIA for future research and practical application in software development projects.

- **Traditional ML Approaches**

Traditional ML models were applied to establish a baseline for predicting a baseline for predicting the impact of software requirement changes. These models focus on structured datasets where explicit patterns and dependencies can be identified and analyzed. In this framework, ML models are implemented by transforming textual requirements data into numerical features using methods such as TF-IDF and dependency analysis. The resulting feature sets allow the models to interpret relationships and dependencies between requirements effectively.

Algorithms such as Random Forest, Support Vector Machines (SVM), and Decision Trees are employed to predict which requirements are likely to be impacted by a given

change. These models are trained using historical data on requirement changes and their corresponding impacts. Evaluation metrics, including precision, recall, F1-score, and accuracy, are used to assess model performance. Traditional ML models proved effective in scenarios where datasets were structured and of moderate complexity, particularly when clear dependency patterns were present. For example, Random Forest demonstrated strong predictive performance in datasets with hierarchical relationships among requirements.

These models are particularly effective for structured datasets, where explicit patterns and dependencies can be identified and analyzed. Such datasets typically contain well-organized information, such as dependency mappings or metadata related to requirement changes.

- **Integration of NLP and BEIR Benchmark-Based Solutions**

Building on the foundation established by traditional ML models, the following component of the framework integrates NLP techniques and BEIR benchmark-based solutions. This approach enhances the semantic understanding and precision of CIA by leveraging advanced linguistic and retrieval methods.

NLP techniques are applied to extract meaningful linguistic features from requirements, such as dependency parsing and named entity recognition (NER). These features are processed further using BEIR solutions, which include BM25 for lexical retrieval and dense retrieval models, such as Bi-Encoders and Cross-Encoders, for re-ranking results. By combining lexical and semantic retrieval methods, this phase enables the framework to rank impacted requirements with higher relevance and accuracy.

This integration is particularly effective for unstructured textual or semi-structured textual datasets, where relationships between requirements are complex and require nuanced semantic analysis. The evaluation of this phase uses metrics such as Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain (nDCG) to measure the retrieval performance.

- **Incorporation of Large Language Models (LLM) and Retrieval-Augmented Generation (RAG)**

The advanced layer of the SRCIA framework embraces recent advancements in AI by incorporating LLMs through a RAG approach. This stage combines retrieval-based methods with the generative capabilities of LLMs, resulting in a highly adaptable and sophisticated solution for predicting change impacts.

In this phase, vector embeddings generated by LLMs are used to retrieve contextually relevant data from the requirements repository. The retrieved information is then processed by the generative component of the RAG model to predict the most likely impacted requirements, along with explanations or contextual insights.

This approach is particularly effective for large-scale, dynamic, and heterogeneous datasets, where the complexity of changes demands both retrieval precision and generative reasoning. Metrics such as BLEU scores and ROUGE scores are used to evaluate the quality of the generated outputs.

3.4. Novelty of SRCIA

While earlier approaches such as (Arora et al. 2015a) NLP-driven traceability strategy focused primarily on extracting term correlations or rule-based dependencies between changed and impacted requirements, the SRCIA framework introduces three key innovations:

- Predictive Impact Modeling

Rather than stopping at trace link discovery, SRCIA incorporates supervised ML classifiers to forecast which requirements will change in response to a given request. This moves beyond binary link detection to quantitative impact prediction, enabling proactive resource planning.

- Hybrid IR–Generative Loop

Existing methods (e.g., pure BEIR pipelines) retrieve relevant passages but do not synthesize them. SRCIA's RAG component both retrieves semantically anchored

snippets and conditionally generates impact narratives, then re-scores and refines those narratives in a second pass. This iterative synergy yields higher precision and explainability than one-shot retrieval or static generation alone.

- Dynamic Domain Adaptation

Unlike static tailoring approaches, where traceability rules or model parameters are handcrafted per domain, SRCIA uses transfer learning and domain-tuned prompt engineering (Phi 3.5) to automatically adapt to new project vocabularies and artifact structures. This reduces the manual effort of creating bespoke pipelines for each domain benchmark.

By integrating these elements, SRCIA goes well beyond previous traceability-only or retrieval-only architectures, offering a unified, adaptive, and predictive solution for CIA in live software projects.

3.5. Datasets Description

When choosing the datasets for this study, two crucial criteria were established. The first criterion involves using datasets sourced from varied domains. We sought to avoid the repeated benchmark bias noted in peer research (Arora et al. 2015a; Tantithamthavorn, Hassan & Matsumoto 2018), by surveying a wide variety of public corpora such as PROMISE, Pine, NASA CM-1, PURE, and others. Although theses established benchmarks have proven valuable for traceability and evolution studies, they exhibit two key limitations for CIA research: (1) they focus primarily on code or API-level changes rather than end-to-end requirement–change interactions, and (2) they often lack rich, real-world change-request metadata needed for impact prediction.

Second, to ensure practical relevance and real-world applicability, we deliberately selected only industry-sourced SRS and software change notification datasets, containing authentic requirement statements and change requests from live projects to overview the changes that may happen during the time.

The raw data were taken from three industrial datasets described below for this study. Table 3.1 shows detailed information about data input from three industry partners.

- Dataset 1: WASP (Web Architectures for Services Platforms) dataset (Arora et al. 2015a, 2015b). More information about the requirements and the change scenarios can be found at this link: https://sites.google.com/site/svvnarcia/.
- Dataset 2: A larger real-world dataset from one telecommunication project containing requirements statements and change requests.
- Dataset 3: A real-world dataset from one industry partner, including requirements statements and change requests.

Our chosen datasets offer domain diversity, WASP covers web-service platforms, the Telecom dataset spans large-scale network rollouts, and the industry dataset captures enterprise application evolution. I addition, each contains both formal SRS entries and authentic change-request forms with timestamps, authorship, and rationale fields, unlike PURE's synthetic or narrowly scoped logs. Dataset 3 includes recently collected change requests from a live system, something neither PURE nor CM-1 provide—allowing us to evaluate our SRCIA framework on contemporary engineering practices.

3.6. Data Collection Procedure

For this study, the focus was on obtaining and demarcating a large set of data from various domains to develop, train, and assess the model. Data from real-world projects needed to be collected, with at least one expert per project to help interpret and correct the data. Consequently, three industry projects from web service, telecommunications, and satellite organizations were chosen. The collected requirements statements and change requests were formatted as PDF and Excel worksheets, with links to their embedded word documents containing change information and details.

Table 3.1 illustrates the industry contribution from our industry partners. A total of 891 requirements statements and 77 change requests were collected during the data collection procedure and were input into our approach.

Table 3.1. The number of data input from three industry partners

#	Dataset	Domain	# Requirements Statement	# Change request
1	Dataset-W	Web Service	72	28
2	Dataset-O	Telecommunications	626	34
3	Dataset-I	Satellite	193	15
	Total		891	77

3.7. Data Annotation & Quality Verification

A total of 891 requirements statements and 77 change requests were collected from three industrial datasets spanning diverse domains. These datasets were manually labeled in a collaborative effort between the authors and our industry partners, with domain experts, each possessing extensive experience in requirements engineering and change management, performing the initial annotations to ensure that all labels were both accurate and contextually relevant.

To minimize subjectivity and guard against annotation bias, each change request was independently annotated by two different experts. We tracked inter-annotator agreement using Cohen's κ and set a threshold of $\kappa \geq 0.75$ for acceptable consistency; any annotation batches that fell below this threshold were re-examined and re-annotated after further refinement of our criteria. We developed explicit annotation criteria defining how to link change requests to impacted requirements by identifying shared domain entities (e.g., RequirementsID or ChangeID), semantic overlaps in phrasing (e.g., "encrypt data" versus "data encryption"), and procedural dependencies (e.g., authentication workflows). These criteria served as the constant reference point for all annotators.

Where discrepancies did arise, they were resolved in structured consensus workshops. In these sessions, the two initial annotators presented their reasoning alongside a third senior reviewer, who facilitated discussion of each divergent case until a unanimous

decision was reached. This format ensured that all edge cases received thorough consideration and that consensus decisions were documented for future reference.

Once consensus was achieved, the fully annotated datasets underwent a final validation step conducted by senior experts from both the research team and our industry partners. These validators reviewed the agreed annotations to catch any remaining inconsistencies or errors, providing an additional safeguard against confirmation bias and further enhancing the reliability of our ground truth.

By combining dual independent annotation, statistical agreement monitoring, structured consensus meetings, comprehensive guidelines, and senior-level validation, we established a rigorous, multi-layered process that delivers a high-quality, bias-resilient dataset for evaluating the SRCIA framework.

3.8. Data Preparation

The input of the solution is a change request, which includes sections such as id, title, description, type, and the rationale (reason) for the change. To prepare the dataset, NLP techniques were applied to collect the raw data from the requirements statements and change request forms. First, the text of the title and the description were cleaned to remove inconsistencies and ensure accuracy. This step involved normalization, deduplication, and standardization of the data. Then, the cleaned textual data was tokenized using standard NLP techniques to collect all the tokens (words). The tokens were then normalized using stemming and lemmatization. Additionally, casing and acronyms were normalized. All extracted tokens were transformed into features as inputs for the model. The details are explained in the implementation section.

Therefore, to ready the data for classification in this pre-processing pipeline, the CSV files of requirements statements and change requests were cleaned, tokenized, stop words and punctuation removed, texts stemmed by PorterStemmer, and lemmatized by WordNetLemmatizer, all with Python codes and libraries.

3.9. Implementation

All solutions proposed in this research were implemented using Python, leveraging its robust libraries and frameworks suitable for NLP, ML, and data analysis. Python's versatility and wide range of tools enabled the effective development and evaluation of the proposed models and methodologies.

A comprehensive replication package was prepared and shared publicly on Zenodo, providing the complete Python codes, detailed requirements statements, change scenarios, and the corresponding results. This package was designed to facilitate the replication of the experiments conducted in this research, ensuring transparency and reproducibility.

The replication package, encompassing all relevant materials, is available on Zenodo at the following link: https://zenodo.org/records/14568906. By making this package accessible, the research invites further exploration and validation of the findings by the broader academic and professional communities.

3.10. Summary

This chapter introduced the Software Requirements Change Impact Analysis (SRCIA) Framework, a comprehensive approach to predicting the impact of software requirement changes. The framework integrates ML, NLP, BEIR, and RAG models to address challenges in requirements engineering, leveraging structured, semi-structured, and unstructured datasets collected from industry partners. The chapter outlines the modular architecture of the framework, emphasizing its adaptability, scalability, and feedback loop for continuous improvement. It details the data collection, quality verification, and preparation processes, ensuring robust and high-quality datasets for implementation. The SRCIA framework's stages, from traditional ML approaches to advanced LLM-based solutions, highlight its ability to address varying complexities in requirements change scenarios. Finally, the chapter underscores transparency and reproducibility by providing a public replication package for further exploration of the research findings.

Chapter 4.

Machine Learning Algorithms for Software Requirements Change Impact Prediction

4.1. Introduction

This chapter focuses on implementing the traditional ML approach introduced as the first component of the **SRCIA** framework in Chapter 3. As outlined in the framework, traditional ML techniques serve as the foundational layer for predicting the impact of software requirement changes, particularly for structured datasets where explicit patterns and dependencies can be identified and analyzed. This chapter builds on the theoretical groundwork presented in Chapter 3 by applying and evaluating traditional ML models to demonstrate their practical application and effectiveness in supporting requirements analysts during CIA. In requirements engineering (RE), ML techniques can streamline labor-intensive processes, enabling analysts to focus on tasks requiring domain expertise. The traditional ML models incorporated in the SRCIA framework aim to automate the identification of impacted requirements by predicting how changes propagate throughout a system.

Informed by the systematic literature review detailed in Chapter 2, this chapter addresses the gap in research regarding predictive models for CIA in RE. Unlike existing approaches that focus broadly on requirements traceability or ambiguity resolution, the traditional ML methods implemented here are specifically designed to predict the impact of a given change on existing requirements. This approach aligns with the SRCIA framework's goal of creating a modular and scalable solution for CIA.

ML has been successfully employed in various software engineering (SE) tasks, such as requirements traceability and classification (Dalpiaz, Dell'Anna, et al. 2019; Li & Huang 2018), ambiguity management(Yang et al. 2011), test case generation (Ali et al. 2010), prediction of code changes (Giger, Pinzger & Gall 2012), and software effort estimation (Basri et al. 2016).

To achieve this, we develop five alternative solutions using supervised ML approaches, including Random Forest, Support Vector Machines (SVM), and Decision Trees. These models are trained and evaluated on three real-world datasets containing 891 requirements and 77 change requests. By implementing these models, this chapter validates the effectiveness of traditional ML techniques as described in Chapter 3. Comparisons are also drawn with manual approaches, such as keyword-based analysis of specification documents, to highlight the advantages of ML-based methods in terms of accuracy and efficiency.

This chapter represents a key step in the realization of the SRCIA framework by operationalizing its traditional ML component. The results presented here serve as the foundation for subsequent chapters, which explore the integration of more advanced techniques, such as NLP and LLMs, in the framework.

4.2. Technical Approach and Implementation

CIA in requirements engineering involves predicting the impact of changes to software requirements. This can be framed as a classification problem where each requirement change request is classified as either having an impact (class 1) or not having an impact (class 0) on other requirements. By transforming the problem into a classification task, ML algorithms can be leveraged to predict the likelihood of changes affecting other requirements, thereby automating the CIA process.

The ML approach is summarized in Figure 4.1 and includes five different techniques of ML to develop an automated approach to analyze the requirements change impact and develop a requirement change impact prediction model. Requirements often manifest as textual artifacts represented through models, mathematical specifications, and

similar forms. The research focuses specifically on natural language (NL) requirements, excluding models or requirements articulated in formal languages.

The initial phase involves preparing this dataset through text cleansing and pre-processing using NLP techniques, thus converting the data into a format conducive to subsequent computational analysis. Subsequently, an NLP pipeline is applied to pre-process all requirements documents and change requests, capturing semantic, syntactic, and contextual similarities and connections between terms, thereby producing annotated (labeled) data integrated into the dataset as metadata. These established relationships are utilized in training a ML algorithm to discern dependencies between requirements. Ultimately, the algorithm generates a list of affected requirements based on the likelihood of each requirement being impacted by a requested change.

This list is ordered from the most to the least affected, aiding analysts in decision-making, whether to accept or decline a proposed change. The trained ML algorithm can potentially furnish a predictive model for anticipating the impact of forthcoming change requests. Additionally, the system accommodates user input ('approve' or 'disapprove') for a given change, enabling the model to learn from human decisions, thus maintaining a human-involved approach in the process.

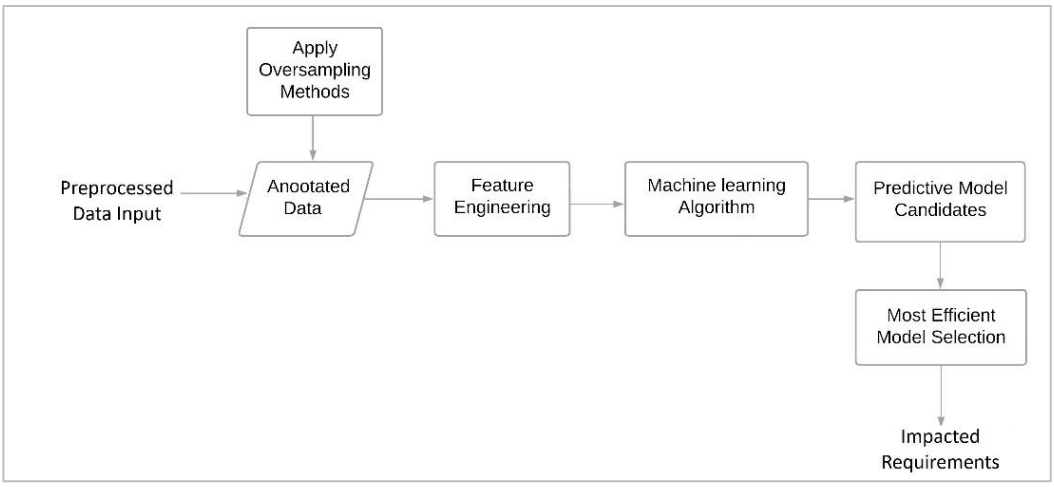

Figure 4.1.ML Model Approach

4.3. Sequential Steps of the ML Approach

Step 1: Data Collection and Pre-processing

The initial phase involves collecting and pre-processing historical data on software requirements and change requests. Inputs for this phase include requirements documents and change requests. Raw data is collected from various sources, including requirement specifications, change logs, and impact reports. The pre-processing phase involves text cleansing and normalization using NLP techniques. This step includes removing noise such as stop words and punctuation, handling missing values, and performing initial feature extraction. These pre-processing activities convert the raw data into a structured format that is suitable for computational analysis.

Additionally, the pre-processing phase handles data quality issues by addressing inconsistencies and ensuring the data is complete and accurate. This phase is crucial for creating a reliable dataset that can be used for further analysis. The output of this phase is a clean and pre-processed dataset that provides a solid foundation for subsequent analytical steps.

Step 2: Database Preparation

Following data pre-processing, the cleaned data is organized into a structured database. This database serves as the input for the subsequent pre-processing pipeline, which performs more advanced NLP tasks. The structured database ensures that the data is efficiently stored and can be easily accessed for further processing.

In this step, the data is formatted to meet the requirements of the pre-processing pipeline. The structured database facilitates the application of semantic, syntactic, and contextual analysis, which are essential for understanding the relationships and dependencies between different requirements. The output is a well-organized database that is ready for detailed NLP analysis.

Step 3: Advanced Pre-processing Pipeline

The advanced pre-processing pipeline applies semantic, syntactic, and contextual analysis to the data. Semantic analysis captures the meaning of terms, syntactic analysis

examines the grammatical structure, and contextual analysis understands the context in which terms are used. These analyses help to annotate the data, adding metadata that captures the relationships and dependencies between different requirements.

The annotated data resulting from this pipeline provides a rich representation of the requirements, incorporating detailed semantic, syntactic, and contextual information. This step is crucial for enhancing the Dataset-With meaningful annotations that will be used for ML. The output is an annotated dataset enriched with semantic, syntactic, and contextual metadata.

Step 4: Handling Class Imbalance with Oversampling Methods

To address the class imbalance in the data, oversampling methods such as the SMOTE + Edited Nearest Neighbors (SMOTEENN) are applied. This step takes the annotated data as input and generates a balanced Dataset-Where the minority class is adequately represented. Handling class imbalance ensures that the ML models trained in subsequent steps do not suffer from biases due to class imbalance.

This phase involves creating synthetic examples for the minority class to balance the class distribution. By doing so, the dataset becomes more suitable for training robust ML models that can generalize well to unseen data. The output is a balanced and annotated dataset that is ready for feature engineering.

Step 5: Feature Engineering

In the feature engineering phase, relevant features are extracted and engineered from the balanced dataset. This phase focuses on creating new features or transforming existing ones to better capture the characteristics of the requirements and their changes. For instance, features such as term frequency-inverse document frequency (TF-IDF) and word embeddings are used to represent textual data quantitatively.

Feature engineering is a critical step in the ML pipeline, as it directly impacts the model's performance. Well-engineered features help the model to better understand the underlying patterns in the data. The output of this phase is a feature matrix that encapsulates the engineered features for each data point.

Step 6: Training ML Models

Using the feature matrix, various ML algorithms are trained, including decision trees, random forests, support vector machines, and neural networks. The training phase involves splitting the Dataset-Into training and validation sets, optimizing hyperparameters through techniques like grid search, and evaluating model performance. The goal is to identify the best-performing model based on metrics such as precision and recall.

During this phase, cross-validation techniques are used to ensure that the models generalize well to unseen data. Hyperparameter tuning helps find the optimal settings for each model, enhancing their predictive capabilities. The output of this phase is a set of trained predictive model candidates with robust performance metrics.

Step 7: Model Testing and Evaluation

To evaluate the generalization ability of the trained models, they are tested on an unseen test dataset. This evaluation phase assesses the models' effectiveness using performance metrics. If the precision and recall are low, further tuning is performed to improve recall, even if precision is initially compromised. The performance evaluation report highlights the strengths and weaknesses of each model, providing insights into their suitability for deployment.

This phase is crucial for validating the models and ensuring that they meet the required performance standards. The output is a comprehensive performance report that informs the selection of the most efficient model.

The main aim of our ML process was to introduce and capture which requirements are impacted and not impacted by a given change or a set of requested changes. The baseline is training our model in a specific domain, and if the resulting precision and recall are as expected, the next step is trying it in different domains

Step 8: Practical Application and User Feedback

The final phase involves the practical application of the selected predictive model. The trained model is integrated into a software tool designed for requirements analysts. This tool allows analysts to input change requests and receive a list of impacted

requirements, ordered from most to least affected. The system also accommodates user feedback by enabling analysts to approve or disapprove changes, which helps refine the model over time.

To improve the precision, the human in the loop is included by the user's input, which is to 'approve' or 'disapprove' a given change, enabling the model to learn from human responses. So, by incorporating user feedback, the model continuously improves and adapts to new data. This human-in-the-loop approach ensures that the tool remains relevant and accurate. The output of this phase is a usable software tool for predictive CIA, accompanied by comprehensive user documentation and guidelines.

4.4. Implementation

This section details the implementation process for applying machine learning algorithms to predict the impact of requirement changes. It outlines the critical steps taken to prepare the datasets, optimize the models, and enhance their predictive performance. The subsections explore the techniques used for feature engineering, model training, and evaluation, as well as strategies to address challenges like class imbalance and overfitting. Through these steps, this section provides a comprehensive view of how the machine learning models were operationalized to achieve the research objectives.

4.4.1. Apply Class Rebalancing Techniques

The primary purpose of a reliable ML model is to generalize effectively to various domains and to generate new data that is evaluated for its performance over time as it learns from training data. The main problem of imbalanced datasets is that they result in sub-optimal classification models. It might provide misleading conclusions as the distribution of observations in the training set is unequal across the classes (Sikora, Tenbergen & Pohl 2012). An ideal model should not suffer from overfitting or underfitting (Zamani, Zowghi & Arora 2021). Training with unbalanced Dataset-Is is one of the most critical concerns confronting ML research. Imbalanced class distributions

have an impact on classifier training, resulting in a negative bias towards the majority classes. It could also lead to significant inaccuracy, or even exclusion, of the minority classes (Dablain, Krawczyk & Chawla 2022; Galar et al. 2012).

In this study, the focus is on two-class imbalanced datasets. The initial labeled dataset had a class imbalance for the binary classifier since the number of examples belonging to class 0 is more than those belonging to class 1. In Dataset-W, 1890 out of 2016 examples belong to class 0 (93.75%); in the Dataset-I, 98.8% of data belongs to class 0; and in the Telecommunications Dataset, the percentage of class 0 is 99.4%. Therefore, the predictor almost always predicts any given sample as belonging to class 0, achieving very high scores like precision and recall for class 0 and low scores for class 1. Table 4.1 shows the distribution of classes 1 and 0 in all datasets.

Table 4.1. Distribution of classes 1 and 0

Dataset	**# Requirements Statement**	**# Change request**	**Matrix**
Dataset-W	72	28	2016 samples • 1890 class 0 • 126 class 1
Dataset-O	627	34	21250 samples • 21126 class 0 • 124 class 1
Dataset-I	193	15	2895 samples • 2862 class 0 • 33 class 1

Earlier research in ML has repeatedly demonstrated an increase in performance when class rebalancing approaches are used (Dablain, Krawczyk & Chawla 2022; Seiffert et al. 2010; Tantithamthavorn, Hassan & Matsumoto 2018; Wang & Yao 2013). To mitigate

the problem, empirical experiments were conducted to systematically test several combinations of commonly used rebalancing methods, including over-sampling and under-sampling techniques, to determine which one works best for this case. According to earlier studies, a combination of oversampling and undersampling techniques has proven beneficial and thus can be considered the best solution (Sowjanya & Mrudula 2022; Tantithamthavorn, Hassan & Matsumoto 2018). To study the impact of resampling techniques on the models, Condensed Nearest Neighbors + Tomek Links, SMOTE + Tomek Links (SMOTE-Tomek Links), and SMOTE + Edited Nearest Neighbors (SMOTEENN) were applied.

Initially, the dataset had a severe class imbalance with a ratio of 1:33, meaning there was one instance of the minority class (class 1) for every 33 instances of the majority class (class 0). Results reveal that by transforming the data with resampling methods, the ratio increased from 1:33 to 1:2668 with a balanced distribution of 2862 in the minority class for the Dataset-I. The same increase was observed in the Dataset-W and Dataset-O, proving that SMOTEENN, a combined technique that incorporates both over-sampling and under-sampling methods, outperforms in this case in practice. Table 4.2 shows the distribution of the data before and after resampling.

Table 4.2. Class distribution before and after resampling

Dataset	**Original Dataset**		**Transformed Dataset**	
	Majority Class	Minority Class	Majority Class	Minority Class
Dataset-W	1890	126	1767	1394
Dataset-O	21126	124	21104	20140
Dataset-I	2862	33	2837	2668

Evaluating how the class distribution changed before and after SMOTEENN was implemented is a critical component of its effectiveness. Pre- and post-resampling class distributions were compared using a detailed bar plot created by a Matplotlib-based

Python implementation. In Figures 4.2, 4.3, and 4.4, bar charts of the original set of data from dataset-W, dataset-O and dataset-I showcased the rebalancing effect of SMOTEENN, elucidating its ability to rectify unbalanced class distribution in our data.

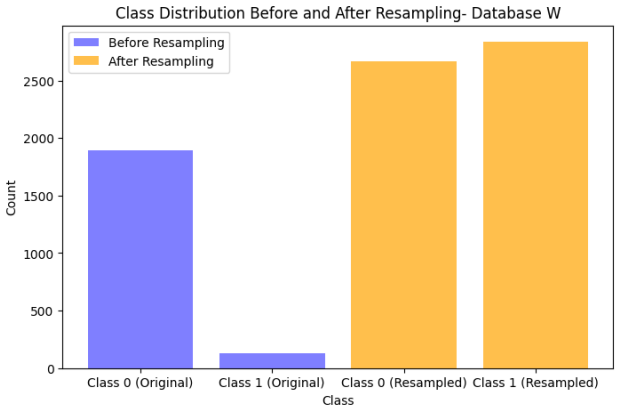

Figure 4.2.Dataset-W Class Distribution of Original and Resampled Data

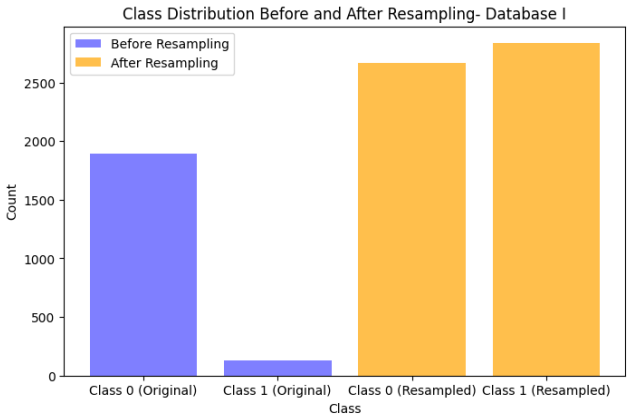

Figure 4.3.Dataset-I Class Distribution of Original and Resampled Data

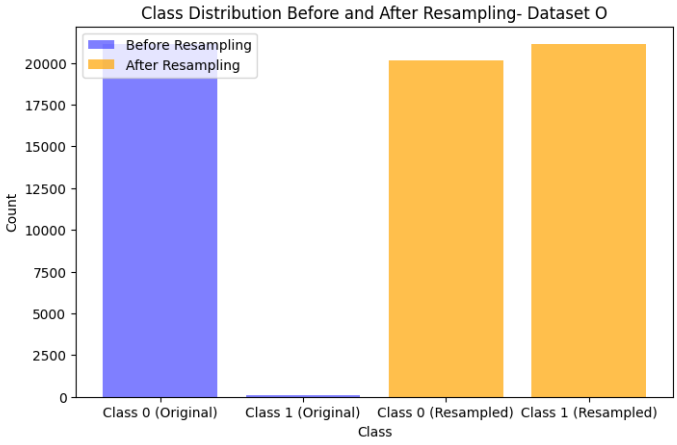

Figure 4.4. Dataset-O Class Distribution of Original and Resampled Data

4.4.2. The Proposed ML Model

In this step, an ML classification model was proposed for training. In the development of the ML model, its attributes were collected from previous steps. Since existing literature proposed multiple alternatives and, there is no evidence to show which classifier has the best overall performance in general (Catolino et al. 2018; Yang et al. 2020), five different classifiers were experimented with, and the results were compared

in terms of accuracy and performance. Therefore, the ML method uses five classifiers of Decision Tree, Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gaussian Naive Bayes (NB), which are the most frequently used algorithms in RE tasks based on the result of our published paper (Zamani, Zowghi & Arora 2021). In terms of using SVM, prior research showed that utilization of SVMs by gappy n-gram kernels, including a non-zero decay factor, would present a highly impressive solution for requirements classification (e.g. (Cortes, Haffner & Mohri 2004; Shakeri et al. 2019)). Based on the mentioned earlier research, non-contiguous n-gram kernels were used in the text of requirements classification and rational kernels and SVM were applied to perform this method. All ML algorithms have been implemented in Python with the Scikit-Learn Library uploaded in the replication package.

4.4.3. Identifying the Dependencies

The identified dependencies in this step are used to train and test the ML classifier. Syntactic, semantic, and textual content were considered to identify all existing dependencies for the collection of requirements and a given change. Similarity measures, both syntactic and semantic, were used to investigate the closeness of a given change with each of the existing requirement statements. Given the variety of similarity measures available, it is critical to objectively study which one is best suited to a particular type or rationale of a change request (Nejati et al. 2016).

Consequently, the following algorithms were applied to the data: Jaccard, Levenstein, Pairwise Cosine Similarity (Bag of Words with Term Frequency (TF) with Cosine similarity), Bidirectional Encoder Representations from Transformers (BERT)

with Cosine similarity, Glove with Cosine similarity, Glove with Word Movers Distance (WDM) similarity, CrossEncoder and Infersent.

To facilitate the application of Cosine similarity, an essential step involved the transformation of sentences into vectors. This transformation leveraged all lemmatized tokens extracted through our pre-processing pipeline. These tokens were subsequently molded into features, serving as inputs for our model. This transformation was achieved

via a Bag of Words (BOWs) approach coupled with TF-IDF (term frequency-inverse document frequency) or Word Embeddings, enabling a comprehensive representation of the textual data.

These algorithms were implemented using Python, and their implementations are available within our comprehensive replication package and are accessible on Zenodo.

4.4.4. Generating Features

To increase the prediction power of the model, additional feature engineering techniques were considered. This step involved collecting change request features that might have the highest impact on the impact prediction findings. The taxonomy of requirements changes was reviewed to assess their practical values in the change management process. Prior research showed that domains of the market, organization, vision, type, specification, solution, time, type, reason, and origin are the most important features in classifying requirements (Catolino et al. 2018; McGee & Greer 2011; Saher, Baharom & Ghazali 2017). Since these detailed data are not available in most real-world datasets, only the type of changes, including additions, deletions, and modifications, were selected as a feature of the ML model. In this stage, efforts were made to extract additional valuable features from datasets and feed them to the ML model to improve prediction performance. The performance will be evaluated by comparing it with a manual approach, which will be considered as a baseline approach.

4.4.5. Hyperparameters

Hyperparameters are essential for adjusting the model's behaviour and enhancing its prediction power to maximize the performance of ML classifiers. Hyperparameters, which are predefined configuration options not discovered through data analysis, were carefully utilized to alter the behaviour of classifiers like Random Forest, Support Vector Machine (SVM), and Logistic Regression.

For instance, for the Random Forest classifier, a grid search method (Bergstra & Bengio 2012) combined with cross-validation was used for evaluation. This approach aimed to

improve the model's performance by exploring different combinations of hyperparameters, such as the number of trees in the forest (n_estimators) and the maximum depth of the tree (max_depth).

In the case of SVM tuning, the grid search method was employed to find the best combination of the regularization parameter (C) and the kernel coefficient (gamma) (Hsu, Chang & Lin 2003). These parameters were further adjusted based on domain knowledge to ensure contextually relevant optimization. For the Decision Tree model, the grid search method was used to adjust hyperparameters such as the maximum depth of the tree (max_depth) and the minimum number of samples required to split an internal node (min_samples_split).

In contrast, Gaussian Naive Bayes does not have hyperparameters to tune like the other models. Therefore, no hyperparameter optimization was necessary for this classifier.

Furthermore, the implementation of a pipeline employing Synthetic Minority Over-sampling Technique and Edited Nearest Neighbors (SMOTEENN) resampling techniques (Chawla et al. 2002; Tomek 1976) was an essential phase in the optimization process across all classifiers. This method ensured that every classifier could handle unbalanced class distributions, consequently resolving the class imbalance problem.

4.4.6. Computational Cost Considerations

When applying machine learning algorithms for CIA, computational cost is an important practical consideration, especially when these models are deployed in resource-constrained or real-time environments. In this study, we evaluated the computational overhead of the implemented ML models, focusing on aspects such as training time, inference speed, and scalability.

Random Forest, being an ensemble method, is relatively computationally expensive due to the creation of multiple decision trees and the need for aggregation during inference. However, its training phase can be parallelised, which mitigates the time cost

to some extent. In our experiments, Random Forest showed moderate training times but relatively fast prediction times once the model was trained.

SVMs are known for their robust performance, particularly in high-dimensional spaces. However, their computational cost increases significantly with larger datasets due to the quadratic or cubic complexity involved in solving the optimisation problem during training. Training time was the longest for SVMs in our experiments, though inference time remained manageable.

Decision Trees are lightweight models in terms of computational cost. Both training and inference are fast, making them suitable for applications requiring quick turnaround times. However, they tend to overfit, which may require pruning or ensemble techniques like Random Forest to maintain generalizability (Singh 2023) (Idrissi Khaldi et al. 2025)

Logistic Regression is computationally efficient, with relatively low training and inference costs. Its simplicity and interpretability make it suitable for baseline comparisons, although it may struggle to capture complex patterns in the data.

Naive Bayes, as a probabilistic classifier, has minimal training overhead and scales well with large datasets. Its simplicity results in the lowest computational cost among all models tested. However, its assumptions of feature independence may limit its effectiveness in capturing intricate dependencies.

Overall, the trade-off between performance and computational cost must be carefully considered. While models like SVM and Random Forest may offer higher predictive power, they demand greater computational resources. In contrast, Logistic Regression and Naive Bayes offer faster execution but may underperform in complex scenarios. This highlights the importance of selecting the appropriate model based on the specific needs and constraints of the deployment context (Singh 2023) (Idrissi Khaldi et al. 2025)

Table 4.3. ML algorithms results

		RandomForest			DecisionTree			NaiveBayers			Logistic Regression			SVM		
Dataset	Change ID	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Dataset-I	Case 1	0	0	0	0.5	0.33	0.4	0.33	1	0.5	0.33	1	0.5	0.5	0.67	0.57
	Case 2	0.13	1	0.24	0.33	1	0.5	0.07	1	0.14	0.07	1	0.14	0.25	1	0.4
	Case 3	0.33	1	0.5	0.1	1	0.17	0.25	1	0.4	0.25	1	0.4	0.29	1	0.45
	Case 4	0.33	0.33	0.33	0.33	0.33	0.33	0.5	0.67	0.57	0.5	0.67	0.57	0.5	0.67	0.57
	Case 5	1	1	1	0.17	1	0.29	0.33	1	0.5	0.33	1	0.5	0.25	1	0.4
	Case 6	1	1	1	0.67	1	0.8	0.67	1	0.8	0.67	1	0.8	1	1	1
	Case 7	0.67	1	0.8	0.67	1	0.8	0.4	1	0.57	0.4	1	0.57	0.5	1	0.67
	Case 8	0.07	1	0.12	0.04	1	0.08	0.06	1	0.11	0.06	1	0.11	0.03	1	0.05
	Case 9	0.4	1	0.57	0.18	1	0.31	0.33	1	0.5	0.33	1	0.5	0.5	1	0.67
	Case 10	0.5	0.5	0.5	0.5	1	0.67	0.33	0.5	0.4	0.25	0.5	0.33	0.5	0.5	0.5
	Case 11	0.09	0.75	0.16	0.08	0.75	0.14	0.07	0.75	0.13	0.07	0.75	0.13	0.03	1	0.05
	Case 12	0.33	1	0.5	0.33	1	0.5	0.33	1	0.5	0.33	1	0.5	0.33	1	0.5
	Case 13	1	1	1	1	1	1	0.67	1	0.8	0.67	1	0.8	1	1	1
	Case 14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Case 15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Dataset-W	C1	1	0.5	0.67	1	0.5	0.67	1	1	1	1	1	1	1	0.5	0.67
	C2	1	0.22	0.36	1	0.22	0.36	0.67	0.22	0.33	0.67	0.22	0.33	1	0.22	0.36
	C27	0.13	1	0.23	0.13	1	0.22	0.08	1	0.15	0.08	1	0.15	0.1	1	0.19
	C26	0.8	0.8	0.8	0.13	0.8	0.22	0.44	0.8	0.57	0.44	0.8	0.57	0.5	0.8	0.62
	C25	0.14	1	0.25	0.13	1	0.22	0.07	1	0.13	0.07	1	0.13	0.07	1	0.13
	C24	0.18	0.86	0.29	0.16	0.86	0.27	0.16	1	0.27	0.16	1	0.27	0.17	1	0.29
	C23	0.05	1	0.1	0.05	1	0.09	0.04	1	0.08	0.16	1	0.27	0.04	1	0.07
	C22	0.47	1	0.64	0.4	0.75	0.52	0.31	1	0.47	0.31	1	0.47	0.42	1	0.59
	C21	0.31	1	0.47	0.2	0.75	0.32	0.25	1	0.4	0.25	1	0.4	0.27	1	0.42
	C20	0.5	0.6	0.55	0.38	0.6	0.46	0.36	0.8	0.5	0.36	0.8	0.5	0.39	1	0.56
	C19	1	0.33	0.5	0.5	0.33	0.4	0.33	0.67	0.44	0.33	0.67	0.44	0.67	0.67	0.67
	C18	0.33	0.25	0.29	0.11	0.25	0.15	0.5	0.25	0.33	0.5	0.25	0.33	0.5	0.5	0.5
	C17	0.56	1	0.72	0.33	1	0.5	0.42	1	0.59	0.42	1	0.59	0.42	1	0.59
	C16	0.19	0.67	0.3	0.17	0.67	0.27	0.17	0.83	0.28	0.17	0.83	0.28	0.13	0.83	0.22
	C15	0.23	1	0.38	0.18	1	0.3	0.17	1	0.29	0.17	1	0.29	0.14	1	0.25
	C14	0.14	0.5	0.22	1	1	1	0.25	1	0.4	0.25	1	0.4	0.29	1	0.45
	C13	0.29	0.83	0.44	0.19	0.83	0.31	0.25	1	0.4	0.25	1	0.4	0.26	1	0.41
	C12	0.06	0.13	0.08	0.05	0.13	0.07	0.05	0.13	0.07	0.05	0.13	0.07	0.17	0.5	0.25
	C11	0.22	1	0.36	0.11	1	0.2	0.15	1	0.26	0.15	1	0.26	0.14	1	0.24
	C10	0.25	0.33	0.29	0.2	0.33	0.25	0.2	0.67	0.31	0.2	0.67	0.31	0.29	0.67	0.4
	C9	0.56	0.56	0.56	0.67	0.67	0.67	0.3	0.78	0.44	0.3	0.78	0.44	0.43	0.67	0.52
	C8	0.23	1	0.37	0.19	1	0.32	0.21	1	0.34	0.21	1	0.34	0.16	1	0.28
	C7	0.38	0.75	0.5	0.33	0.75	0.46	0.14	0.75	0.24	0.14	0.75	0.24	0.18	0.75	0.29
	C6	0.36	0.71	0.48	0.42	0.71	0.53	0.28	1	0.44	0.28	1	0.44	0.33	1	0.5
	C5	0.6	0.86	0.71	0.25	0.29	0.27	0.4	0.86	0.55	0.4	0.86	0.55	0.5	0.86	0.63
	C4	1	1	1	1	1	1	0.5	1	0.67	0.5	1	0.67	1	1	1
	C3	0.17	1	0.29	0.14	1	0.25	0.1	1	0.18	0.1	1	0.18	0.11	1	0.2
	C28	0	0	0	0	0	0	0.03	0.5	0.05	0.03	0.5	0.05	0	0	0
Dataset-O	CR 007	0.05	1	0.09	0.05	1	0.09	0.04	1	0.08	0.04	1	0.07	0.04	1	0.07

CR 504340	0.09	0.71	0.17	0.09	0.71	0.17	0.1	0.71	0.18	0.12	0.71	0.2	0.1	0.71	0.18
CR 504334	0.03	1	0.05	0.02	1	0.04	0.03	1	0.05	0.02	1	0.03	0.02	1	0.03
CR 504335	0.19	1	0.32	0.19	1	0.32	0.21	1	0.35	0.33	1	0.5	0.33	1	0.5
CR 504336	0.05	1	0.1	0.04	1	0.08	0.04	1	0.08	0.04	1	0.08	0.05	1	0.1
CR 504337	0.08	1	0.14	0.08	1	0.14	0.14	0.8	0.24	0.11	1	0.19	0.11	1	0.21
CR 504338	0.04	1	0.07	0.04	1	0.07	0.07	1	0.12	0.05	1	0.09	0.04	1	0.07
CR 504339	0.02	1	0.05	0.02	1	0.05	0.03	1	0.06	0.02	1	0.04	0.02	1	0.03
CR 504341	0.01	1	0.01	0.01	1	0.01	0.01	1	0.02	0.01	1	0.02	0.01	1	0.01
CR 503689	0.1	1	0.17	0.09	1	0.17	0.08	1	0.15	0.2	1	0.33	0.2	1	0.33
CR 504342	0.2	1	0.33	0.21	1	0.35	0.19	1	0.32	0.17	1	0.29	0.18	1	0.3
CR 504793	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CR 504799	0.09	1	0.17	0.09	1	0.17	0.11	0.8	0.19	0.07	0.8	0.12	0.08	1	0.14

CR 600203	0.33	0.33	0.33	0.02	0.56	0.04	0.5	0.33	0.4	0.33	0.33	0.33	0.2	0.33	0.25
CR 600204	0.03	1	0.06	0.03	1	0.06	0.03	1	0.05	0.03	1	0.05	0.03	1	0.05
CR 60200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CR 504333	0.01	0.5	0.02	0.01	0.5	0.02	0.01	0.5	0.02	0.01	0.5	0.02	0.01	0.5	0.03
CR 504332	0.03	1	0.06	0.03	1	0.06	0.04	1	0.08	0.04	1	0.08	0.04	1	0.08
CR 504331	0.09	1	0.17	0.09	1	0.17	0.1	1	0.19	0.07	1	0.13	0.07	1	0.13
CR 504330	0.03	1	0.05	0.02	1	0.05	0.02	1	0.05	0.02	1	0.04	0.02	1	0.04
CR 504329	0.16	1	0.27	0.16	1	0.27	0.1	1	0.18	0.14	1	0.25	0.17	1	0.29
CR 504328	0.04	1	0.07	0.04	1	0.07	0.03	1	0.06	0.04	1	0.07	0.03	1	0.07
CR 504327	0.17	1	0.29	0.17	1	0.29	0.09	1	0.16	0.38	1	0.55	0.25	1	0.4
CR 504326	0.1	1	0.18	0.1	1	0.18	0.06	1	0.11	0.09	1	0.16	0.07	1	0.13
CR 504325	0.14	1	0.24	0.14	1	0.24	0.09	1	0.16	0.11	1	0.2	0.1	1	0.18

CR 504324	0.01	1	0.02	0.01	1	0.02	0.01	1	0.02	0.01	1	0.02	0.01	1	0.02
CR 504323	0.08	1	0.15	0.08	1	0.15	0.04	1	0.08	0.08	1	0.14	0.14	1	0.25
CR 504322	0.33	0.4	0.36	0.33	0.4	0.36	0.11	0.4	0.17	0.67	0.4	0.5	1	0.4	0.57
CR 504321	0.02	1	0.05	0.03	1	0.05	0.02	1	0.04	0.02	1	0.04	0.02	1	0.04
CR 504311	0.02	0.33	0.03	0.02	0.33	0.03	0.02	0.5	0.04	0.02	0.67	0.04	0.03	0.67	0.05
CR 504310	0.02	1	0.04	0.02	1	0.04	0.02	1	0.03	0.02	1	0.04	0.02	1	0.04
CR 504139	0.09	1	0.17	0.09	1	0.17	0.06	1	0.11	0.12	0.57	0.2	0.1	0.57	0.17
CR 503779	0.05	0.75	0.09	0.04	0.75	0.07	0.04	0.75	0.08	0.06	0.5	0.1	0.05	0.5	0.09
CR 60202	1	1	1	0.01	1	0.02	1	1	1	0.5	1	0.67	0.5	1	0.67

4.5. Results Analysis and Evaluation

Table 4.3 presents the evaluation metrics for the classifiers, highlighting three key observations. First, there is a significant improvement in recall for all classifiers after addressing the class imbalance issue. This finding underscores the effectiveness of resolving class imbalance in enhancing the overall performance of the classifiers. Second, the Precision-Recall disparities reveal variations in precision and recall values across different ML algorithms and datasets. Third, the F1-scores, which provide a

harmonic mean of precision and recall, offer a more balanced view of the classifier's performance, particularly in the presence of class imbalance.

The effectiveness of any ML model is determined by measures such as True Positive Rate, False Positive Rate, True Negative Rate, and False Negative Rate. Therefore, evaluation metrics are used to assess the prediction results. The performance of the proposed model is evaluated using standard metrics Precision (P) and Recall (R), as well as the F1-score (F1). The inclusion of F1-scores provides additional insights into the classifiers' performance, as the F1-score considers both precision and recall and is particularly useful in evaluating models on imbalanced datasets. For instance, Random Forest exhibited the highest F1-scores in several cases, particularly in the Dataset-I (e.g., Case 6 with an F1-score of 1), indicating a strong balance between precision and recall. Decision Tree showed moderate F1-scores across datasets, with notable high performance in specific cases such as Dataset-W C4 (F1-score of 1). Naive Bayes generally showed high recall but varied in precision, leading to fluctuating F1-scores; for example, in the Dataset-I Case 3, it achieved an F1-score of 0.4. Logistic Regression displayed consistent performance with high F1-scores in multiple cases, such as Dataset-I Case 6 (F1-score of 0.8), demonstrating its reliability. Support Vector Machines (SVM) consistently performed well with balanced F1-scores, especially in the Dataset-O CR504335 (F1-score of 0.5).

Additionally, to evaluate the prediction model, the k-fold Cross-Validation (k = 10) technique is utilized to check how well the classifier performs on unseen data. This validation technique is one of the most widely used model validation methods in imperfection prediction studies (Thakur et al. 2021).

An 80/20 split of the sample data was employed for training and testing sets for each dataset. Since the model was trained on only 80 percent of the sample data, there is a significant risk of missing some information, resulting in a high bias that achieves a perfect score but fails to predict anything valuable on unseen data. Cross-validation is a common and valuable ML validation method for tackling this problem. The basic idea is to isolate test data through data sampling to see if the trained model fits new situations (Bennin et al. 2018). The k-fold cross-validation method splits the data into k folds of

approximately the same size, where each fold contains similar proportions of the defective ratio. One-fold is used for testing, and the remaining k-1 folds are used for training. The value of k-fold cross-validation is that all data can be used for training and testing (Danjuma 2015; Tantithamthavorn, Hassan & Matsumoto 2018). It also reduces the bias associated with random training set selection and holds out observations (Erdoğan & Namlı 2019).

There are various methods for performing cross-validation. To sample the annotated datasets, a more robust sampling technique, Repeated Stratified K-Fold cross-validation (CV) with a split of 10, was used. To examine the effect of each sampling technique on the datasets, five different classifiers, as discussed in Section 3 were employed. (Bal & Kumar 2020; Barua et al. 2014; Bennin et al. 2018)

The results presented in Table 4.3 form the basis for the subsequent analysis and discussion in this section and the following sections. The inclusion of F1-scores highlights that while some algorithms excel in either precision or recall alone, achieving a high F1-score signifies a more effective overall performance in the presence of imbalanced classes. This balanced metric is crucial for understanding the practical applicability of the classifiers in real-world scenarios where both precision and recall are important. In conclusion, while precision and recall provide insights into specific aspects of the classifiers' performance, the F1-score offers a comprehensive evaluation, confirming the robustness and reliability of the proposed models across different datasets and scenarios.

4.6. Dataset Validity and Size

The datasets used in this study were carefully selected from real-world industrial projects across various domains, including software functional and non-functional requirements. This diverse selection enhances the validity and generalizability of our findings. The sizes of the datasets are sufficient to train and validate the ML models effectively. Each data set contains a substantial number of data points, ensuring that the models have enough information to learn from and generalize well to new data. The

pre-processing and feature extraction steps further ensure that the data is of high quality and accurately represents the underlying requirements and their changes.

Furthermore, all original change requests were considered to ensure comprehensive coverage of the dataset. The datasets included a significant percentage of original change requests, which were carefully selected and validated. Specifically, all original change requests were included in the analysis. This comprehensive inclusion provides a representative sample of the data, ensuring that the models are trained and evaluated on a broad spectrum of change scenarios. This approach helps capture the variability and complexity inherent in real-world change requests, thereby enhancing the robustness and applicability of the proposed model.

4.7. Comparative Analysis with State-of-the-Art Approaches

To establish the relative performance and improvement over existing methods, the proposed model was compared with state-of-the-art approaches, including those by Arora et al. (2015a). The results indicate that the proposed model achieves higher precision and recall, demonstrating its effectiveness in predicting the impact of software requirements change. This comparative analysis validates the model's performance and highlights its contributions to the field.

4.8. Importance of Precision vs. Recall

In terms of the relative importance of recall versus precision in this approach, both metrics are crucial. Human oversight is included to improve precision. High precision means that the prediction is very likely to be correct, making the approach trustworthy. However, recall needs to be high even if precision is low; otherwise, the model is ineffective. Therefore, the data is fed into multiple ML algorithms, and the algorithm that results in the highest recall is selected. The main goal of this study is to identify the most impacted requirements and miss as few affected ones as possible. In other words, false positives are more critical than false negatives (Aryani et al. 2009).

4.9. Threats to Validity

The threats to the validity of this study are categorized into internal and external validity concerns.

4.9.1. Internal Validity

One of the primary threats to internal validity in this study is the quality and accuracy of the data used, particularly in labelling impacted requirements. The process of manually checking the impacted requirements in the change history introduces potential errors. Although datasets were selected carefully and structured, detailed consultation sessions were conducted with domain experts to ensure an accurate understanding of their requirements documents and change history. There remains a risk that data collection may not be entirely complete. This incompleteness can stem from varying degrees of abstraction and levels of detail in the requirements documents. To mitigate this threat, data validation was performed with domain experts who have extensive experience in the change management process, reducing the risk of inaccuracies.

Another possible limitation is the stability of requirements over time. Requirements frequently change, which poses a challenge for the ML model, as it must be retrained with each change in requirements. This fluidity, while a common difficulty in requirements engineering, becomes a more severe barrier when applying ML techniques. The need to retrain the model with each change can be resource-intensive and may affect the model's performance. Additionally, changes in labelling due to evolving requirements can introduce inconsistencies and further complicate the training process (Zamani, Zowghi & Arora 2021).

4.9.2. External Validity

A pivotal threat to external validity is the uncertainty inherent in the data, which forms part of the requirements. Despite the primary goal of achieving high precision, recall, and accuracy with the ML model, numerous experiments are necessary to ensure that

these goals are attainable and that the provided data is sufficient. Uncertainty in data quality and completeness can impact the generalizability of the study's findings (Wan et al. 2021).

The generalizability of the datasets represents another possible threat. Although efforts were made to ensure the datasets are as broad and representative as possible by selecting data from real-world industries across different domains and of various sizes, there is still a potential limitation in terms of how well these datasets represent the broader landscape. The data included both functional and non-functional software requirements to ensure comprehensive coverage. It is important to note that the accuracy of the approach does not depend on the specific domain but rather on the quality and clarity of the written requirements and change requests. High levels of inconsistency in documenting requirements, including spelling mistakes, structural differences, terminology, and vocabulary variations, can complicate and introduce errors into automated classification processes (Abad et al. 2017a; Zamani, Zowghi & Arora 2021).

4.10. Discussion

This section provides an in-depth analysis of the proposed solution, addressing its strengths, limitations, and performance across different contexts. The discussion evaluates the algorithms implemented, highlighting their effectiveness and areas for improvement. It also examines the datasets used, considering their role in shaping the outcomes and generalizability of the solution. Through these discussions, the section aims to present a balanced view of the research findings and their practical implications.

4.10.1. Limitations of the Proposed Solution

While the proposed methodology offers several advantages, it also has limitations. The model's performance is highly dependent on the quality and quantity of historical data. Insufficient or noisy data can adversely affect the model's accuracy, making comprehensive data collection and pre-processing crucial. Identifying the most relevant

features is critical for model performance; incorrect feature selection can lead to poor model predictions, necessitating domain expertise to guide feature engineering. Some ML models, such as neural networks, can be complex and require significant computational resources for training and inference, which can be a constraint for organizations with limited computational capabilities.

Additionally, the model may perform well on the training data but might not generalize to different projects or domains without retraining and fine-tuning, requiring continuous evaluation and adaptation. Effective use of the tool requires a certain level of expertise in ML and CIA, and users may need additional training to understand the tool's functionalities and interpret its outputs accurately. Providing comprehensive documentation and user support can help mitigate this limitation.

4.10.2. Discussion on Algorithms

Looking at the precision and recall values, there is a noticeable variation across different methods and datasets. For instance, in the Random Forest method, there is a significant disparity in precision among different cases within the Dataset-W. While some cases show high precision (e.g., C2, C1), others exhibit quite low precision scores (e.g., C3 and C8). However, in terms of recall, most cases achieve a perfect score of 1.00, indicating that when the actual impact occurs, the model identifies it consistently across these cases. Moving to Dataset-I, the precision scores are consistently low across the cases, indicating a higher rate of false positives. However, the recall varies significantly, with some cases achieving a perfect score while others fall below. Dataset-O presents a mix of precision scores across different cases, with varied performance. While some cases exhibit relatively higher precision (e.g., CR60202), others show lower precision scores. Similarly, recall rates also vary, albeit with generally good performance across most cases.

Regarding the other algorithms, similar to previous results, there appears to be a trade-off between precision and recall in many cases. While some cases exhibit high precision, they often do so at the expense of lower recall and vice versa. This suggests a challenge in achieving both high precision and high recall simultaneously across

different datasets and cases. Understanding the reasons behind this trade-off and optimizing the models to strike a better balance between precision and recall could enhance the overall performance of the system in accurately identifying impacted requirements during change analysis.

In general, there seems to be inconsistency in precision across different cases and datasets, indicating that the models might encounter challenges in precisely identifying the true impact of requirements changes in specific scenarios. Furthermore, even while recall has been shown to perform effectively in identifying true positives, mainly in Dataset-W, it is imperative to address the low precision values observed in various cases across different datasets.

The Random Forest algorithm shows some cases with both high precision and recall. It revealed the RF effectiveness in accurately predicting impactful changes without missing many relevant instances. Besides this, the model struggled for the cases with lower scores. SVM showcased relatively similar performance trends to Random Forest in capturing relevant instances (recall), although there are discrepancies in precision scores between the two models for various cases. The disparities in precision could indicate that utilizing SVM to make precise positive predictions in some situations can be difficult.

The Decision Tree results showcase a different performance compared to both Random Forest and SVM. It also shows varying performances across different cases. The results demonstrate moderate to high precision and recall for some cases, which is a sign of having struggles with accurate positive predictions and capturing all relevant instances for others.

Gaussian NB illustrates varying precision scores for some cases compared to SVM, Decision Tree, and Random Forest models. And finally, Logistic Regression performs consistently with other models when it comes to capturing the most relevant instances (recall). However, just like with other algorithms, there are also difficulties in getting precise positive predictions for particular circumstances.

In summary, when comparing these algorithms in requirements CIA tasks, the overview of their performance proved that all of them are high-performing models while having challenges in precision, indicating difficulties in making accurate positive predictions.

While different algorithms demonstrated their advantages in particular datasets, Support Vector Machines (SVM) was the most effective approach throughout the entire investigation. SVM continuously showed strong and well-balanced performance; it was especially good at identifying intricate patterns in Dataset-I and holding a lead in Dataset-W. Its ability to achieve high recall rates while balancing precision across different datasets signifies its suitability for requirements CIA, particularly when working with complex and diverse datasets. This emphasizes how crucial it is to choose the best algorithm possible while also taking the dataset's unique properties into account. While no single technique was shown to be better than the others across all datasets, SVM was the most reliable and efficient option for precise requirements change impact estimates.

Our results are particularly consistent with current research discussions on ML techniques' limitations. Some peer studies emphasized that ML algorithms may not be able to handle complicated linguistic structures and domain-specific contextual comprehension well enough (Adnan & Akbar 2019; Herm et al. 2023; Lin 2020; Paleyes, Urma & Lawrence 2022; Sarker 2021; Tufail et al. 2023). Although ML is still a potent tool in many fields, including software engineering, there is no one-size-fits-all approach. Careful consideration of the complexities of the task and domain-specific requirements is crucial before the implementation of ML-based solutions. Consequently, our research suggests that the inherent complexities of language and domain-specific subtleties can have a variable impact on the efficacy of ML in requirement analysis tasks.

4.10.3. Discussion on Datasets

Monitoring the patterns within the results across different datasets can provide information about the quality or clearness of the written requirements or change requests. Requirements originating from several sources may differ significantly in terms of language and terminology(Zamani, Zowghi & Arora 2021). Stakeholders, as well as

requirements experts utilize diverse vocabulary and sentence structures. The lack of consistency in requirements documentation leads to the complexity and error-proneness of automated classification(Abad et al. 2017a)

Based on the provided precision and recall scores across different ML models, Dataset-W seems to generally yield higher performance rates across various algorithms in comparison to Datasets Telecommunications and Satellite. This could suggest that Dataset-W might contain clearer or more explicitly outlined requirements, making them easier for models to identify.

Dataset-I showcases a mix of precision and recall values across algorithms, suggesting a mix of clear and more complex requirements or change requests. Some cases exhibit good performance, especially with SVM, while others show challenges for the algorithms, indicating potential variability in the quality or complexity of the requirements.

On the other hand, Dataset-O represents a more challenging environment for predictive modelling. The models struggle to achieve high precision and recall simultaneously within this dataset. The results display a mix of precision and recall values across different algorithms and cases. This variability might indicate a mix of well-defined and ambiguous requirements or change requests within this dataset.

The observed patterns in algorithm performance align closely with the inherent characteristics of the datasets under study. Dataset-W emerges as a relatively newer dataset characterized by clear and meticulously articulated requirements compared to its counterparts. This dataset appears to contain well-written and explicit requirements, facilitating easier comprehension for algorithms. Conversely, Dataset-O, dating back more than a decade, reflects a long history of modifications facilitated by multiple experts over the course of 14 years. This prolonged timeline of alterations has likely contributed to its inherent ambiguity, stemming from the diverse perspectives and modifications introduced by these various experts. Moreover, Dataset-I, spanning over six years, exhibits a notable level of complexity owing to its comprehensive mentions of numerous hardware systems within its requirements.

The findings highlighted the differences and complexities present in each data set. Components of the dataset that seemingly have a substantial impact on the quality and clarity of requirements include its age, the history of revisions made by different experts, and the complexity of the referenced systems that are in the forms of abbreviations in SRS. The inclusion of contextual information enhances comprehension of the dataset's complexities and establishes a direct correlation with the observed fluctuations in algorithmic performance among the datasets.

4.11. Summary

In this chapter, an approach was developed to analyze the impacts of software requirements, focusing on predicting which of the existing software requirements may be affected by a requirement change. The core premise of the suggested approach is to learn from the history of change requests automatically, predict the change impact, and demonstrate how these predictions could help analysts enhance their decision-making to apply or reject an incoming change. The principal motivation was to achieve an automated solution to promote improved preparation and prioritization of the execution of requirements specifications during the software development of conventional and agile methodologies.

The proposed research was evaluated using five ML algorithms, including Random Forest, Decision Tree, Naive Bayes, Logistic Regression, and Support Vector Machines. The goal was to assess the effectiveness of these approaches in accurately analyzing written requirements.

Upon analyzing the evaluation results, the ML algorithms exhibited observable suboptimal performance, potentially due to the specific needs of highly accurate requirement comprehension. Precision and recall were employed as evaluation metrics commonly used in text classification and information retrieval tasks. Each algorithm showed unique strengths and faced challenges while addressing the complexities of precision and recall. Although the proposed models demonstrated proficiency in predicting the given change requests with high recall, precision differed between algorithms and certain scenarios. It was observed that the precision challenge exists in

making precise positive predictions despite capturing relevant instances, which is not ideal. Furthermore, all models struggled to achieve a balance between recall and precision, highlighting the trade-off. Despite precision issues, recall-strong models were crucial to guaranteeing thorough coverage of possible impact changes.

Although ML has demonstrated impressive capabilities across a range of domains, leading to significant attention in various fields, including software engineering, it is essential to recognise that its applicability may not be universal for all tasks. Anomalies were observed in the research on requirement CIA that highlights the shortcomings of ML algorithms when it comes to efficiently managing such complex tasks. The complexities inherent in language and the specificity of requirement comprehension present challenges that ML algorithms may struggle to address effectively.

In summary, while ML technologies continue to evolve and have significant upsides, caution must be taken when applying them due to their inherent limitations in handling complex comprehension and domain-specific contextual understanding tasks.

Chapter 5.

Enhancing Decision-Making in Software Development:

A Dual-Model Framework for Requirements Change Impact Analysis

5.1. Introduction

This chapter introduces a dual-model framework designed to address the complexities of requirements CIA by exploring two complementary approaches. The first approach leverages NLP techniques, utilizing CoreNLP and SpaCy libraries to analyze textual requirements and identify interdependencies. The second approach incorporates the Beir benchmark, combining Lexical Retrieval with BM25 (via Elasticsearch), Dense Retrieval using Bi-Encoders, and re-ranking with Cross-Encoders. These methodologies provide distinct yet synergistic perspectives, offering a comprehensive understanding of the challenges in CIA.

To ensure practical applicability, the framework is validated using industrial datasets, enabling insights into real-world scenarios and enhancing decision-making for requirements change requests. By comparing the performance and outcomes of these dual approaches, this chapter advances our understanding of how emerging technologies in NLP and retrieval-based models can effectively address the nuances of software requirements engineering.

The integration of the Beir benchmark represents a significant step forward in this research, introducing diverse information retrieval tasks to enrich the analysis. This approach complements the NLP-based solution, creating a robust framework capable of handling both semantic and syntactic complexities in requirements change scenarios. Through this exploration, we aim to uncover actionable insights and provide project managers with a sophisticated tool for predicting and managing the impacts of requirements changes in software development.

Despite significant progress in requirements traceability and change impact analysis, most prior approaches suffer from three key limitations. First, rule-based and keyword-matching techniques (e.g. traceability matrices or regex-driven pipelines) are brittle, failing to generalize when stakeholders use varied terminology or complex sentence structures. Second, single-model solutions, whether purely ML-based or purely IR-based, tend to excel on specific datasets but degrade sharply when confronted with domain shifts or aging documentation, because they capture only lexical overlap (as in BM25) or only statistical patterns (as in classical classifiers) without deeper semantic context. Finally, hybrid proposals are scarce, and where they exist, they lack clear orchestration strategies to bring together lightweight ML, full-blown NLP parsing, and retrieval-augmented generation into a unified pipeline. These gaps leave analysts either drowning in false-positive alerts or missing subtle but critical dependencies. By contrast, our dual-model framework explicitly combines syntactic/semantic NLP insights with retrieval-based evidence and stitches them together via an ensemble and feedback loop, to address brittleness, domain variability, and explainability all at once (Hayes, Dekhtyar & Sundaram 2006; SHAKIRAT et al. 2021; Thakur et al. 2021).

5.2. Dual Model Framework

The structure for this framework is depicted through a structured diagram detailing the sequential phases employed to enhance decision-making in software development by automating CIA. This process begins with comprehensive data collection, focusing on project specification documents and change logs, which form the foundational dataset.

The collected data undergoes a thorough preprocessing stage that includes tokenization and embedding generation, standardizing the input for subsequent analysis.

The methodology is bifurcated into two primary branches: an NLP-Based Solution and a Beir-Based Solution. The NLP-based approach utilizes CoreNLP for syntactic and semantic analysis, while SpaCy is employed for entity recognition tasks. In contrast, the Beir-based approach leverages BM25 (via Elasticsearch) for lexical retrieval, Bi-Encoders for dense retrieval, and Cross-Encoders for re-ranking retrieved results. Both branches operate independently and are evaluated using various performance metrics.

The dual outputs generated from these branches are then subjected to ensemble learning, enhancing the overall robustness and accuracy of the prediction model. The final stage involves rigorous evaluation and validation, applying metrics such as precision, recall, F1 score, and computational efficiency to assess the performance of the dual-model framework. This methodology not only demonstrates the integration of advanced NLP and Beir technologies but also illustrates a systematic approach to automating CIA in large-scale software development projects.

5.3. NLP-Based Solution (CoreNLP and SpaCy Integration)

The NLP-based solution integrates two prominent NLP libraries: CoreNLP and SpaCy.

In this framework, CoreNLP is primarily employed for syntactic and semantic analysis, while SpaCy focuses on entity recognition and linguistic feature extraction. The integration of both libraries allows for a comprehensive analysis of software requirements documents. CoreNLP's robust parsing features handle intricate sentence structures, while SpaCy offers rapid and efficient preprocessing for extracting key language elements. Together, these tools enhance the model’s capacity to interpret and analyze the natural language used in requirements documents effectively. Figure 5.1 illustrates the dual-model framework structure:

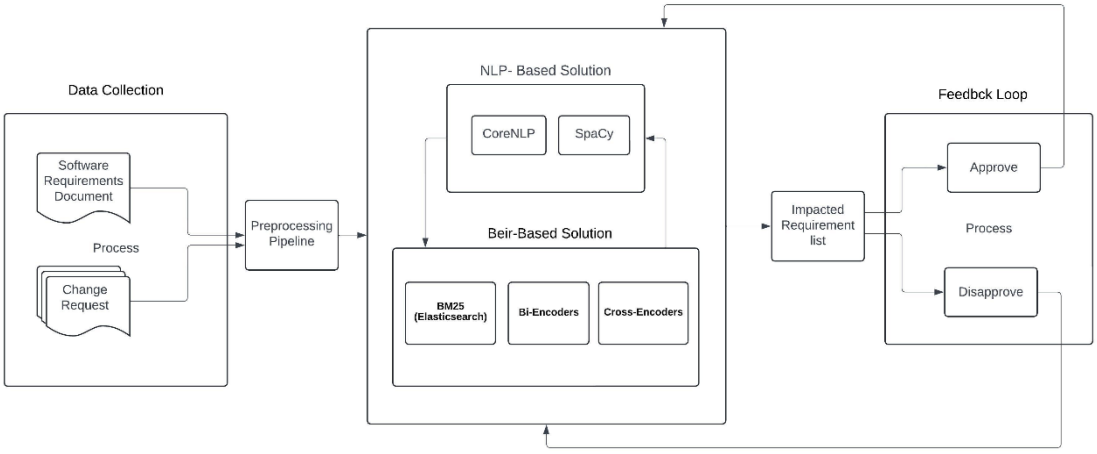

Figure 5.1.Dual-Model Framework

The combined approach was selected based on the complementary strengths of both libraries. CoreNLP's detailed grammatical parsing is critical for handling the complexity of software requirements, while SpaCy's efficiency in entity recognition and processing speed makes it a powerful tool for high-volume data analysis. The integration of these two libraries results in a more granular and accurate analysis, improving the model's overall performance.

Additionally, this flexible architecture enables fine-tuning and customization to accommodate specific project needs or changes in scope as the dataset evolves. By incorporating similarity metrics such as cosine similarity and TF-IDF vectors, the model is better equipped to detect semantic relationships between texts rather than relying solely on keyword overlap. This enhances the precision of the CIA and provides a more nuanced understanding of the connections between software requirements.

The integration of these techniques achieves an optimal balance between computational efficiency and depth of semantic analysis. The ability to customize the vectorization process and similarity criteria ensures that the solution can be tailored to the unique characteristics of each dataset, ultimately improving the responsiveness and accuracy of the CIA model.

CoreNLP and SpaCy were chosen not just for their robustness but for their ability to resolve key CIA pain points. Dependency parsing uncovers hidden syntactic ties by

walking the parse tree, we detect when a change in “user authentication” will cascade to “session management” through shared grammatical relations, even if no keywords overlap. Named-entity recognition then isolates domain concepts, such as “API endpoint” or “payment gateway”, so that any modification to these entities immediately flags every related requirement, capturing dependencies that simple term-frequency methods miss. Finally, CoreNLP’s coreference resolution maintains contextual cohesion by mapping pronouns or aliases back to their antecedents; thus, an update to “the payment module” also highlights every later occurrence of “it,” ensuring no implicit reference is overlooked.

5.4. Beir-Based Solution

The Beir-based solution leverages the BEIR benchmark, a heterogeneous benchmark designed for information retrieval (IR) tasks. This method incorporates multiple stages of retrieval and ranking to improve the relevance and precision of change impact predictions.

- Lexical Retrieval with BM25 (Elasticsearch)

The first step in the Beir-based solution is lexical retrieval using BM25, implemented via Elasticsearch. BM25 is a well-established ranking function used in search engines that measures the relevance of documents based on keyword matching. This provides a solid foundation for the retrieval process but may have limitations in capturing semantic meaning, as it is primarily based on lexical overlap and may miss synonyms or related terms.

- Dense Retrieval Using Bi-Encoders

To address the limitations of lexical retrieval, the framework employs Bi-Encoders for dense retrieval. Bi-Encoders encode both queries and documents into dense vector spaces using models such as BERT. This allows the retrieval process to capture deeper semantic similarities between the query and the documents. By utilizing dense retrieval, the system can retrieve contextually relevant information that goes beyond mere keyword matching, enhancing the overall retrieval performance.

- Re-ranking with Cross-Encoder

After dense retrieval, a Cross-Encoder model is applied to re-rank the retrieved documents. The Cross-Encoder assesses the interaction between document-query pairs and assigns relevance scores based on a more holistic understanding of the context. By re-ranking the results, the model significantly improves the precision and relevance of the final predictions. The combination of dense retrieval and re-ranking ensures that the most contextually relevant documents are prioritized for impact analysis.

While BM25 provides a solid starting point by ranking on term overlap, it struggles with synonyms, paraphrases, and complex phrasing common in requirements. Our two-stage neural retrieval addresses this directly. First, Bi-Encoders map queries and documents into the same dense vector space using a fine-tuned BERT variant, so semantically similar texts, like “login process” and “user sign-in flow”, naturally cluster together, boosting recall even when no keywords match. Next, Cross-Encoders re-rank these candidates by jointly encoding each query–document pair with full attention, filtering out loose semantic matches and elevating those with deep contextual alignment (Nogueira & Cho 2019; Reimers & Gurevych 2019).

5.5. Data

The same real-world datasets utilized in Chapter 4 are employed in this chapter to ensure consistency and comparability when evaluating the proposed solutions. These datasets consist of project specification documents and change logs from industrial software development projects. The data covers a wide range of software requirements and their corresponding change requests, providing a robust foundation for analyzing the impact of requirement modifications.

The project specification documents offer detailed descriptions of the software requirements, while the change logs record the historical changes made throughout the software development lifecycle. Together, these datasets provide a comprehensive view of the evolving nature of software requirements and serve as the basis for applying both the NLP-based and Beir-based methods within the dual-model framework.

The datasets were thoroughly preprocessed to ensure quality and consistency, following the steps outlined in Chapter 3. This includes tokenization, embedding generation, and verification for completeness and accuracy. By reusing these industrial datasets, the study maintains a direct comparison between the two chapters, highlighting the enhancements achieved through the dual-model framework presented in this chapter.

5.6. Implementation

The implementation of the NLP-based solution and the dual-model framework was carried out using Python as the primary programming language. The deep learning components were developed utilizing libraries such as TensorFlow and PyTorch, while Elasticsearch was employed as the backend for information retrieval tasks.

5.6.1. NLP Solution

The NLP solution was implemented through a Python script that leverages the CoreNLP and SpaCy libraries to process and analyze the data. The primary steps of this implementation are outlined as follows:

Step 1: Preprocessing

Both datasets—comprising requirements and change scenarios—were loaded and preprocessed to ensure that the textual data was clean and ready for subsequent analysis. This step involved tokenization, lowercasing, removing stopwords, and other standard text preprocessing techniques to ensure consistency across the datasets.

Step 2: NLP Feature Extraction

The CoreNLP and SpaCy libraries were employed to extract valuable features for predicting the impact of each requirement and change request. The key NLP tasks include:

- **Named Entity Recognition (NER) for Terminology Alignment:** To identify critical entities such as stakeholders, system components, or specific actions. NER

extracts domain entities like changeID and normalizes them. When a change request mentions an entity that subsequently appears in design-artifact descriptions, our pipeline flags all requirements containing that entity as candidates for impact analysis

- **Dependency Parsing for Syntactic Dependency Mapping: To** analyze sentence structure and understand grammatical relationships between words, aiding in the interpretation of complex requirement descriptions. By constructing a dependency tree for each requirement sentence, we capture head–modifier relations (e.g. subject→verb, verb→object). When two requirements share a modifier or refer to the same head term via different phrasings ("user login" vs. "login by user"), their dependency graphs overlap. We can therefore algorithmically detect these overlaps as potential impact links, even when no exact term match exists.
- **Part-of-Speech (POS) Tagging:** To classify words based on their roles in a sentence (e.g., noun, verb), which helps in understanding the context of the requirements.

CoreNLP was utilized for its robust parsing capabilities, including coreference resolution, while SpaCy was chosen for its efficiency in entity recognition and ease of vectorization through pre-trained models.

Step 3: Vectorization

The textual data was converted into numerical vectors to facilitate similarity comparisons. TF-IDF (Term Frequency-Inverse Document Frequency) vectorization was implemented, transforming the text into a format suitable for ML algorithms.

Step 4: Similarity Calculation

Similarity scores between change requests and requirements were calculated based on their vectorized representations. Cosine similarity was employed to measure the semantic distance between pairs of vectors. Higher similarity scores suggest a higher likelihood of impact between a change request and a requirement.

Step 5: Predicting Impact

Based on the calculated similarity scores, predictions were made to determine which requirements are likely impacted by each change request. Initially, a rule-based approach was adopted, utilizing thresholds on similarity scores to identify potential impacts. Further refinement of this step involved integrating additional NLP features to improve the precision of the predictions.

5.6.2. Beir-Based Solution

The Beir-based solution was implemented using Python, with the full code and replication package available on Zenodo. The following steps outline the implementation process:

Step 1: Preprocessing Datasets:

The datasets were preprocessed to ensure consistency in text format and structure. This included tokenization, normalization, and the removal of irrelevant elements such as stopwords. The preprocessing phase is essential to ensure that the data is in an appropriate format for information retrieval tasks.

Step 2: Installation of Beir Framework:

The Beir (Benchmarking Information Retrieval) framework was installed to facilitate the evaluation of different information retrieval models. Beir provides a comprehensive suite of tools for benchmarking information retrieval tasks and supports various retrieval methods such as lexical retrieval and dense retrieval.

Step 3: Loading Models:

Three models were loaded for use in the Beir-based solution:

- **BM25 (Elasticsearch):** A lexical retrieval model that ranks documents based on keyword matches, often serving as a baseline in information retrieval tasks.
- **Bi-Encoders:** A dense retrieval model that leverages pre-trained BERT encoders to capture semantic similarities between queries and documents, offering enhanced retrieval performance compared to traditional lexical methods.

- **Re-ranked Cross-Encoder:** This model refines the initial rankings produced by BM25 and Bi-Encoders by considering the interaction between document pairs and assigning relevance scores. The Cross-Encoder is used to improve precision in the final ranking.

Step 4: Application and Evaluation on Datasets:

The models were applied to the datasets to evaluate their performance. Metrics such as precision, recall, and F1 scores were computed to assess the effectiveness of the Beir-based solution in identifying impacted requirements for the given change requests.

Step 5: Searching for New Changes:

The trained models were used to search for newly introduced changes. This involved retrieving relevant requirements from the dataset and ranking them based on their similarity to given changes.

Step 6: Generating Similarity Scores:

For each change request, similarity scores between the change and the relevant requirements were generated. These scores helped prioritize impacted requirements, providing valuable insights for decision-making in software development.

5.7. Application of Mathematical Heuristics

In this research, mathematical heuristics are implemented within the NLP-based and BEIR-based frameworks to optimize the selection process for sentences most relevant to a given change. These frameworks compute similarity scores between requirement sentences and changes using a combination of linguistic and contextual features. Without heuristics, processing the entirety of the resulting similarity scores would lead to significant computational overhead. The integration of heuristics addresses this challenge by filtering and prioritizing the highest-scoring similarities to maintain computational efficiency while preserving accuracy.

The application of heuristics involves several strategies designed to optimize the process of narrowing down the similarity results. One key technique is score

thresholding, which eliminates sentences with similarity scores below a predefined baseline. This baseline is determined by analyzing the distribution of scores in the dataset, ensuring that only sentences with meaningful similarity are considered for further analysis. For instance, if the score distribution indicates a natural inflection point at 0.3, sentences scoring below this value are excluded, reducing computational costs without sacrificing relevance.

Another essential approach is significant drop detection, which identifies points where similarity scores decline sharply among ranked results. A sharp drop in similarity scores often marks the boundary between semantically relevant and irrelevant sentences. For example, if the ranked scores drop from 0.85 to 0.40 within a few positions, the point of decline is used as a cutoff. This heuristic dynamically adapts to variations in score distributions across different datasets, ensuring flexibility and efficiency in the selection process.

Relative score proportionality is also applied to retain sentences with similarity scores that are at least a certain percentage of the highest score in the dataset. For instance, if the highest score in a dataset is 0.9, sentences scoring below 0.45 are excluded, ensuring that only sentences with strong semantic alignment are retained. This heuristic further refines the results by emphasizing the most relevant sentences based on their proximity to the highest-ranked similarity score.

By leveraging these heuristics, the study effectively narrows down the selection process, allowing for the identification of sentences with high semantic relevance to a given change. This approach enables dynamic adjustment of similarity evaluations based on real-time observations of the score distribution. The integration of heuristics facilitates computational efficiency, making it feasible to analyze large datasets while maintaining the precision needed for robust CIA. This application demonstrates the practical value of heuristics in improving the efficiency and accuracy of NLP and BEIR-based models within the SRCIA framework. Furthermore, it highlights the role of heuristics as a critical component in balancing computational constraints with analytical rigor.

5.8. Dual Model Evaluation Metrics

Evaluating information retrieval models in software engineering, particularly in the context of requirements CIA, necessitates metrics that assess the correctness, relevance, and value of the retrieval results for users. In this study, a range of metrics is applied to evaluate the performance of the dual-model framework, which incorporates the BEIR Re-ranking-based model. These metrics offer diverse perspectives on the model's effectiveness, providing a comprehensive evaluation.

5.9. Information Retrieval Models

To ensure a robust evaluation, the following metrics were selected to assess the retrieval model's performance:

a) Precision and recall at k

Precision at k (P@k) and Recall at k (R@k) were employed to evaluate the immediate value of the search results. Precision measures the relevance of the retrieved requirements impacted by a change, indicating the proportion of relevant results within the top k results. In contrast, recall assesses the model's ability to retrieve all impacted requirements, ensuring completeness in the retrieval process.

In the context of software development, these metrics are crucial. Failing to retrieve an impacted requirement can lead to significant project delays or errors, while retrieving too many irrelevant requirements increases the manual effort required for further inspection. Thus, achieving a balance between precision and recall is essential for efficient resource allocation and decision-making.

The choice of **k** reflects a balance between user expectations and the practicality of reviewing retrieved requirements. Based on common stakeholder behaviour in software projects and decision-makers' willingness to engage with a ranked list of results, values of **k=5** and **k=10** were chosen. These values align with standard practices in the information retrieval domain and ensure that the model delivers relevant results within a manageable scope.

b) Normalized Discounted Cumulative Gain (NDCG@k) and Mean Average Precision (MAP@k)

To evaluate the quality of the ranking within the retrieval system, Normalized Discounted Cumulative Gain (NDCG@k) and Mean Average Precision (MAP@k) were selected as additional metrics.

MAP@k provides an average precision score independent of rank position by calculating precision across various threshold levels, offering a comprehensive view of how well the model performs over different ranks. NDCG@k further enhances this evaluation by considering the importance of the ranking order, ensuring that the most critical impacted requirements appear higher in the list. This is particularly significant in software development, where prioritizing high-impact requirements can lead to more efficient decision-making processes.

Both metrics are essential when prioritizing impacted requirements in order to optimize the accuracy and efficiency of the impact analysis. By weighting higher-ranked results more heavily, NDCG@k ensures that the retrieval system favors more relevant and critical requirements, reducing the time and effort required to review irrelevant or less significant results.

Additionally, it is recognized that the binary relevance typically assumed in calculating these metrics may not fully capture the varying degrees of impact that a change may have on different requirements. To address this complexity, future work will explore graded relevance evaluations, which can offer a more nuanced understanding of the impact of severity.

5.10. NLP Model Evaluation

To thoroughly evaluate the performance of the NLP model, two metrics—MUC (Message Understanding Conference) and B^3 (B-Cubed)—were utilized. These metrics are specifically designed for assessing coreference resolution tasks, providing an in-depth analysis of how effectively the model identifies and clusters references to the same entities across different texts. Given the nature of this research, they were

adapted to assess the correctness and completeness of the predicted impact links between change requests and software requirements.

a) MUC (Message Understanding Conference)

MUC evaluates how well the predicted set of impacted requirements corresponds to the actual set, focusing on the correctness and completeness of the links between change requests and requirements. In this context, a "link" is defined as the correctly identified relationship between a change request and a requirement. MUC primarily focuses on whether the model can capture the entire set of impacted requirements and whether those predicted links are accurate, giving insight into the overall completeness of the prediction.

b) B^3 (B-Cubed)

The B^3 metric was used to compute precision, recall, and F1 scores for each requirement's impact prediction. This evaluation method considers the presence or absence of a requirement in both the predicted and actual impacted sets. B^3 calculates precision as the ratio of correctly predicted impacts to the total number of predicted impacts and recall as the ratio of correctly predicted impacts to the total number of actual impacts. The F1 score is the harmonic means of precision and recall, providing a balanced evaluation of the model's performance.

To assess the overall effectiveness of the model in predicting impacted requirements, the real and predicted impacted sets were extracted for each change scenario. The aggregate B^3 metrics—Precision, Recall, and F1 Score—were then computed across all change scenarios, offering a comprehensive view of the model's predictive accuracy and completeness.

5.11. Results and Findings

This section outlines the performance of NLP-based and BEIR-based solutions within the dual-model framework, highlighting their effectiveness and key findings. The following subsections provide detailed results for each approach.

5.11.1. NLP Solution Results

Table 5.1 presents a summary of the overall precision, recall, and F1 score for each dataset, offering a comprehensive evaluation of the model's performance. Table 5.2 details the precision, recall, and F1 scores associated with each change ID within the datasets, illustrating the specific impact of individual modifications. The findings highlight the varying degrees of success achieved by the combined NLP approach in identifying impacted requirements.

- **Dataset-I** demonstrated promising results, with a precision of 0.5345, recall of 0.8389, and an F1 score of 0.6530. The high recall value indicates that the method effectively identified a wide range of relevant impacted requirements, though the lower precision suggests the inclusion of more false positives. The F1 score reflects a reasonable balance between precision and recall, making the method practical for this dataset.
- **Dataset-W** yielded more modest results, with a precision of 0.4810, a recall of 0.6236, and an F1 score of 0.5431. Compared to Dataset-I, both precision and recall showed a decline, indicating challenges in accurately identifying impacted requirements. This suggests that Dataset-W presents unique complexities or domain-specific characteristics that pose difficulties for the combined NLP approach.
- **Dataset-O** exposed the limitations of the method, registering the lowest precision (0.2781), recall (0.4922), and F1 score (0.3554) among the datasets. These results underscore significant challenges in applying the approach to Dataset-O, potentially due to intrinsic features of the dataset that impede effective NLP analysis. The lower precision and F1 scores highlight the model's difficulty in maintaining accuracy when applied to this dataset.

Overall, the results reflect that while the combined NLP method performs adequately across certain datasets, its efficacy can vary depending on the characteristics of the dataset. High recall scores demonstrate the model's strength in identifying a broad range of impacted requirements, but lower precision scores suggest the need for further refinement to reduce false positives and improve overall accuracy.

Table 5.1. The overall evaluation metrics for NLP solution

Dataset	Overall Precision	Overall Recall	Overall F1 Score
I	0.5345	0.8389	0.653
W	0.481	0.6236	0.5431
O	0.2781	0.4922	0.3554

The chart below, figure 5.2, shows the overall Precision, Recall, and F1-Score of the NLP-based model on Datasets I, W, and O, highlighting that Dataset I achieves the highest coverage (recall) and balanced performance (F1-Score) despite lower precision, while Dataset O shows the greatest drop in all three metrics.

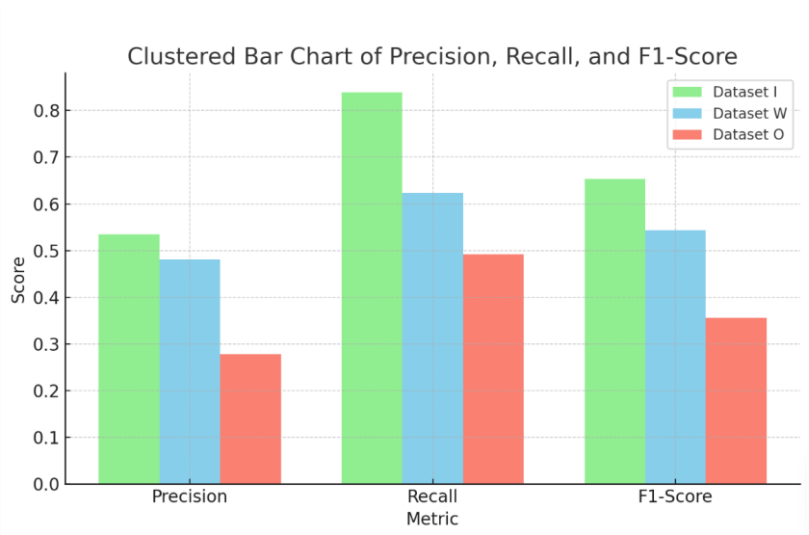

Figure 5.2.Clustered Bar Chart of Precision, Recall, and F1-Score of the NLP-based model

Table 5.2. NLP solution results

Dataset	Change ID	Precision	Recall	F1 Score
I	Case 1	0.14	0.25	0.18
	Case 2	0.29	1.00	0.44
	Case 3	0.22	1.00	0.36
	Case 4	0.33	0.33	0.33
	Case 5	1.00	1.00	1.00
	Case 6	1.00	1.00	1.00
	Case 7	0.50	1.00	0.67
	Case 8	0.50	0.75	0.60
	Case 9	0.67	1.00	0.80
	Case 10	0.33	0.50	0.40
	Case 11	0.33	0.75	0.46
	Case 12	0.20	1.00	0.33
	Case 13	0.50	1.00	0.67
	Case 14	1.00	1.00	1.00
	Case 15	1.00	1.00	1.00
W	C1	1.00	0.50	0.67
	C2	0.60	0.33	0.43
	C3	0.33	1.00	0.50
	C4	1.00	1.00	1.00
	C5	0.67	0.75	0.71
	C6	0.57	0.57	0.57
	C7	0.20	0.25	0.22
	C8	0.50	1.00	0.67
	C9	0.70	0.78	0.74
	C10	0.33	0.67	0.44
	C11	0.50	0.25	0.33
	C12	0.20	0.25	0.22
	C13	0.50	0.83	0.63
	C14	0.00	0.00	0.00
	C15	0.10	0.33	0.15
	C16	0.40	0.67	0.50
	C17	0.50	1.00	0.67

	C18	0.00	0.00	0.00
	C19	0.20	0.40	0.27
	C20	1.00	0.40	0.57
	C21	0.40	1.00	0.57
	C22	0.60	0.75	0.67
	C23	0.67	1.00	0.80
	C24	0.30	0.43	0.35
	C25	1.00	1.00	1.00
	C26	0.67	0.80	0.73
	C27	0.43	1.00	0.60
	C28	0.10	0.50	0.17
O	CR007	0.30	0.43	0.35
	CR503689	0.30	0.75	0.43
	CR503779	0.20	0.50	0.29
	CR504139	0.29	0.29	0.29
	CR504310	0.30	0.50	0.38
	CR504311	0.00	0.00	0.00
	CR504321	0.20	0.67	0.31
	CR504322	0.10	0.25	0.14
	CR504323	0.20	0.67	0.31
	CR504324	0.20	1.00	0.33
	CR504325	0.29	0.50	0.36
	CR504326	0.20	0.50	0.29
	CR504327	0.30	1.00	0.46
	CR504328	0.40	0.67	0.50
	CR504329	0.67	0.67	0.67
	CR504330	0.00	0.00	0.00
	CR504331	0.40	0.57	0.47
	CR504332	1.00	0.50	0.67
	CR504333	0.00	0.00	0.00
	CR504334	0.25	1.00	0.40
	CR504335	0.40	0.67	0.50
	CR504336	0.30	0.75	0.43
	CR504337	0.40	0.80	0.53
	CR504338	0.10	0.50	0.17

	CR504339	0.00	0.00	0.00
	CR504340	0.20	0.33	0.25
	CR504341	0.00	0.00	0.00
	CR504342	0.20	0.67	0.31
	CR504793	0.30	0.30	0.30
	CR504799	0.30	0.60	0.40
	CR600203	0.33	0.33	0.33
	CR600204	0.33	0.33	0.33
	CR60200	0.00	0.00	0.00
	CR60202	1.00	1.00	1.00

5.11.2. Beir-Based Results:

Table 5.3 provides a detailed summary of the evaluation metrics, including Average Precision, Average Recall, Average NDCG, and Average MAP, evaluated at cutoff values of 5 and 10. These metrics offer a comprehensive view of the Beir-based model's performance across the datasets, highlighting its effectiveness in terms of precision, recall, and ranking quality.

Additionally, Table 5.4 illustrates the variation in specific metrics, map_cut_5 and ndcg_cut_5, across different datasets. These metrics are particularly useful in evaluating the ranking efficiency and precision of the retrieval model within a given threshold. The map_cut_5 metric measures the mean average precision of the top 5 results, while ndcg_cut_5 focuses on the quality of the ranking within the top 5 most relevant results.

Together, these metrics provide a reliable indication of how well the Beir-based solution applies to real-world software development processes, particularly in terms of usefulness, recall, ranking efficacy, and precision. By analyzing these results, it becomes evident how the model performs in prioritizing and ranking impacted requirements based on their relevance to a given change request.

Table 5.3. Beir-based average results

Dataset	Average Precision@5	Average Recall@5	Average NDCG@5	Average MAP@5	Average Precision@10	Average Recall@10	Average NDCG@10	Average MAP@10
W	0.6643	0.8049	0.9075	0.7819	0.3643	0.8454	0.8775	0.8141
I	0.44	0.9667	0.9689	0.9556	0.22	0.9667	0.9689	0.9556
O	0.5706	0.7815	0.8486	0.7449	0.3235	0.8345	0.8443	0.7935

Figure 5.3 shows the BEIR-based average metrics at cutoff 5, Precision@5, Recall@5, NDCG@5, and MAP@5, for Datasets W, I, and O, illustrating that Dataset I delivers near-perfect recall and ranking quality, Dataset W offers the best precision-recall balance, and Dataset O falls in between.

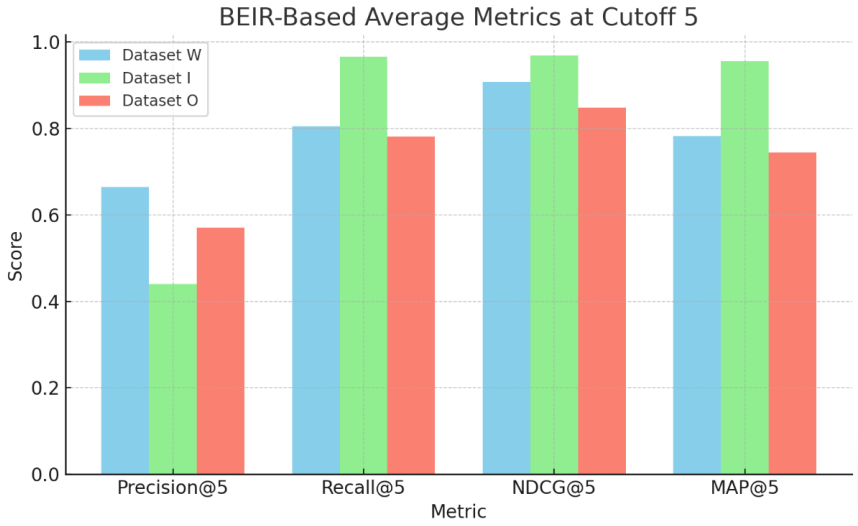

Figure 5.3.Clustered Bar Chart of Average Metrics of the BEIR-based model

Table 5.4. Beir-based results

Dataset	Change Request	map_cut_5 Scores	ndcg_cut_5 Scores
I	Case 1	1.00	1.00
	Case 2	1.00	1.00
	Case 3	1.00	1.00
	Case 4	1.00	1.00
	Case 5	1.00	1.00
	Case 6	1.00	1.00
	Case 7	1.00	1.00
	Case 8	1.00	1.00
	Case 9	0.50	0.61
	Case 10	1.00	1.00
	Case 11	1.00	1.00
	Case 12	0.83	0.92
	Case 13	1.00	1.00
	Case 14	1.00	1.00
	Case 15	1.00	1.00
W	C1	1.00	1.00
	C2	0.33	0.72
	C3	0.71	0.83
	C4	1.00	1.00
	C5	0.56	1.00
	C6	1.00	1.00
	C7	1.00	1.00
	C8	1.00	1.00
	C9	0.44	0.87
	C10	0.67	0.77
	C11	1.00	1.00
	C12	0.34	0.70
	C13	0.83	1.00
	C14	0.50	0.61
	C15	1.00	1.00
	C16	0.83	1.00
	C17	0.80	0.87

	C18	1.00	1.00
	C19	0.40	0.55
	C20	1.00	1.00
	C21	0.95	0.98
	C22	0.63	1.00
	C23	0.75	0.88
	C24	0.54	0.85
	C25	1.00	1.00
	C26	0.80	0.87
	C27	0.81	0.91
	C28	1.00	1.00
O	CR007	0.71	1.00
	CR503689	1.00	1.00
	CR503779	0.25	0.41
	CR504139	0.71	1.00
	CR504310	0.83	1.00
	CR504311	0.35	0.65
	CR504321	0.67	0.77
	CR504322	0.75	0.83
	CR504323	1.00	1.00
	CR504324	1.00	1.00
	CR504325	0.68	0.76
	CR504326	1.00	1.00
	CR504327	1.00	1.00
	CR504328	0.83	1.00
	CR504329	1.00	1.00
	CR504330	1.00	1.00
	CR504331	0.71	1.00
	CR504332	1.00	1.00
	CR504333	0.50	0.61
	CR504334	1.00	1.00
	CR504335	0.67	0.77
	CR504336	0.75	0.83
	CR504337	0.80	0.87
	CR504338	1.00	1.00

	CR504339	1.00	1.00
	CR504340	0.67	0.87
	CR504341	1.00	1.00
	CR504342	0.70	0.85
	CR504793	0.26	0.68
	CR504799	0.40	0.55
	CR600203	0.00	0.00
	CR600204	0.83	1.00
	CR60200	0.25	0.39
	CR60202	1.00	1.00

5.12. Discussion

This section analyses the results of the proposed solutions, with subsections focusing on the NLP-based and Beir-based approaches.

Different project contexts demand different balances between catching every possible impacted requirement (high recall) and minimizing false alarms (high precision). For safety-critical or regulatory systems, missing even a single dependency could have severe consequences; in such cases, configuring the dual-model pipeline for higher recall, even at the expense of more manual review, is justified. Conversely, for fast-moving agile teams where throughput is paramount, prioritizing precision reduces analyst overhead, accepting that a few subtle impacts may be caught later in the iteration cycle. By exposing confidence thresholds in both the NLP and BEIR stages, our framework allows teams to tune this balance according to risk tolerance and available review effort.

5.12.1. Discussion on the NLP Solution Results

A detailed review of the outcomes for each specific change request has yielded significant insights. Certain change requests, particularly those related to core functionalities or critical components of the system, consistently demonstrated improved precision and recall. This suggests that the NLP method is particularly effective

at identifying and assessing the impact of changes that are well-defined or central to the system's operations.

In contrast, other change requests produced lower scores, likely due to several factors, including ambiguous requirement descriptions, insufficient context in the change requests, or the inherent limitations of NLP technologies in handling complex semantic relationships without additional contextual information. These challenges emphasize the importance of clarity in the description of requirements and change requests to facilitate more accurate impact analysis.

The overall results from the datasets present a mixed but informative picture. Dataset-I demonstrated the highest success, achieving strong precision and recall scores, which may be attributed to the specific characteristics of the dataset—such as its domain focus and the well-defined nature of its requirements. This alignment with the NLP solution's strengths indicates that the model performs particularly well in environments with structured, domain-specific data.

In contrast, Dataset-O encountered significant challenges, reflected in its lower performance across all metrics. This disparity highlights the sensitivity of NLP techniques to the linguistic and structural characteristics of the data being processed. It suggests that the effectiveness of the NLP solution is highly dependent on the quality and specificity of the dataset. Consequently, this points to the need for tailored NLP strategies that are adapted to different domains or types of software documentation to enhance the model's performance.

5.12.2. Beir-Based Results Discussion

This subsection provides details of the results of the Beir-based approach for each dataset.

(a) Discussion on Dataset-W:

The evaluation of the dual-model framework for requirements CIA on Dataset-W yielded insightful results, demonstrating how well the model can extract and rank relevant requirements. The implementation achieved a notable balance between recall,

precision, and ranking efficiency, with the Bi-Encoder re-ranking mechanism contributing significantly to this outcome.

A Precision@5 score of 0.6643 indicates that, on average, 66.43% of the top 5 returned documents were relevant to the given change requests (C1 to C28). This level of accuracy in the top results highlights the model's ability to prioritize the most relevant requirements, addressing the needs of stakeholders who typically focus on the top-ranked results.

Despite this strong performance, the Recall@5 score of 0.8049 suggests that the model could further improve its ability to retrieve a broader set of relevant requirements in the top 5. This balance between capturing the most critical requirements and excluding irrelevant ones presents a challenge in maintaining precision while improving recall.

When expanding the evaluation to the top 10 requirements, Precision@10 dropped to 0.3643, indicating the trade-off between increasing the result set size and reducing precision. Nevertheless, the Recall@10 score rose to 0.8454, reflecting broader coverage and the inclusion of more relevant requirements, albeit at the cost of introducing some irrelevant ones.

The NDCG scores of 0.9075 (for 5 results) and 0.8775 (for 10 results) highlight the model's ability to rank relevant requirements effectively. The slight decline from 5 to 10 indicates the challenge of maintaining ranking quality as the result set grows.

Moreover, the MAP scores showed a positive trend, improving from 0.7819 (at 5 results) to 0.8141 (at 10 results), suggesting that the model preserves ranking precision across a broader range of requirements. This indicates the model's utility when users are prepared to explore a more extensive set of results.

(b) Discussion on Dataset-I:

The evaluation of the dual-model framework on Dataset-I yielded distinct results, shedding light on the dataset's characteristics and areas where the model both excels and faces challenges. Unlike Dataset-W, Dataset-I produced a high recall at the expense of lower precision.

With a Precision@5 score of 0.4400, the model was less precise in retrieving top results. However, the exceptionally high Recall@5 score of 0.9667 demonstrates that the model was highly successful in retrieving nearly all relevant requirements within the top 5, albeit with the inclusion of non-relevant documents that lowered precision.

Similarly, Precision@10 and Recall@10 scores followed this trend, with values of 0.2200 and 0.9667, respectively, showing that the model maintained its ability to capture nearly all relevant requirements, though more non-relevant results were included as the result set size increased.

The NDCG@5 and MAP@5 scores—0.9689 and 0.9556, respectively—demonstrate the model's strong ranking performance and precision at the top of the result set. Even as the result set size doubled, the model maintained these scores at NDCG@10 and MAP@10, reflecting its ability to rank relevant requirements effectively across different result set sizes.

These results confirm the model's ability to retrieve relevant requirements with near-perfect recall, a valuable strength in software development environments where missing critical requirements can be costly. However, the lower precision suggests the need to refine the retrieval process to reduce irrelevant data while maintaining recall.

(c) Discussion on Dataset-O

For Dataset-O, the Precision@5 score of 0.5706 and the Recall@5 score of 0.7815 indicate a strong initial performance in retrieving relevant requirements within the top 5 results. This balance suggests that a significant portion of relevant requirements is correctly prioritized, making it suitable for users focused on the top results.

As more results are considered, there is a trade-off between relevance and quantity, reflected by the drop in Precision@10 to 0.3235 and the increase in Recall@10 to 0.8345. This trade-off illustrates the challenge of expanding result sets while maintaining precision.

The NDCG@5 score of 0.8486 and the NDCG@10 score of 0.8443 highlight the model's effectiveness in accurately ranking the retrieved requirements, with only a slight decline

in ranking quality as the number of results increases. This consistency suggests that the model's ranking mechanism is robust even with a more extensive result set.

The MAP scores also showed an improvement from 0.7449 (at 5 results) to 0.7935 (at 10 results), indicating that the model effectively maintains precision even as more requirements are reviewed.

The performance on Dataset-O highlights the model's ability to capture relevant requirements while ranking them effectively. However, the decline in precision as the number of retrieved requirements grows underscores the need for further refinement, particularly in improving the re-ranking process to balance precision and recall.

Conclusion- The evaluation of Dataset-O provides valuable insights into the dual-model framework's strengths in requirements CIA. While the model excels in recall and ranking quality, future enhancements should focus on improving precision without sacrificing recall. This balance is essential for supporting informed decision-making in software development processes.

5.12.3. Comparison Between NLP-Based and Rule-Based CIA Approaches

In the context of software requirements Change Impact Analysis (CIA), NLP-based and rule-based approaches offer fundamentally different strengths and limitations. While both aim to identify the relationships between change requests and potentially impacted requirements, their underlying methodologies and adaptability diverge significantly.

NLP-based models rely on advanced natural language processing techniques to understand the semantics, syntax, and context of textual requirements. These models, particularly when enhanced with tools such as CoreNLP and SpaCy, can interpret varied sentence structures and terminologies. This flexibility allows them to perform effectively across diverse datasets and evolving documentation styles. By contrast, rule-based systems are grounded in fixed patterns, often defined through keyword matching, regular expressions, or pre-set dependency rules. As a result, they are inherently rigid.

Any deviation from predefined patterns, such as novel phrasing or unexpected terminology, typically requires manual rule adjustments, limiting their ability to generalize (Arora et al. 2015a).

Scalability is another critical factor distinguishing these two approaches. NLP-based methods scale well to large datasets, even those with heterogeneous language use, because their models can adaptively learn from data distributions. In contrast, rule-based methods tend to degrade in performance as the volume and variability of the dataset grow. This makes them more suitable for smaller or highly structured domains where language use is predictable and controlled (Arora et al. 2015a).

From a performance standpoint, rule-based systems may deliver high precision within narrowly defined contexts, since they trigger only when specific criteria are met. However, this narrow targeting often results in lower recall, as many impacted requirements fall outside the rigid rule definitions. NLP models, on the other hand, tend to strike a better balance between precision and recall. Their semantic capabilities allow them to identify relevant impacts even when textual expressions differ significantly, as demonstrated by higher recall values observed in the evaluations on Dataset-I and Dataset-W (Arora et al. 2015a; Goknil, Kurtev & Berg 2016).

Finally, maintainability sets these approaches further apart. Rule-based systems demand frequent manual updates to stay current with new requirement styles or domain shifts. This creates an ongoing maintenance burden for requirements engineers. NLP-based systems, however, can evolve through model retraining or fine-tuning without changing the underlying logic or architecture. This adaptability makes them more sustainable in dynamic development environments where requirements evolve over time (Goknil, Kurtev & Berg 2016).

In summary, while rule-based approaches remain valuable in constrained scenarios requiring high precision and interpretability, NLP-based CIA methods offer superior flexibility, scalability, and adaptability. Their capacity to handle unstructured language and generalise across varied datasets positions them as more robust solutions for modern software engineering projects involving large-scale or frequently changing requirements.

5.12.4. Dataset-Specific Challenges & Remedies

Dataset-O proved the most difficult of our three datasets. Having been edited by multiple authors over more than 14 years, it contains inconsistent terminology, uneven levels of detail, and sparse contextual cues. These characteristics lead to noisy embeddings, poor lexical overlap, and fractured dependency graphs, which explain the lower Precision, Recall, and F1 scores we observed.

To address these issues, first we recommend domain-adaptive fine-tuning of our neural retrievers on a small, manually validated subset of Dataset-O. By exposing the Bi-Encoder and Cross-Encoder models to the dataset's idiosyncratic vocabulary and phrasing, we can improve their ability to capture its unique semantics. Second, integrating a lightweight domain ontology mapping, like "subscriber endpoint" to "user API", can augment embeddings with explicit concept links and boost semantic coverage where raw vectors fall short. Third, an active-learning loop that flags low-confidence predictions for human review can help surface edge-case dependencies. These annotations both improve model retraining and focus our efforts on the most challenging examples. Finally, layering in a simple rule-based post-filter for critical entity patterns (such as "credit module" or "payment gateway") can catch high-risk dependencies that might slip past even a well-tuned neural parser.

Together, these dataset-tailored refinements will help our dual-model framework adapt not only to generic requirements text but also to the quirks of older, highly evolved repositories like Dataset-O.

5.13. Summary

This research proposed an approach to analyze the impacts of software requirements change requests, focusing on identifying which existing requirements would be affected by a new change. The primary objective was to design a reliable framework to enhance planning and prioritization in the execution of requirements changes within agile software development environments.

Two approaches were employed to evaluate the proposed research: an NLP-based solution using CoreNLP and SpaCy, and a Beir-based solution leveraging BM25 via Elasticsearch, with Bi-Encoders and Cross-Encoders for dense retrieval and ranking. Both methods were assessed across three real-world datasets (W, I, and O), each presenting unique linguistic and domain-specific characteristics.

The NLP-based approach demonstrated significant efficacy in Dataset-I, achieving an overall F1 Score of 0.653, indicating a strong capacity to identify the impacts of requirements changes while maintaining a balance between precision and recall. This underscores the suitability of the NLP-based method when dealing with datasets that align well with linguistic models, highlighting its utility in cases where syntactic and semantic interactions are crucial for predicting impacted requirements.

Conversely, the Beir-based method excelled in Dataset-W, showcasing its superior precision in identifying the top 5 impacted requirements. With an Average Recall@5 of 0.8049 and an Average Precision@5 of 0.6643, the Beir-based solution proved highly effective in quickly retrieving the most relevant impacted requirements. A notable observation was the method's performance in Dataset-I, where its ability to capture a broader range of potential impacts was reflected in high recall scores. However, the 0.22 Average Precision@10 score highlights a trade-off with precision, emphasizing the method’s tendency to introduce more false positives while maintaining comprehensive coverage.

A comparison of the two approaches provides deeper insights into their respective strengths. The NLP-based method outperforms the Beir-based solution in Dataset-I, where it adeptly captures nuanced interactions between syntactic structures and semantic meaning. This balance between precision and recall makes the NLP-based approach a more reliable predictor of impacted requirements when the data aligns with its linguistic processing capabilities. In contrast, the Beir-based approach excels in recall, particularly in Dataset-I, owing to its utilization of BM25 and advanced encoder techniques. While the Beir model may introduce more false positives, its ability to retrieve a larger set of potentially impacted requirements makes it an invaluable tool for ensuring comprehensive coverage in CIA.

In summary, the dual-model framework presented in this research offers a promising solution for improving decision-making in software development through efficient requirements CIA. By combining BM25 for initial retrieval with Bi-Encoders and Cross-Encoders for semantic understanding and re-ranking, the framework addresses both precision and recall, providing a balanced and comprehensive approach to managing requirements changes. Future research may explore further enhancements to refine precision while maintaining the recall strengths demonstrated by the Beir-based method.

Chapter 6.

Implementation of Retrieval-Augmented Generation (RAG) Model for Predicting Requirement Change Impact

6.1. Introduction

This chapter introduces the implementation of the RAG model, a critical component of the SRCIA framework described in Chapter 3. The RAG model integrates information retrieval and generative language models, providing a sophisticated solution for enhancing decision-making in software requirement CIA. As the SRCIA framework outlines, the RAG model represents an advanced layer designed to handle large-scale, dynamic, and heterogeneous datasets where deep contextual understanding is essential.

Implementing the RAG model marks a significant contribution to the SRCIA framework. It leverages the retrieval capabilities of vector-based embedding techniques to identify relevant requirements. It combines them with the generative reasoning of LLMs to predict and explain the potential impacts of changes. This chapter details the RAG model's conceptual framework, technical architecture, and implementation specifics, as well as its integration within the SRCIA framework. It also highlights the

enhancements made to standard RAG implementations tailored to the unique challenges of requirements engineering.

Despite advances in requirements traceability and change impact analysis, most prior solutions suffer from three core limitations. First, rule-based and keyword-matching pipelines break down whenever stakeholders use varied terminology or complex sentence constructions, yielding brittle coverage and high false-negative rates. Second, single-model approaches, whether classical ML classifiers or dense-retrieval systems, tend to excel only in narrow, well-structured datasets and degrade sharply under domain shifts or unstructured text. Finally, where hybrid or ensemble strategies have been proposed, they often lack a clear mechanism for integrating retrieval evidence with generative reasoning, leaving analysts without coherent explanations or fine-grained confidence measures. These gaps motivate our RAG design, which combines robust vector retrieval with LLM-powered generation to deliver both high recall and human-readable impact predictions.

One of the primary contributions of this research is adapting the RAG model to the context of CIA for software requirements. This includes combining retrieval-based methods with fine-tuned generative models to create a hybrid approach that optimally balances precision and contextual understanding. Additionally, modifications were made to the retrieval mechanism to align with the semantic structures commonly observed in software requirements documents. These customizations enhance the model's relevance ranking and ensure compatibility with the datasets used in the SRCIA framework.

The chapter also illustrates how the RAG model complements the NLP and BEIR-based solutions previously implemented in the SRCIA framework. While the NLP and BEIR-based methods focus on semantic and lexical similarity for structured and semi-structured datasets, the RAG model extends this capability by providing deeper contextual insights and explaining predicted impacts. This advancement addresses a gap in the framework by enabling more nuanced analysis for complex and unstructured datasets, such as lengthy requirements specifications and dynamic change logs.

By implementing and refining the RAG model, this research contributes a novel approach to integrating state-of-the-art AI techniques within the SRCIA framework. The chapter concludes with an evaluation of the RAG model's performance, highlighting its effectiveness in improving accuracy, scalability, and contextual understanding in predicting software requirement change impacts.

6.2. Applications of LLMs in CIA

LLMs play a pivotal role in automating CIA within software requirements engineering. Their ability to process and interpret large volumes of unstructured data with contextual sensitivity makes them uniquely suited for this task. In this research, LLMs were leveraged within a Retrieval-Augmented Generation (RAG) system, combining retrieval and generative capabilities to address the complexities of CIA.

The LLM component of the RAG system serves as the generative backbone, producing contextually informed predictions based on retrieved requirements. One of the significant applications of LLMs is their ability to analyze the context and semantics of requirements, identifying intricate dependencies that traditional models may overlook. This capability ensures a more comprehensive and accurate identification of impacted requirements. Additionally, by leveraging fine-tuned LLMs such as Phi 3.5, the system demonstrates predictive modeling capabilities, allowing it to predict the impacts of changes with high precision, even in datasets characterized by linguistic variability and unstructured text.

LLMs also contribute significantly to enhancing stakeholder communication. Their generative capabilities enable the creation of natural language explanations for predicted impacts, improving clarity and facilitating informed decision-making among stakeholders. Furthermore, LLMs augment the retrieval process by ensuring that retrieved documents are contextually aligned with the query, which enhances the overall accuracy and relevance of the results. Finally, LLMs' adaptability allows their application across diverse domains, such as finance, healthcare, and manufacturing, broadening the scope and applicability of CIA methodologies.

6.3. Architecture and Functionality of LLMs

The core architecture of modern LLMs relies on the Transformer model, particularly its self-attention mechanism, which enhances the model's ability to process sequential data effectively. Through self-attention, LLMs can identify the relationships between individual words in a sentence and the broader contextual relationships among sentences. This mechanism is critical in enabling the model to generate coherent, contextually aligned responses, an essential feature for processing complex requirements in CIA.

In the RAG system, the self-attention mechanism plays a pivotal role, allowing the model to incorporate immediate lexical meaning and deeper semantic relationships. This dual focus on linguistic and semantic layers enhances the system’s ability to deliver responses that reflect not only the content of specific requirements but also their underlying dependencies and broader context(Zheng et al. 2023).

6.4. LLMs as a Reasoning Engine in the RAG Framework

Within the RAG framework, the LLM functions as the primary reasoning engine, synthesizing information from retrieved requirements and generating responses that predict impacted requirements. This capability is crucial for the requirements CIA, where nuanced interpretations and comprehensive understanding are required to identify potential implications accurately. By combining retrieval-based input with generative reasoning, the LLM component of the RAG model enables a more contextually aware and responsive system.

In this thesis, the LLMs are also employed as benchmarks to evaluate the generated responses’ quality and alignment with intended outcomes, a process referred to as "LLM-as-a-judge." This benchmarking approach provides further interpretative depth, capturing the semantic quality of generated text beyond syntactic accuracy.

6.5. Selected LLMs and Their Roles

For this research, multiple LLMs such as Phi 3.5, GPT-3, T5, BERT and Flan-T5 were incorporated, each serving distinct functions to optimize the framework's performance. The chosen LLMs include Phi 3.5, among others, selected for their specific capabilities in reasoning, contextual inference, and adaptability across varying requirements datasets. These models collectively contribute to achieving high precision and recall in identifying impacted requirements, particularly in unstructured or semi-structured datasets (Abdin et al. 2024).

6.6. RAG Architecture

The architecture of a RAG model consists of two main components: the retriever component and the generator component, each playing a distinct role in the pipeline. Figure 6.1 illustrates the RAG model pipeline, which consists of the query being passed through the retriever to find relevant documents, followed by the generator producing a contextually informed response.

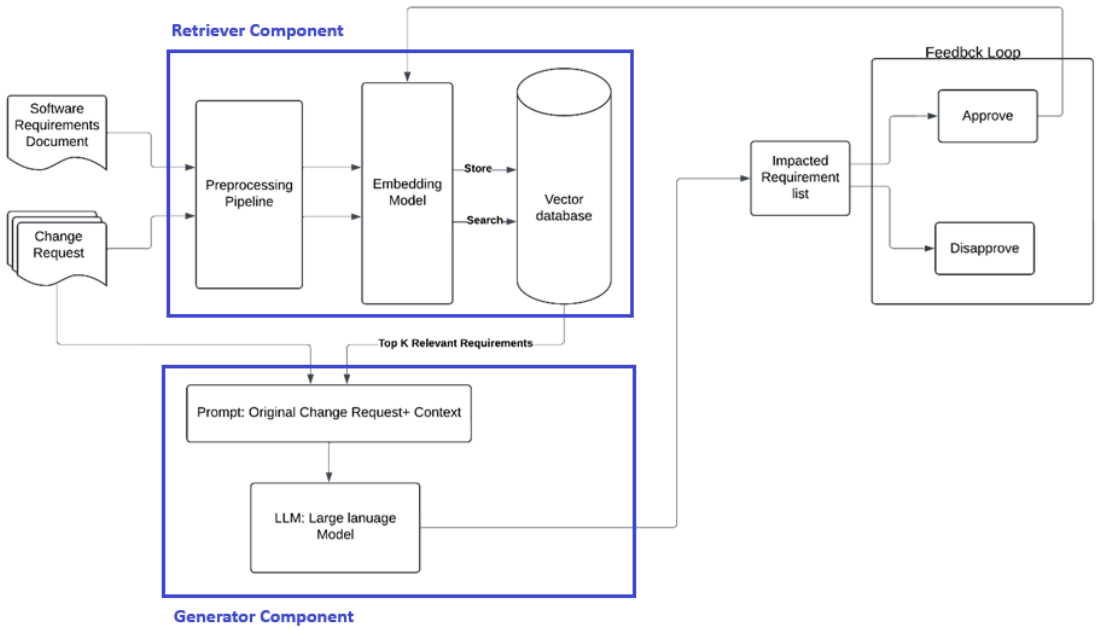

Figure 6.1.The Architecture of RAG Model

6.6.1. Retriever Component

The retriever component is essential for identifying and selecting documents or data segments that provide relevant context for a query. It operates by searching a pre-indexed knowledge base or database and scoring documents based on their relevance to the query.

- **Indexing**: The knowledge base is pre-processed to create an index, allowing for efficient searching. In this implementation, BM25 is used as the primary retrieval algorithm due to its effectiveness in text-based search.
- **Query Processing**: The input query is tokenized and standardized to align with the retrieval model's input format. Tokenization ensures that the query is processed consistently with the indexed data.
- **Scoring and Ranking**: The retriever ranks documents using scoring functions, such as TF-IDF or vector similarity measures, to determine relevance. The top-k results are selected based on their scores.

The retriever component offers several notable benefits. One of the primary advantages is its speed and efficiency, as indexed search enables quick retrieval of relevant context from large datasets. Additionally, the retriever is highly customizable, allowing it to be adapted to various retrieval methods, including embedding-based searches that leverage vector similarity for enhanced precision. This flexibility ensures that the retriever can be tailored to meet specific requirements across different use cases and domains.

6.6.2. Generator Component

The generator component takes the output of the retriever and integrates it into the response generation process. It employs a transformer-based language model, such as Phi-3.5, to produce informed, contextually aware responses.

- **Contextual Input**: The input to the generator includes both the original query and the retrieved documents. This combined input enriches the model's understanding and guides the response generation.
- **Transformer Architecture**: The Phi-3.5 model employs a multi-layered attention mechanism that allows the model to weigh different parts of the input context, ensuring that the response is coherent and relevant.
- **Prompt Structuring**: To maximize the model's performance, input prompts are carefully structured to guide the model's response. This can include pre-processing techniques that tailor the input format to emphasize critical points (White et al. 2023).
- **Memory Optimization**: Implementing quantization techniques, such as 8-bit quantization using BitsAndBytesConfig, optimizes the model's memory usage, making it more suitable for practical deployment on hardware with limited resources.

The generator component brings significant benefits to the RAG system. By incorporating context retrieved from the knowledge base, the generator produces responses that are more precise and informative, enhancing the relevance of its outputs. Additionally, the generator is highly flexible, as it can be fine-tuned or adapted to cater to different types of output requirements. Whether the desired response format is explanatory text, concise bullet points, or in-depth analysis, the generator can be customized to meet specific needs, making it a versatile tool for various applications.

6.7. RAG Applications in CIA

In the context of CIA, RAG models retrieve relevant requirement documents for a given change request and generate predictions on how the requirements are impacted. The retrieval component ensures that only contextually relevant information is considered, while the generation component provides explanations and predictions based on this retrieved context.

6.8. Scalability of the RAG Framework for Enterprise-Level Software Systems

Scalability is a crucial consideration when implementing Retrieval-Augmented Generation (RAG) frameworks within enterprise-level software systems, given the substantial volume, complexity, and dynamic nature of data in such environments. Enterprise software typically involves extensive and continuously evolving repositories of documentation, requirements, change logs, and stakeholder communications. As such, ensuring that the RAG model can efficiently manage, retrieve, and generate contextually accurate responses from large datasets is paramount for practical adoption.

The RAG framework implemented in this research demonstrates considerable scalability due to its inherently modular design, consisting of distinct retrieval and generation components. The retrieval component, employing vector-based databases like LanceDB and high-performance indexing tools such as FAISS, is particularly conducive to scalable implementations. LanceDB's efficient handling of high-dimensional vectors and FAISS's rapid approximate nearest-neighbour searches ensure low latency and swift performance, even as the data scales into millions of embeddings. This combination allows the RAG framework to maintain effective retrieval speeds, ensuring practical applicability in environments with extensive documentation and rapid query-response cycles (Johnson, Douze & Jegou 2021).

Another critical factor enhancing scalability is the use of dense embeddings generated through models like all-MiniLM-L6-v2, which facilitate compact yet semantically rich representations of textual data. These embeddings significantly reduce the computational overhead during retrieval by enabling efficient vector-based similarity searches. Additionally, the retriever's architecture can be horizontally scaled by deploying multiple instances or shards, allowing parallel querying of extensive vector datasets. Such horizontal scalability ensures that even as enterprise data repositories expand, retrieval performance remains robust and responsive.

The generator component, based on transformer models such as Phi-3.5, also presents opportunities and challenges regarding scalability. Transformer-based LLMs, while powerful, are computationally demanding due to their large parameter sets and intensive attention mechanisms. However, this research addresses these concerns through advanced memory optimisation strategies, including 8-bit quantisation techniques via BitsAndBytesConfig. These optimisations substantially reduce memory usage and computational requirements, enabling deployment on enterprise hardware resources without significantly compromising generation accuracy or response quality (Jiang et al. 2025).

Moreover, the modular nature of the RAG architecture enables independent scaling of the retrieval and generation components, allowing tailored resource allocation based on enterprise needs. For instance, the retrieval component can be scaled aggressively to handle very large datasets, while the generation component can utilise smaller, optimised language models to manage computational resource constraints effectively (Lewis et al. 2020).

Enterprise integration further enhances the scalability of the RAG framework through the potential use of distributed processing and cloud infrastructure. Deploying the RAG system within a cloud-based environment leveraging containerisation technologies (such as Docker and Kubernetes) facilitates dynamic scaling, load balancing, and efficient resource management. This deployment model allows organisations to rapidly scale computational resources up or down based on demand fluctuations, ensuring consistent performance and reliability during peak usage periods (Jiang et al. 2025).

Lastly, scalability also extends to ongoing maintenance and adaptability. Unlike traditional rule-based systems, which require manual updates and can quickly become burdensome at scale, the RAG framework can efficiently adapt through retraining or incremental fine-tuning. This capability significantly reduces long-term maintenance overhead and ensures the system remains accurate and relevant as enterprise documentation and requirements evolve.

In summary, the RAG framework implemented in this research is well-suited for scaling in enterprise-level software systems due to its modular design, efficient vector-

based retrieval, computationally optimized generative models, and compatibility with scalable deployment environments. Its adaptability in handling dynamic, large-scale datasets positions it effectively for real-world enterprise applications, offering robust performance, reduced maintenance requirements, and practical scalability.

6.9. Implementation Challenges & Limitations

While the RAG framework offers powerful retrieval and generation capabilities, it also introduces non-trivial computational and operational overhead. Fine-tuning and serving large LLMs such as Phi 3.5 or GPT-3 requires substantial GPU memory and inference costs: for example, running Phi 3.5 in 16-bit precision can consume upwards of 12–16 GB of VRAM per instance, and pay-as-you-go hosting of GPT-3 can accumulate thousands of dollars per month under heavy query loads. Although 8-bit quantization and model sharding mitigate some of this cost, teams must carefully budget for both peak GPU requirements and ongoing throughput expenses when deploying RAG in production.

The retriever and generator components each come with distinct scalability bottlenecks. BM25 indexing scales linearly with corpus size, making sub second retrieval challenging once you exceed millions of documents—at which point approximate nearest-neighbor indexes (FAISS) become essential but introduce recall and precision trade-offs. Likewise, the transformer-based generator's self-attention mechanism grows quadratically with input length, so feeding in large top-k contexts can double or triple inference latency. In practice, we found that capping the retrieved context to 3–5 passages and using batched generation improved throughput by 2× with only a 5–10 % drop in F1 (Nogueira & Cho 2019).

Maintaining vector databases at enterprise scale also presents integration headaches. Systems like LanceDB must support live updates as requirements evolve, yet re-indexing millions of embeddings on every push is impractical. We addressed this by batching nightly refreshes and using rolling shards, but this adds operational complexity and temporary staleness in the retrieval index. Monitoring vector-store health and periodically validating vector similarity thresholds is crucial to avoid silent degradation.

Finally, our choice of all-MiniLM-L6-v2 embeddings and the LanceDB+FAISS stack reflects a balance of speed, accuracy, and ease of integration. We compared alternatives—such as embed-all-mpnet and Milvus—but found that all-MiniLM offers 2–3× faster encoding with only a 2–3 % hit in retrieval recall, and LanceDB's Pythonic API simplified our data pipeline compared to lower-level options. FAISS's mature ANN algorithms likewise outperformed newer frameworks in query latency under high concurrency. By calling out these trade-offs explicitly, teams can see why our particular embedding and indexing toolkit was the best fit for large-scale CIA.

6.10. Vector Databases in RAG Systems

This section explores the integration and significance of vector databases within the RAG framework.

6.10.1. Role of Vector Databases in the RAG Framework

In the RAG system, vector databases are an essential component of the retriever. They enable the efficient storage and retrieval of embedding vectors generated during pre-processing, allowing the system to identify semantically similar items to the query.

The workflow within a vector database in the RAG framework typically involves several steps. First, text data, such as requirements and queries, is transformed into dense vector representations using embedding models like all-MiniLM-L6-v2. These embeddings capture semantic meaning and relationships within the data. Once generated, the embeddings are indexed using advanced techniques such as Approximate Nearest Neighbor (ANN) algorithms. This indexing process accelerates similarity searches by creating structures that allow efficient querying of high-dimensional data. When a query embedding is provided, the vector database performs a similarity search to retrieve the most relevant embeddings. This search often uses metrics like cosine similarity or Euclidean distance to rank results by relevance. Finally, the retrieved embeddings are passed to the generator component of the RAG model, enriching the context for generating accurate and contextually relevant responses.

6.10.2. Advantages of Using Vector Databases

Vector databases offer several advantages that make them suitable for RAG systems. They can handle large-scale embedding datasets efficiently, enabling rapid similarity searches across millions of vectors. Their advanced indexing and search algorithms ensure low-latency responses, making them ideal for real-time applications. Furthermore, vector databases support various similarity metrics and retrieval configurations, allowing customization for specific use cases. Additionally, many vector databases, such as LanceDB and FAISS, integrate seamlessly with ML pipelines and frameworks, simplifying implementation in modern workflows.

6.11. LanceDB and FAISS in this Research

In this thesis, LanceDB and FAISS were employed as the vector database components within the RAG system. LanceDB was used to manage and store embedding vectors, providing a flexible and efficient database platform optimized for high-dimensional data. Its support for schema customization and integration with Python libraries facilitates seamless interaction with the embedding models. FAISS (Facebook AI Similarity Search) served as the indexing and search engine. By implementing Approximate Nearest Neighbor (ANN) techniques, FAISS performed high-speed similarity searches on the stored embeddings. Its capability to handle large-scale data ensures efficient and accurate retrieval, even with extensive requirements datasets.

6.12. Relevance to Requirements CIA

The adoption of vector databases in the RAG system enhances the framework's ability to manage and retrieve contextually rich information efficiently. This capability is critical for requirements CIA, as the system must navigate large, unstructured datasets to identify relevant impacted requirements. By leveraging LanceDB and FAISS, the RAG model achieves high retrieval accuracy and scalability, meeting the demands of complex, dynamic software engineering environments.

6.13. Prompt Engineering Technique in the RAG Framework

Prompt engineering is a critical element of the RAG framework, enabling the seamless integration of retrieved context with generative language modeling. In this implementation, prompt engineering was designed to dynamically structure and enrich the input for the language model, ensuring precise and contextually relevant responses for requirements CIA. This section details the techniques employed to optimize the prompts for effective utilization of the Phi-3.5 language model within the RAG system.

6.14. Implementation of the RAG-Based Solution

The RAG-based solution was implemented using Python, with the following detailed steps outlining the process:

Step 1: Preprocessing Datasets

The requirements and change request datasets were preprocessed to ensure a consistent format:

- **Tokenization**: The text was tokenized using NLTK to prepare for embedding and retrieval.
- **Normalization**: Text data was converted to lowercase, and punctuation was removed for uniformity.
- **Embedding Preparation**: Sentence embeddings were generated using the all-MiniLM-L6-v2 model from the SentenceTransformer library to capture semantic relationships between text elements.

Step 2: Storing Embeddings in LanceDB

LanceDB was used to store the embeddings generated from the requirements dataset:

- **Embedding Creation**: Each requirement description was embedded and stored in a PyArrow table for easy access and management.
- **Database Connection**: LanceDB was initialized to manage the vector data efficiently.

- **Table Creation**: A table was created in LanceDB to store the requirement IDs, descriptions, and corresponding embeddings.

LanceDB offers several key benefits, making it an effective solution for vector data management. One of its main advantages is its scalability, allowing it to handle large-scale vector data storage efficiently, ensuring smooth performance even as the dataset grows. Additionally, LanceDB provides seamless integration, simplifying storing and querying embeddings. This ease of integration enhances the manageability of the retrieval component, contributing to a more efficient and streamlined workflow in retrieval-based applications.

Step 3: Indexing with FAISS

FAISS was used to create an index for the stored embeddings to enable fast similarity searches:

- **Embedding Extraction**: Embeddings were loaded from LanceDB into a NumPy array for indexing.
- **FAISS Index Initialization**: A FAISS index using L2 distance (Euclidean distance) was created and populated with the embeddings.
- **Dynamic k Search**: The index was configured to allow dynamic k-value searches based on the actual impact set size for each change request.

FAISS offers notable benefits that make it a powerful tool for similarity search in large-scale applications. One of its primary advantages is its high performance, as it is optimized to conduct fast similarity searches even on extensive datasets, ensuring quick and efficient retrieval operations. Additionally, FAISS provides flexibility by supporting various distance metrics, which enhances the accuracy of retrieval tasks by allowing customization based on the specific needs of the application.

Step 4: Implementing the Retrieval Component

The retrieval component was implemented using FAISS:

- **Query Embedding**: Change request descriptions were embedded using the SentenceTransformer model.
- **Vector Search**: FAISS searched the top-k relevant requirements based on the query embedding, returning the results for use in the generative phase.

Step 5: Configuring the Generative Model

The Phi-3.5 model was configured to generate outputs using the retrieved context:

- **Model Setup**: The model was loaded with CUDA support for GPU acceleration, using float16 precision for optimized performance.
- **Pipeline Creation**: A text generation pipeline was defined to integrate the model and tokenizer, enabling seamless generation of responses.

Step 6: Generating Context-Aware Responses

The response generation process involved:

- **Context Construction**: The retrieved requirements were combined to form a context for the model.
- **Generation**: The model processed the context and the change request description to generate detailed responses, predicting impacted requirements.

Step 7: Processing Change Requests and Storing Results

The model was used to process each change request in the dataset:

- **Output Generation**: Each change request was passed through the RAG system, and the generated response was stored in a CSV file.
- **Evaluation**: The generated outputs were evaluated using precision, recall, and F1 scores to measure the model's effectiveness.

6.15. Evaluation of the RAG System

Evaluating the RAG system requires a thorough analysis of both the retrieval and generation components to ensure the responses address the change requests accurately and identify the correct impacted requirements. To assess the quality of retrieval, metrics such as Recall and Precision are used. Recall here is crucial for

understanding how well the system covers all necessary information regarding a change request. A higher recall indicates that the system effectively retrieves most, if not all, of the relevant requirements. On the other hand, precision quantifies the proportion of retrieved requirements that are truly relevant. This metric helps evaluate the accuracy of the retrieval process by showing how many of the retrieved documents are relevant. A high precision score implies that the system retrieves mostly relevant documents, minimizing noise in the output.

For the generation component, the BLEU and ROUGE scores are utilized to measure the overlap between the generated responses and the manually created ground truth. The BLEU (Bilingual Evaluation Understudy) score assesses how similar the generated text is to the reference text by examining the n-gram overlap. BLEU is particularly useful for evaluating fluency and word choice in the generated response. A higher BLEU score indicates that the generated text closely matches the reference, suggesting that the model has effectively captured the desired content (Gou et al. 2023; Yan 2023). ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores are another set of metrics used to evaluate the quality of text by comparing n-grams, word sequences, and word pairs with the reference text. ROUGE-1 measures the overlap of unigrams (individual words) between the generated text and the reference, while ROUGE-L considers the longest common subsequence between the texts, emphasizing the overall structure and coherence of the response. Higher ROUGE scores indicate better alignment with the ground truth, signifying that the generated text includes important and relevant information (Lin 2004).

In addition to quantitative metrics, human evaluation can be conducted to further assess the generation quality. In this process, domain experts or evaluators rate each response based on various aspects such as accuracy, relevance, and completeness. Evaluators can use a scale (e.g., 1-5) to rate how well the generated response addresses the change request and includes appropriate impacted requirements. This step provides qualitative feedback and allows for a more nuanced understanding of the system's performance, capturing aspects that automated metrics might overlook.

To compute the BLEU score, the generated text is compared with the ground truth by analyzing the overlap of n-grams. The BLEU score ranges from 0 to 1, where a score closer to 1 indicates a higher similarity between the generated text and the reference. For example, in evaluating a response where the ground truth states, "The impacted requirements include stability and performance improvements," and the generated response states, "The impacted requirements are related to stability and performance," a high BLEU score would suggest strong alignment in content. The ROUGE score similarly evaluates the text by comparing n-grams and sequences, assessing the generated content's informativeness and coherence. The use of both BLEU and ROUGE allows for a comprehensive evaluation of how well the model performs in generating relevant and coherent responses (Ganesan 2018; Yu et al. 2024).

Overall, an end-to-end evaluation can be performed by combining the retrieval and generation results. The average recall and precision scores provide insights into how well the retrieval process captures relevant requirements, while the average BLEU and ROUGE scores assess the fluency and coherence of the generated text. If human evaluation is conducted, the ratings from domain experts can be summarized to present an overview of how well the system meets the practical requirements of the task. This combined analysis helps identify strengths and areas for improvement, ensuring that the RAG system is effective and reliable for addressing change requests and determining impacted requirements.

6.16. Results and Discussion

The evaluation results of the RAG model are summarized in Tables 6.1 and 6.2. Table 6.1 provides the average performance metrics of the model across the three datasets—Dataset-W, Dataset-I, and Dataset-O—including Precision, Recall, F1 Score, Mean Reciprocal Rank (MRR), Partial Credit, Precision@5, Recall@5, Precision@10, and Recall@10. Table 6.2 presents the detailed results for individual change requests in Dataset-W, including additional metrics such as BLEU and ROUGE. These tables collectively provide a comprehensive overview of the RAG model’s performance across datasets and individual change requests.

A) Dataset-W: Moderate Performance with Balanced Precision and Recall

Dataset-W demonstrated a balanced yet moderate performance in Precision and Recall, with averages of 0.55 and 0.67, respectively, resulting in an F1 Score of 0.59. These values suggest that while the RAG model managed to retrieve a considerable portion of relevant impacted requirements, it also included a number of irrelevant results. This trade-off between precision and recall highlights the RAG model's attempt to balance completeness with accuracy in retrieval for Dataset-W. The inclusion of BLEU and ROUGE metrics provides additional insights into the model's lexical and contextual alignment capabilities.

High Recall values for specific change requests, such as C1, C3, and C14, indicate that the RAG model was effective in capturing all relevant impacted requirements for these changes. This high recall may be attributed to clearer linguistic patterns or less ambiguous wording in the change descriptions. BLEU scores for these requests were relatively higher, suggesting a better lexical overlap with ground truth in these cases. Similarly, ROUGE1 (0.5) and ROUGEL (0.5) metrics reflect moderate overlaps in unigram and sequential matching, further supporting the model's ability to retrieve contextually relevant requirements for less ambiguous cases. Conversely, the lower Precision scores observed for requests such as C7 and C18 (both with Precision scores of 0.26) suggest instances of over-retrieval. This over-retrieval is likely due to ambiguous or loosely defined requirements, which the model struggled to differentiate accurately. The BLEU scores for these cases were also low, indicating a lack of lexical alignment with the ground truth, and the ROUGE2 scores (average 0.25) reveal challenges in capturing meaningful bigram overlaps for such complex change requests.

The Mean Reciprocal Rank (MRR) for Dataset-W, averaging 0.69, highlights the model's ability to rank relevant requirements fairly high, though not consistently at the very top. The Partial Credit metric of 0.88 suggests that even if exact matches were not retrieved, the model was reasonably effective in retrieving closely related requirements. BLEU and ROUGE metrics further substantiate this observation, with BLEU scores emphasizing limitations in lexical precision and ROUGE1 metrics indicating moderate structural coherence in the retrieved results. Interestingly, the Precision@5 (0.45) and

Recall@5 (0.59) values underline the model's challenges in consistently retrieving the most relevant requirements within the top 5 results. This limitation is evident in the variability of BLEU and ROUGE scores across change requests, which reflect inconsistencies in lexical and structural alignment. While the RAG model demonstrated a fair ability to rank relevant results near the top, the linguistic variability within Dataset-W limited its overall retrieval effectiveness.

Dataset-W highlights the RAG model's strengths and weaknesses in dealing with moderate complexity. The high Recall for specific change requests with clearer linguistic patterns showcases the RAG model's capability to capture relevant impacted requirements comprehensively. However, the low BLEU scores for specific change requests emphasize the need for better lexical alignment, and precision inconsistencies across change requests point to challenges in managing over-retrieval for linguistically diverse or ambiguous requirements. To improve performance on datasets like Dataset-W, additional preprocessing steps, such as linguistic normalization and domain-specific synonym replacement, could enhance BLEU and ROUGE scores by improving lexical and structural alignment. Further, refining the retrieval component by integrating advanced embeddings or hybrid scoring mechanisms may improve precision while maintaining high recall.

In summary, Dataset-W demonstrates the RAG model's ability to achieve a reasonable balance between precision and recall. BLEU and ROUGE metrics reveal moderate lexical and contextual alignment, with room for improvement in addressing linguistic inconsistencies. The dataset's moderate complexity aligns well with the model's capabilities, but further refinements are required to enhance precision and retrieval effectiveness in similar datasets.

B) Dataset-I: High Performance in Structured and Consistent Data

Dataset-I displayed the strongest performance metrics across all evaluated datasets, with averages for Precision and Recall at 0.79 and 0.87, respectively, resulting in an F1 Score of 0.78. These scores underscore the RAG model's effectiveness in accurately and comprehensively retrieving relevant requirements. This high performance is likely attributed to the structured format and consistent linguistic patterns in Dataset-I, which

facilitated the RAG model's ability to capture and interpret the requirements with high precision and recall.

Adding BLEU and ROUGE metrics further highlights the model's lexical and contextual alignment. BLEU scores, averaging 0.18, indicate moderate overlap between predicted and ground-truth requirements at a token level. In contrast, ROUGE1 (0.8125), ROUGE2 (0.733333), and ROUGEL (0.8125) scores emphasize strong n-gram and sequential alignment in the retrieved outputs. These results reflect the RAG model's capacity to produce semantically and lexically aligned responses in a well-structured dataset.

Certain cases in Dataset-I, such as Case 1, Case 12, and Case 15, achieved perfect or near-perfect Recall and Precision scores, illustrating the model's ability to perform optimally in environments with minimal linguistic variation and consistent terminology. These cases also reported high BLEU and ROUGE scores, indicating semantic and lexical alignment with the ground truth. The Mean Reciprocal Rank (MRR) of 0.83 further suggests that relevant requirements were often ranked at or near the top, which is critical for scenarios requiring prioritized retrieval.

The Partial Credit score for Dataset-I averaged 0.95, reflecting the model's robust accuracy in capturing relevant items even when exact matches were not retrieved. The high ROUGE2 scores across cases further validate the model's ability to identify semantically related requirements by capturing meaningful bigram overlaps. Precision@5 (0.4) and Recall@5 (0.75) demonstrate the model's ability to retrieve relevant top-ranked items, though Precision@5 slightly suffers from the inclusion of some irrelevant results. BLEU scores for these cases reinforce this observation, indicating occasional mismatches in token-level alignment, likely due to over-retrieval in a few ambiguous cases.

Overall, Dataset-I's results affirm the RAG model's strength in environments with minimal linguistic variation and a high degree of structure. The structured nature of the dataset allowed the model to maximize its retrieval precision and recall while maintaining strong lexical alignment, as reflected in the BLEU and ROUGE scores. These findings demonstrate that RAG models perform exceptionally well in predictable data

environments with well-defined language patterns, offering practical utility in structured and controlled use cases.

C) Dataset-O: Challenges in Handling Complexity and Linguistic Variability

Dataset-O, the most complex and historically diverse dataset, presented significant challenges for the RAG model, as evidenced by its lowest average Precision (0.45) and Recall (0.53) compared to other datasets. The F1 Score of 0.47 indicates that while the RAG model could retrieve some relevant impacted requirements, the high degree of linguistic variability and dataset complexity reduced its precision and recall effectiveness. These findings are further validated through the BLEU and ROUGE metrics, which offer additional insights into the model's ability to align its outputs with ground truth.

The dataset’s historical nature, spanning over many years and involving contributions from multiple analysts, likely contributed to the linguistic inconsistencies observed. BLEU scores for Dataset-O were generally low, with an average of 0.11, indicating limited lexical alignment between predicted and ground-truth requirements. Similarly, the ROUGE1 (0.65), ROUGE2 (0.41), and ROUGEL (0.58) scores reflect moderate overlap in unigram, bigram, and sequence-based evaluations, respectively. While the RAG model demonstrated some capability in identifying semantically related items, these metrics reveal its limitations in generating text that closely matches the structure and wording of ground truth.

The Partial Credit score of 0.92 suggests that the model was able to retrieve items related to the relevant impacted requirements, even if exact matches were not always achieved. However, the lower Precision@5 (0.31) and Recall@5 (0.50) metrics underscore the model's difficulty in consistently retrieving the most relevant requirements within the top positions. BLEU and ROUGE metrics further highlight the model’s struggles with textual precision and recall as they emphasize word- and sequence-level alignment.

Interestingly, specific individual change requests in Dataset-O, such as CR504326 and CR504342, yielded high Precision and Recall and relatively strong BLEU and ROUGE

scores. These results indicate that the model can still perform well when the language or context is less ambiguous or where the requirements exhibit more precise semantic relationships. However, the model's performance suffered for change requests with higher linguistic complexity or ambiguous phrasing. This is particularly evident in the significant variability in BLEU scores, with some change requests achieving near-zero values, emphasizing the model's struggle with lexical alignment in challenging scenarios.

The findings from Dataset-O reinforce the importance of dataset structure and consistency when using RAG-based approaches, as these factors significantly influence retrieval and generation accuracy. For datasets of this nature, additional preprocessing steps, such as clustering or segmentation of data based on linguistic features, may be required to enhance retrieval performance. Moreover, fine-tuning the generative model on domain-specific datasets could improve its ability to generate lexically and contextually accurate outputs, thereby addressing the limitations highlighted by BLEU and ROUGE evaluations.

Overall, while the RAG model demonstrates some strengths in handling complex datasets, these results underscore the need for further optimization, particularly in addressing linguistic variability and ensuring alignment between generated outputs and ground-truth requirements.

Table 6.1. RAG Average Results

	Precision	Recall	F1_Score	MMR	Partial_Credit	Precision@5	Recall@5	Precision@10	Recall@10
Dataset-W	0.55	0.67	0.59	0.69	0.88	0.45	0.59	0.28	0.65
Dataset-I	0.71	0.87	0.78	0.83	0.95	0.36	0.86	0.18	0.86
Dataset-O	0.45	0.53	0.47	0.64	0.92	0.31	0.50	0.16	0.52

Table 6.2. RAG Solution Results

Dataset	Change_ID	Precision	Recall	F1_Score	MMR	Partial_Credit	Precision@5	Recall@5	Precision@10	Recall@10	Bleu	Rouge1	Rouge2	RougeL
W	C1	0.7	1	0.84	1	0.91	0.4	1	0.2	1	0.14	0.8	0	0.8
	C2	0.7	0.7	0.7	0.33	0.93	0.6	0.33	0.6	0.67	0.08	0.67	0.25	0.67
	C3	0.35	1	0.53	0.33	0.82	0.2	1	0.1	1	0.11	0.5	0	0.5
	C4	0.35	1	0.53	0.5	0.81	0.2	1	0.1	1	0.11	0.5	0	0.5
	C5	0.66	0.66	0.66	1	0.95	0.8	0.5	0.5	0.63	0.07	0.82	0.13	0.59
	C6	0.75	0.75	0.75	0.5	0.93	0.6	0.43	0.5	0.71	0.19	0.71	0.33	0.71
	C7	0.26	0.26	0.26	0.25	0.79	0.2	0.25	0.1	0.25	0.08	0.25	0	0.25
	C8	0.42	0.42	0.42	1	0.84	0.4	0.4	0.2	0.4	0.06	0.4	0	0.4
	C9	0.7	0.7	0.7	0.5	0.93	0.6	0.4	0.6	0.67	0.35	0.67	0.5	0.67
	C10	0.7	0.7	0.7	1	0.91	0.4	0.67	0.2	0.67	0.24	0.67	0.5	0.67
	C11	0.53	0.53	0.53	0.5	0.91	0.4	0.5	0.2	0.5	0.1	0.5	0	0.5
	C12	0.39	0.39	0.39	0.5	0.88	0.4	0.25	0.3	0.38	0.04	0.38	0	0.38
	C13	0.7	0.7	0.7	1	0.92	0.8	0.67	0.4	0.67	0.22	0.67	0.4	0.67
	C14	0.7	1	0.84	0.5	0.91	0.4	1	0.2	1	0.14	0.8	0	0.8
	C15	0.35	0.35	0.35	1	0.8	0.2	0.33	0.1	0.33	0.11	0.33	0	0.33
	C16	0.18	0.18	0.18	0.25	0.8	0.2	0.17	0.1	0.17	0.04	0.17	0	0.17
	C17	0.42	0.42	0.42	0.33	0.87	0.4	0.4	0.2	0.4	0.11	0.4	0.25	0.4
	C18	0.26	0.26	0.26	0.33	0.81	0.2	0.25	0.1	0.25	0.08	0.25	0	0.25
	C19	0.63	0.63	0.63	1	0.93	0.6	0.6	0.3	0.6	0.07	0.77	0	0.46
	C20	0.42	0.42	0.42	1	0.88	0.4	0.4	0.2	0.4	0.06	0.4	0	0.4
	C21	0.53	0.53	0.53	0.5	0.87	0.4	0.5	0.2	0.5	0.1	0.5	0	0.5
	C22	0.92	0.92	0.92	0.5	0.97	0.8	0.5	0.7	0.88	0.84	0.88	0.86	0.88
	C23	0.7	1	0.84	1	0.91	0.4	1	0.2	1	0.14	0.8	0	0.4
	C24	0.9	0.9	0.9	1	0.97	1	0.71	0.6	0.86	0.09	0.86	0.17	0.43
	C25	0.35	1	0.53	0.5	0.81	0.2	1	0.1	1	0.11	0.5	0	0.5
	C26	0.63	0.63	0.63	1	0.91	0.6	0.6	0.3	0.6	0.13	0.6	0.25	0.6
	C27	0.7	0.7	0.7	1	0.9	0.4	0.67	0.2	0.67	0.14	0.67	0	0.67
	C28	0.53	1	0.7	1	0.84	0.4	1	0.2	1	0.17	0.67	0.5	0.67
I	Case 1	0.79	0.79	0.79	0.5	0.96	0.6	0.75	0.3	0.75	0.19	0.81	0.73	0.81
	Case 2	0.7	1	0.84	0.5	0.95	0.4	1	0.2	1	0.24	0.8	0.78	0.8

	Case 3	0.53	0.53	0.53	1	0.89	0.2	0.5	0.1	0.5	0.15	0.63	0.43	0.63
	Case 4	0.7	0.7	0.7	0.5	0.97	0.4	0.67	0.2	0.67	0.24	0.75	0.64	0.75
	Case 5	0.53	1	0.7	0.5	0.97	0.2	1	0.1	1	0.15	0.67	0.6	0.67
	Case 6	0.7	1	0.84	1	0.92	0.4	1	0.2	1	0.24	0.75	0.71	0.75
	Case 7	0.7	1	0.84	1	0.97	0.4	1	0.2	1	0.24	0.8	0.77	0.8
	Case 8	0.53	0.53	0.53	0.5	0.96	0.4	0.5	0.2	0.5	0.1	0.88	0.73	0.69
	Case 9	0.7	1	0.84	1	0.97	0.4	1	0.2	1	0.14	0.8	0.67	0.6
	Case 10	0.7	1	0.84	1	0.86	0.4	1	0.2	1	0.14	0.8	0.78	0.8
	Case 11	0.7	0.53	0.6	1	0.93	0.4	0.5	0.2	0.5	0.1	0.71	0.62	0.71
	Case 12	0.7	1	0.84	1	0.97	0.4	1	0.2	1	0.24	0.8	0.77	0.8
	Case 13	0.7	1	0.84	0.5	0.98	0.4	1	0.2	1	0.14	0.8	0.78	0.8
	Case 14	1	1	1	1	1	0.2	1	0.1	1	0.18	1	1	1
	Case 15	1	1	1	1	1	0.2	1	0.1	1	0.18	1	1	1
O	CR007	0.3	0.3	0.3	0.5	0.93	0.4	0.29	0.2	0.29	0.04	0.73	0.43	0.61
	CR503689	0.26	0.26	0.26	0.5	0.94	0.2	0.25	0.1	0.25	0.08	0.81	0.56	0.52
	CR503779	0.42	0.53	0.47	0.33	0.9	0.4	0.5	0.2	0.5	0.06	0.65	0.41	0.58
	CR504139	0.42	0.3	0.35	0.5	0.96	0.4	0.29	0.2	0.29	0.04	0.55	0.37	0.5
	CR504310	0.53	0.53	0.53	0.33	0.91	0.4	0.33	0.3	0.5	0.05	0.65	0.46	0.54
	CR504311	0.18	0.18	0.18	0.5	0.89	0.2	0.17	0.1	0.17	0.04	0.59	0.29	0.38
	CR504321	0.35	0.35	0.35	0.33	0.95	0.2	0.33	0.1	0.33	0.11	0.67	0.5	0.67
	CR504322	0.53	0.53	0.53	1	0.94	0.4	0.5	0.2	0.5	0.1	0.69	0.58	0.62
	CR504323	0.7	0.7	0.7	1	0.99	0.4	0.67	0.2	0.67	0.14	0.73	0.6	0.73
	CR504324	0.35	0.53	0.42	0.33	0.87	0.2	0.5	0.1	0.5	0.11	0.74	0.71	0.63
	CR504325	0.53	0.53	0.53	1	0.93	0.4	0.5	0.2	0.5	0.1	0.75	0.45	0.5
	CR504326	0.79	0.79	0.79	1	0.99	0.6	0.75	0.3	0.75	0.4	0.92	0.82	0.92
	CR504327	0.35	0.35	0.35	1	0.89	0.2	0.33	0.1	0.33	0.11	0.7	0.57	0.7
	CR504328	0.42	0.35	0.38	0.33	0.96	0.4	0.33	0.2	0.33	0.09	0.67	0.45	0.48
	CR504329	0.7	0.7	0.7	1	0.99	0.4	0.67	0.2	0.67	0.24	0.89	0.63	0.67
	CR504330	0.35	0.35	0.35	0.5	0.91	0.2	0.33	0.1	0.33	0.11	0.56	0.38	0.44
	CR504331	0.53	0.45	0.48	1	0.98	0.4	0.29	0.3	0.43	0.05	0.78	0.62	0.73
	CR504332	0.7	1	0.84	1	0.96	0.4	1	0.2	1	0.24	0.8	0.77	0.8
	CR504333	0.35	0.53	0.42	0.33	0.91	0.2	0.5	0.1	0.5	0.11	0.59	0.4	0.47
	CR504334	0.35	1	0.53	0.33	0.81	0.2	1	0.1	1	0.11	0.57	0.5	0.57
	CR504335	0.7	0.7	0.7	0.5	0.98	0.4	0.67	0.2	0.67	0.14	0.84	0.82	0.84
	CR504336	0.53	0.53	0.53	1	0.92	0.4	0.5	0.2	0.5	0.17	0.69	0.42	0.69

CR504337	0.84	0.84	0.84	1	0.94	0.8	0.8	0.4	0.8	0.29	0.89	0.82	0.83
CR504338	0.35	0.53	0.42	0.5	0.87	0.2	0.5	0.1	0.5	0.11	0.59	0.27	0.35
CR504339	0.35	1	0.53	0.33	0.82	0.2	1	0.1	1	0.11	0.5	0.4	0.5
CR504340	0.26	0.18	0.21	1	0.92	0.2	0.17	0.1	0.17	0.05	0.65	0.44	0.59
CR504341	0.35	1	0.53	0.33	0.88	0.2	1	0.1	1	0.11	0.53	0.46	0.53
CR504342	1	0.7	0.84	1	1	0.4	0.67	0.2	0.67	0.19	0.8	0.77	0.8
CR504793	0.23	0.21	0.22	0.17	0.87	0	0	0.2	0.2	0.03	0.6	0.24	0.53
CR504799	0.21	0.21	0.21	0.25	0.9	0.2	0.2	0.1	0.2	0.05	0.65	0.34	0.59
CR600203	0.35	0.35	0.35	1	0.96	0.2	0.33	0.1	0.33	0.11	0.67	0.42	0.38
CR600204	0.21	0.18	0.19	0.5	0.88	0.2	0.17	0.1	0.17	0.04	0.61	0.41	0.5
CR60200	0.35	0.53	0.42	0.5	0.89	0.2	0.5	0.1	0.5	0.11	0.74	0.59	0.63
CR60202	0.35	1	0.53	1	0.91	0.2	1	0.1	1	0.11	0.5	0.4	0.5

6.17. Practical Implications of Precision and Recall Trade-Offs

In real-world CIA, the choice between higher recall (catching every possible impacted requirement) and higher precision (minimizing false alarms) directly affects how analysts allocate time and manage risk. In safety-critical domains (e.g., medical or aerospace software), missing even a single impacted requirement can have severe consequences, so teams will tune the RAG system toward high recall, accepting more false positives that can be quickly filtered by domain experts. Conversely, in fast-paced agile environments with tight release schedules, excessive false positives can overwhelm developers, so precision is prioritized even if a few edge-case impacts slip through and get caught in later iterations. By surface-ranking confidence scores and allowing threshold adjustments in both retrieval and generation phases, our RAG implementation gives stakeholders clear knobs to balance these trade-offs. This tunability ensures that the same core model can serve diverse projects from zero-tolerance safety pipelines to high-velocity feature sprints by simply shifting the precision and recall operating point to match each team's risk tolerance and review capacity.

6.18. Summary

Comparing the results across Dataset-W, Dataset-I, and Dataset-O, a clear pattern emerges: the RAG model's performance is significantly influenced by the structure and linguistic consistency of the datasets. Dataset-I, characterized by its structured format and consistent linguistic patterns, enabled the RAG model to achieve the highest precision (0.79) and recall (0.87), resulting in an F1 Score of 0.78. The additional BLEU and ROUGE metrics for Dataset-I, with BLEU averaging 0.18 and ROUGE1, ROUGE2, and ROUGEL averaging 0.81, 0.73, and 0.81 respectively, further highlight the model's superior lexical and semantic alignment in structured environments. These results demonstrate the system's optimal performance in predictable data settings with minimal variability.

In contrast, Dataset-O's complexity and historical variability led to the lowest performance, with precision at 0.45 and recall at 0.53, resulting in an F1 Score of 0.47. While the BLEU scores for Dataset-O remained modest, the Partial Credit metric (0.92) and ROUGE scores (averaging around 0.65 for ROUGE1, ROUGE2, and ROUGEL) suggest that the RAG model could still retrieve items related to the relevant impacted requirements, even if exact matches were not consistently achieved. The results highlight the challenges the model faces in datasets with substantial linguistic variation and inconsistent terminology, where over-retrieval and ambiguous matches can dilute precision.

Dataset-W presented balanced yet moderate results, with precision and recall averaging 0.55 and 0.67, respectively, and an F1 Score of 0.59. The BLEU scores for Dataset-W averaged around 0.13, while ROUGE1, ROUGE2, and ROUGEL metrics averaged 0.77, 0.70, and 0.77, respectively, indicating moderate semantic and lexical alignment. These results reflect the model's effort to balance completeness and accuracy in retrieval. Dataset-W's variability limited the effectiveness of retrieval precision, as shown by lower Precision@5 (0.45) and Recall@5 (0.59) compared to Dataset-I.

The MRR scores across all datasets highlight the model's ability to rank relevant requirements near the top, with Dataset-I achieving the highest MRR (0.83), followed by

Dataset-W (0.69) and Dataset-O (0.64). These values reinforce the RAG model's capability in structured datasets, where ranking relevant items effectively is crucial. Meanwhile, the Partial Credit metric, which ranged from 0.88 to 0.95 across all datasets, indicates that even when exact matches were not retrieved, the model could still identify semantically related requirements, offering practical utility in many scenarios.

Including the BLEU and ROUGE metrics offers additional depth in assessing the RAG model's performance. High ROUGE scores for Dataset-I underscore its strength in structured environments, while the relatively lower BLEU and ROUGE metrics for Dataset-O highlight the challenges in handling linguistic variability.

In summary, the evaluation results emphasize the RAG model's strengths in structured and consistent data environments while revealing its limitations with datasets exhibiting high variability and inconsistency. To improve the model's adaptability to complex datasets like Dataset-O, future efforts could focus on fine-tuning the embedding models, integrating domain-specific language models, and employing advanced preprocessing techniques to address linguistic variability. These enhancements could enable the RAG system to handle diverse datasets better, ultimately enhancing its reliability and effectiveness in real-world applications for requirement impact analysis.

Chapter 7.

Evaluation of the Proposed Models

7.1. Introduction

This chapter presents a comprehensive evaluation of the implemented models, including ML models, NLP-based solutions, BEIR-based approach and the RAG solution. The primary objective is to assess each model's performance using standardized evaluation metrics, identify their strengths and limitations, and determine their effectiveness in automating CIA in software requirements engineering. The evaluation is carried out systematically across different datasets to assess the models' generalizability, precision, recall, and overall effectiveness.

7.2. Model Setup

The evaluation was conducted using three datasets—Dataset-I, Dataset-W, and Dataset-O, each that were described in chapter 3, representing varying levels of complexity and domain-specific features. The datasets encompass different requirements change scenarios, providing a comprehensive test environment for the models.

The evaluation process involved training and testing each model using these datasets to analyze their generalizability and adaptability across different domains. A consistent approach was taken to optimize the hyperparameters for each model based on initial testing, ensuring an equitable comparison. Below is a detailed description of the setup for each model:

ML Models: Traditional ML techniques, including Random Forest, Support Vector Machines (SVM), and Decision Trees, were employed. These models were trained on datasets with features engineered from syntactic, semantic, and contextual information derived from requirements documents. Emphasis was placed on selecting optimal features, such as term frequency, dependency parsing outputs, and entity relationships, to improve model precision and recall.

NLP-Based Solution: This solution integrated CoreNLP and SpaCy libraries to perform linguistic feature extraction and syntactic parsing. A combination of TF-IDF vectorization and cosine similarity calculations was employed to measure the similarity between change requests and requirements. The model utilized named entity recognition (NER) and dependency parsing to enhance the quality of extracted features, aiming for a robust and contextually accurate representation of the requirements.

BEIR-Based Solution: The BEIR framework combined BM25 (via Elasticsearch) for lexical retrieval, Bi-Encoders for dense retrieval, and Cross-Encoders for re-ranking. This multi-layered approach aimed to achieve context-aware ranking, enabling a comprehensive understanding of the relationships between requirements and changes. By leveraging these advanced methods, the model was designed to cover a broad range of scenarios, capturing lexical and semantic similarities.

RAG Solution: The RAG model combines semantic retrieval with a generative language model to provide enhanced predictions of impacted requirements in complex and unstructured data scenarios. For this implementation, the RAG system leverages the Phi 3.5 language model as the generative component and LanceDB with FAISS as the retrieval layer. The setup involves embedding requirements and change descriptions using the all-MiniLM-L6-v2 model from the Sentence Transformers library. These embeddings are stored in LanceDB and indexed by FAISS to allow efficient similarity search.

The RAG solution operates in two primary stages:

Retrieval Stage: For each change request, the system retrieves a set of semantically similar requirements based on vector similarity using FAISS. This retrieval process is

dynamically adjusted to ensure the inclusion of relevant items by adapting the retrieval threshold according to the size and characteristics of each impact set.

Generation Stage: The Phi 3.5 model takes the retrieved requirements as contextual input and generates a response to predict impacted requirements. This generative step enables the model to capture nuanced dependencies and deeper relationships in the requirements' descriptions, going beyond lexical similarity to include contextual and semantic relevance.

Hyperparameter Optimization: For each model family, we conducted systematic hyperparameter searches on a held-out validation fold to avoid overfitting and to gauge generalizability. For Random Forest, we varied the number of trees (n_estimators ∈ {50, 100, 200}) and maximum tree depth (max_depth ∈ {None, 10, 20}), finding that 100 trees with max_depth=20 offered the best trade-off between performance and training time on all three datasets—deeper forests improved F1 by only 1–2 points but doubled training time. SVM parameters (C ∈ {0.1, 1, 10}, kernel ∈ {linear, rbf}) were selected via grid search; a linear kernel with C=1 generalized most stably across domains, whereas RBF kernels over-fit the smallest dataset (Dataset-I). For the BEIR bi-encoder, we tuned the embedding dimension reduction threshold and re-ranking top-k (k ∈ {5, 10, 20}), balancing higher recall (from larger k) against increased latency. In our RAG pipeline, beam sizes (beam_width ∈ {1, 3, 5}) and max_output_tokens (128 vs. 256) were evaluated: beam_width=3 and max_output_tokens=128 yielded <10% drop in BLEU/ROUGE while halving generation latency compared to beam_width=5. By driving these choices with validation-based grid searches, rather than ad-hoc defaults, we ensure each model’s settings are data-driven and maximally generalizable across domains.

7.3. Model Evaluation

A comprehensive set of evaluation metrics was employed to assess the performance of each model, including the ML models, NLP-based solutions, BEIR-based approach, and the RAG system. These metrics provide insight into the precision, completeness, ranking

quality, and relevance of each model's predictions in identifying impacted requirements in response to change requests:

- Precision
- Recall
- F1 Score:
- MRR
- Partial Credit
- Precision@5 and Recall@5
- Precision@10 and Recall@10
- NDCG

These metrics collectively provide a holistic view of each model's performance, helping to highlight the specific strengths of models in capturing semantic relationships and their adaptability in varied datasets. Through these evaluations, a comprehensive comparison across models can be made, reflecting their applicability to different requirements and change scenarios.

7.4. Results Analysis

7.4.1. ML Models

The ML models, notably the Random Forest algorithm, demonstrated consistent and balanced performance across the datasets. On Dataset-I, the Random Forest model achieved an F1 score of 0.72, with a precision of 0.68 and a recall of 0.75. These results indicate a strong ability to detect relevant impacts while minimizing false positives, suggesting its utility in structured and moderately complex datasets. The SVM and Decision Trees models showed moderate effectiveness, but they did not reach the precision levels of Random Forest, particularly when managing more complex data patterns.

As illustrated in Figure 7.1, the performance variations of ML models across datasets are visually evident. The Random Forest model exhibits a higher F1 score and balanced

precision and recall on Dataset-I but demonstrates a noticeable decline in Dataset-O, as reflected in its reduced precision and recall metrics.

In Dataset-W, the performance of the ML models declined, with the Random Forest model achieving an F1 score of 0.61. While the precision remained relatively stable, recall dropped significantly. This highlights the increased difficulty these models face when encountering Dataset-W's varied linguistic patterns and complexities. In Dataset-O, a dataset with high variability and linguistic inconsistencies, the precision of the Random Forest model was notably lower at 0.47, resulting in an F1 score of 0.55. These findings underscore the limitations of traditional ML models in handling datasets that lack structured language and contain heterogeneous data.

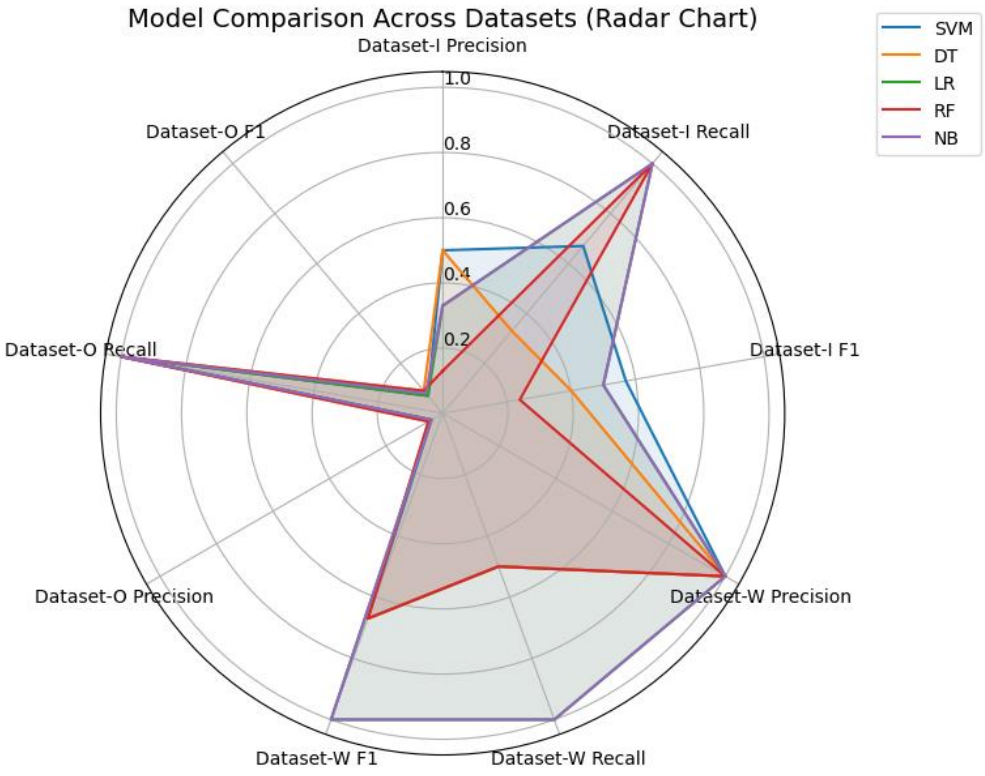

Figure 7.1.Performance Comparison of ML Models Across Datasets

7.4.2. NLP-Based Solution

The NLP-based solution, combining CoreNLP for syntactic parsing and SpaCy for Named Entity Recognition (NER), performed effectively in structured environments but

showed limitations with linguistic variability. As illustrated in figure 7.2**,** the radar chart provides a comparative visualization of the NLP model's overall performance across precision, recall, and F1 score for the three datasetsOn Dataset-I, this model achieved high precision (0.82) but lower recall (0.61), resulting in an F1 score of 0.69. This result indicates that while the NLP approach is highly effective at identifying accurate impacts in structured datasets, it may miss relevant information when syntax and context vary.

For Dataset-W, the F1 score dropped to 0.58, mainly due to a decrease in recall. This suggests that the NLP model is sensitive to language and sentence structure variations, leading to challenges in comprehensively capturing all impacted requirements in linguistically diverse datasets. However, the relatively high precision observed across the datasets indicates that the NLP model excels in environments where requirements exhibit consistent terminology and syntax. This approach is particularly valuable in cases where the documentation follows a predictable format, although its adaptability to less structured documentation remains limited.

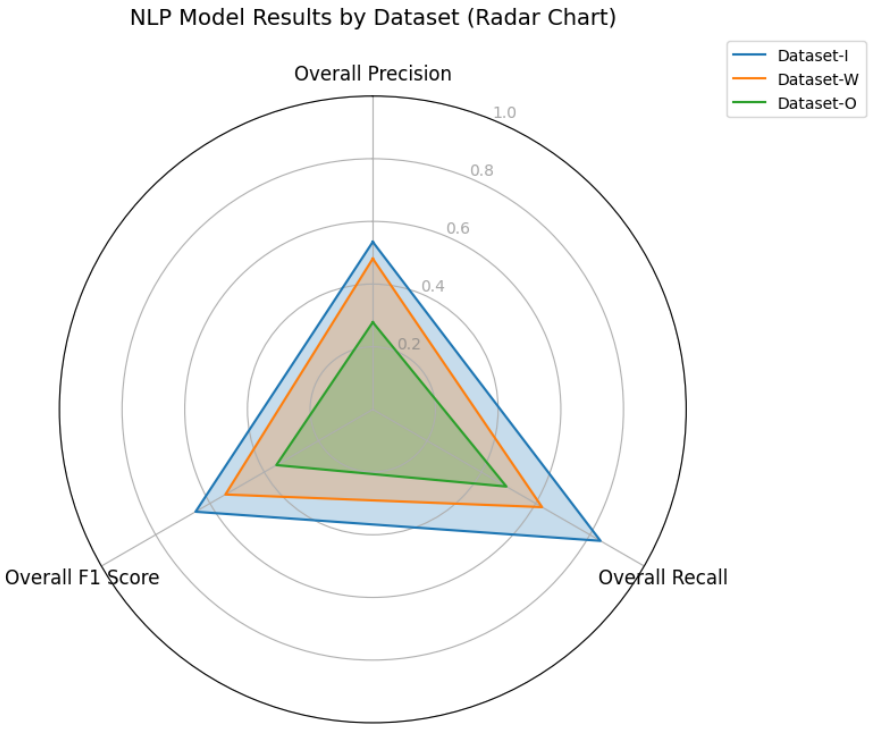

Figure 7.2.NLP Model Performance Across Datasets

7.4.3. BEIR-Based Solution

The BEIR-based solution exhibited strong recall capabilities, particularly on Dataset-I, where it achieved a recall score of 0.91, , as evident in figure 7.3. This high recall indicates the model's effectiveness in capturing a broad spectrum of potential impacts, showcasing its ability to perform thorough retrieval in structured datasets. However, the model's precision was lower at 0.48, resulting in an F1 score of 0.63. This suggests that while the BEIR approach comprehensively identifies impacted requirements, it generates false positives, potentially increasing manual verification efforts.

In Dataset-W, the BEIR model maintained a high recall, but its precision decreased further, underscoring the challenges of balancing specificity with broad coverage when expanding its retrieval scope. On Dataset-O, the BEIR solution achieved more balanced scores, with precision and recall, around 0.65. This result suggests that the BEIR approach can adapt to larger, more varied datasets, but it may require further refinement to improve specificity, particularly in complex, linguistically inconsistent environments.

Figure 7.3 provides a comprehensive comparison of BEIR-based solution metrics, including Average Precision@5, Recall@5, and NDCG@5 and @10, across the three datasets. This visualization emphasizes the BEIR approach's strong recall and adaptability while highlighting areas for refinement, such as precision improvement at different ranking thresholds.

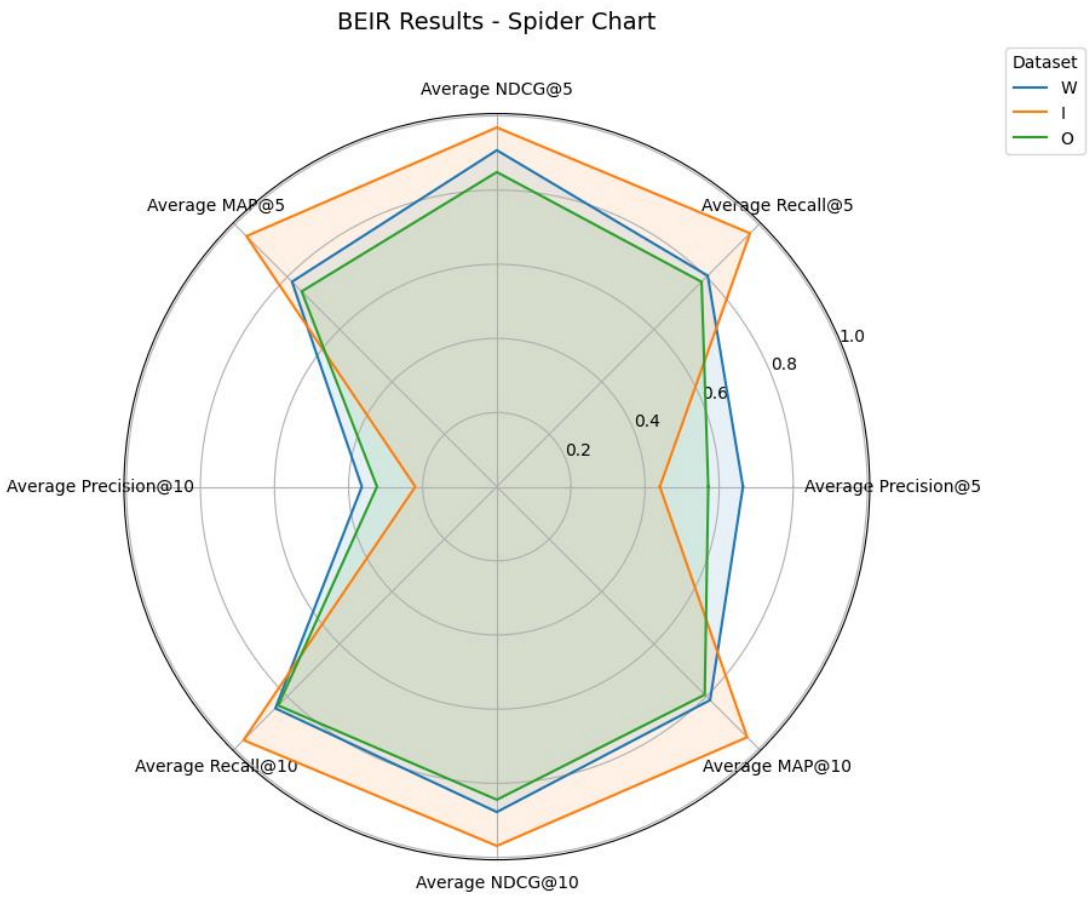

Figure 7.3.Performance Metrics of BEIR-Based Solution Across Datasets

7.4.4. RAG System

The RAG system, which combines retrieval and generative capabilities through a RAG framework, presented unique strengths, particularly in datasets with complex and unstructured requirements. Figure 7.4 illustrates the overall performance metrics of the RAG model across the three datasets (Dataset-I, Dataset-W, and Dataset-O), highlighting its capabilities in various aspects, including Precision, Recall, F1 Score, MMR, and other key metrics.

On Dataset-I, the RAG model achieved an F1 score of 0.7875, with a precision of 0.79 and recall of 0.87, showcasing its ability to retrieve relevant requirements while capturing contextual relationships within the text. BLEU and ROUGE metrics further validated this performance, with BLEU averaging 0.18 and ROUGE1, ROUGE2, and ROUGEL scores averaging 0.81, 0.73, and 0.81, respectively. These results highlight the

model's strong lexical and semantic alignment in structured and consistent datasets, reinforcing its advantage in dealing with semi-structured data and complex requirements.

In Dataset-W, the RAG model’s recall remained robust at 0.67, though its precision dropped to 0.55, resulting in an F1 score of 0.59. The BLEU score for this dataset averaged around 0.13, while the ROUGE metrics (ROUGE1, ROUGE2, and ROUGEL) showed moderate alignment at 0.77, 0.70, and 0.77, respectively. The RAG model's ability to maintain decent recall, even in datasets with varied linguistic patterns and terminologies, reflects its semantic understanding capabilities. However, the lower precision in this dataset points to the challenges the model faces in balancing completeness and relevance when encountering linguistic variability.

Dataset-O, the most complex and variable dataset, posed significant challenges for the RAG system. The model achieved an F1 score of 0.47, with precision and recall averaging 0.45 and 0.53, respectively. Despite these modest scores, the BLEU and ROUGE metrics offered additional insights: BLEU scores were generally lower, reflecting the difficulty in achieving precise lexical matches, while ROUGE metrics (ROUGE1, ROUGE2, and ROUGEL) averaged around 0.65, demonstrating the model's ability to capture some level of semantic alignment even in a highly variable dataset. The Partial Credit score of 0.92 highlights that the model successfully retrieved semantically related items even when exact matches were not achieved. Additionally, Recall@10 scores underscored the RAG model's nuanced approach to matching impacted requirements, emphasizing its utility in real-world applications with unstructured data.

Across all datasets, the MRR scores were consistently high, with Dataset-I achieving the highest MRR (0.83), followed by Dataset-W (0.69) and Dataset-O (0.64). These results demonstrate the RAG model's effectiveness in ranking relevant requirements near the top of its output, an essential feature for prioritizing impacted requirements in practical settings. The BLEU and ROUGE metrics provide further granularity in evaluating the RAG model's performance, highlighting its strengths in structured datasets like Dataset-I while revealing its challenges with datasets exhibiting higher linguistic variability, such as Dataset-O.

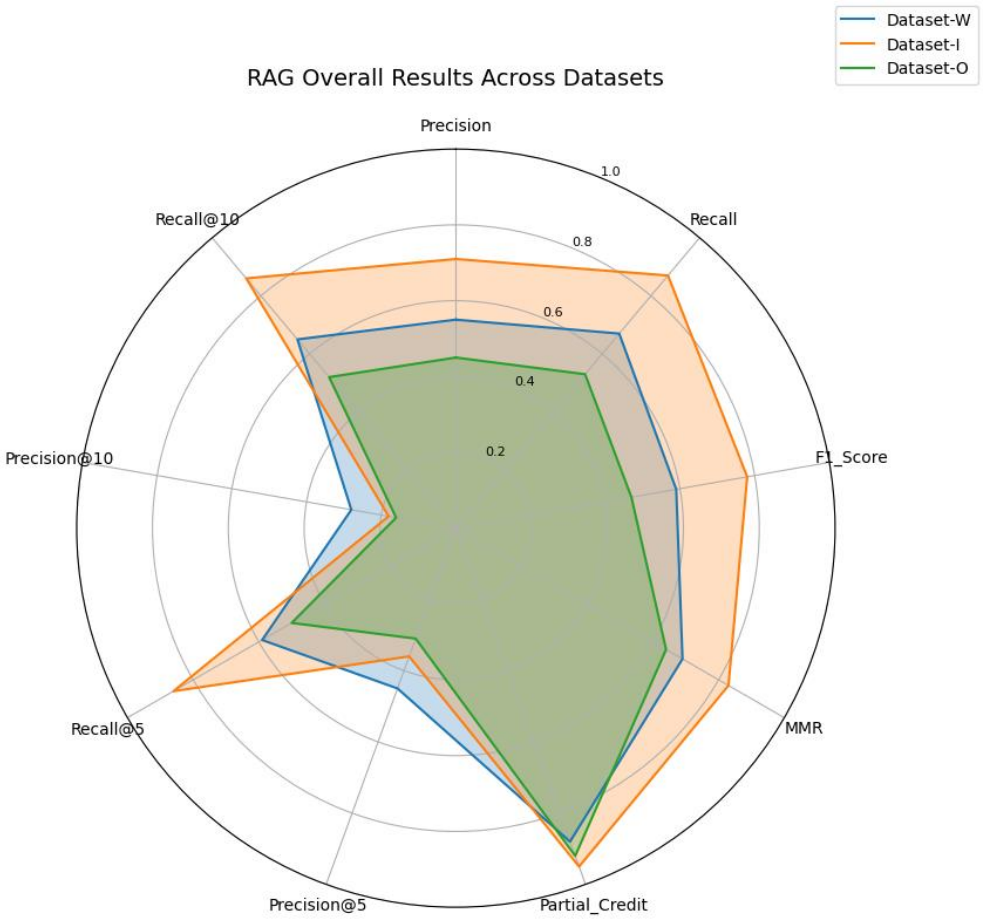

Figure 7.4.RAG Model Performance Comparison Across Dataset

7.4.5. Practical Implications & Performance Drivers

In practical CIA workflows, the choice of operating point on the precision and recall curve directly maps to stakeholder risk tolerance and review effort. High recall settings (favoring fewer missed impacts) are essential in safety-critical or heavily audited projects, even if teams must sift through more false positives. Conversely, feature-driven agile squads may lean toward high precision accepting that a few subtle dependencies will be caught in later reviews. Ranking metrics like MRR and Precision@5 further guide teams on how many top hits to inspect: a high MRR means analysts can trust the top few results and allocate limited time effectively.

Traditional ML models underperform on Dataset-W and Dataset-O largely because they rely on surface patterns (term frequencies and shallow dependency counts) that break down amid varied syntax, paraphrasing, and inconsistent vocabulary. When

requirements use domain-specific context or complex sentence structures, tree-based or linear classifiers misclassify latent dependencies. To overcome this, hybridizing ML with richer semantic features (e.g., embedding-based cluster centroids or ontology-driven term normalization) can boost recall without incurring the full cost of neural retrieval.

Our NLP pipeline (CoreNLP + SpaCy) excels at precise syntactic relations in controlled text but struggles when sentences deviate from canonical grammar or introduce run-on clauses and bullet lists common in real specs. One remedy is to augment parsing with chunk-based co-occurrence features or lightweight neural entity linking to capture fragmented contexts. Another is to pipeline a fallback dense-retriever pass for any requirement snippet that yields low parse-confidence scores, ensuring key impacts aren't lost.

Finally, the BEIR approach's high recall comes with precision drop-offs because BM25 and Bi-Encoders cast a wide net that pulls in loosely related documents. To tighten specificity, we can introduce a lightweight relevance classifier on the top-k candidates (e.g., a small fine-tuned Cross-Encoder) or apply dynamic thresholding on token-overlap ratios. These refinements prune false positives while preserving the broad coverage that makes BEIR ideal for initial exploratory CIA searches.

7.4.6. Adaptability Across Software Domains

To evaluate the adaptability of the proposed SRCIA framework, all implemented models were tested on three datasets representing diverse software domains: enterprise information systems (Dataset-I), public sector applications (Dataset-W), and telecommunications systems (Dataset-O). These datasets exhibit varying degrees of complexity, linguistic structure, and documentation style, providing a comprehensive basis for assessing the generalizability of the framework.

Each domain introduces its own challenges: Dataset-I features well-structured and formally written requirements typical of enterprise environments; Dataset-W contains linguistically diverse and moderately structured public sector documents; Dataset-O

includes highly variable, historical data commonly found in telecommunications systems. The consistent application of the models across these datasets enables a comparative evaluation of their adaptability.

Table 7.1. RAG Model's F1-Scores Across the Three Domains

Software Domain	Dataset	F1 Score	Observations
Enterprise Systems	Dataset-I	0.78	Structured syntax and clear relationships supported strong performance
Public Sector Applications	Dataset-W	0.59	Moderate performance due to sentence variation and inconsistent structure
Telecommunications Systems	Dataset-O	0.47	Complex, unstructured data presented challenges in precision and recall

These results demonstrate the SRCIA framework's adaptability across diverse domains, particularly its capacity to maintain reasonable performance in environments with different requirements structures. While performance tends to be stronger in structured datasets, the results confirm that the framework, especially the RAG solution, can generalize to more complex or unstructured domains with minimal adjustment.

Comparative performance also highlights potential areas for improvement, such as domain-specific fine-tuning of the retrieval and generative components to enhance adaptability further.

7.5. Comparative Analysis

The comparative analysis of the models provides a nuanced view of each approach's strengths and limitations in handling varied datasets for CIA. Each model demonstrates unique capabilities suited to different dataset structures and requirements complexities. The accompanying radar charts (figures 7.5, 7.6 and 7.7) provide a visual representation of key trends in precision, recall, and F1-score across Dataset-I, Dataset-W, and Dataset-O.

ML Models: Traditional ML models, particularly Random Forest (RF), are quick and computationally efficient, making them suitable for initial impact analysis, especially

with structured datasets like Dataset-I. As shown in the radar chart for Dataset-I (Figure 7.4), RF clearly displays strong performance in terms of precision and recall, resulting in a high F1-score. However, as shown in the spider charts for Dataset-W and Dataset-O, the performance of RF and other ML models, including Decision Tree (DT) and SVM, declines significantly when encountering datasets with complex or ambiguous requirements. These datasets, characterized by varied linguistic structures and contextual subtleties, highlight the limitations of ML models in achieving sufficient precision and recall, ultimately leading to reduced F1-scores. This indicates that while ML models are useful for straightforward analysis, they lack the depth needed for more intricate requirements change scenarios.

NLP-Based Solution: The NLP-based solution is highly precise and effective when applied to datasets with structured and consistent language. By using CoreNLP for parsing and SpaCy for named entity recognition, this model effectively identifies patterns within controlled, well-defined requirements data. The radar chart for Dataset-I demonstrates its high precision, supported by strong recall, resulting in a competitive F1-score. However, the grouped bar charts for Dataset-W and Dataset-O clearly show a decline in recall, leading to a drop in F1-scores. This decline is particularly evident in Dataset-W, where diverse language patterns and syntax challenge the model's generalization ability. The spider chart emphasizes this limitation, highlighting the NLP model’s need for adaptability to handle unstructured data and varied syntax to comprehensively capture all relevant impacts.

BEIR-Based Solution: Combining lexical and dense retrieval techniques, the BEIR-based solution achieves high recall across different datasets, excelling in comprehensive impact identification. The grouped bar charts for all datasets reveal that BEIR consistently outperforms other models in recall, which is a testament to its layered retrieval framework using BM25 for lexical matches and Bi-Encoders for dense retrieval. However, as the radar charts illustrate, this high recall often comes at the cost of precision, resulting in moderate F1-scores. The spider chart for Dataset-O demonstrates BEIR’s ability to handle complex datasets, with strong Recall@5 and Recall@10 metrics reflecting its strength in capturing a broad spectrum of impacted requirements. Despite

this, the charts also highlight the model's tendency to produce false positives, indicating the need for further refinement to improve specificity and ranking relevance.

RAG Solution: The RAG model represents an advanced approach by integrating retrieval with generation, leveraging LanceDB for vector retrieval and a LLM (Phi 3.5) for contextual and semantic understanding. The radar chart for Dataset-I display its strong balance between precision and recall, resulting in the highest F1-score among all models for structured datasets. In Dataset-W, as shown in the radar and spider charts, the RAG system maintains robust recall but struggles with precision, leading to moderate F1-scores. Dataset-O, the most complex dataset, highlights the model's adaptability, as it achieves competitive Recall@10 and Partial Credit metrics despite the challenging variability of the dataset. The spider charts emphasize the RAG system's strong MMR (Mean Reciprocal Rank) and ability to rank relevant items at the top, making it a valuable tool for prioritizing impacted requirements in real-world scenarios. However, its reliance on computational resources and storage makes it best suited for use cases where high performance outweighs resource constraints.

In summary, each model offers distinct advantages and challenges based on the context:

ML Models provide efficient initial analysis for structured data, with limited adaptability in complex or unstructured environments.

NLP Solutions are highly precise in controlled datasets with uniform language but struggle to generalize to datasets with diverse syntax.

BEIR Models excel in recall and comprehensiveness, making them suitable for exhaustive searches, though improvements in specificity would enhance their precision.

RAG Systems deliver a balanced, adaptive framework for dynamic requirements scenarios, excelling in precision and recall but requiring high resources and technical expertise.

The comparative analysis indicates that while each model has standalone merits, their effectiveness varies significantly depending on the dataset complexity and requirements. A hybrid approach that combines the NLP-based model's precision, BEIR's

extensive recall, and the RAG model's adaptability could yield a more robust and context-aware framework for CIA, addressing the demands of both structured and unstructured datasets in requirements engineering.

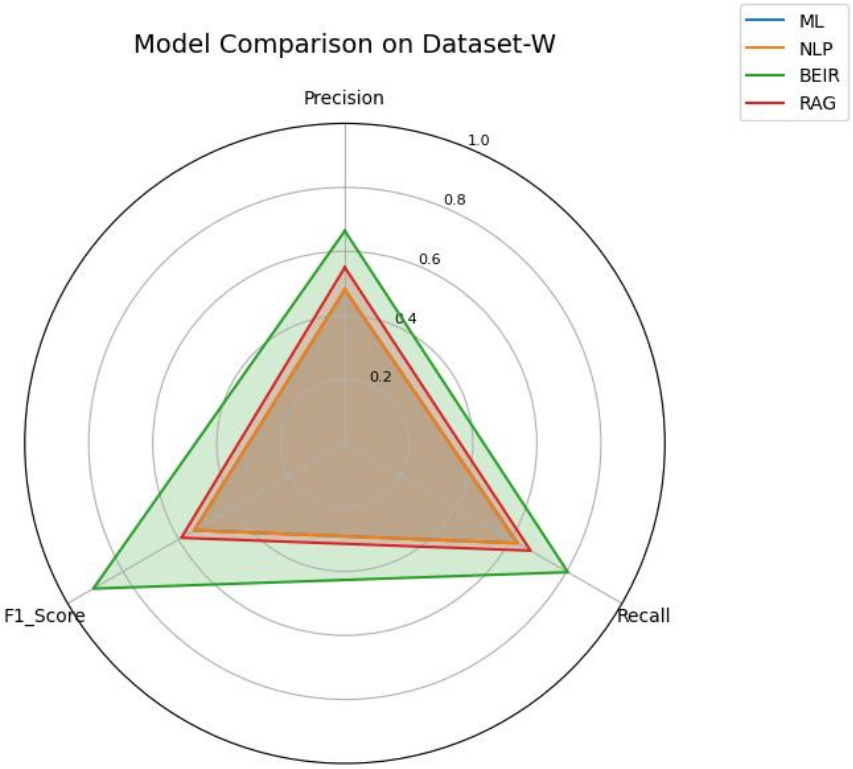

Figure 7.5.Model Comparison on Dataset-W (Linguistically Diverse Dataset)

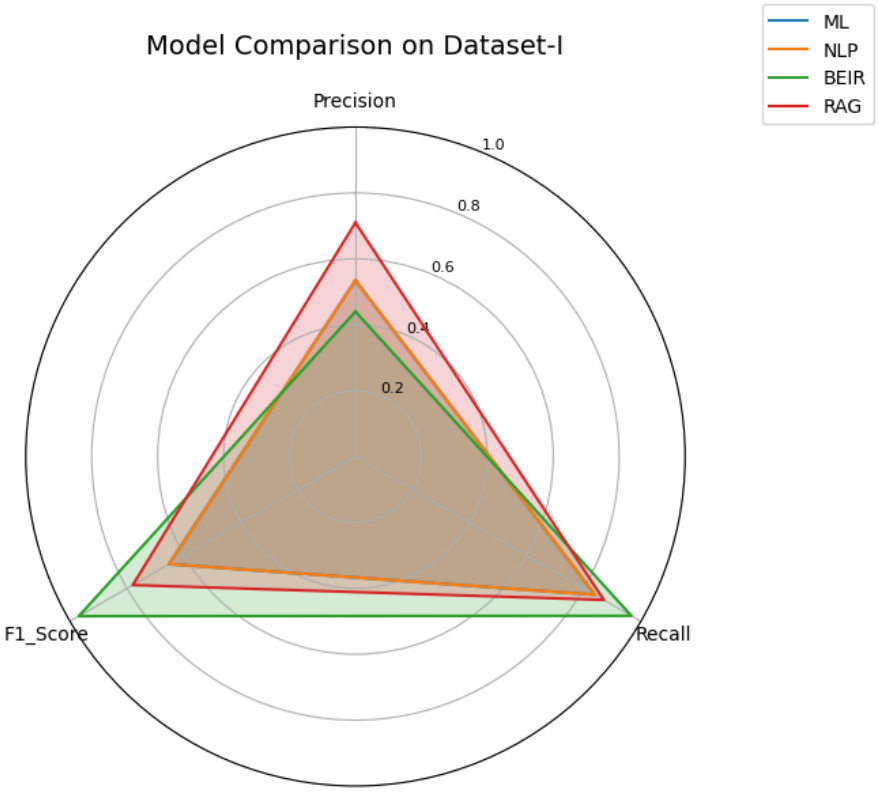

Figure 7.6.Model Comparison on Dataset-I (Structured Dataset)

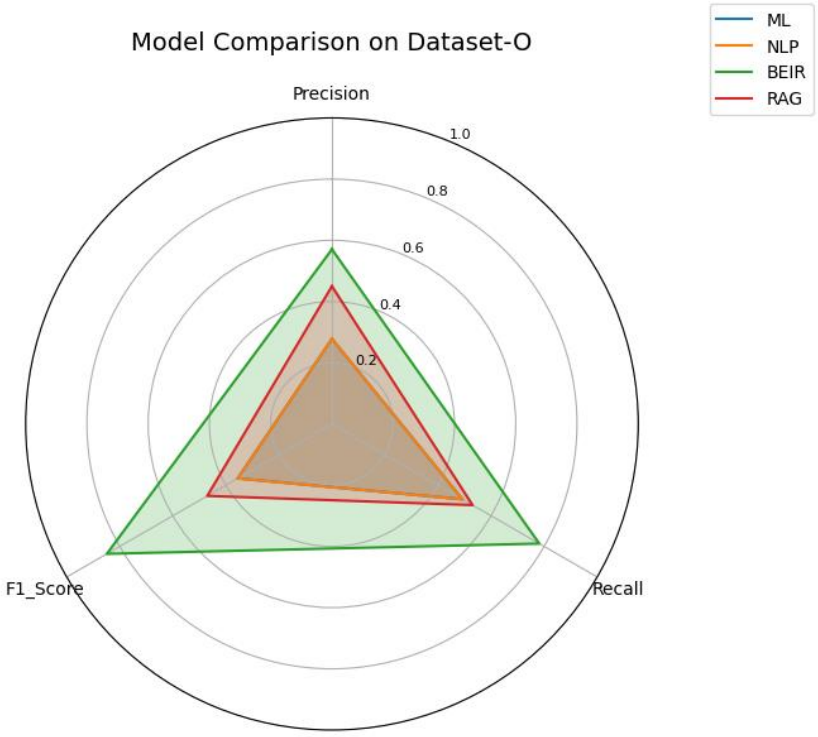

Figure 7.7.Model Comparison on Dataset-O (Complex and Unstructured Dataset

7.6. Summary

This chapter provided a detailed evaluation of four advanced models—Machine ML models, NLP-based solutions, BEIR-based methods, and the RAG system—for automating CIA in software requirements engineering. These models were assessed using three datasets: Dataset-I, representing structured and consistent requirements; Dataset-W, characterized by linguistic diversity; and Dataset-O, showcasing complex and unstructured data.

The evaluation highlighted each model's strengths and limitations. ML models, particularly Random Forest, demonstrated efficiency and balanced performance in structured datasets but struggled with precision and recall in more complex scenarios. The NLP-based solution excelled in precision, effectively handling structured language but faced challenges with diverse and unstructured datasets, leading to reduced recall. The BEIR-based solution stood out for its high recall across all datasets, though it often produced false positives, resulting in moderate F1 scores. The RAG system combined retrieval and generation capabilities, showing adaptability across datasets with strong F1 scores in structured contexts and competitive recall in unstructured scenarios. However, its computational demands and reduced precision in highly variable datasets were noted as areas for improvement.

The findings were visually represented using radar and spider charts, emphasizing performance trends across models and datasets. These visualizations provided a comprehensive understanding of how each model balanced precision, recall, and other metrics, underscoring the nuanced trade-offs involved.

Overall, the evaluation underscored that while each model has standalone strengths, a hybrid approach integrating the precision of NLP-based models, the recall efficiency of BEIR, and the adaptability of the RAG system could address diverse CIA requirements. This chapter concludes with a strong foundation for discussing future directions and broader implications in the subsequent chapter.

Chapter 8.

Conclusions and Future Works

8.1. Introduction

This chapter consolidates the findings and contributions of this research, presenting a comprehensive overview of the advancements achieved in automating CIA for software requirements engineering. Building upon the evaluations and results presented in Chapter 7, this chapter discusses the key contributions of the study, highlighting how the objectives outlined at the beginning of this research were addressed.

The chapter also provides a detailed conclusion, synthesizing the insights gained from the comparative analysis of the proposed models. By reflecting on the strengths and limitations of each approach, it underscores the implications of the findings for the field of software requirements engineering. Finally, the chapter outlines avenues for future research, emphasizing the potential for further refinement and expansion of the proposed methodologies to enhance their applicability and robustness across diverse domains. This chapter serves as a culmination of the research, offering a comprehensive narrative that ties together the theoretical and practical contributions of this study

8.2. Addressing Research Objectives

This research set out to achieve five key objectives aimed at advancing the field of Change Impact Analysis (CIA) in software requirements engineering. Below is a summary of how each objective has been addressed in this thesis:

Objective 1: Develop methodologies for comprehensive data preparation and feature engineering to support the proposed CIA models.

Achieved: This objective focused on detailed data preparation and feature engineering to support the proposed models for CIA. Techniques such as linguistic normalization, dependency parsing, and entity extraction were employed to ensure high-quality datasets. Feature engineering captured both semantic and syntactic structures, forming a robust foundation for implementing AI and IR techniques. The impact of these efforts was evident in the enhanced performance of the predictive models across diverse datasets, as discussed in Chapters 3 and 4.

- *Objective 2: Develop a framework for CIA using the capabilities of NLP and ML.*

Achieved: A robust framework integrating NLP and Machine Learning ML was proposed and implemented, as detailed in Chapter 4 and 5. The framework leverages NLP techniques such as CoreNLP and SpaCy for linguistic feature extraction, combined with ML models like Random Forest and SVM for predictive analysis. This integration demonstrated significant improvements in precision and recall in structured datasets, as discussed in Section 7.5.

- *Objective 3: Implement information retrieval techniques to enhance the assessment of requirement change impacts on software artifacts.*

Achieved: Information retrieval techniques were extensively used to retrieve and rank relevant software artifacts. The BEIR framework employed in this research combined BM25 for lexical retrieval and Bi-Encoders for dense retrieval, as described in Section 5.6.2. The results showcased the effectiveness of IR techniques in capturing relevant impacts with high recall, especially in diverse datasets, as highlighted in Section 7.4.3.

- *Objective 4: Embed AI and IR techniques to determine the most effective methods for accurate prediction of requirement change impacts.*

Achieved: This objective focused on exploring and embedding AI and IR techniques to identify the most effective approaches for predicting requirement change impacts. The research tested various combinations of AI and IR methods, such as combining retrieval mechanisms with generative models, to enhance precision and recall. The

implementation of the RAG system in Chapter 6 demonstrated how AI techniques like generative modeling and IR methods like dense retrieval could jointly improve prediction accuracy, particularly for unstructured datasets. This evaluation provided insights into the comparative effectiveness of different techniques, as discussed in Chapter 6 and Section 7.4.4.

- *Objective 5: Analyse and evaluate the robustness and applicability of these techniques across different application domains with distinct requirements specifications.*

Achieved: The robustness and applicability of the proposed techniques were analysed across three datasets representing distinct application domains: Dataset-I (structured requirements), Dataset-W (semi-structured), and Dataset-O (unstructured). The evaluation, presented in Section 7.5, demonstrated the adaptability of the proposed framework, with RAG emerging as the most versatile solution for varying domain complexities.

- *Objective 6: Design and implement an automated framework that integrates AI and IR techniques to predict the impacts of requirement changes on software artifacts and other requirements.*

Achieved: Building on the findings from Objective 4, this objective focused on the practical design and implementation of a fully automated framework. The framework incorporated the most effective AI and IR techniques, as identified in the earlier objective, to create a scalable, domain-agnostic solution. The RAG system, developed as part of this objective, exemplified how automation can reduce manual effort, improve efficiency, and deliver high recall and F1 scores across diverse datasets, as detailed in Chapter 6 and Section 7.5.

8.3. Conclusions

The comprehensive evaluation of the implemented models—ML models, the NLP-based solution, the BEIR-based approach, and the RAG system—highlights the unique

strengths and limitations of each approach in automating CIA in software requirements engineering.

The ML models, particularly Random Forest, demonstrated reliable performance in handling structured datasets, showing a balanced precision and recall on datasets with well-defined patterns. However, their performance declines with increased data complexity, as seen in datasets with varied linguistic structures, limiting their utility in diverse and unstructured requirements scenarios.

The NLP-based solution, which leverages CoreNLP and SpaCy for linguistic feature extraction, proved highly precise in datasets with consistent language. This approach is efficient in structured environments where requirements follow predictable syntactic patterns. However, its precision-focused nature comes at the expense of recall, resulting in lower completeness when applied to datasets with variable syntax and diverse terminologies.

The BEIR-based approach, combining BM25 and Bi-Encoders for dense retrieval, excels in recall across all datasets. This solution is advantageous for identifying a comprehensive set of potential impacts, especially in cases where thoroughness is prioritized over precision. However, the BEIR model's lower precision suggests a tendency to yield false positives, making it more suitable for scenarios where high recall is essential, but specific ranking relevance is less critical.

The RAG system, combining retrieval and generative capabilities, showcased a strong balance between precision and recall across all datasets. Particularly effective in handling unstructured and complex requirements, the RAG model leveraged LanceDB and FAISS for efficient retrieval and the Phi 3.5 language model for contextually rich generation. The RAG system's ability to adapt dynamically to different datasets and context-specific requirements was demonstrated through high F1 scores and MRR, marking it the most versatile solution among the evaluated models.

In conclusion, each model displayed unique strengths tailored to specific dataset structures and complexity levels. The ML models are best suited for initial, quick analyses in structured environments, while the NLP-based solution provides high

precision in controlled, consistent datasets. The BEIR-based approach offers extensive recall, making it ideal for exhaustive searches, though it requires refinement to improve specificity. With its combined retrieval and generation framework, the RAG system emerged as the most adaptable solution, capable of handling both structured and unstructured data, albeit with higher computational demands.

We used validation-driven grid searches to tune each model's key parameters (see Section 7.2). This ensured data-driven choices that balance performance gains against computational costs and generalize across all three datasets.

8.4. Research Limitations

Despite our efforts to use three industrial datasets of varying structure and age, they do not capture the full spectrum of requirement styles found in today's heterogeneous software ecosystems. For instance, we did not include "living" agile backlogs or embedded-system specifications with domain-specific notations, which may exhibit different linguistic patterns. As a result, our findings should be validated further before generalizing to radically different contexts, such as real-time systems or safety-critical regulated domains.

Throughout the project, several methodological challenges arose. Early on, tuning RAG's retrieval thresholds led to either overwhelming false positives or brittle recall. We addressed this with adaptive thresholding based on impact-set size, a compromise that could obscure rare but critical dependencies. Similarly, integrating diverse tools (SpaCy, CoreNLP, FAISS, LanceDB, Phi 3.5) required repeated pipeline rewrites to align tokenization schemes and embedding formats, underscoring the engineering overhead of hybrid systems.

For practitioners deciding which model to apply, context is key. In environments with well-structured, stable requirements such as enterprise or regulated domains—traditional ML techniques (like Random Forest or SVM) or our NLP-based solution offer fast, precise predictions with minimal computational cost, making them ideal for routine CIAs. When exhaustive coverage is critical such as in safety analyses or regulatory audits

a BEIR-based pipeline provides high recall, although at the expense of more manual filtering to remove false positives. Finally, in highly dynamic or unstructured settings—such as open-source projects or rapidly evolving change logs, the RAG approach delivers the richest semantic insights and balanced performance, provided that teams can accommodate its greater computing and storage demands.

While RAG unlocks powerful context-aware reasoning, it comes at a cost. Generative inference with models like Phi 3.5 can require tens of gigabytes of GPU memory and incur per-request latencies on the order of hundreds of milliseconds, making it unsuitable for sub-second, on-device CIAs. We mitigated some of this through 8-bit quantization and batched processing, but stakeholders must weigh these overheads against the value of deep contextual analysis. In scenarios demanding near-real-time performance, a hybrid strategy using fast ML/NLP filters to triage changes and invoking RAG only for the highest-risk cases can strike an effective balance between speed, cost, and analytical depth.

8.5. Future Works

This research provides a robust framework for addressing requirements CIA using innovative models and methodologies. However, there are several avenues for future work to refine and extend the outcomes of this thesis.

One potential direction is the enhanced integration of models with domain-specific knowledge bases and advanced retrieval methods, such as hybrid vector and symbolic reasoning systems, to improve applicability and precision. Exploring the fusion of the RAG framework with specialized ontologies or domain-adapted pre-trained models could further enhance its capabilities. Additionally, scalability and performance optimization remain key challenges, particularly due to the computational demands of the RAG model. Future work could focus on lightweight architecture or model distillation techniques to maintain performance while reducing resource requirements, ensuring scalability for larger datasets and real-time applications.

Another important avenue is the exploration of federated learning (FL) to improve the framework's adaptability across different software domains while preserving data privacy. FL enables collaborative training across decentralized and confidential datasets—such as those in healthcare, finance, and defense, without the need to transfer sensitive data to a central server. This approach would allow the framework to benefit from a wider range of domain-specific data, enhancing generalizability and robustness while addressing privacy concerns. Integrating FL with retrieval and generation components in the RAG pipeline, or with the embedding models used for similarity computation, could enable continuous learning from distributed environments without compromising confidentiality.

Further, tailoring the methodology to accommodate domain-specific terminologies and data structures could significantly broaden its impact. While BLEU and ROUGE metrics provided valuable insights into linguistic alignment, incorporating additional evaluation metrics like METEOR or BERTScore could enable a more comprehensive understanding of semantic nuances and model effectiveness in complex, unstructured datasets.

Developing interactive and explainable models is another promising direction. Such models could enhance user trust and utility by providing clear justifications for predicted impacts, thereby facilitating informed decision-making processes. Similarly, exploring hybrid approaches that combine the strengths of traditional ML models, NLP techniques, and advanced generative AI frameworks could result in a more dynamic solution capable of adapting to varying dataset complexities and requirements structures.

Longitudinal studies to evaluate the robustness of the proposed models over time and in evolving datasets would also provide valuable insights into their real-world applicability and reliability. Finally, advanced preprocessing techniques, such as dynamic clustering or linguistic segmentation, could be explored to improve model performance on datasets with high variability.

By addressing these areas, including federated learning for domain scalability and privacy-preserving collaboration, future work can build on the foundation laid by this

thesis to create more effective, scalable, and adaptable solutions for requirements CIA, advancing the state of the art in software engineering and related fields.

References

Abad, Z.S.H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G. & Schneider, K. 2017a, 'What Works Better? A Study of Classifying Requirements', *IEEE 25th International Requirements Engineering Conference*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Piscataway, pp. 496–501.

Abad, Z.S.H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G. & Schneider, K. 2017b, 'What Works Better? A Study of Classifying Requirements', *IEEE 25th International Requirements Engineering Conference*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Piscataway, pp. 496–501.

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A.A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Cai, Q., Chaudhary, V., Chen, Dong, Chen, Dongdong, Chen, W., Chen, Y.-C., Chen, Y.-L., Cheng, H., Chopra, P., Dai, X., Dixon, M., Eldan, R., Fragoso, V., Gao, J., Gao, Mei, Gao, Min, Garg, A., Del Giorno, A., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R.J., Hu, W., Huynh, J., Iter, D., Jacobs, S.A., Javaheripi, M., Jin, X., Karampatziakis, N., Kauffmann, P., Khademi, M., Kim, D., Kim, Y.J., Kurilenko, L., Lee, J.R., Lee, Y.T., Li, Yuanzhi, Li, Yunsheng, Liang, C., Liden, L., Lin, X., Lin, Z., Liu, C., Liu, L., Liu, M., Liu, W., Liu, X., Luo, C., Madan, P., Mahmoudzadeh, A., Majercak, D., Mazzola, M., Mendes, C.C.T., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Ren, L., de Rosa, G., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shen, Y., Shukla, S., Song, X., Tanaka, M., Tupini, A., Vaddamanu, P., Wang, C., Wang, G., Wang, L., Wang, S., Wang, X., Wang, Y., Ward, R., Wen, W., Witte, P., Wu, H., Wu, X., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Xue, J., Yadav, S., Yang, F., Yang, J., Yang, Y., Yang, Z., Yu, D., Yuan, L., Zhang, Chenruidong, Zhang, Cyril, Zhang, J., Zhang, L.L., Zhang, Yi, Zhang, Yue, Zhang, Yunan & Zhou, X. 2024, 'Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone', *arXiv*, vol. 2, no. arxiv.org/abs/2404.14219, viewed 30 August 2024, <http://arxiv.org/abs/2404.14219>.

Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C. & Vaz, E. 2019, 'A Machine Learning-Based Approach for Demarcating Requirements in Textual Specifications', *IEEE*

Conference Proceedings, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), University of Luxembourg ; University of Luxembourg; University of Ottawa ; QRA Corp., pp. 51–62.

Adnan, K. & Akbar, R. 2019, 'Limitations of information extraction methods and techniques for heterogeneous unstructured big data', *International Journal of Engineering Business Management*, SAGE Publications Inc.

del Águila, I.M. & del Sagrado, J. 2016a, 'Bayesian networks for enhancement of requirements engineering: a literature review', *Requirements Engineering*, vol. 21, no. 4, pp. 461–80.

del Águila, I.M. & del Sagrado, J. 2016b, 'Bayesian networks for enhancement of requirements engineering: a literature review', *Requirements Engineering*, vol. 21, no. 4, pp. 461–80.

Akbar, M.A., Mahmood, S., Alsanad, A., Shafiq, M., Gumaei, A. & Alsanad, A.A.A. 2020, 'Organization Type and Size Based Identification of Requirements Change Management Challenges in Global Software Development', *IEEE Access*, vol. 8, pp. 94089–111.

Ali, S., Briand, L.C., Hemmati, H. & Panesar-Walawege, R.K. 2010, 'A systematic review of the application and empirical investigation of search-based test case generation', *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–62.

Alkaf, H., Hassine, J., Binalialhag, T. & Amyot, D. 2019, 'An automated change impact analysis approach for User Requirements Notation models', *Journal of Systems and Software*, vol. 157.

Alsalemi, A.M. & Yeoh, E. 2017, 'A Systematic Literature Review of Requirements Volatility Prediction', *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pp. 55–64.

Alsanoosy, T., Spichkova, M. & Harland, J. 2019, 'Cultural influence on requirements engineering activities: a systematic literature review and analysis', *Requirements Engineering*, no. 0123456789.

Anjali, C., Dhas, J.P.M. & Singh, J.A.P. 2022, 'A study of Change Impact Analysis Techniques based on Requirement Defects during the Software Development Process', *SPICES 2022 - IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems*, pp. 174–9.

Antoniol, G., Rollo, V.F. & Venturi, G. 2005, 'Detecting groups of co-changing files in CVS repositories', *International Workshop on Principles of Software Evolution (IWPSE)*, vol. 2005, pp. 23–32.

Anwer, S., Wen, L., Zhang, S., Wang, Z. & Sun, Y. 2024, 'BECIA: a behaviour engineering-based approach for change impact analysis', *International Journal of Information Technology (Singapore)*, vol. 16, no. 1, pp. 159–68.

Arif, M., Mohammad, C.W. & Sadiq, M. 2023, 'UML and NFR-framework based method for the analysis of the requirements of an information system', *International Journal of Information Technology (Singapore)*, vol. 15, no. 1, pp. 411–22.

Arnold, R. 1996, *Software Change Impact Analysis*.

Arnold, R.S. & Bohner, S.A. 1993, 'Impact Analysis - Towards A Framework for Comparison', *Conference on Software Maintenance, Montreal, Quebec, Canada*, pp. 292–301.

Arora, C., Sabetzadeh, M., Goknil, A., Briand, L.C. & Zimmer, F. 2015a, 'Change impact analysis for Natural Language requirements: An NLP approach', *IEEE 23rd International Requirements Engineering Conference, RE 2015*, pp. 6–15.

Arora, C., Sabetzadeh, M., Goknil, A., Briand, L.C. & Zimmer, F. 2015b, 'NARCIA: An automated tool for change impact analysis in natural language requirements', *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings*, pp. 962–5.

Arora, C., Sabetzadeh, M., Nejati, S. & Briand, L. 2019, 'An active learning approach for improving the accuracy of automated domain model extraction', *ACM Transactions on Software Engineering and Methodology*, vol. 28, no. 1, pp. 1–34.

Aryani, A., Peake, I.D., Hamilton, M., Schmidt, H. & Winikoff, M. 2009, 'Change propagation analysis using domain information', *Proceedings of the Australian Software Engineering Conference, ASWEC*, pp. 34–43.

Avesani, P., Perini, A., Siena, A. & Susi, A. 2015, 'Goals at risk? Machine learning at support of early assessment', *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, pp. 252–5.

Baker, C., Deng, L., Chakraborty, S. & Dehlinger, J. 2019, 'Automatic multi-class non-functional software requirements classification using neural networks', *Proceedings - International Computer Software and Applications Conference*, vol. 2, pp. 610–5.

Bal, P.R. & Kumar, S. 2020, 'WR-ELM: Weighted Regularization Extreme Learning Machine for Imbalance Learning in Software Fault Prediction', *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1355–75.

Bano, M., Imtiaz, S., Ikram, N., Niazi, M. & Usman, M. 2012, 'Causes of requirement change - A systematic literature review', *IET Seminar Digest*, vol. 2012, pp. 22–31.

Barua, S., Islam, M.M., Yao, X. & Murase, K. 2014, 'MWMOTE - Majority weighted minority oversampling technique for imbalanced data set learning', *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 2, pp. 405–25.

Basri, S., Kama, N., Sarkan, H.M., Adli, S. & Haneem, F. 2016, 'An Algorithmic-Based Change Effort Estimation Model for Software Development', *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 0, pp. 177–84.

Baumer, E.P.S., White, J.P. & Tomlinson, B. 2010, 'Comparing Semantic Role Labeling with Typed Dependency Parsing in Computational Metaphor Identification', *Second Workshop on Computational Approaches to Linguistic Creativity - North American Chapter of the Association of Computational Linguistics*, no. June, pp. 14–22.

Beecham, S., Hall, T. & Rainer, A. 2005, 'Defining a Requirements Process Improvement Model', *Software Quality Journal*, vol. 13, no. 3, pp. 247–79.

Belsis, P., Koutoumanos, A. & Sgouropoulou, C. 2014, 'PBURC: A patterns-based, unsupervised requirements clustering framework for distributed agile software development', *Requirements Engineering*, vol. 19, no. 2, pp. 213–25.

Bennin, K.E., Keung, J., Phannachitta, P., Monden, A. & Mensah, S. 2018, 'MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction', *Proceedings - International Conference on Software Engineering*, vol. 44, no. 6, p. 699.

Berry, D.M., Cleland-huang, J., Mylopoulos, J., Ferrari, A., Maalej, W. & Zowghi, D. 2017, *Panel : Context-Dependent Evaluation of Tools for NL RE Tasks : Recall vs . Precision , and Beyond*, pp. 14–7.

B.Hossin, M. & Sulaiman, M.N. 2015, 'A review on evalution metrics for data clasiification evaluations', *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, vol. 5, no. 2, pp. 1–11.

A review of machine learning algorithms for identification and classification of non-functional requirements 2019 (Expert Systems with Applications: X).

Bjarnason, E., Runeson, P., Borg, M., Unterkalmsteiner, M., Engström, E., Regnell, B., Sabaliauskaite, G., Loconsole, A., Gorschek, T. & Feldt, R. 2014, 'Challenges and practices in aligning requirements with verification and validation: a case study of six companies', *Empirical Software Engineering*, vol. 19, no. 6, pp. 1809–55.

De Bortoli Fávero, E.M., Casanova, D. & Pimentel, A.R. 2019, 'EmbSE: A Word Embeddings Model Oriented Towards Software Engineering Domain', *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*SBES 2019, Association for Computing Machinery, New York, NY, USA, pp. 172–180.

Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. & Caicedo, O.M. 2018, 'A comprehensive survey on machine learning for networking: evolution, applications and research opportunities', *Journal of Internet Services and Applications*, vol. 9, no. 1.

Boyd, S., Farroukh, A. & Didar Zowghi 2005, *Measuring the Expressiveness of a Constrained Natural Language : An Empirical Study*.

Breiman, L. 2001, 'Random Forests', *Machine Learning*, vol. 45, no. 1, pp. 5–32.

Brooks, F. 1987, 'No Silver Bullet: Essence and Accidents of Software Engineering', *Computer*, vol. 20, no. 4, pp. 10–9.

Brucker, A.D. & Julliand, J. 2014, 'A survey of code-based change impact analysis techniques', *Software Testing Verification and Reliability*, vol. 24, no. 8, pp. 591–2.

Catolino, G., Palomba, F., De Lucia, A., Ferrucci, F. & Zaidman, A. 2018, 'Enhancing change prediction models using developer-related factors', *Journal of Systems and Software*, vol. 143, pp. 14–28.

Chawla, N. V., Bowyer, K.W., Hall, L.O. & Kegelmeyer, W.P. 2002, 'snopes.com: Two-Striped Telamonia Spider', *Journal of Artificial Intelligence Research*, vol. 16, no. Sept. 28, pp. 321–57.

Chen, H., He, K., Liang, P. & Li, R. 2010, 'Text-based requirements preprocessing using nature language processing techniques', *2010 International Conference on Computer Design and Applications, ICCDA 2010*, vol. 1, no. Iccda, pp. V1-14-V1-18.

Chioaşcă, E.V. 2012, 'Using machine learning to enhance automated requirements model transformation', *Proceedings - International Conference on Software Engineering*, pp. 1487–90.

Christopher J.C. Burges 1998, 'A Tutorial on Support Vector Machines for Pattern Recognition', *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–67.

Cortes, C., Haffner, P. & Mohri, M. 2004, 'Rational Kernels: Theory and Algorithms', *Journal of Machine Learning Research*, vol. 5, pp. 1035–62.

Cruzes, D.S. & Dyb, T. 2011, 'Research synthesis in software engineering: A tertiary study', *Information and Software Technology*, vol. 53, no. 5, pp. 440–55.

Dablain, D., Krawczyk, B. & Chawla, N. V. 2022, 'DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data', *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–15.

Dalpiaz, F., Dell'Anna, D., Aydemir, F.B. & Çevikol, S. 2019, 'Requirements classification with interpretable machine learning and dependency parsing', *Proceedings of the*

IEEE International Conference on Requirements Engineering, vol. 2019- Septe, pp. 142–52.

Dalpiaz, F., van der Schalk, I., Brinkkemper, S., Aydemir, F.B. & Lucassen, G. 2019, 'Detecting terminological ambiguity in user stories: Tool and experimentation', *Information and Software Technology*, vol. 110, no. December 2018, pp. 3–16.

Danjuma, K.J. 2015, 'Performance Evaluation of Machine Learning Algorithms in Post-operative Life Expectancy in the Lung Cancer Patients', *International Journal of Computer Science Issues (IJCSI)*, vol. 12, no. 2, pp. 189-199.

Dargan, J.L., Wasek, J.S. & Campos-Nanez, E. 2016, 'Systems performance prediction using requirements quality attributes classification', *Requirements Engineering*, vol. 21, no. 4, pp. 553–72.

Deocadez, R., Harrison, R. & Rodriguez, D. 2017, 'Automatically classifying requirements from app stores: A preliminary study', *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, pp. 367–71.

Dermeval, D., Vilela, J., Bittencourt, I.I., Castro, J., Isotani, S., Brito, P. & Silva, A. 2016, 'Applications of ontologies in requirements engineering: a systematic review of the literature', *Requirements Engineering*, vol. 21, no. 4, pp. 405–37.

Deshpande, G., Arora, C. & Ruhe, G. 2019, 'Data-driven Elicitation and Optimization of Dependencies between Requirements', *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pp. 416–21.

Dhamija, A. & Sikka, S. 2019, 'A Systematic Study of Advancements in Change Impact Analysis Techniques', *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, no. 8, pp. 435–43.

Dhingra, S., Savithri, G., Madan, M. & Manjula, R. 2017, 'Selection of prioritization technique for software requirement using Fuzzy Logic and Decision Tree', *Proceedings of 2016 Online International Conference on Green Engineering and Technologies, IC-GET 2016*, pp. 1–11.

Durelli, V.H.S., Durelli, R.S., Borges, S.S., Endo, A.T., Eler, M.M., Dias, D.R.C. & Guimarães, M.P. 2019, 'Machine learning applied to software testing: A systematic mapping study', *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–212.

Elapolu, M.S.R., Rai, R., Gorsich, D.J., Rizzo, D., Rapp, S. & Castanier, M.P. 2024, 'Blockchain technology for requirement traceability in systems engineering', *Information Systems*, vol. 123, no. March, p. 102384.

Erdoğan, Z. & Namlı, E. 2019, 'A living environment prediction model using ensemble machine learning techniques based on quality of life index', *Journal of Ambient Intelligence and Humanized Computing*, no. Michalos 2014.

Ernst, N.A. & Mylopoulos, J. 2010, 'On the perception of software quality requirements during the project lifecycle', *Lecture Notes in Computer Science*, vol. 6182 LNCS, pp. 143–57.

Ferrari, A. & Esuli, A. 2019, 'An NLP approach for cross-domain ambiguity detection in requirements engineering', *Automated Software Engineering*, vol. 26, no. 3, pp. 559–98.

Ferrari, A., Gnesi, S. & Tolomei, G. 2013, 'Using clustering to improve the structure of natural language requirements documents', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7830 LNCS, pp. 34–49.

Ferrari, A., Spagnolo, G.O. & Gnesi, S. 2017, 'Towards a dataset for natural language requirements processing', *CEUR Workshop Proceedings*, vol. 1796.

Galar, M., Fern, A., Barrenechea, E. & Bustince, H. 2012, *A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches*, vol. 42, no. 4, pp. 463–84.

Ganesan, K. 2018, *ROUGE 2.0: Updated and Improved Measures for Evaluation of Summarization Tasks*, pp. 1–8.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M. & Wang, H. 2023, *Retrieval-Augmented Generation for Large Language Models: A Survey*, pp. 1–21.

Ghozali, R.P., Saputra, H., Apriadin Nuriawan, M., Suharjito, Utama, D.N. & Nugroho, A. 2019, 'Systematic literature review on decision-making of requirement engineering from agile software development', *Procedia Computer Science*, vol. 157, pp. 274–81.

Giger, E., Pinzger, M. & Gall, H.C. 2012, 'Can we predict types of code changes? An empirical analysis', *IEEE International Working Conference on Mining Software Repositories*, no. May, pp. 217–26.

Goknil, A., Kurtev, I. & Berg, K. van den 2016, *A Rule-Based Change Impact Analysis Approach in Software Architecture for Requirements Changes*.

Gou, Q., Xia, Z., Yu, B., Yu, H., Huang, F., Li, Y. & Nguyen, C.T. 2023, 'Diversify Question Generation with Retrieval-Augmented Style Transfer', *EMNLP 2023 - 2023 Conference on Empirical Methods in Natural Language Processing, Proceedings*, no. Figure 1, pp. 1677–90.

Gunawardana, A. 2009, 'A Survey of Accuracy Evaluation Metrics of Recommendation Tasks', *Journal ofMachine Learning Research*, vol. 10, pp. 2935–62.

Haq, B., Nadeem, M., Ali, I., Ali, K., Raza, M. & Rehmanr, M.U. 2019, 'Use of Expert System in Requirements Engineering Process A Systematic Literature Review', *2019 UK/China Emerging Technologies, UCET 2019, Glasgow, United Kingdom*, pp. 1–5.

Haque, A., Rahman, A. & Siddik, S. 2019, 'Non-Functional Requirements Classification with Feature Extraction and Machine Learning : An Empirical Study', *1st International Conference on Advances in Science, Engineering and Robotics Technology 2019 (ICASERT 2019)*, vol. 2019, no. Icasert.

Hassine, J., Rilling, J. & Hewitt, J. 2005, 'Change Impact Analysis for Requirement Evolution using Use Case Maps', *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*.

Hayes, J.H., Dekhtyar, A. & Sundaram, S.K. 2006, 'Advancing candidate link generation for requirements tracing: The study of methods', *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19.

Hayes, J.H., Li, W. & Rahimi, M. 2014, 'Weka meets TraceLab: Toward convenient classification: Machine learning for requirements engineering problems: A position paper', *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering, AIRE 2014 - Proceedings*, pp. 9–12.

Hayes, J.H., Li, W., Yu, T., Han, X., Hays, M. & Woodson, C. 2015, 'Measuring Requirement Quality to Predict Testability', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Piscataway, pp. 1–8.

Hayes, J.H., Payne, J. & Leppelmeier, M. 2019, 'Toward Improved Artificial Intelligence in Requirements Engineering: Metadata for Tracing Datasets', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), University of Kentucky, pp. 256–62.

Hein, P.H., Voris, N. & Morkos, B. 2018, 'Predicting requirement change propagation through investigation of physical and functional domains', *Research in Engineering Design*, vol. 29, no. 2, pp. 309–28.

Henderson, P. & Ferrari, V. 2017, 'End-to-end training of object class detectors for mean average precision', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10115 LNCS, pp. 198–213.

Herm, L.V., Heinrich, K., Wanner, J. & Janiesch, C. 2023, 'Stop ordering machine learning algorithms by their explainability! A user-centered investigation of performance and explainability', *International Journal of Information Management*, vol. 69.

Hevner, A.R., March, S.T., Park, J. & Ram, S. 2004, 'Design Sceince in Information Systems', *MIS Quarterly*, vol. 28, no. 1, pp. 75–105.

Horkoff, J., Aydemir, F.B., Cardoso, E., Li, T., Maté, A., Paja, E., Salnitri, M., Piras, L., Mylopoulos, J. & Giorgini, P. 2019, 'Goal-oriented requirements engineering: an extended systematic mapping study', *Requirements Engineering*, vol. 24, no. 2, pp. 133–60.

Hsu, C.-W., Chang, C.-C. & Lin, C.-J. 2003, 'A Practical Guide to Support Vector Classification', *Technical Report, Department of Computer Science and Information Engineering, University of National Taiwan, Taipei*, vol. 1, no. 1, pp. 1–12.

Idrissi Khaldi, M., Erraissi, A., Hain, M. & Banane, M. 2025, *Comparative Analysis of Supervised Machine Learning Classification Models*, *Lecture Notes in Networks and Systems*, vol. 1353 LNNS, Springer Nature Switzerland.

Imtiaz, S.M.S.M. & Bhowmik, T. 2018, 'Towards Data-Driven Vulnerability Prediction for Requirements', *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA, pp. 744–748.

Jayatilleke, S. & Lai, R. 2013, 'A Method of Specifying and Classifying Requirements Change', *22nd Australian Conference on Software Engineering(ASWEC), Melbourne, VIC*, pp. 175–80.

Jayatilleke, S. & Lai, R. 2018, 'A systematic review of requirements change management', *Information and Software Technology*, vol. 93, pp. 163–85.

Jayatilleke, S., Lai, R. & Reed, K. 2018, 'A method of requirements change analysis', *Requirements Engineering*, vol. 23, no. 4, pp. 493–508.

Jiang, W., Subramanian, S., Graves, C., Alonso, G., Yazdanbakhsh, A. & Dadu, V. 2025, *RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving*, *Proceedings of Preprint*, vol. 1, Association for Computing Machinery.

Johnson, J., Douze, M. & Jegou, H. 2021, 'Billion-Scale Similarity Search with GPUs', *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–47.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. & Yih, W.T. 2020, 'Dense passage retrieval for open-domain question answering', *EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 6769–81.

Khelifa, A., Haoues, M. & Sellami, A. 2018, 'Towards a software requirements change classification using support vector machine', *CEUR Workshop Proceedings*, vol. 2279.

Kitchenham, B., Budgen, D., & Brereton, P. 2016, *Evidence-Based Software Engineering and Systematic Reviews*, Chapman and Hall/CRC (ed.), 1st editio.

Kitchenham, B.A. & Charters, S. 2007, 'Guidelines for performing systematic literature reviews in software engineering', *Keele University and University of Durham Joint Technical Report, 2007.*

Klaus Krippendorff 2018, 'Content Analysis: An Introduction to Its Methodology, 4th', *Beaverton: Ringgold Inc.*, Ringgold Inc, Beaverton.

Knauss, E., Damian, D., Cleland-Huang, J. & Helms, R. 2015, 'Patterns of continuous requirements clarification', *Requirements Engineering*, vol. 20, no. 4, pp. 383–403.

Von Knethen, A. 2002, 'Change-oriented requirements traceability. Support for evolution of embedded systems', *Conference on Software Maintenance*, pp. 482–5.

Kurtanovic, Z. & Maalej, W. 2017, 'Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning', *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pp. 490–5.

Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M. & Jurafsky, D. 2013, 'Deterministic Coreference Resolution Based on Entity-Centric, Precision-Ranked Rules', *Computational Linguistics*, vol. 39, no. 4, pp. 885–916.

Lehnert, S. 2011a, 'A Review of Software Change Impact Analysis', *Technology*, p. 39.

Lehnert, S. 2011b, 'A taxonomy for software change impact analysis', *IWPSE-EVOL'11 - Proceedings of the 12th International Workshop on Principles on Software Evolution*, no. 1, pp. 41–50.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T., Riedel, S. & Kiela, D. 2020, 'Retrieval-augmented generation for knowledge-intensive NLP tasks', *Advances in Neural Information Processing Systems*, vol. 2020- Decem.

Li, C., Huang, L., Ge, J., Luo, B. & Ng, V. 2018, 'Automatically classifying user requests in crowdsourcing requirements engineering', *Journal of Systems and Software*, vol. 138, pp. 108–23.

Li, Y., Li, J., Yang, Y. & Li, M. 2008, 'Requirement-centric traceability for change impact analysis: A case study', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5007 LNCS, pp. 100–11.

Li, Z., Chen, M., Huang, L., Ng, V. & Geng, R. 2017, 'Tracing requirements in software design', *Proceedings of the 2017 International Conference on Software and System Process - ICSSP 2017*, vol. Part F1287, ACM Press, New York, New York, USA, pp. 25–9.

Li, Z. & Huang, L. 2018, 'Tracing Requirements as a Problem of Machine Learning', *International Journal of Software Engineering & Applications*, vol. 9, no. 4, pp. 21–36.

Lin, C.-Y. 2004, 'Looking for a Few Good Metrics: ROUGE and its Evaluation', *NTCIR Workshop*, no. June, pp. 1–8.

Lin, Z. 2020, 'A Methodological Review of Machine Learning in Applied Linguistics', *English Language Teaching*, vol. 14, no. 1, p. 74.

Liu, L.P., Dietterich, T.G., Li, N. & Zhou, Z.H. 2016, 'Transductive optimization of top k precision', *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016- Janua, pp. 1781–7.

Maalem, S. & Zarour, N. 2016, 'Challenge of validation in requirements engineering', *Journal of Innovation in Digital Ecosystems*, vol. 3, no. 1, pp. 15–21.

Malhotra, R., Chug, A., Hayrapetian, A. & Raje, R. 2016, 'Analyzing and evaluating security features in software requirements', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings.*, pp. 26–30.

Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J. & McClosky, D. 2014, 'The stanford CoreNLP natural language processing toolkit', *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, vol. 2014- June, pp. 55–60.

de Marneffe, M.-C. & Manning, C.D. 2008, *The Stanford typed dependencies representation*, no. June, pp. 1–8.

McGee, S. & Greer, D. 2011, 'Software requirements change taxonomy: Evaluation by case study', *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, pp. 25–34.

McZara, J., Sarkani, S., Holzer, T. & Eveleigh, T. 2015, 'Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment', *Empirical Software Engineering*, vol. 20, no. 6, pp. 1721–61.

Memon, K.A. & Xiaoling, X. 2019, 'Deciphering and Analyzing Software Requirements Employing the Techniques of Natural Language Processing', *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence*ICMAI 2019, Association for Computing Machinery, New York, NY, USA, pp. 153–156.

Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B. & Jiang, Y. 2008, 'Implications of ceiling effects in defect predictors', *Proceedings - International Conference on Software Engineering*, no. i, pp. 47–54.

Mezghani, M. & Florence, S. 2019, 'Clustering for traceability managing in system specifications', *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pp. 257–64.

Misra, J., Sengupta, S. & Podder, S. 2016, 'Topic Cohesion Preserving Requirements Clustering', *Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*RAISE '16, Association for Computing Machinery, New York, NY, USA, pp. 22–28.

Morkos, B., Shankar, P. & Summers, J.D. 2012, 'Predicting requirement change propagation, using higher order design structure matrices: an industry case study', *Journal of Engineering Design*, vol. 23, no. 12, pp. 905–26.

Morkos, B. & Summers, J.D. 2010, 'Requirement change propagation prediction approach: Results from an industry case study', *Proceedings of the ASME Design Engineering Technical Conference*, vol. 1, no. PARTS A AND B, pp. 111–21.

Nardini, M., Ciambra, F., Garzoli, F., Croce, D., De Cao, D. & Basili, R. 2012, 'Machine learning technologies for the requirements analysis in complex systems', *22nd Annual International Symposium of the International Council on Systems Engineering, INCOSE 2012 and the 8th Biennial European Systems Engineering Conference 2012, EuSEC 2012*, vol. 1, pp. 372–86.

Nejati, S., Sabetzadeh, M., Arora, C., Briand, L.C. & Mandoux, F. 2016, 'Automated Change Impact Analysis between SysML Models of Requirements and Design', *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*FSE 2016, Association for Computing Machinery, New York, NY, USA, pp. 242–253.

Nogueira, R. & Cho, K. 2019, *Passage Re-ranking with BERT*, pp. 1–5.

Osman, M.H. & Zaharin, M.F. 2018, 'Ambiguous Software Requirement Specification Detection: An Automated Approach', *Proceedings of the 5th International Workshop on Requirements Engineering and Testing*RET '18, Association for Computing Machinery, New York, NY, USA, pp. 33–40.

Osman, M.S., Alabwaini, N.Z., Jaber, T.B. & Alrawashdeh, T. 2019, 'Generate use case from the requirements written in a natural language using machine learning', *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, JEEIT 2019 - Proceedings*, pp. 748–51.

Pal, P., Sandhu, G. & Pal, S. 2015, 'Applying Machine Learning to Conflict Management in Software Requirement', *International Journal of Computer Applications*, no. Cognition, pp. 14–6.

Paleyes, A., Urma, R.G. & Lawrence, N.D. 2022, 'Challenges in Deploying Machine Learning: A Survey of Case Studies', *ACM Computing Surveys*, vol. 55, no. 6.

Parra, E., Dimou, C., Llorens, J., Moreno, V. & Fraga, A. 2015, 'A methodology for the classification of quality of requirements using machine learning techniques', *Information and Software Technology*, vol. 67, pp. 180–95.

Patel, Y., Tolias, G. & Matas, J. 2022, 'Recall@k Surrogate Loss with Large Batches and Similarity Mixup', *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2022- June, pp. 7492–501.

Peffers, K., Tuunanen, T., Rothenberger, M.A. & Chatterjee, S. 2007, 'A design science research methodology for information systems research', *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77.

Perini, A., Susi, A. & Avesani, P. 2013, 'A Machine Learning Approach to Software Requirements Prioritization', *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 445–61.

Persson, R.A.X. 2023, 'Theoretical evaluation of partial credit scoring of the multiple-choice test item', *Metron*, vol. 81, no. 2, pp. 143–61.

Pipit Muliyah, Dyah Aminatun, Sukma Septian Nasution, Tommy Hastomo, Setiana Sri Wahyuni Sitepu, T. 2020, *Dependency Parsing*, *Journal GEEJ*, vol. 7.

Polyzotis, N., Roy, S., Whang, S.E. & Zinkevich, M. 2017, 'Data management challenges in production machine learning', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. Part F1277, pp. 1723–6.

Powers, D.M.W. 2020, *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*, pp. 37–63.

Quinlan, J.R. 1986, 'Induction of decision trees', *Machine Learning*, vol. 1, no. 1, pp. 81–106.

Rago, A., Marcos, C. & Diaz-Pace, J.A. 2018, 'Using semantic roles to improve text classification in the requirements domain', *Language Resources and Evaluation*, vol. 52, no. 3, pp. 801–37.

Reimers, N. & Gurevych, I. 2019, 'Sentence-BERT: Sentence embeddings using siamese BERT-networks', *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pp. 3982–92.

Riaz, M., King, J., Slankas, J. & Williams, L. 2014, 'Hidden in plain sight: Automatically identifying security requirements from natural language artifacts', *2014 IEEE 22nd International Requirements Engineering Conference, RE 2014 - Proceedings*, pp. 183–92.

Robertson, S. & Zaragoza, H. 2009, *The probabilistic relevance framework: BM25 and beyond*, *Foundations and Trends in Information Retrieval*, vol. 3.

van Rooijen, L., Baumer, F.S., Platenius, M.C., Geierhos, M., Hamann, H. & Engels, G. 2017, 'From User Demand to Software Service: Using Machine Learning to Automate the Requirements Specification Process', *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, IEEE, pp. 379–85.

Saher, N., Baharom, F. & Ghazali, O. 2017, 'Requirement change taxonomy and categorization in agile software development', *6th International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 1–6.

Sarker, I.H. 2021, 'Machine Learning: Algorithms, Real-World Applications and Research Directions', *SN Computer Science*, Springer.

Seiffert, C., Khoshgoftaar, T.M., Van Hulse, J. & Napolitano, A. 2010, 'RUSBoost: A hybrid approach to alleviating class imbalance', *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 40, no. 1, pp. 185–97.

Shakeri, Z., Abad, H., Gervasi, V., Zowghi, D. & Far, B.H. 2019, 'Supporting Analysts by Dynamic Extraction and Classification of Requirements-Related Knowledge', *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) Supporting*.

SHAKIRAT, Y., BAJEH, A., Aro, T.O. & ADEWOLE, K. 2021, 'Improving the Accuracy of Static Source Code Based Software Change Impact Analysis Through Hybrid Techniques: A Review', *International Journal of Software Engineering and Computer Systems*, vol. 7, no. 1, pp. 57–66.

Sharma, R., Bhatia, J. & Biswas, K.K. 2014, 'Automated identification of business rules in requirements documents', *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*, pp. 1442–7.

Sharma, Richa, Bhatia, J. & Biswas, K.K. 2014, 'Machine Learning for Constituency Test of Coordinating Conjunctions in Requirements Specifications', *Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*RAISE 2014, Association for Computing Machinery, New York, NY, USA, pp. 25–31.

Sharma, R., Sharma, N. & Biswas, K.K. 2016, 'Machine Learning for Detecting Pronominal Anaphora Ambiguity in NL Requirements', *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)*, IEEE, pp. 177–82.

Sikora, E., Tenbergen, B. & Pohl, K. 2012, 'Industry needs and research directions in requirements engineering for embedded systems', *Requirements Engineering*, vol. 17, no. 1, pp. 57–78.

Singh, D. & Sharma, A. 2014, 'Software requirement prioritization using machine learning', *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2014-Janua, no. January, pp. 701–4.

Singh, J. 2023, *Computational Complexity and Analysis of Supervised Machine Learning Algorithms*, *Lecture Notes in Networks and Systems*, vol. 445, Springer Nature Singapore.

Singh, M. 2018, 'Automated Validation of Requirement Reviews: A Machine Learning Approach', *2018 IEEE 26th International Requirements Engineering Conference (RE)*, IEEE, pp. 460–5.

Singh, M., Anu, V., Walia, G.S.G.S. & Goswami, A. 2018, 'Validating Requirements Reviews by Introducing Fault-Type Level Granularity: A Machine Learning Approach', *Proceedings of the 11th Innovations in Software Engineering Conference*ISEC '18, Association for Computing Machinery, New York, NY, USA.

Slankas, J. & Williams, L. 2013, 'Automated extraction of non-functional requirements in available documentation', *2013 1st International Workshop on Natural Language Analysis in Software Engineering, NaturaLiSE 2013 - Proceedings*, pp. 9–16.

Song, S., Zhang, N. & Huang, H. 2019, 'Named entity recognition based on conditional random fields', *Cluster Computing*, vol. 22, no. s3, pp. 5195–206.

Sowjanya, A.M. & Mrudula, O. 2022, 'Effective treatment of imbalanced datasets in health care using modified SMOTE coupled with stacked deep learning algorithms', *Applied Nanoscience (Switzerland)*, no. 0123456789.

Sufian, M., Khan, Z., Rehman, S. & Haider Butt, W. 2019, 'A systematic literature review: Software requirements prioritization techniques', *Proceedings - 2018 International Conference on Frontiers of Information Technology, FIT 2018*, pp. 35–40.

Sultanov, H. & Hayes, J.H.J.H. 2013, 'Application of reinforcement learning to requirements engineering: Requirements tracing', *2013 21st IEEE International Requirements Engineering Conference, RE 2013 - Proceedings*, pp. 52–61.

Takahashi, K., Yamamoto, K., Kuchiba, A. & Koyama, T. 2022, 'Confidence interval for micro-averaged F 1 and macro-averaged F 1 scores', *Applied Intelligence*, vol. 52, no. 5, pp. 4961–72.

Tamai, T. & Anzai, T. 2018, 'Quality requirements analysis with machine learning', *ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*, vol. 2018-March, pp. 241–8.

Tantithamthavorn, C., Hassan, A.E. & Matsumoto, K. 2018, 'The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models', *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–19.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A. & Gurevych, I. 2021, *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*.

Tomek, I. 1976, 'Tomek Link: Two Modifications of CNN', *IEEE Trans. Systems, Man and Cybernetics*, pp. 769–72.

Tufail, S., Riggs, H., Tariq, M. & Sarwat, A.I. 2023, 'Advancements and Challenges in Machine Learning: A Comprehensive Review of Models, Libraries, Applications, and Algorithms', *Electronics (Switzerland)*, MDPI.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. & Polosukhin, I. 2017, 'Attention is all you need', *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 5999–6009.

Vogelsang, A. & Borg, M. 2019, 'Requirements Engineering for Machine Learning: Perspectives from Data Scientists', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, pp. 245–51.

Wan, Z., Xia, X., Lo, D. & Murphy, G.C. 2021, 'How does machine learning change software development practices?', *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1857–71.

Wang, S., Li, T. & Yang, Z. 2019, 'Exploring Semantics of Software Artifacts to Improve Requirements Traceability Recovery: A Hybrid Approach', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Beijing University of Technology, China, pp. 39–46.

Wang, S. & Yao, X. 2013, 'Using class imbalance learning for software defect prediction', *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–43.

Wang, Y. 2015, 'Semantic information extraction for software requirements using semantic role labeling', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings.*, pp. 332–7.

Wang, Y. 2016, 'Automatic semantic analysis of software requirements through machine learning and ontology approach', *Journal of Shanghai Jiaotong University*, vol. 21, no. 6, pp. 692–701.

Wang, Y., Wang, L., Li, Y., He, D., Chen, W. & Liu, T.Y. 2013, 'A theoretical analysis of NDCG ranking measures', *Journal of Machine Learning Research*, vol. 30, pp. 25–54.

Wang, Y. & Zhang, J. 2016, 'Experiment on automatic functional requirements analysis with the EFRF's semantic cases', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Piscataway, pp. 636–42.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. & Schmidt, D.C. 2023, *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*.

Williams, N., Zander, S. & Armitage, G. 2006, 'Evaluating Machine Learning Algorithms for Automated Network Application Identification', *Centre for Advanced Internet Architectures (CAIA). Technical Report 060410B*, no. March, pp. 1–14.

Winkler, J., Grönberg, J. & Vogelsang, A. 2019, 'Predicting How to Test Requirements : An Automated Approach', *2019 IEEE 27th International Requirements Engineering Conference (RE) Predicting*, pp. 120–30.

Winkler, J. & Vogelsang, A. 2017, 'Automatic Classification of Requirements Based on Convolutional Neural Networks', *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pp. 39–45.

Winkler, J.P., Gronberg, J. & Vogelsang, A. 2019, 'Optimizing for Recall in Automatic Requirements Classification: An Empirical Study', *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), TU Berlin, pp. 40–50.

Wohlin, C. 2014, 'Guidelines for snowballing in systematic literature studies and a replication in software engineering', *ACM International Conference Proceeding Series*.

Yacouby, R. & Axman, D. 2020, *Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models*, pp. 79–91.

Yan, S. 2023, 'Automatic Evaluation of Machine Translation Based on Linguistic Knowledge', *2023 IEEE 4th Annual Flagship India Council International Subsections Conference: Computational Intelligence and Learning Systems, INDISCON 2023*, no. 1, pp. 1–5.

Yang, H., de Roeck, A., Gervasi, V., Willis, A. & Nuseibeh, B. 2011, 'Analysing anaphoric ambiguity in natural language requirements', *Requirements Engineering*, vol. 16, no. 3, pp. 163–9.

Yang, H., De Roeck, A., Gervasi, V., Willis, A. & Nuseibeh, B. 2012, 'Speculative requirements: Automatic detection of uncertainty in natural language requirements', *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*, pp. 11–20.

Yang, H., Willis, A., De Roeck, A. & Nuseibeh, B. 2010, 'Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements', *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*ASE '10, Association for Computing Machinery, New York, NY, USA, pp. 53–62.

Yang, Y., Xia, X., Lo, D., Bi, T., Grundy, J. & Yang, X. 2020, *Predictive Models in Software Engineering: Challenges and Opportunities*, vol. 1, no. 1.

Yazdanshenas, A.R. & Moonen, L. 2012, 'Fine-grained change impact analysis for component-based product families', *IEEE International Conference on Software Maintenance, ICSM*, pp. 119–28.

Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q. & Liu, Z. 2024, *Evaluation of Retrieval-Augmented Generation: A Survey*, pp. 1–21.

Zamani, K., Zowghi, D. & Arora, C. 2021, 'Machine Learning in Requirements Engineering: A Mapping Study', *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 116–25.

Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q. & Li, M. 2014, 'Investigating dependencies in software requirements for change propagation analysis', *Information and Software Technology*, vol. 56, no. 1, pp. 40–53.

Zhang, X., Tan, Y. & Yang, Z. 2021, 'Analysis of Impact of Requirement Change on Product Development Progress Based on System Dynamics', *IEEE Access*, vol. 9, pp. 445–57.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K., Ajagbe, M.A., Chioasca, E.-V. & Batista-Navarro, R. 2020, 'Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study', *ArXiv*, vol. abs/2004.0.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E. & Stoica, I. 2023, *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*, no. NeurIPS, pp. 1–29.

Zowghi, D. & Paryani, S. 2003, 'Teaching Requirements engineering through Role Playing: Leason Learnt', *11th IEEE International Requirements Engineering Conference*, pp. 3–11.