
UNIVERSITY OF THE AEGEAN
**SCHOOL OF ENGINEERING**

DEPARTMENT OF INFORMATION AND COMMUNICATION SYSTEMS

MASTER'S PROGRAM
INFORMATION AND COMMUNICATION SYSTEMS ENGINEERING


# Review of AI based requirements analysis, verification and validation

MASTER'S THESIS

by


Christina Aikaterini Makri


**Supervisor: K. Kritikos**

**Members of the Examination Committee: S. Kokolakis, I. Charalabidis**

Samos, August 2025

This page is intentionally blank.

# Acknowledgments

This thesis, entitled “Review of AI-Based Requirements Analysis, Verification and Validation”, was carried out as part of the Master’s Program in Information and Communication Systems Engineering at the University of the Aegean.

I would like to express my deepest gratitude to my supervising professor, Dr. Kritikos, whose continuous support, insightful guidance, and detailed feedback accompanied me throughout this work. His immediacy and presence were invaluable and truly helped me bring this thesis to completion. Through his guidance, I was able to grow as a researcher and strengthen my ability to approach work with clarity and rigor.

My heartfelt thanks go to my loved ones, who supported me with strength and kindness throughout this process. Their patience during the long periods I was devoted to this research and their constant encouragement made the completion of this demanding journey possible.



By

CHRISTINA AIKATERINI MAKRI

Department of Information and Communication Systems

UNIVERSITY OF THE AEGEAN

This page is intentionally blank.

# Table of Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| AHP | Analytical Hierarchy Process |
| AI | Artificial Intelligence |
| ALBERT | A Lite BERT |
| API | Application Programming Interface |
| BERT | Bidirectional Encoder Representations from Transformers |
| BM25 | Best Matching 25 (Ranking Function) |
| BST | Binary Search Tree |
| CoNLL | Conference on Natural Language Learning |
| DAMIR | Dataset for Anaphoric aMbiguity In Requirements |
| EBA | Execute-Before-After |
| GloVe | Global Vectors for Word Representation |
| GUI | Graphical User Interface |
| HIPAA | Health Insurance Portability and Accountability Act |
| IGA | Interactive Genetic Algorithm |
| ISO/IEC/IEEE | International Organization for Standardization / International Electrotechnical Commission / Institute of Electrical and Electronics Engineers |
| LLM | Large Language Model |
| loPo | Leave-One-Project-Out |
| ML | Machine Learning |
| MLensemble | Machine Learning Ensemble |
| MLFE | Machine Learning Feature Embeddings |
| MLLF | Machine Learning Language Features |
| MLV | Mechanical Lung Ventilator |
| NaPiRE | Naming the Pain in Requirements Engineering |
| NL | Natural Language |
| NLP | Natural Language Processing |
| NLPcoref | NLP Coreference |
| NLTK | Natural Language Toolkit |
| POS tagging | Part-of-Speech tagging |
| PSO | Particle Swarm Optimization |
| QA | Question Answering |
| RA | Requirements Analysis |
| RCWV | Requirements Cost Weight Value |
| RE | Requirements Engineering |
| ReqEval | Requirements Evaluation Dataset |
| RIWV | Requirements Importance Weight Value |
| RPV | Requirement Priority Value |
| RVV | Requirements Verification and Validation |
| SBADR | Score-Based Ambiguity Detector and Resolver |
| SE | Software Engineering |
| SpanBERT | Span-Based BERT |

| | |
|---|---|
| SPV | Stakeholder Priority Value |
| SRS | Software Requirements Specification |
| SVM | Support Vector Machine |
| T5 | Text-To-Text Transfer Transformer |
| TF-IDF | Term Frequency-Inverse Document Frequency |
| UML | Unified Modeling Language |

# Abstract

The Requirements Engineering (RE) process plays a critical role in determining software project success, with its Requirements Analysis (RA) and Requirements Verification and Validation (RVV) activities being essential for ensuring requirement completeness and quality. This thesis makes a specialized contribution by focusing explicitly on RA and RVV, clearly defining their scope and systematically exploring how Artificial Intelligence (AI) can enhance these phases of RE. This work provides a targeted systematic review of AI-based RA and RVV approaches between 2020 and 2024, a period marked by rapid advancements in AI technologies.

The review covers tasks, such as classification, prioritisation and quality factors issue detection like ambiguity detection and completeness checking. In the context of this review, the proposed methodologies, datasets, tools, and performance metrics were analysed, highlighting both advancements and challenges. One main observation obtained is that the studied approaches employ a variety of techniques, including Natural Language Processing (NLP), Machine Learning (ML), Deep Learning (DL), and Large Language Models (LLMs). Further, the core review findings show that AI can significantly improve the efficiency in RA and RVV tasks, providing a potential to accelerate project delivery and software projects costs reduction, thereby shaping future directions in software engineering practice. However, limitations still remain in performance, dataset quality, and cross-domain applicability. As such, this review offers updated perspective and insights to guide further research and practical adoption of AI-based RA and RVV approaches. Overall, the work demonstrates how AI can serve as a transformative driver for future improvements in software engineering efficiency and quality.

**Keywords:** *Requirements Engineering, Requirements Analysis, Requirements Verification, Requirements Validation, Artificial Intelligence, Machine Learning, NLP, Deep Learning, Large Language Models*

# Περίληψη

Η Μηχανική Απαιτήσεων (Requirements Engineering – RE) παίζει κρίσιμο ρόλο στην επιτυχία ενός έργου λογισμικού, με τις φάσεις της Ανάλυσης Απαιτήσεων (Requirements Analysis – RA) και της Επαλήθευσης και Επικύρωσης Απαιτήσεων (Requirements Verification and Validation – RVV) να είναι απαραίτητες για τη διασφάλιση της πληρότητας και της ποιότητας των απαιτήσεων. Η παρούσα διπλωματική εργασία προσφέρει μια εξειδικευμένη συνεισφορά, εστιάζοντας αποκλειστικά στις φάσεις RA και RVV, παρέχοντας σαφείς ορισμούς και εξετάζοντας συστηματικά πώς η Τεχνητή Νοημοσύνη (AI) μπορεί να ενισχύσει αυτές τις φάσεις της RE. Η εργασία παρέχει μια συστηματική ανασκόπηση των προσεγγίσεων RA και RVV βασισμένων στην AI για την περίοδο 2020 εως 2024, μια περίοδο που χαρακτηρίζεται από ραγδαίες εξελίξεις στις τεχνολογίες AI.

Η ανασκόπηση καλύπτει δραστηριότητες, όπως ταξινόμηση, προτεραιοποίηση και ανίχνευση θεμάτων ποιότητας, όπως η ανίχνευση ασάφειας και ο έλεγχος πληρότητας απαιτήσεων. Στο πλαίσιο της ανασκόπησης, αναλύθηκαν οι προτεινόμενες μεθοδολογίες, τα σύνολα δεδομένων, τα εργαλεία και οι μετρικές απόδοσης, αναδεικνύοντας τόσο τις προόδους όσο και τις προκλήσεις. Μία βασική παρατήρηση είναι ότι οι προσεγγίσεις που εξετάστηκαν αξιοποιούν ποικιλία τεχνικών, όπως Επεξεργασία Φυσικής Γλώσσας (NLP), Μηχανική Μάθηση (ML), Βαθιά Μάθηση (DL) και Μεγάλα Γλωσσικά Μοντέλα (LLMs). Επιπλέον, τα βασικά ευρήματα δείχνουν ότι η AI μπορεί να βελτιώσει σημαντικά την αποδοτικότητα στις φάσεις RA και RVV, παρέχοντας τη προοπτική να επιταχύνει την παράδοση έργων και να μειώσει το κόστος ανάπτυξης λογισμικού, διαμορφώνοντας έτσι μελλοντικές κατευθύνσεις στη μηχανική λογισμικού. Ωστόσο, εξακολουθούν να υπάρχουν περιορισμοί σχετικά με την απόδοση, την ποιότητα των συνόλων δεδομένων και την εφαρμογή σε διαφορετικούς τομείς. Συνεπώς, η παρούσα ανασκόπηση προσφέρει μία επικαιροποιημένη οπτική και χρήσιμες κατευθύνσεις για την περαιτέρω μελλοντική έρευνα και την πρακτική υιοθέτηση προσεγγίσεων βασισμένων σε AI, για τις φάσεις RA και RVV. Συνολικά, η εργασία αναδεικνύει πώς η AI μπορεί να αποτελέσει κινητήριο μοχλό για μελλοντικές βελτιώσεις στην αποδοτικότητα και την ποιότητα της μηχανικής λογισμικού.

**Λέξεις-Κλειδιά**: *Μηχανική Απαιτήσεων, Ανάλυση Απαιτήσεων, Επαλήθευση Απαιτήσεων, Επικύρωση Απαιτήσεων, Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Βαθιά Μάθηση, Μεγάλα Γλωσσικά Μοντέλα*

# 1 *Introduction*

In software engineering, the success of a project depends not only on the quality of its design and implementation but also on the clarity, completeness, and consistency of its initial requirements. Requirements Engineering (RE), being a part of software engineering discipline, is responsible for identifying, analysing, documenting, and validating what a software system should do. The practice of RE, however, is inherently human-dependent, and relies on the expertise, experience, and communication skills of analysts, engineers and stakeholders. These parties must collaborate closely to overcome misunderstandings, negotiate conflicting priorities and requirements as well as define, align and validate the actual requirements of the software to develop. When this orchestration fails, it often cascades into downstream activities and can result in requirements, design or code rework, defects, and significant project delays.

The NaPiRE (Naming the Pain in Requirements Engineering) initiative is a survey series that has been replicated in several countries worldwide. Its empirical findings from the most recent study [1], indicate that underspecified, incomplete, vague or too abstract requirements and communication flaws between the involved parties are among the most frequent problems related to project failures. Within the Requirements Engineering process, the activities that are directly and mostly responsible for these types of problems are Requirements Analysis (RA), which classifies and organizes the input, and Requirements Validation and Verification (RVV), which checks the quality characteristics of requirements. Both are crucial for ensuring that the right (software) product is constructed and built right. However, they can be both time-consuming, prone to errors, and heavily dependent on human effort.

The past decade has seen an explosion of Artificial Intelligence (AI) in a variety of fields, including Natural Language Processing (NLP) and Machine Learning (ML). This evolution suggests that AI could be leveraged to support RE activities. In particular, the broad and growing use of AI may have the potential to support the proper classification of requirements, the identification of ambiguous or vague requirements, the assessment of their completeness, and their prioritization. While these activities are usually performed by domain experts with expert judgment and manual reviews, they could be accelerated or assisted by AI.

Although several reviews have been performed on the use of AI in software engineering in general or in RE, this thesis focuses on two important and key activities of RE, the RA and RVV ones, by conducting a systematic structured review of AI-based approaches applied to RA and RVV. The contributions of this thesis are as follows.

First, a structured classification of the most recent state-of-the-art AI-assisted RA and RVV approaches is presented, organized into multiple hierarchical categorizations so as to provide a complete and extended understanding of what tasks are being automated, how, and by which techniques. Second, a comparative analysis and evaluation of these approaches is

conducted that helps researchers and practitioners to understand which are the best approaches under which sets of criteria and which are the main research gaps and relevant future directions to cover them.
Beyond these contributions, the thesis can be useful in research and practice. Researchers can use the results as a starting point when planning new studies on AI for RE; for example by avoiding already well explored settings and focusing on the gaps that were identified. Practitioners can use the collected evidence to better judge when AI-based support for RA or RVV is feasible, which kinds of tasks are currently mature, and where manual review is still necessary. The work also helps both groups to set more appropriate expectations about metrics, datasets and evaluation depth. In this way, the thesis supports future informed decisions about designing, applying and assessing AI techniques in RE.

The remainder of this thesis is structured as follows. Chapter 2 supplies a background in Software Engineering, Artificial Intelligence and Requirements Engineering. Chapter 3 highlights the research methodology followed in the literature review. Chapter 4 presents the analysis of the selected studies, hierarchical categorizations of them and their respective comparative evaluation. Chapter 5 highlights the limitations identified during the analysis and supplies future work directions for advancing and improving or optimising the subject studied. Finally, Chapter 6 concludes this thesis.

# 2 *Background*

This chapter is divided into three main parts: Section 2.1 introduces SE, Section 2.2 examines RE process with emphasis on RA and RVV, and Section 2.3 presents AI and its relevant subfields. Together, these sections provide the theoretical background for the thesis and enable a proper understanding of its main contributions as these will be unfolded in later chapters of this thesis report.

## 2.1 Software Engineering

### 2.1.1 Software Engineering Definition

Software engineering (SE) is the application of engineering principles to methodologically develop and maintain software systems. The software development process includes a specific number of phases from requirements elicitation until ongoing maintenance of the software after its deployment. The core objective is to develop software that functions reliably and remains maintainable while fulfilling (continuously evolving) user requirements.

Ian Sommerville defines software engineering as "an engineering discipline that is concerned with all aspects of software production, from the early stages of system specification through to maintaining the system after it has gone into use" [2]. In fact, SE requires project planning and quality control mechanisms alongside core design and programming activities to manage the entire development process.

Through SE complex systems can be managed via the use of well-defined processes and specialized tools. The field enables both structured plan-driven as well as agile development approaches. As software becomes more important in our daily lives, using these engineering practices is essential for building reliable and useful systems.

### 2.1.2 Software Engineering Activities

SE involves a complete framework of operations that targets the organized development and maintenance of software systems. Each activity throughout the software lifecycle plays a part in developing software solutions that are high-quality while remaining cost-efficient. The key activities typically included in any software development process are described next.

#### 2.1.2.1 Core Software Engineering Life cycle Phases / Activities

*1. Requirements Engineering*

Stakeholders' needs are identified and analyzed during this foundational stage. The process involves phases like requirements extraction, analysis, specification, verification and validation. Usually, the entire project depends on establishing accurate requirements to ensure its success. Further analysis of this phase follows later in the current chapter.

*2. System / Software Design*

Design activities convert requirements into both architectural and detailed design plans covering both structural and behavioral aspects. The high-level design establishes the main system components with their relationships whereas the detailed design describes internal constructs like algorithms and data models.

*3. Implementation (Coding)*

Executable code is written in a suitable programming language to bring the design to reality. The implementation phase usually follows coding standards while integrating system components to create a functional whole.

*4. Verification and Validation (V&V)*

VV activities verify that the software functions according to its intended specification. Verification proves that the software has been constructed properly while validation checks whether it satisfies the user requirements. Unit testing, integration testing, system testing, and acceptance testing are key activities that belong to this phase.

*5. Deployment*

The deployment phase consists of launching the software on production systems usually done after user training sessions and system configuration tasks.

*6. Maintenance and Evolution*

After deployment the software might typically require additions or modifications due to new requirements emerging, existing requirements updating, late identified bugs that need resolution, or environmental changes.

#### 2.1.2.2 Software Engineering Supporting Activities

Further supporting activities within the framework of SE act proactively to ensure organization, consistency, and quality across all stages of software development. In particular, the following supporting activities play crucial role in sustaining effective software delivery.

*1. Project Management*

Project management is a supporting activity of SE which involves project planning, project scheduling, risk management, and progress tracking. It ensures that technical efforts align with deadlines, budgets, and stakeholder expectations.

*2. Configuration and Change Management*

This activity manages the different versions of documents as well as of code and additional artifacts. The process maintains traceability and consistent integration across development and evolution/maintenance phases while supporting collaborative work environments.

*3. Software Quality Assurance*

Software Quality Assurance covers the entire software lifecycle to maintain compliance with established quality standards and procedures as well as testing practices. SQA includes activities, such as audits, reviews, documentation control, and process compliance checks, to maintain high quality standards and practices across development.

*4. Measurement and Process Improvement*

Organizations can drive continuous improvement by monitoring and analyzing product and process metrics to uncover inefficiencies and bottlenecks so as to improve existing development processes based on well-defined goals.

### 2.1.3 Software Engineering Importance

Software Engineering plays a critical role in creating high-quality, scalable, and maintainable software systems in today's rapidly grown digital world. Critical infrastructure and services depend on software functioning properly, which requires disciplined engineering practices to achieve reliable performance and future adaptability.

SE manages complexity by using clear design structures, breaking systems into smaller parts, reusing existing software and focusing on the most important details through analysis. This makes it easier to build and maintain large systems. It also improves quality by checking the software at every stage and following quality standards throughout the development process.

Furthermore, SE helps in reducing costs and finishing projects faster by detecting problems either in requirements, design or code early, reusing proven solutions, and by conducting careful planning. It also helps teams work together better by using common rules, documentation practices along with version control. Maintenance remains essential because software systems need to adapt to new requirements. SE supports this with clear documentation and change management strategies. During the development of safety critical systems SE maintains regulatory compliance by performing rigorous testing and applying formal methods. [55]

Lastly, SE enables innovative technology development by supplying essential foundations for AI and other emerging systems like cloud computing and cyber-physical systems, ensuring they are built reliably and sustainably. [2], [3]

## 2.2 Requirements Engineering

Requirements Engineering (RE) is one of the most important phases conducted in the software engineering process. In this process, system's needs and constraints are defined and documented, aiming to create a well-structured set of correctly registered requirements. [22]

The ISO/IEC/IEEE 29148:2018 standard serves as the authoritative framework for RE, defining this discipline, categorising its processes, and specifying detailed activities for each RE phase. It defines RE as an interdisciplinary process that connects acquirers and suppliers to agree on, document, and manage system requirements throughout a project's life cycle.

According to the standard, RE involves activities, such as eliciting, analysing, specifying, validating, and maintaining requirements, which collectively support the creation of the Software Requirements Specification (SRS). The SRS represents the transformation of the initial stakeholder needs, which are often vague or inconsistent, into a well-structured and verifiable set of valid requirements, further explained below.

Every set of requirements must be linked to a system, software, or service. The intent is to establish a common understanding among stakeholders (such as acquirers, users, customers, operators, and suppliers) regarding the system's intended functions, qualities, and limitations. Furthermore, it should be ensured that the requirements can be tested against real-world needs and serve as a baseline to verify subsequent design and implementation procedures within the software engineering process. [5], [10].

### 2.2.1 Requirements

The software specification is an overall document that includes the software requirements set and constitutes the primary output of the RE process. It is usually written in the form of natural language (usually such form is unstructured, but other forms also exist, including structured ones, like the actor-action or user story format) [25] and specifies how the system should behave along with constraints over the system's functional or non-functional characteristics [5].

There are different types of requirements at different levels of abstraction, with a content that depends on the perspectives of the stakeholders posing them, where a stakeholder is anyone with an interest in the system, such as end users, customers, project managers, or the organization. At the highest abstraction level, a business goal that expresses high-level requirements and constraints derived from policies, business procedures, or strategic objectives of the developer organization is considered as a *business requirement*. At a lower abstraction level, a *user requirement* describes the services or functions that users expect the system to provide and the conditions under which it should operate. Finally, there are detailed requirements that specialise the user ones that are called *system requirements*. Requirements can be further classified as *functional* and *non-functional* depending on the system aspect that they concern. Functional requirements describe the (functional) behavior that the system must have, while non-functional (or quality [5]) requirements describe the qualities that characterize such behavior (transportability, survivability, flexibility and portability), or the qualities of the system itself [7] or the constraints it must meet.[5]

In addition, it is possible that certain constraints, representing mandatory conditions, are introduced concerning either single requirements or sets of requirements, which limit the solution space. These may include interfaces to pre-existing systems (e.g., fixed formats, protocols, or content), physical limitations (e.g., fitting a component within a constrained hardware space), compliance with laws, limitations in available time or budget, reliance on existing technology platforms, maintenance restrictions, or the capabilities and limitations of end-users and operators.

Requirements should be written using standard, commonly accepted phrasing, based on the ISO/IEC/IEEE 29148:2018 guidelines. They need to focus on what the system should do, not how it should be built or how it will implement its main functionality, and they must avoid including design decisions. Also, unclear, vague, or unverifiable language should be avoided, since it can lead to misinterpretations or the inability to verify the respective requirement later on.

More specifically, the word "shall" is used for mandatory requirements. On the other hand, words like "is," "are," or "was" indicate non-requirements, as they just give background information. The verb "will" is typically used to describe future intentions, but not something that is binding. "Should" is used for preferred system behaviors or goals that are recommended but not required. These are often referred to as preferences. "May" is used to indicate optional features that are permitted but whose inclusion is entirely at the developer's discretion. Although both "should" and "may" describe non-binding provisions, "should" implies a recommendation, while "may" signals that the feature is allowed without any particular expectation. The word "must" should be avoided, because it might confuse readers or be mistakenly treated as a requirement when it is not meant to be.

In addition, negative requirements expressed as "shall not" should be rephrased into a positive form whenever possible. It is also recommended to use the active voice (the system shall) rather than passive forms like (it is required that). Finally, vague phrases, such as "shall be able to", should be replaced with clear and verifiable statements of capability. [5]

### 2.2.2 Importance of Requirements Engineering

The importance of RE lies in its foundational role in ensuring that a solution aligns with stakeholder needs and business goals. Careful planning and execution of requirement-related activities are essential to the success of a project. If sufficient time is not allocated for RE or if the associated tasks are poorly planned, the result is often a solution that fails to deliver value to stakeholders or requires significant re-work to meet the actual stakeholder needs. Proper RE helps avoid inconsistent and incomplete requirements, improves communication among stakeholders, supports traceability, and ultimately contributes to higher quality and more cost-effective solutions [6].

RE also plays a vital role for software change management after software has been deployed. It clarifies what the system is expected to do and what limits it must meet. It also assists future developers in understanding what the system was meant to do, which is important for fixing problems and making updates, especially in cases when the current developers are not the same who realised changes or implemented the system in the past [6]. Furthermore, when requirements are properly documented and traced, it becomes easier to estimate the impact and cost of changes. Traceability links requirements to related design elements, test cases, and code, making it possible to identify which parts of the system will be affected by a requirement change and how much effort is needed to perform the necessary work. [6], [56]

### 2.2.3 Requirements Engineering Process

Having established RE and requirements, we now examine in more detail the phases of RE, including Requirements Analysis and Requirements Verification & Validation that are the focus of this study. In parallel, it is worth mentioning that there is noticeable terminological variability in the literature and the different related work implemented in the area of RE

regarding the phases and activities of RE. Therefore, it is very important to strictly define which are the RE phases studied in this thesis and which activities are involved in these phases.

The RE process includes the following phases: feasibility study, requirements elicitation, requirements analysis, requirements specification, and requirements verification & validation.

These phases are iterative such that the activities (involved) are interleaved, as shown in Figure 1. The activities are sometimes revisited throughout the software engineering lifecycle. For example, new requirements might arise during the verification & validation phase, which would trigger the activity of specifying new requirements (requirement specification), which typically takes place within the requirements elicitation or analysis phase. In such cases, the process loops back to earlier phases to update the requirements set and ensure that the changes are reflected consistently throughout the system.

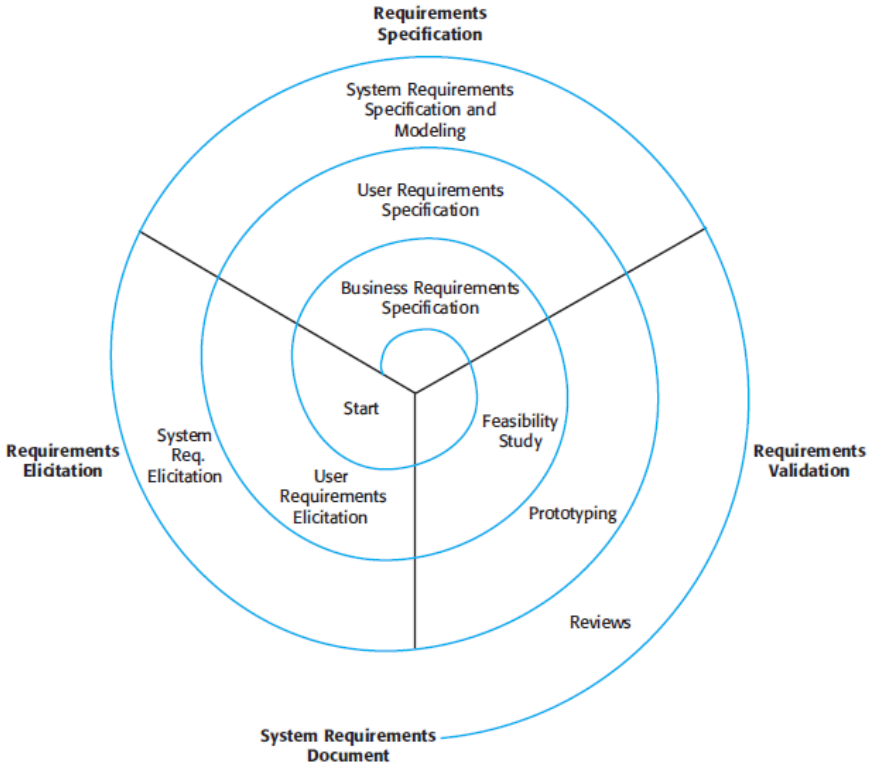


*Figure 1. Illustration of iterative requirements engineering phases*

### 2.2.3.1 Feasibility Study

Feasibility study's role is to assess the practicability of the proposed system. It checks the business value, achievability within time and budget, and compatibility with the current technical landscape. The activities of this phase are assessing economic and technical feasibility, estimating costs and benefits, and assessing the compatibility with business strategies. [2], [6]

### 2.2.3.2 Requirements Elicitation

Following the feasibility assessment, the next step aims at identifying and understanding what stakeholders expect from the system. Requirements elicitation is the phase focused on identifying and capturing stakeholder needs. Before requirements can be further analysed, it is necessary to determine who the stakeholders are and understand their expectations from the system functionality, the expected performance of the system, and the operational constraints that must be respected. Requirements might also originate from the (application) domain, or the interaction between systems.

The aforementioned goals are achieved through the analysis of the operational context plus the direct elicitation and identification of implicit stakeholder needs based on domain knowledge. These needs are systematically transformed into requirements, considering the system constraints, software life cycle considerations, and critical quality factors, such as safety, security, and reliability. Common ways to uncover requirements include stakeholder interviews and workshops, observation (including ethnographic studies) and technical documentation reviews. [3], [6], [12]

#### 2.2.3.3 Requirements Analysis

The next phase is requirements analysis (RA), sometimes referred to as prioritization and negotiation in the literature. It includes several key activities applied to the requirements gathered during elicitation. These include the classification and the organization of requirements and their prioritisation. Analysts also identify overlaps or conflicts and resolve them through negotiation with stakeholders, while defining performance measures enables the assessment of requirement achievement. Typically performing such activities through the analysis phase results in informal or preliminary requirements documents. [2]

The classification and organisation activity takes as input the preliminary unstructured version of the requirements. Its main purpose is to cluster the requirements into groups and organise them within each group. The most common basis for this grouping is the system architecture, with requirements allocated to relevant sub-systems.[2] Also, additional schemes are often applied, such as separating functional from non-functional requirements. [25],[35],[36]

In the prioritisation activity, the ranking or weighing of the requirements is performed to indicate a priority, a timing or other relevant importance. Prioritisation is correlated to the classification activity. The higher the priority, the more essential the requirement is for meeting the overall goals of the software. A requirement is often classified on a fixed-point scale, such as mandatory, highly desirable, desirable, or optional. [6]

The negotiation between competing or conflicting requirements, includes identifying and resolving conflicts. Conflicts can take the form of, i.e. mutually incompatible features, antithesis over requirements or disputes between desired functionality or constraints, available budget and delivery schedule. Typically, stakeholder(s) consulting is necessary to reach a consensus on an appropriate trade-off, while the adoption of a prioritisation technique can facilitate focusing on satisfying the most critical requirements first and then making decisions about what to defer or modify during negotiation [56]. It is often important for contractual reasons that such decisions are traceable to the actual stakeholder. [2]

Performance measures are criteria that enable the assessment of whether a requirement has been technically achieved. During the analysis phase, such measures should be identified to support the evaluation of whether each requirement fulfills the intended stakeholder outcomes. Although often associated with non-functional requirements, performance measures can also be applied to functional requirements where quantifiable criteria are needed. [5]

Requirements analysis plays a critical role in shaping high-quality software by refining and evaluating stakeholder needs to ensure they are complete, consistent, feasible, and well-understood. This step helps identify and negotiate conflicts, classify requirements, and define measurable criteria for later verification. The refinement and correction activities prevent defects from entering the later stages of development, reducing rework, cost, and delay. It is highlighted that a faulty analysis can lead to ambiguity and incompleteness, two of the most common causes of software failure. By successfully resolving these in the analysis phase, not only is software failure risk mitigated, but also a common understanding of the system goals is produced. [2], [3]

#### 2.2.3.4 Requirements Specification

Once requirements are elicited and analyzed, they are formally documented during the requirements specification activity. The aim of this phase is to produce a Software Requirements Specification (SRS) document that captures all user and system requirements. While some requirements are already written and refined during the analysis phase, the goal of the specification phase is to formally document these requirements in a structured format.

User requirements describe what the users expect the system to do. They are usually written in natural language and should focus on the external behavior of the system, structured notations, or technical detail. Simple tables, forms, and intuitive diagrams can be used to improve clarity.

System requirements are more detailed and technical. They expand upon the user requirements and are used by developers as a basis for system design and by clients as a reference in contractual agreements. These may be written in natural language but often include structured formats, graphical models, such as UML sequence or state charts, or even mathematical specifications for critical systems.

Although the system specification should ideally avoid implementation details, some design information may be necessary to include. For example, to address external interfaces, architectural constraints, or safety regulations. [2], [3]

#### 2.2.3.5 Requirements Verification & Validation

In the RE process, requirements validation is an explicit and critical phase, while requirements verification is a broader quality assurance activity that also applies within RE. Although these terms are often used interchangeably, they represent different but complementary perspectives on requirements quality.

Similarly to requirements validation, the requirements verification process primarily focuses on the quality of the requirements. According to Sommerville [2] and BABOK [6], this phase ensures that the documented requirements are clear, consistent, feasible, traceable, and compliant with agreed formats and standards and provides a sound basis for subsequent design and development activities in the software lifecycle. Within RE, verification is typically conducted through structured reviews, quality checklists, formal analysis, traceability checks, and compliance checking. In this context, verification addresses quality factors from a syntactic perspective, further described below. [2], [6], [13]

Requirements validation, in contrast, ensures that the documented requirements are the right requirements, that they correctly reflect stakeholder needs, business goals, and the intended operational environment. [2], [6] Validation activities confirm qualities, such as correctness, relevance and completeness of the requirements from a semantic point of view, focusing on justifiability and alignment with real-world needs. Typical validation techniques include stakeholder reviews, prototyping, simulation, acceptance criteria definition and agreement.

The quality of requirements can be assessed across multiple characteristics, with each activity focusing on different aspects of the same characteristic.

**Correct**: In verification, a requirement is correct if it accurately describes a necessary system property, which represents the syntactic point of view. In contrast, a requirement is correct in validation if it truly reflects an actual stakeholder expectation, representing the semantic point of view.

**Complete**: In verification, a requirement is complete if it contains all the necessary information for implementation without needing additional clarification, while there are no missing dependencies, and all necessary conditions are defined. A complete set of requirements may be assessed across several dimensions, including document completeness (all expected parts present), goal completeness (alignment with business objectives), scenario completeness (coverage of usage and operational cases), environment completeness (consideration of external constraints), interface completeness (definition of interactions with external systems and actors), and functional/non-functional completeness (coverage of required behaviors and quality attributes). In validation, completeness refers to whether all stakeholder needs and relevant scenarios are indeed captured.

**Consistent**: In verification, consistency means that requirements do not contradict each other or create internal conflicts within the specification. In validation, consistency means that stakeholder expectations are not in conflict with one another and that the requirements reflect an agreed view of the system's objectives.

**Unambiguous**: In verification, a requirement is unambiguous if it is written in precise, measurable, and clear language that avoids vague terms and ensures syntactic clarity. In validation, unambiguous means that all stakeholders interpret the requirement in the same way, avoiding semantic misunderstandings during communication.

**Feasible**: In verification, a requirement is feasible if it is realistic, given the technology, time, and resources available. In validation, feasibility refers to whether the requirement is practical and valuable from a business and operational perspective.

**Appropriate**: In verification, a requirement is appropriate if its level of detail and abstraction matches the system level it describes, avoiding premature design decisions or over-specification that could restrict implementation creativity.
In validation, appropriate means that the content and intent of the requirement are relevant to stakeholder expectations and the purpose of the system, not omitting what is truly needed.

**Traceable**: In verification, traceability means that each requirement has a clear origin and can be linked to verification artifacts, such as test cases or acceptance criteria, ensuring accountability and the ability to confirm implementation. In validation, traceability ensures that requirements are linked back to stakeholder needs and business goals, confirming that every documented requirement is aligned with the intended purpose.

**Delimited**: In verification, delimitation means the specification clearly defines scope, included and excluded functionality, and constraints, while in validation it ensures that these documented boundaries align with stakeholder intentions and shared understanding.

**Readable**: In verification, requirements must use a clear, structured, and standardized language while in validation, it means that diverse stakeholders, such as developers, testers, and business analysts, can consistently understand and interpret them.

**Modifiable**: In verification, modifiability ensures that requirements are structured so that changes can be made without affecting (i.e., causing unnecessary changes to) other parts of the document, while in validation it ensures that evolving stakeholder needs can be incorporated without compromising overall alignment.

**Verifiable**: In verification, a requirement is verifiable if tests, inspections, or demonstrations can confirm its satisfaction, while in validation it ensures that requirements are stated so stakeholders can judge whether their needs are truly met.

**Prioritized**: In verification, prioritization ensures that the documented ordering of requirements follows agreed criteria, such as urgency or dependencies, while in validation it confirms that the ranking reflects stakeholder value and business goals.

**Endorsed**: In verification, endorsement means that requirements have passed systematic review and approval procedures, while in validation it confirms that stakeholders explicitly accept them as valid, complete, and aligned with (their) objectives.

**Singular:** In verification, a requirement is singular if it expresses only one discrete capability, condition, or constraint, ensuring that it can be uniquely identified. In validation, singular means that each requirement represents a single stakeholder intention or objective, rather than combining multiple needs or decisions within the same statement.

**Necessary:** In verification, a requirement is necessary if it defines an essential capability, constraint, or quality factor without which the system would exhibit a deficiency that cannot be satisfied alternatively by other requirements. In validation, necessary means that the requirement corresponds to an actual stakeholder or operational need that remains relevant and current, ensuring that no obsolete requirements are retained.

The more quality factors are satisfied, the higher is the overall quality of requirements. However, it is impossible to reach a point of perfection where all factors are completely met,

as different factors can pull in opposite directions so trade-offs between them can exist. For example, the unambiguous criterion/factor promotes meticulous requirements, possibly using mathematics, which may compromise or conflict with the readable criterion.

Verification techniques support the systematic assessment of requirements against quality criteria. The most common technique is the review, where the system requirements are assessed for errors, omissions, invalid assumptions, lack of clarity and deviation from accepted practice. The review might be performed by different roles involved in the project, like software engineers and system architects who determine whether the quality of the requirements is sufficient. The reviewers can be assisted when conducting the review activity via checklists or quality criteria. It must be noted that reviews are also conducted in the context of validation. However, in this case, such reviews attempt to assess whether the requirements reflect the stakeholder needs and intentions so they are conducted by or involve stakeholders of the software under development. [56]

Quality checklist is a structured tool used during inspections or walkthroughs to systematically verify that each requirement meets predefined quality standards (e.g., clarity, consistency, completeness). It helps uncover omissions, ambiguities, or formatting issues early in the process.

Traceability checks ensure that requirements are linked to related artifacts (design elements, test cases, etc.). Techniques like a Requirements Traceability Matrix (RTM) provide visibility both forward and backward across the development lifecycle. It must be highlighted that this technique also covers validation as requirements have to be linked to their origins (e.g., the stakeholders posing them reflecting their needs).

Compliance checking involves verifying that requirements conform to relevant regulatory standards, legal norms, or organizational policies (e.g., data protection regulations). It ensures the requirements are aligned with mandatory norms and can include automated or manual audit processes. [2], [56]

On the other hand, validation techniques confirm that requirements are the right requirements from a stakeholder and business perspective. Justifiability ensures that each requirement has a clear source and rationale, explaining why it is needed for the system or project. Unnecessary or arbitrary requirements not (directly or indirectly) related to a project goal or a constraint, or based on personal stakeholder preferences or their assumptions, should not be included in the specifications. As noted in the literature, personal stakeholder preferences or requirements may appear during elicitation, but they can harm the development process if they do not contribute directly to the project goals. In the context of validation, the following techniques can be applied.

By implementing prototypes, alignment between software engineers and stakeholders can be enabled. This happens through the demonstration of a sample of requirements, to assist with clarifications and necessary adjustments of requirements during the validation phase to guarantee that requirements really meet the real stateholder needs and intentions.

Simulation is the creation of a simplified model of a real system to study its behavior and facilitate requirements validation. It focuses on the aspects that matter for the analysis, leaving out unnecessary details, and can be used to validate a system and how it would

perform under different conditions. Simulations usually depict event scheduling or process interaction within a system and they generally include entities, events and activities. Please note that simulation servers the purposes of both validation and verification. In the context of validation, it enables to check whether the system meets the real needs of stakeholders. In the context of verification, it enables to check whether the system behaviour matches the requirements, not exhibiting logical inconsistencies, missing steps or contradictions, as well as the requirements feasibility, especially focusing on whether they lead to feasible and correct system actions.

Last, it must be highlighted that acceptance criteria are measurable conditions that requirements must satisfy to be considered complete and acceptable, essentially serving as criteria for requirement acceptance. In essence, their definition is not a validation technique but rather a task that supports software validation in general as they can be utilised for the specification and implementation of acceptance tests where acceptance testing is a validation activity. However, as indicated previously, they can also support requirements validation, especially in the context of applying validation techniques like stakeholder reviews and walkthroughs. [2], [13], [56]

In summary, both activities are essential in RE phase, and although they consist of distinct tasks, requirements verification and validation (RVV) contribute to early requirements defect detection, stakeholder alignment, and ensuring that the requirements provide a solid foundation for design and implementation. Requirements verification safeguards the form and quality of the specification, while requirements validation ensures the external correctness and relevance of its content.

### 2.2.4 Challenges in Requirements Engineering Practice

Although RE and its phases are theoretically well-structured and analytical, in practice there are certain factors that can lead to problems and negatively affect the software engineering process.

Often, the effort and resources allocated in the RE process are minimized due to some underlying reasons, such as limited time, budget, resource constraints, insufficient training and skills, uncertainty and ambiguity in early stages, which teams consider challenging to handle. Another contributing factor is the adoption of approaches that prioritize implementation early or promote minimal documentation in the software development process in an effort to produce solutions faster. For example, agile methods favor minimal documentation, which can lead to underspecified and undocumented requirements, especially non-functional requirements, weakening the RE process [31].

Unfortunately, such practices lead to significant challenges in the later stages of development as issues related to inconsistent, incomplete and incorrect requirements become increasingly difficult to resolve, often resulting in higher development costs, project delays, and lower-quality software systems. [22]

Among the most frequent problems in RE process are incomplete or hidden requirements, following the communication gaps between the involved stakeholders and the rest of the

project team (for example due to stakeholders time constraints to invest on explaining what they need), both of which can potentially lead to project failure.

Furthermore, moving targets (changing goals, business processes and / or requirements) and requirements evolution over time, are also frequent issues in RE. Large organizations that follow plan-driven development might face negative impact, such as project delays, extended engagement of resources beyond original plan, and customer dissatisfaction, due to requirements changing mid-project and because these approaches assume requirements will remain relatively stable once specified, and do not usually handle late changes. Other factors complicating the process are underspecified requirements that are too abstract and might be interpreted differently by stakeholders, time and resources constraints, inconsistent requirements, missing access to domain knowledge or lack of domain or technical expertise. [9], [10]

In addition, common issues related to the natural language-based specification of stated requirements' ambiguity, incompleteness, inconsistency and incorrectness, which lead to misinterpretations, untestable requirements, untraced requirements to their origin, no consensus among stakeholders on what needs to be built, and conflicting requirements. In such cases, it is mandatory to identify these gaps and refine the requirements accordingly. [22] These challenges, particularly in analysis, validation and verification activities, highlight the need to adopt intelligent, automated or semi-automated solutions to support them. The next chapters investigate which AI-based techniques have been proposed to address at which degree such issues in the Requirements Analysis and Requirements Verification & Validation phases.

## 2.3 Artificial Intelligence

### 2.3.1 Artificial Intelligence Definition

Artificial Intelligence (AI) refers to a branch of Computer Science that focuses on developing methods to embed intelligent, automated behavior into computer and software systems, which includes cognitive activities, such as perception, analysis, and reaction. Several definitions have been proposed to capture the concept of intelligent behavior, in particular characteristics, such as autonomy, adaptability, and human-like reasoning. AI is defined as a discipline or branch that focuses on developing methods and algorithms that enable producing or training machines that display human-like behavior by reasoning and learning from experience while solving problems and adapting to new situations [4]. The International Organization for Standardization (ISO) defines AI as computer systems which execute tasks typically requiring human intelligence. Such tasks include language understanding, pattern recognition, decision-making, or learning from data that require some form of intelligence [52]. A definition shared by IBM states that AI is a technology that allows a machine to complete tasks that require human intellect or capabilities, such as understanding, reasoning, creativity, and decision-making, by exploiting machine learning and deep learning models [41]. These definitions help us understand the three primary characteristics of AI: (1) it can

function autonomously by also exhibiting human/intelligent behavior, (2) it can adapt to various circumstances and new situations, and (3) it can process large quantities of complex information at the same level as humans.

### 2.3.2 Types of Artificial Intelligence

Artificial Intelligence can be classified based on the cognitive capabilities it exhibits. This perspective focuses on the extent to which AI systems can perceive, reason, learn, generalize, and make decisions, providing a structured understanding of their levels of intelligence and autonomy. In this respect, such a classification is called capabilities-based classification, which comprises mainly three AI types: *Artificial Narrow Intelligence (ANI)*, *Artificial General Intelligence (AGI)*, and *Artificial Superintelligence (ASI)*.

Artificial Narrow Intelligence (ANI), also called Weak AI, functions within specific tasks or limited domains. It does not understand situations, and it cannot leverage knowledge outside of its specific domain. Most AI systems today are ANI. Some examples include voice assistants like Siri and Alexa, email applications with spam filters, and facial recognition systems. Those AI systems are very effective in their domains, but simply lack the capabilities to do anything outside of their training and thus of their domains.

Artificial General Intelligence (AGI) represents a theoretical and advanced category of artificial intelligence systems. AGI systems would possess the capability to perform similar intellectual tasks as humans while demonstrating adaptability, reasoning skills and learning abilities across various fields/domains. Although there is substantial research and interest in AGI, it still remains unrealized while its development continues as a long-term goal within AI research.

Artificial Superintelligence (ASI) refers to a hypothetical, advanced stage of AI development where machines surpass human abilities in every cognitive area including logic processing and creative thinking as well as emotional understanding and social skills. ASI systems would exceed human performance in multiple tasks and exhibit advanced decision-making with innovative capabilities and superior adaptive abilities beyond human capability. AI development at this advanced stage remains a theoretical idea which generates philosophical discussions and ethical evaluations. [41], [52]

### 2.3.3 Core Subfields of Artificial Intelligence

Artificial Intelligence includes several main subfields that shape both its theory and its practical use. Each subfield focuses on specific techniques, ways of learning, or problem-solving methods that work together to make AI systems effective.

#### 2.3.3.1 Machine Learning

Machine Learning (ML) stands as a core field in AI which focuses on developing systems that enhance their task accuracy and performance through experience (exposure to data) without direct programming instructions. A system learns when its task performance

improves due to data exposure or environmental interaction [4]. Most ML algorithms operate through three main parts: a decision process to make predictions or classifications, an error function to evaluate how accurate those predictions are, and an optimization process that iteratively updates the model parameters to minimize the error and improve accuracy. Optimization can be performed using methods like gradient descent and its variants, such as Adam optimizer [102], or through metaheuristic techniques like Particle Swarm Optimization (PSO), which are inspired by collective behavior in nature. [42] , [89]

A ML pipeline generally includes several key steps:

- problem definition, where the task and goals are clearly specified.
- data collection, which involves gathering relevant and representative datasets.
- data preprocessing, where raw data is cleaned, transformed, and prepared for modeling.
- model training, in which the algorithm learns patterns from the processed data and
- performance evaluation, where the trained model is tested against defined metrics to assess its accuracy, generalization, and suitability for deployment.

Each pipeline step is critical, especially in domains like SE where data quality and representation vary widely depending on the software lifecycle stage [7].

The value of ML lies in its ability to adapt in situations where static rule-based systems fail. For instance, in dynamic environments or when dealing with vast and heterogeneous (software) data, such as Git logs, user reviews, or issue trackers, ML models provide scalable and flexible solutions that learn optimal data and interaction patterns from historical records to guide better decision-making [7].

ML learning methods (focusing on how ML systems are trained and learn from data) can be broadly categorized into the following types:

- *Supervised Learning or classic ML* depends on human intervention to allow the system to learn and perform specific tasks [43]**.** The supervised learning process entails training a model using a labeled dataset which includes input data matched with the correct output labels. The algorithm develops its ability to connect inputs with outputs by making repeated weights adjustments. This adjustment is part of the cross-validation process to ensure that the model does not overfit (performs very well on training data but fails to generalize to new data) or underfit (performs poorly because it has not learned enough from the data). Supervised learning is generally divided into two categories of algorithms, the Classification and Regression ones. In the first category, algorithms group data based on the prediction of categorical labels that derive from input data. In the latter category, algorithms detect relationships between two or more variables to predict a real or continuous value [10]. Some methods commonly used in supervised learning include neural networks, Naïve Bayes, linear regression, logistic regression, random forest, and support vector machine (SVM). [4], [21].

- *Unsupervised Learning* analyzes unlabeled data to discover hidden structures or to classify data (points) based on their similarities (clustering), to find relationships between variables in the dataset (association), or reduce the number of data dimensions (dimensionality reduction). This category comprises unsupervised learning methods, which include clustering algorithms, such as k-means and DBSCAN, together with dimensionality reduction techniques like principal component analysis (PCA). [7], [24]

- *Reinforcement Learning* allows a computer or agent to determine the best behaviour by interacting with its surrounding environment and receiving rewards or penalties as feedback during trial-and-error processes. The goal of the agent involves accumulating the highest possible rewards through time. Examples of widely used methods include dynamic programming, Monte Carlo, temporal difference learning and its variations, such as SARSA and Q-learning, policy gradient methods, and actor–critic methods. [86], [87] Dynamic fields, such as robotics alongside gaming, autonomous control systems and natural language processing, demonstrate high utilization of Reinforcement Learning as their respective environments require adaptive decision-making processes [7], [52].

- *Semi-Supervised Learning* acts as a hybrid approach which uses both labeled and unlabeled data to boost learning performance, when labeled data is scarce or costly to obtain/gather. The model to produce is initially tained on the limited labeled data set to direct the learning process. Then, it is used to extract features or conduct classification in a larger unlabeled data set. Semi-supervised learning addresses supervised learning algorithms' challenge of insufficient labeled data and proves beneficial when labeling enough data becomes too expensive [36], [42].

- *Self-Supervised Learning* enables models to gain insights from data without depending on human-created labels. The model generates pseudo-labels or supervisory signals from raw input data to create its own supervision instead of relying on human-provided labels. [45]

In the following, two well-known and widely used ML methods are analysed.

**Artificial Neural Networks**

Artificial Neural Networks (ANNs), also known as Neural Networks, are a family of machine learning models inspired by the human brain. They form the computational foundation of deep learning (as analysed below). They consist of layers of nodes (neurons), including an input layer, one or more hidden layers, and an output layer. Each connection has a weight that can be adjusted during the training of the model to minimize the prediction error. Neural networks are used in tasks like pattern recognition and play an important role in applications, including natural language translation, image recognition, speech recognition, and image creation. They support supervised learning as well as unsupervised and semi-supervised learning. While ANNs can be used in both shallow and deep architectures, networks with

multiple hidden layers are called Deep Neural Networks (DNNs), and they form the core of Deep Learning.

**Deep Learning**

Deep Learning (DL) or deep neural networks, rely on the architecture of ANNs with multiple hidden layers, to learn from large amounts of unstructured raw data. Unlike classic machine learning models that require humans to manually select important features, deep learning can automatically learn the best features from the raw data during training [41], [52]. As such, in DL, the manual intervention required is minimized in comparison to classic ML, due to feature extraction automation. DL also has the capability to handle large datasets which is particularly beneficial for organizations' unstructured data, which is estimated to be over 80% of the total organization data. More data points are required to improve accuracy in DL while ML models can generally rely on less data but are less scalable and require more feature engineering and manual intervention [43].

DL models are especially useful for tasks where data is high-dimensional or unstructured, such as images, speech, and natural language. DL models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been used in software engineering for identifying non-functional requirements, predicting bugs, or summarizing code automatically [22], [25].

#### 2.3.3.2 Natural Language Processing (NLP)

Natural Language Processing represents the AI field focused on developing machine capabilities to understand, interpret, and generate human language. It combines computational linguistics, machine learning, and deep learning to analyze text at multiple levels: Natural Language Processing analyzes language through phonetic, morphological, lexical, syntactic, semantic, and pragmatic dimensions [4]. Several NLP tasks typically help process human text and voice data in ways that assist the computer in making sense of what it is ingesting. Some of these tasks include coreference resolution, named entity recognition, part-of-speech tagging (POS tagging) and word sense disambiguation. [46]

RE has transformed NLP into a specialized area referred to as NLP4RE. As requirements artifacts, such as specifications, interview notes, or legal documents, use natural language writing styles they become prime subjects for automated linguistic analysis. In this respect, to process such artifacts, NLP techniques and tools along with linguistic resources are applied. [26].

There are three major types of NLP approaches:

- The first one is *Rule-Based NLP*. Such approaches/methods apply predefined grammatical rules, often arranged in a decision tree: a step-by-step flow where each condition (e.g., an if-then rule) leads to the next decision. The tree determines which

rules to apply and in what order. Such methods cannot learn from examples and work only for very specific tasks.

- The second type is *Statistical NLP*. Statistical methods map linguistic elements (e.g. words, grammar) to mathematical representations, enabling methods, such as part-of-speech tagging and language modeling or regression.
- The final type is *Deep Learning NLP*, which has become the dominant approach in NLP and uses deep neural networks trained on big amounts of raw data (text or voice), to extract patterns and features and become more accurate. Models like Recurrent Neural Networks (RNNs), Sequence-to-Sequence (seq2seq) architectures, and Transformer models (e.g. BERT, GPT) have greatly advanced NLP performance. Transformer models, in particular, form the backbone of modern Large Language Models (LLMs), such as GPT, BERT, and LLaMA. In the sequence, we explicate the main types of models utilised in Deep Learning NLP:
  - *RNNs* are neural networks designed to work with sequential data. They keep a memory of previous inputs to understand the context and make predictions about the current input in the sequence. RNNs can struggle with long-term dependencies, so methods like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) were developed to address this issue. Bidirectional RNNs (BRNNs) consider both past and future context, and encoder-decoder RNNs are used for tasks like machine translation.
  - *Sequence-to-sequence models (seq2seq)*, are built on RNNs and have mostly been used for translating text from a domain to another domain, for example from German to English, learning the ways words and phrases of different languages relate to each other.
  - *Transformer models*: By utilizing self-attention and tokenisation, transformer models can evaluate word relationships without considering their position in a sentence. These models can be trained by using self-supervised learning. Such models improve context management capabilities particularly when working with extended text sequences. Google's BERT (Bidirectional Encoder Representations from Transformers) stands out as a model that reads text bidirectionally and uses self-supervised learning during training. BERT has become foundational for many current NLP applications, including Google's search engine where BERT is until now its basic component. Additionally, an extension of BERT, SpanBERT further enhances contextual understanding by pre-training on contiguous text spans instead of individual tokens. This allows it to capture relationships between phrases more effectively. [93]
  - *Autoregressive models*: These models are trained to predict the next word in a sequence based on the words that came before. This capability has been critical for generating coherent and contextually relevant text. Some well-known examples are GPT, LLaMA, and Claude. Some models, like GPT, are also *foundation models* (see below).
  - *Foundation models*: These models refer to large, general-purpose models that have been pre-trained on big amounts of unlabeled data and utilize self-supervised

learning, with the ability to adapt to other domains with minimal fine-tuning. GPT-3, BERT, and DALL·E 2 stand out as they can create text, images or answers from a single prompt for tasks they were not specifically trained to handle. Some foundation models are autoregressive (GPT). The models' power comes from integrating self-supervised learning with transfer learning to enable the usage of knowledge from one application area to be applied to different areas. Foundation models enable natural language processing tasks to require less labeled data while expanding AI deployment scalability and flexibility for various industry applications. [46], [47]

#### 2.3.3.3 Computer Vision

Computer Vision allows machines to extract meaning from visual inputs like images and videos. While image processing modifies visual data, computer vision aims to comprehend visuals by extracting features and identifying objects to create 3D models from 2D images [4], [52].

The fields of medical imaging, robotic guidance systems, facial recognition technology, autonomous vehicle navigation and industrial quality assurance all utilize computer vision applications. The standard computer vision techniques include object detection methods along with segmentation processes and optical character recognition (OCR) [4].

#### 2.3.3.4 Robotics

Robotics creates autonomous or semi-autonomous systems to interact with their environment through the integration of sensing capabilities with AI and mechanical action. The field of robotics brings together multiple artificial intelligence functionalities, including visual processing abilities along with planning capabilities and reasoning processes. Automation tasks like welding, assembly, and material handling heavily utilize industrial robots [4].

#### 2.3.3.5 Expert Systems

Expert Systems represent AI programs that simulate the decision-making functions of human experts. These systems function through a knowledge base of rules and inference mechanisms which utilize logical processes to generate conclusions or provide recommendations. The application of expert systems extends to domains, such as medical diagnosis, configuration systems setup as well as troubleshooting [4].

#### 2.3.3.6 Knowledge Representation and Reasoning

Knowledge representation and reasoning (KRR) is a field that enables AI systems to structure, organize, and utilize knowledge to make decisions or solve problems, forming a foundation for human-like reasoning [4]. Some of the essential techniques in this AI field include ontologies and semantic networks. The main goal is to build systems that can reason about their surroundings and provide solutions to complex challenges using stored knowledge [4]. KRR connects to multiple AI subfields and enables their application like Expert Systems, NLP tasks, robotics, and cognitive computing systems.

#### 2.3.3.7 Planning and Decision-Making

The practice of planning and decision-making requires developing action sequences that lead toward achieving predetermined objectives. Problem-solving agents within AI systems simulate environments to test strategies through computational search and optimization techniques. Robotics, logistics fields and game AI systems require these capabilities as fundamental components [4].

#### 2.3.3.8 Speech Recognition and Generation

Speech processing involves both recognition and synthesis. Recognition systems process spoken words into text by using acoustic models and pattern recognition whereas synthesis systems create spoken language from text through Text-to-Speech (TTS). The complexity of speech recognition arises from the differences in pronunciation styles and accents among speakers as well as the presence of background noise [4].

### 2.3.4 Applications and Advantages of AI in Software Engineering

Software Engineering (SE) nowadays uses Artificial Intelligence (AI) to create advanced solutions throughout the entire software lifecycle. The range of AI capabilities from predictive analysis to automated processes brings revolutionary improvements to productivity and quality in software development and maintenance. This chapter explores the functionalities of AI systems throughout the different software lifecycle phases/stages and describes the primary advantages their application provides.

#### 2.3.4.1 Applications of AI Across the Software Engineering Life Cycle

Software engineering core activities utilize AI techniques effectively throughout all stages from planning projects, to deploying code. Software Project Management benefits from AI which delivers advanced project scheduling capabilities and includes effort estimation and cost prediction functionalities. Traditional estimation techniques tend to produce errors due to human bias alongside unclear requirements and changing project scope. AI models reduce estimation risks by using historical project data to deliver predictions with improved accuracy and context sensitivity. Certain methods employ probabilistic techniques, such as Bayesian inference, to integrate uncertainty modeling that facilitates scenario-based "what-if" planning for improved risk management [43].

Building upon the project planning phase, AI in Software Design and Architecture automates architectural decision-making, and produces design artifacts, such as UML diagrams, system models and user interface layouts. Design processes frequently require specialized knowledge and subjective judgment. AI tools that learn from historical designs and system goals minimize design flaws while enforcing consistent design patterns and promoting reusability, which proves beneficial for large distributed teams [59].

Following design activities, AI tools utilized in software implementation and programming including code completion engines along with intelligent refactoring assistants and program synthesizers like GitHub Copilot that automate repetitive coding activities and recommend proper APIs while maintaining consistent coding patterns. The provided systems

enable developers to eliminate redundant code segments while minimizing syntax mistakes and enhancing the productivity of regular coding processes [59].

In addition, the field of Code Generation and Synthesis continues to experience significant growth. AI models have the capability to generate code from natural language inputs and turn formal specifications into functional software. Domains that demand extensive automation and minimal error rates benefit greatly from this application particularly in the areas of embedded systems, automotive software development and scientific computing [20].

Furthermore, AI makes substantial improvements to Software Testing and Quality Assurance processes. The testing phase stands out as one of the most demanding activity in SE because it consumes substantial time and resources especially when applied to safety-critical applications which need extensive test coverage. AI makes the creation of test cases faster while optimizing their arrangement and reducing excess cases. Fault prediction algorithms detect vulnerable modules before issues arise and deep learning models both localize bugs and generate potential solutions. The approach cuts down on manual work while accelerating the validation process and enhances the efficiency of testing [59], [43].

In cases of failure resolution, debugging can be enhanced by identifying failure causing inputs as well as ranking potential faulty spots with suggested fixes through data mining and inference techniques. The process of traditional debugging requires significant time while relying heavily on the skills of the developer. AI tools help reduce the time needed for investigations which supports more rapid error resolutions in extensive codebases [4].

Beyond the initial development phases, AI models in Software Maintenance and Evolution perform tasks, such as predicting future defects while detecting code smells and supporting regression testing during maintenance activities**.** A significant part of a project's long-term cost comes from maintenance activities. Teams can use AI to concentrate on high-risk areas and predict potential changes in the software or in specific components while automating routine tasks, such as patch validation and dependency tracking, which enhances both software stability and maintainability [59].

Lastly, the application of AI in Software Engineering practices, including DevOps and CI/CD, supports automated monitoring along with anomaly detection and performance optimization throughout the DevOps pipeline, from configuration and integration to delivery and deployment. Intelligent software agents evaluate telemetry data to make decisions about building software, testing it, or triggering rollbacks which leads to faster software delivery cycles [3].

#### 2.3.4.2 Advantages of AI in Software Engineering

The integration of AI into software development delivers multiple benefits which improve both software technical standards and development processes.

One of the most immediate advantages is efficiency and automation. AI helps minimize manual labour by automating various tasks within the software lifecycle. For instance, through the use of generative models and NLP tools developers process and produce source code more rapidly which can boost their productivity. As a result of AI application (as, e.g.,

in software implementation), software products benefit from reduced development timelines leading to quicker (time-to-market) deliveries [7].

Additionally, AI models demonstrate excellent scalability in complex environments while allowing for fine-tuning to meet the unique needs of industries, such as healthcare and finance, or embedded systems. In parallel, General AI models can solve specific domain challenges using techniques, such as transfer learning and domain adaptation, which enhances AI versatility and industry-wide impact [18], [20].

# 3 *Review Methodology*

The research followed the widely adopted guidelines by Kitchenham et al. (2007) [14], which serve as a structured framework for literature review processes in software engineering studies. These guidelines mandate that the review methodology follows a series of steps to ensure both rigour and transparency through the review process.

The review methodology applied maps to a process consisting of four steps. Initially, the process involved creating the review objective, which determined the main research questions to answer. Then, a set of search terms was constructed from the research questions to retrieve the relevant literature available at scientific repositories and databases. Following the article search, the applied methodology ensured selected studies maintained quality and relevance by applying inclusion, exclusion and quality criteria through the filtering process/step on the initial results of the search. In the fourth step a statistical analysis of the selected studies was implemented. The remaining of this chapter is dedicated to presenting these steps and their results. In particular, each step is orderly described in a separate (chapter) section along with its main rationale and results.

## 3.1 Review Objective and Research Questions

The objective of this study is to review, identify and analyze recent AI-based approaches and tools that support Requirements Analysis, Validation and Verification in the field of Requirements Engineering. Through this analysis, the study aims to compare the proposed approaches and tools and identify their novelties, strengths and limitations. In addition, it aims to uncover related gaps as well as propose future work directions to address and resolve them.

The research questions this study aims to answer are defined below:

**RQ1**: *What are the existing approaches that utilize Artificial Intelligence for Requirements Analysis, Validation and Verification?*

This research question goal aims at identifying AI-based approaches in RE, specifically targeting those that offer implementations and functionalities for the analysis or validation & verification phases of RE. RQ1 is answered in Section 4.2.

***RQ2**: How can these approaches be categorized?*

This question the deep categorisation of the identified approaches based on common characteristics, such as the AI technology used (e.g., NLP, ML), or the RE activity supported. RQ2 is answered in Section 4.1.

**RQ3**: *Which approaches can be considered as best based on a certain set of criteria?*

This question aims to evaluate and compare the identified approaches based on their own respective conducted assessment by relying on empirical results, benefits or usability aspects when these are reported by them. RQ3 is answered in detail in Section 4.3.

**RQ4***: Which are the main research limitations and gaps in the existing approaches?*

This question aims to report common issues, challenges, or limitations in current AI-based approaches for RA and RVV, so as to capture which main research problems remain unresolved. RQ4 is answered in detail in Chapter 5.

**RQ5***: Which research directions and trends need to be followed in the future to improve or optimise AI-based requirements analysis, validation and verification?*

This question explores directions for future work to address the identified main research limitations and gaps, including new methods, evaluation strategies, datasets, or unexplored tasks in RA and RVV. RQ5 is answered in detail in Chapter 5.

## 3.2 Article Search

In order to conduct the review, a search pattern was designed and applied over existing scientific repositories and databases. This pattern included three main parts: one related to AI, another related to RE and the last one related to approaches. All these parts were connected via the AND logical operator (logical conjunction).

The first part ensured that AI-related terms are disjunctively included in the searched articles, such as "artificial intelligence" and "machine learning". The second part ensured the disjunctive appearance of RE-related terms, such as "requirement validation", "requirement verification" and "requirement analysis", designating the main research area of focus. The third part covered the target of research, which relates to concrete approaches in requirements analysis or validation. Thus, it included terms related to the proposition of an approach, such as methods or techniques. The overall pattern thus took the next final form:

(("Artificial Intelligence" OR AI OR "machine learning" OR ML OR "deep learning" OR OR DL OR "natural language processing" OR NLP) AND ("Requirements Engineering" OR "Requirements Analysis" OR "Requirements Validation" OR "Requirements Verification") AND (tool OR method OR approach OR framework OR technique OR solution OR service OR facility)).

The aforementioned pattern was searched over the title and abstract of the articles. Further, an inclusion filter was directly applied in the search to limit the results to articles published in the period 2020–2024. This timeframe was selected to examine the recent advancements and breakthroughs in the field and reflect current trends, considering the rapid growth of AI technologies in recent years. It also helped to reduce the initial volume of the results. The rest of the filtering criteria were applied afterward, as described in the following section (Section 3.3).

The exploited search tool/academic source was primarily Google Scholar while other sources included IEEE Xplore, ACM Digital Library, Semantic Scholar, Springer and Elsevier. These selected scientific databases/sources are widely used, well-recognized, reputable and most credible in scientific research so they have been selected to ensure the quality and breadth of the reviewed literature. [32]

## 3.3 Article Filtering

The search was restricted to peer-reviewed journal and conference articles to make sure that the selected studies were scientifically reliable and consistent. These types of publications are evaluated by scientists, academics and researchers before publication such that they are considered trustworthy. The evaluation helps filter out studies that are insufficient or incorrect. Non-peer-reviewed sources, for example white papers or blogs, were not taken into consideration as they miss formal quality assessment.

The initial search returned ~6,000 results across all databases. A selection strategy helped produce a manageable subset from the large initial result set to maintain feasibility and relevance. To limit the scope of the research, if a database produced more than 100 results, the top 100 ranked by relevance were selected for screening. If a database produced fewer than 100 results, all available articles were included. After combining the results from all databases, approximately 500 papers were gathered for initial screening.

After the database searches were conducted, the resulting articles were manually reviewed and filtered according to the inclusion and exclusion criteria. These criteria, which also addressed the relevance of the study to the research questions, were applied to the titles, keywords, and abstracts of each article. Duplicate articles were removed during this process.

Based on this, 60 potential articles were selected for further review. These articles were then assessed based on the quality criteria. After applying the quality filtering, 18 articles that met all the criteria were considered as primary studies and included in the thesis analysis.

### 3.3.1 Inclusion and Exclusion Filtering Criteria

*Inclusion Criteria*:

1. Articles that present AI-based approaches specifically for Requirements Analysis, Requirements Validation or Requirements Verification and have the potential to answer at least one research question.
2. Articles that are fully written in English so as to be able to understand them.
3. Articles published in peer-reviewed journals or conferences between 2020 and 2024 based on the rationale mentioned in the previous section (Section 3.2).

*Exclusion Criteria*:

1. Articles that focus on AI techniques for other RE activities (e.g., elicitation, specification)
2. Articles that discuss RE practices for AI or ML systems (i.e., the reverse direction).
3. Gray or yellow literature, such as book chapters and blogs
4. Duplicated articles (same articles from different search sources as well as earlier versions of more extended collected articles were also excluded)
5. Articles that do not answer any of the research questions

### 3.3.2 Quality Filtering Criteria

In addition to inclusion and exclusion criteria filtering, studies were further assessed based on their quality to ensure they could contribute meaningfully to the review so as to produce consistent and scientifically valid results. Articles were excluded if they:

1. Lacked a clear description of their objective, the proposed approach or evaluation.
2. Were difficult to understand, or poorly structured.
3. Failed to present a transparent, scientifically-valid and understandable evaluation or validation of the proposed approach/method/technique.

## 3.4 Quantitative Analysis

The final set of selected studies is quantitatively analyzed further, based on three aspects: publication type, publisher, and publication year. The findings are demonstrated with two pie charts and one bar chart, respectively.

In Figure 2, the distribution of the selected articles by publication type is presented. Most of the selected articles are journal articles (66,7%), while conference papers represent 33,3% of the total, final set. This finding highlights the focus of this review on journal articles, which typically undergo a more rigorous review process and provide more extensive methodological details and research results than conference papers, potentially enhancing the depth of analysis [49]. However, the further inclusion of conference papers allows to cover new, cutting-edge research, which might not be so mature yet but is still quite innovative and breakthrough.

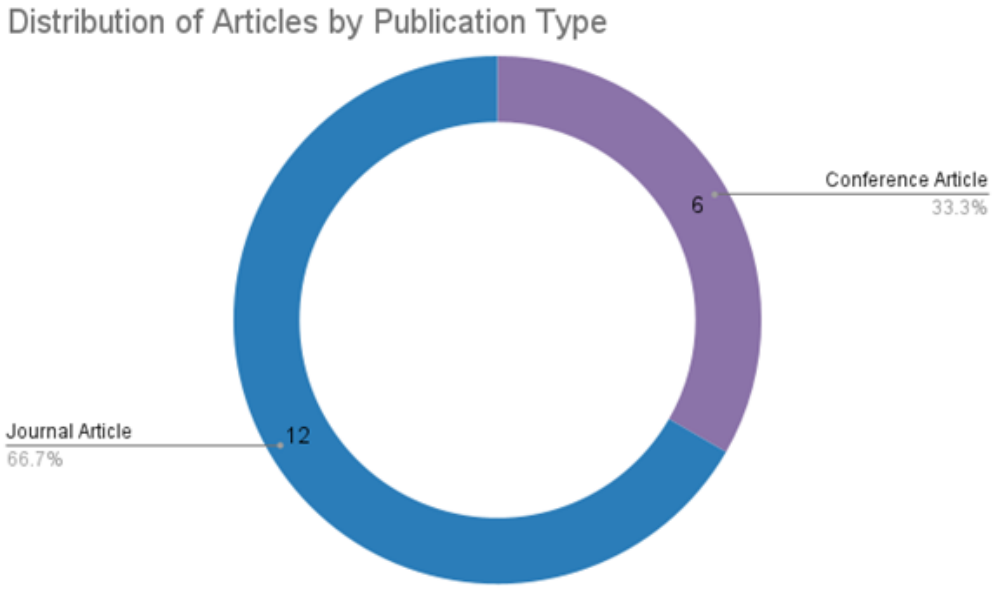


*Figure 2. Distribution of selected articles by publication type*

Figure 3 shows the distribution of articles by publisher in the form of a pie chart. The largest share comes from IEEE (33.3%), followed by Elsevier (27.8%) and Springer (22.2%). The remaining publications are distributed across MDPI, ACM, and Wiley (each 5.6%). IEEE, Elsevier, and Springer are widely recognized as leading and reputable publishers in the fields of computer science, engineering, and software development, known for their rigorous peer-review processes and impact in the research community. The dominance of such publishers in the dataset is a positive factor for the forthcoming analysis, as it suggests that a substantial portion of the included studies are likely to meet high standards of quality, methodological robustness, and academic credibility.

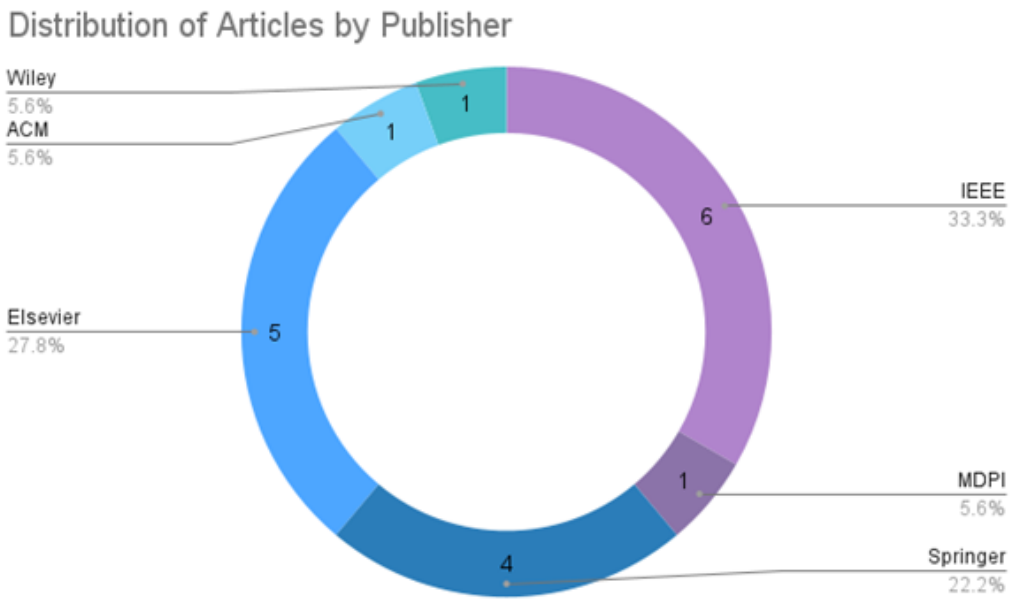


*Figure 3. Distribution of selected articles by publisher*

In Figure 4, the number of articles per publication year is presented as a bar chart. The number of articles is similar across all the years searched, with 2022 and 2024 being the years with the highest number of articles relative with the objective studied. This figure, however, may underrepresent 2024's output, as the bibliography selection was completed before the year's end. Furthermore, recent advancements in AI, particularly in LLMs and Generative AI might stimulate increased research activity in applying AI to RE. Thus, the number of publications will be surely higher in 2024 and is expected to be even higher in 2025.

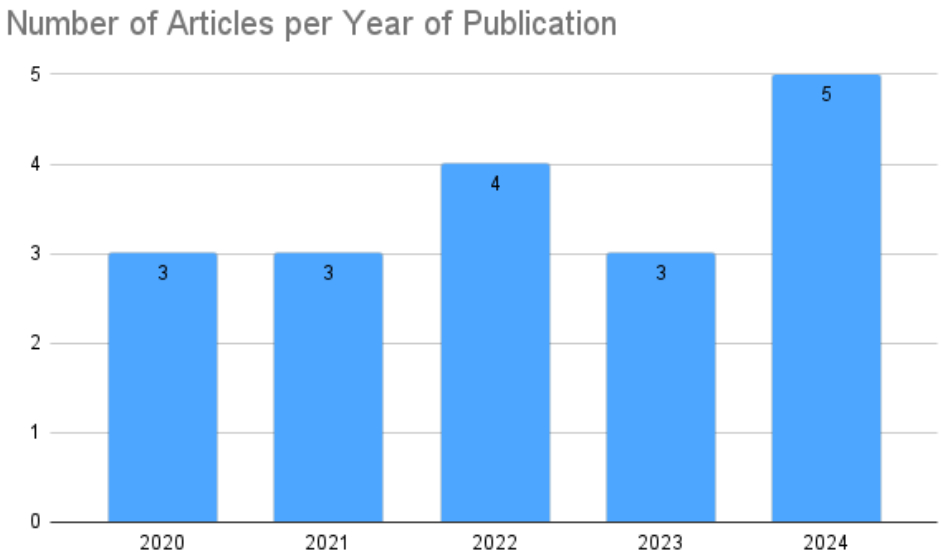


*Figure 4. Distribution of selected articles per year of publication*

# 4 *Literature Analysis*

This chapter presents the analysis of the selected studies that apply AI to RE, focusing on the phases of RA and RVV. First, a series of hierarchical categorizations is introduced to organize the studies across multiple relevant dimensions, including RE activities, AI approaches and techniques, and supporting tools. These categorizations are illustrated through diagrams that provide a visualization of how AI is employed in RA and RVV tasks in the studies. Following this, the chapter separately analyses the selected articles and ends with a thorough comparative analysis, comparing each study/article in terms of its targeted RE activity, applied AI technique, methodology, tools, datasets, and evaluation metrics.

## 4.1 Hierarchical Categorisation

To better understand how AI supports the RE phases of focus, i.e., RA and RVV, across the selected articles, different hierarchical categorizations are being presented and analysed next, based on multiple dimensions. Each diagram organizes its content into hierarchical categories that represent increasingly specific levels of detail within the broader landscape of AI-assisted RE. On the highest level, the diagrams identify the main categories (e.g., RE tasks, AI methods or tool categories) and on the lower levels these are broken down into activity types, individual methods, techniques, tools and tasks or subtasks. The diagrams illustrate the connections between reviewed studies and multiple aspects of RA and RVV while demonstrating how tasks, methods and tools interrelate within each RE phase.

The first categorisation dimension relates to the RE phases/activities/tasks on which the selected approaches focus. The consideration of this dimension has resulted in a hierarchical categorisation of the selected RE phases/activities/tasks addressed in the reviewed literature. The purpose of this categorisation is to contextualize the diverse range of activities/tasks within RE that are being automated or enhanced through AI.

For each of the two following diagrams the primary node is labeled with the respective RE phase of the surveyed approaches, and is further broken down into more detailed levels of activities (e.g., classification), and specific tasks (e.g., functional requirements classification, non-functional requirements classification).

The first diagram in Figure 5 involves the RA phase and the different activities/tasks included in that phase that the reviewed approaches cover. The most focused activity is classification of requirements with a significantly higher number of AI-based approaches proposed to address/improve it.

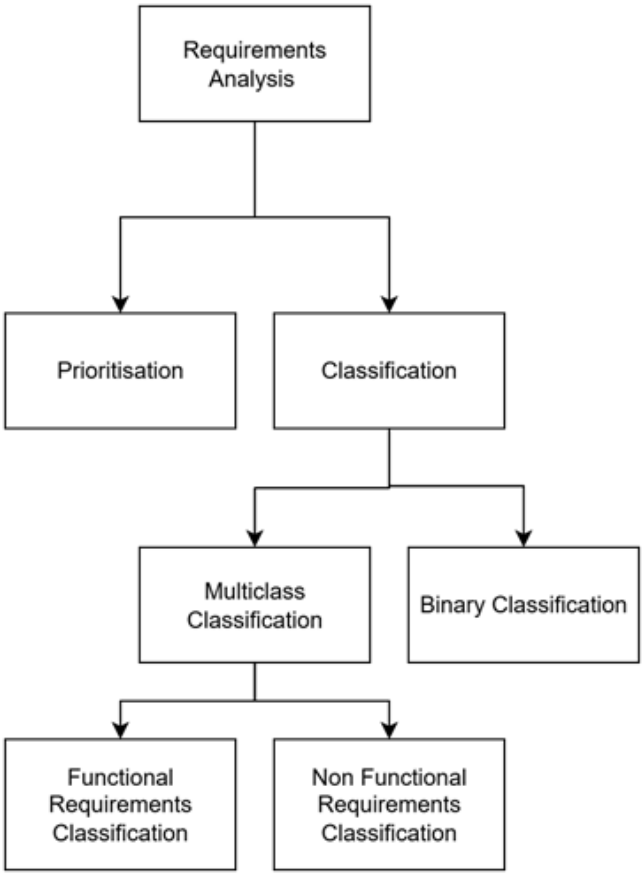


*Figure 5. RA Activities Categorisation*

The second categorisation diagram in Figure 6 involves the RVV phase and the activities/tasks that AI methods contribute to this phase. The activity addressed most for this RE phase is ambiguity detection in requirements.

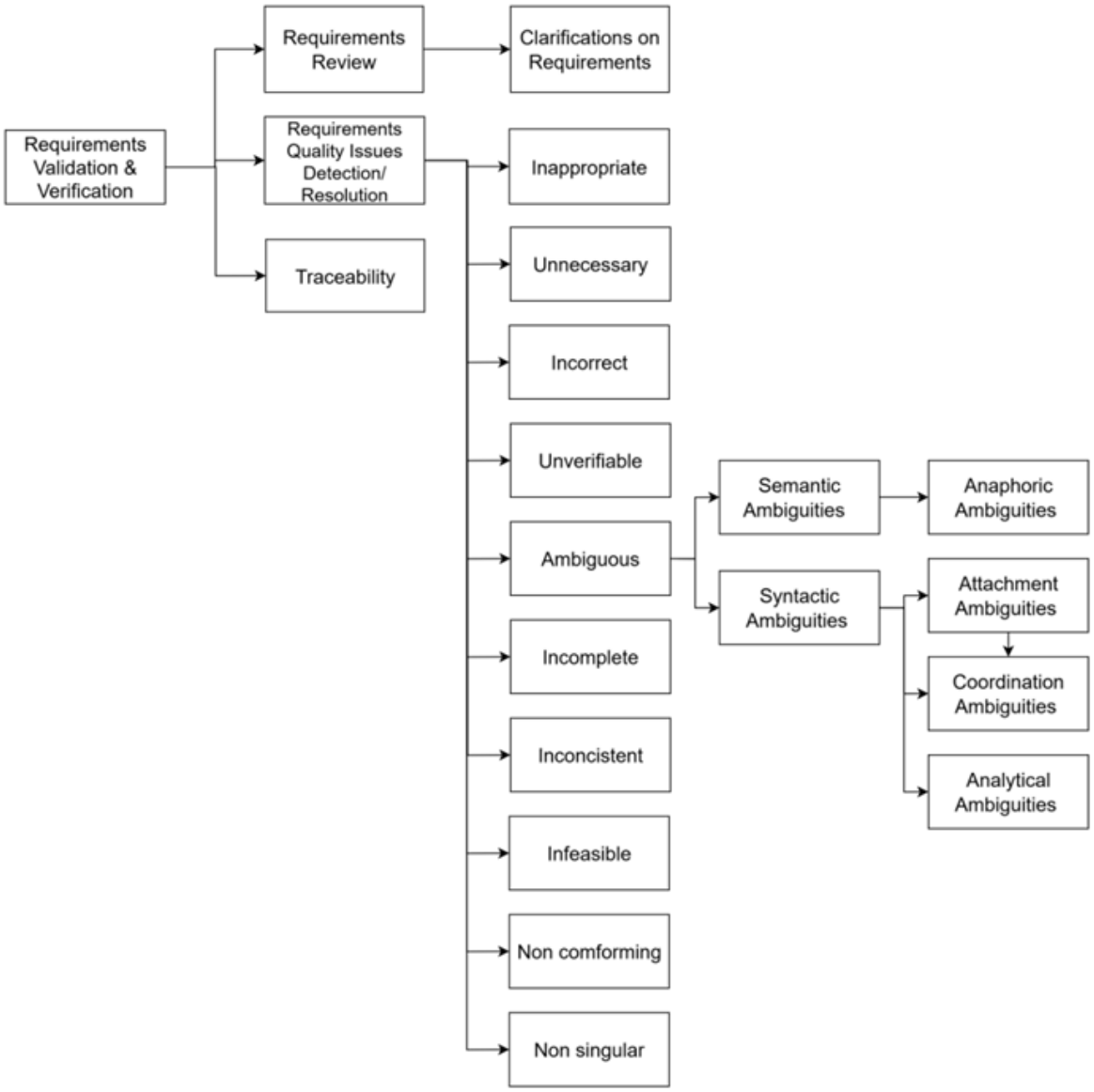


*Figure 6. RVV Activities Categorisation*

Below are a series of diagrams displaying the categorisation of the AI approaches and architectures used in the selected studies in RE (so this is the categorisation dimension now). Each diagram corresponds to a specific RE phase, RA and RVV, and shows how the different subfields of AI (e.g. classical machine learning, deep learning, transfer learning, large language models) are applied to automate or improve specific tasks in each phase. Following this, there is a further break down to more specific techniques utilized in the articles.

Figure 7 illustrates a deep hierarchical structure of AI approaches applied to RA, illustrating how different methods are organized into broader categories and progressively refined into specific techniques, reaching a level of four in the tree. This depth highlights the granularity of categorization of the techniques utilised and provides emphasis on the complexity of AI methods in this domain.

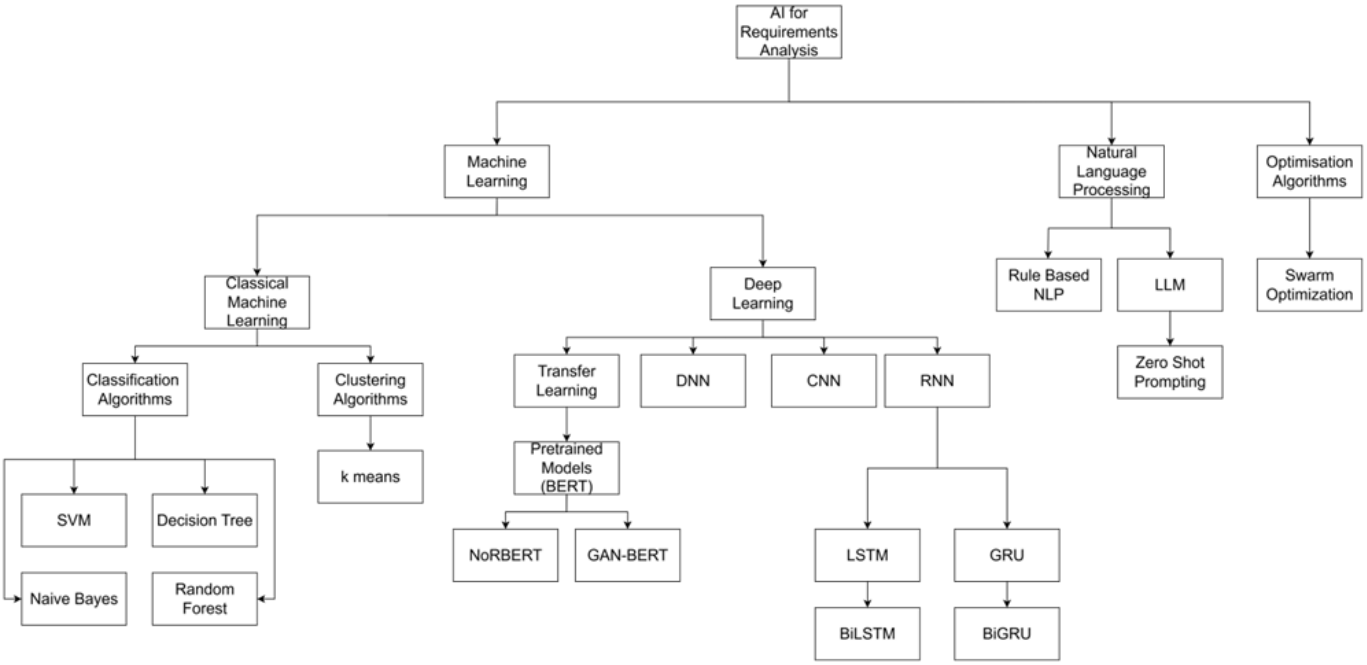


*Figure 7. Hierarchical Categorization of AI Techniques used in RA*

Next, Figure 8 shows the hierarchy of AI techniques used for RVV, according to the reviewed literature. The top-level node represents the RE phase under study, and the subsequent nodes represent the AI subfield and respective techniques used within each reviewed study. From the diagram it can be easily deducted that compared to RA techniques, the variety of RVV techniques utilized is minimal.

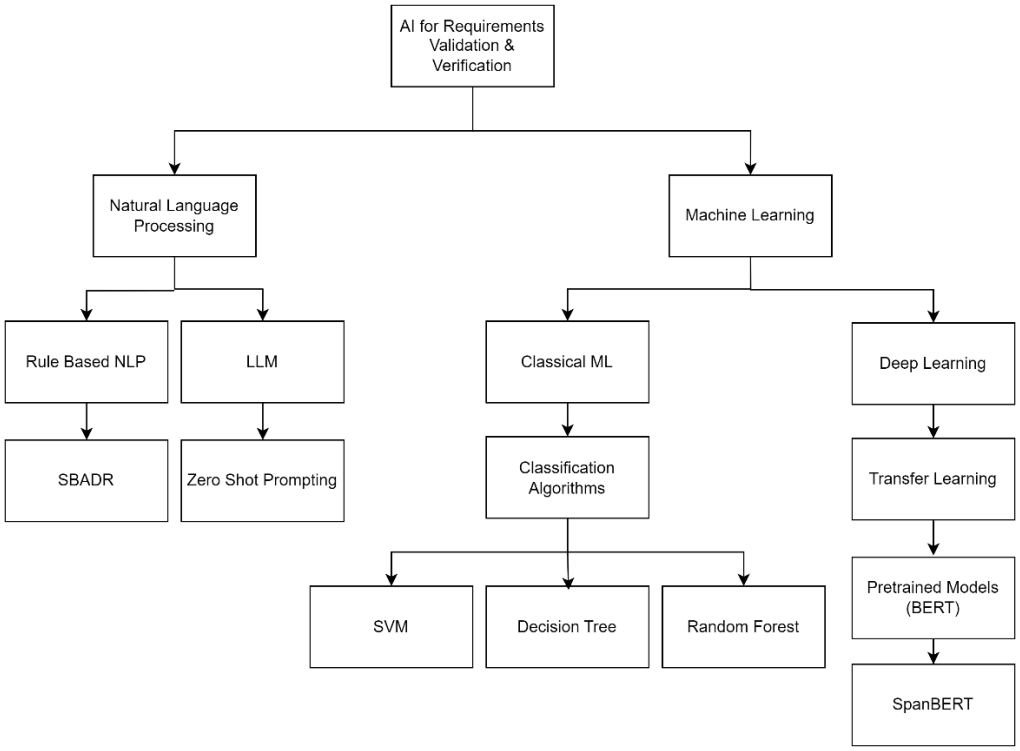


*Figure 8. Hierarchical Categorization of AI Techniques Used in RVV*

The next set of diagrams present a structured taxonomy of the tools employed across the two relevant phases on which the selected AI-assisted RE approaches focused.

The diagram in Figure 9 illustrates a taxonomy of tools and technologies used in RA, organized into distinct functional categories. This categorization helps to clarify the diverse software, frameworks, and utilities that support AI-based RE approaches. At the upper level, Development Infrastructure includes Programming Languages, Development Tools and Deep Learning Frameworks, which provide the environment and frameworks for model implementation and experimentation. Core AI Models & Algorithms include utilities for learning and optimization, such as vectorizers and optimizers, which contribute to model construction and training. Processing Pipelines & Supporting Libraries include utilities like NLP toolkits and machine learning libraries, which support activities, such as data preprocessing and feature extraction. Finally, Domain & Task Specific Tools include pattern-matching and annotation utilities, enabling the adaptation of AI models to RE-specific tasks.

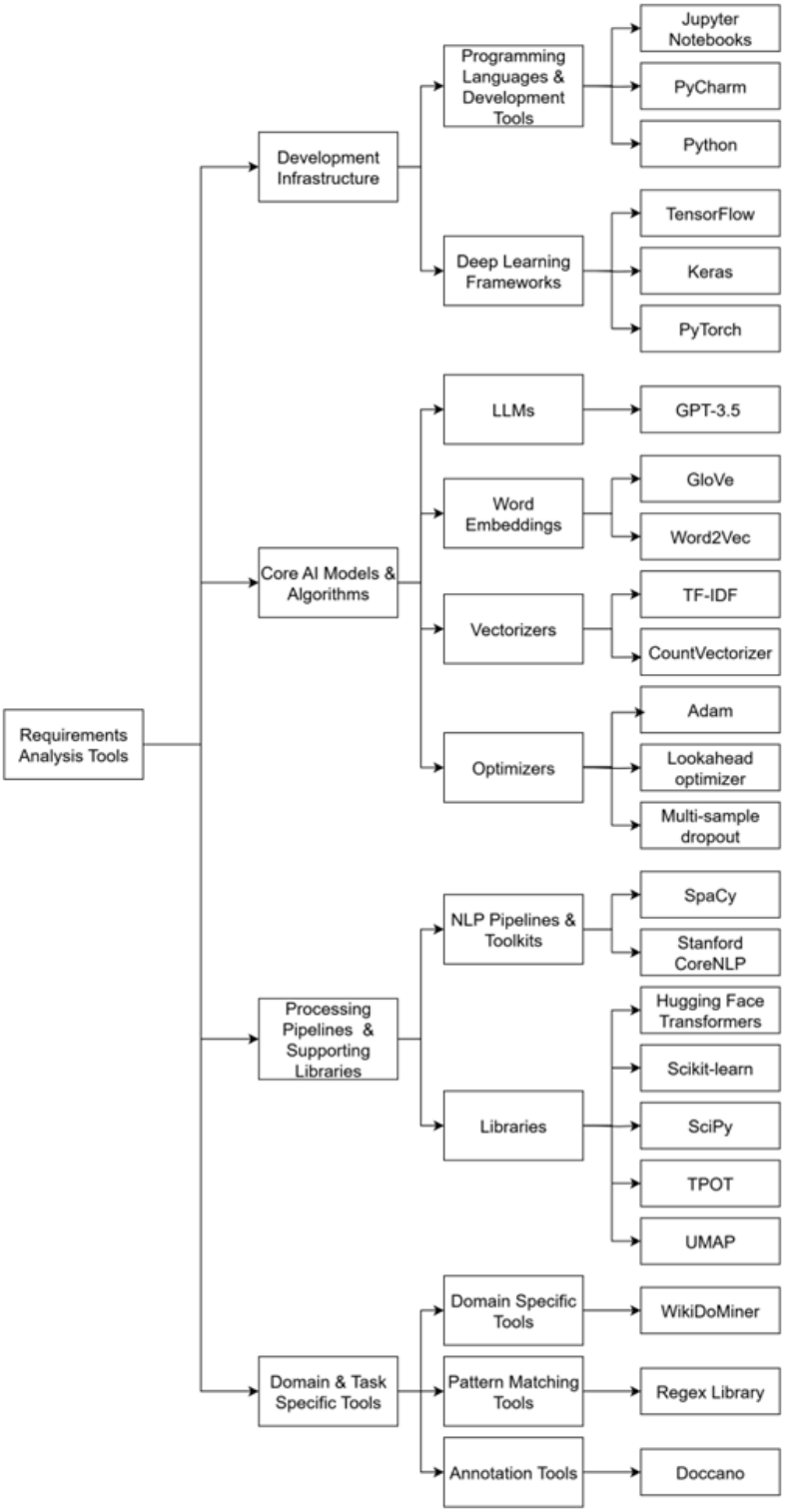


*Figure 9. Hierarchical Categorization of Tools used in RA*

Figure 10 diagram presents a structured categorisation of tools used for RVV, organized into functional categories. It highlights the software environments, libraries, and frameworks that facilitate the requirements verification and validation approaches reviewed. The range of tools represented here appears narrower while in contrast, RA utilized a broader set of tools, (propably) due to the greater number of studies that addressed RA in the relevant literature.

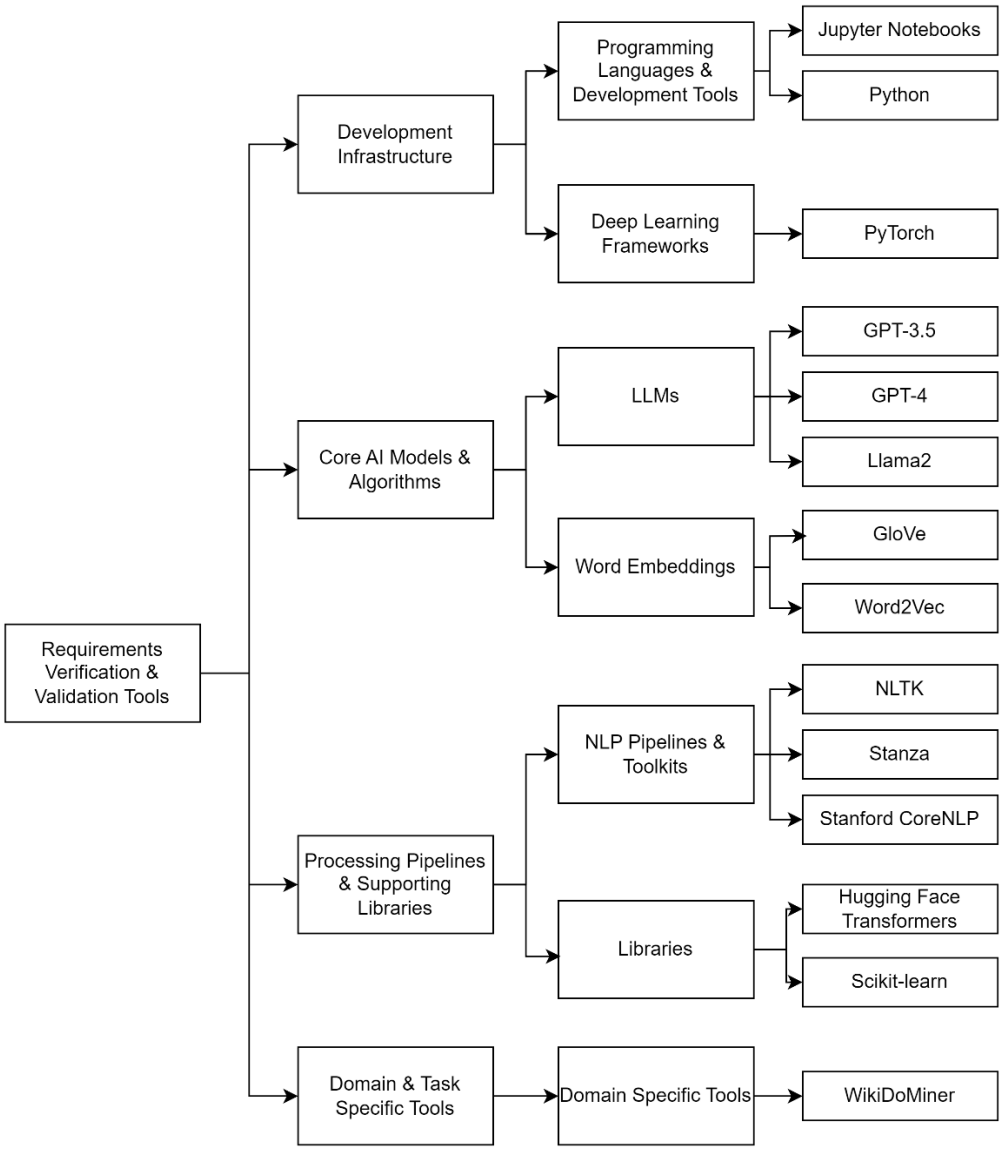


*Figure 10. Hierarchical Categorization of Tools used in RVV*

## 4.2 Articles Analysis

The following analysis offers a detailed structured review of chosen articles. In particular, each selected article will be reviewed according to the targeted RE activity (e.g., classification, ambiguity detection, prioritization), the AI technique used, the implemented methodology and the involved tools and frameworks, as well as the datasets and metrics used.

The rationale behind this analysis is to collect and structure the main technological and methodological characteristics of each approach, to detect novelties and gaps between these approaches, and to provide a basis for the subsequent comparison between them. First the articles that address RA phase activities are examined and RVV articles follow next in the next two sub-sections, respectively.

### 4.2.1 Articles based on Requirements Analysis

*1. Requirements Engineering using Generative AI: Prompts and Prompting Patterns*

The article [21] explored the application of Generative AI, specifically the GPT-3.5 Turbo[1] model, in RE, focusing on the evaluation of five different prompting patterns in two RE tasks: requirements binary classification into functional and non functional requirements which belongs in RA, and requirements traceability task which is an RVV task. The five prompting patterns individually assessed were *Cognitive Verifier*, *Context Manager*, *Persona*, *Question Refinement*, and *Template*, each designed to structure prompts differently to optimize task performance.

The *Cognitive Verifier* prompt breaks the task into smaller pieces, asks clarifying questions if needed, and combines all partial answers into a final output. The *Context Manager* prompt explains the reasoning and assumptions behind its output and handles any uncertainties or limitations to provide a better response. The *Persona* prompt makes the LLM act as an RE domain expert to provide professional classifications. The *Question Refinement* prompt allows the model to suggest clearer or more specific rephrasing of the main task (i.e., the prompt given to the model by the user), and to ask the user whether they would prefer to use the refined prompt version instead, to reduce ambiguity. Finally, the *Template* prompt requires the model to present the output in a strict, structured list.

Two datasets were used in the experiments. The PROMISE NFR dataset [82] was used for the binary classification of requirements into FR and NFR. It consists of 621 labelled requirements, out of which 253 are functional and 368 non-functional. On the other hand, the PURE dataset [83] was used for the requirements traceability task. It contains publicly available requirement documents from multiple domains. This study utilized two sets of software requirement specifications (SRS) taken from the PURE dataset, but the exact number of requirements was not mentioned. Before running the experiments, the considered datasets were cleaned from unnecessary information like hyperlinks (PURE) and curated to

[1] OpenAI: GPT-3.5 Turbo: https://platform.openai.com/docs/models/gpt-3.5-turbo

have both a ground truth and a cleaned unlabelled version (PROMISE NFR). By using Python the authors created a programmatic script to automatically create prompts (based on the aforementioned patterns) and interface requirements to GPT 3.5.

The study introduced an evaluation framework designed to systematically assess the effectiveness of prompt patterns in the RA tasks. The conduction of the evaluation consisted of three main steps: (1) executing the script per examined prompt to generate the respective prediction results (i.e., binary classification or conflicting requirements detection), (2) comparing the results with ground truth annotations, and (3) using the comparative results to calculate evaluation metrics to assess the performance of prompt patterns for the specified RA task. This process was repeated five times, one for each prompting pattern, with each experiment involving multiple executions (five runs per pattern) to account for variability in model responses. The resulting metrics (precision, recall, F1-score, accuracy) were averaged across these runs. Additionally, experiments were conducted under three different temperature settings (0.0, 0.4 and 1) to assess the impact of temperature variation on performance.

The authors propose the *Question Refinement* as the most consistent prompt compared to the other patterns, across all temperature settings. Temperature refers to a parameter controlling the randomness of GPT-3.5 response, where 0 produces the most deterministic outputs, 0.4 balances accuracy and variability, and 1 increases randomness or creativity of the LLM, but reduces consistency.

The evaluation of the *Question Refinement* prompt pattern demonstrated variability across metrics, especially when comparing the binary classification and requirements traceability tasks. For binary classification, the precision ranged between 80.8% and 84.6% depending on the temperature setting, while recall ranged from 83.7% to 87.4%, F1-Score[2] from 81.0% to 85.2%. Accuracy[3] was consistently above 89%. These results indicate a relatively balanced performance in distinguishing functional and non-functional requirements. However, in the requirements traceability task, the performance metrics were lower. Precision[4] ranged between 43.2% and 45.4%, while recall was slightly higher, varying between 60.7% and 64.0%. The F1-score remained low as well across all temperatures, ranging from 51.0% to 53.2%. Despite this low precision and F1 values, accuracy remained consistently high (between 91.3% and 91.8%). This discrepancy between accuracy and the other metrics is due to the imbalance in the dataset. Since most requirements did not have any links, the model correctly predicted a large number of "no link" cases. As a result, the overall accuracy remained high, even though the model was not effective at identifying the actual linked requirements, as reflected by the lower precision and F1-scores.

Nevertheless, the authors recommend *Question Refinement* as the most effective and stable prompting pattern compared to others, due to its consistent ranking across tasks and the

[2] F1-Score: The harmonic mean of Precision and Recall. It balances the model's ability to correctly identify relevant instances (Recall) and to correctly classify its predictions (Precision). [15]

[3] Accuracy: The proportion of correctly classified instances over the total number of instances. [15]

[4] Precision: The proportion of all the model's positive classifications that are actually positive. [40]

lowest standard deviation observed in all metrics which was below 2.5%, indicating a predictable and reproducible performance irrespective of the temperature setting. *Cognitive Verifier* ranked second, showing slightly lower precision in the traceability task and higher standard deviation, but it maintained high precision and recall in the binary classification task, further supporting its overall effectiveness.

One of the strengths of this study is the proposed evaluation framework, providing empirical results based on precision, recall, F-score, and accuracy, which can be used to assess the performance of prompts in RE tasks. Also, the authors demonstrate that RE activities can be improved through prompt engineering without extra training of the model, particularly in the binary classification task. However, in the traceability task, the lower performance metrics suggest that other factors, such as the model's limited exposure to specific domains or the complexity of establishing traceability, may have constrained its effectiveness. Finally, the examination of the results across different temperature values allowed to check the results consistency under different settings. However, the authors note that these findings may not generalize to other datasets or RE tasks beyond the two examined here.

A limitation of the study is that it is dependent on GPT-3.5 Turbo, without considering alternative LLM architectures or models. Since then, newer and more advanced LLM versions (such as GPT-4 and GPT-5) have been released, which may achieve better performance on the same tasks. Another weakness is the lack of human evaluation, which could have provided additional insights into the practical usability of the proposed prompt patterns beyond numerical performance metrics. Last, an important limitation is that the prompts are considered orthogonal to each other and the study evaluates each prompt pattern only individually. The combination of multiple prompt patterns [60] and the investigation of other techniques, such as few-shot prompting and chain-of-thought prompting [61], represent promising directions for future research of these tasks.

*2. Enhancing Software Requirements Classification with Artificial Intelligence with Semisupervised GAN-BERT Technique*

The article [71] presents a semisupervised DL-based framework that leverages a customized GAN-BERT technique for the classification of software requirements. The approach utilizes BERT embeddings with a Generative Adversarial Network (GAN) to improve performance. This combination aims to integrate transfer learning capabilities with semi-supervised learning to reduce dependence on labeled data. The classification focuses on the distinction between functional and nonfunctional requirements, by considering five specific NFR types: security, availability, performance, operational, and usability.

The authors define the prediction of software quality attributes from requirements as a text classification problem, where requirements are encoded in a high dimensional semantic vector space. The quality attributes are the labels, assigned by domain experts. A challenge highlighted is the crossdomain problem where the distribution of features is different between the training and testing datasets, thus decreasing performance. The approach attempts to adapt to this limitation.

In this study the BERT architecture is combined with a semisupervised GAN framework to classify software requirements. BERT converts each input requirement into an embedding. These embeddings are passed to a discriminator, which is a multilayer perceptron (MLP), to classify each input into one of c+1 classes. The first c classes are the actual quality attributes, and the additional class is used to detect "fake" inputs, created by a generator. The generator is an MLP as well, and it is trained to produce embeddings that resemble those generated from real requirements. During training, the BERT weights are updated together with the discriminator, and after training, the generator is removed. This structure combines transfer learning, through the use of BERT pretrained on general corpora, with semisupervised learning enabled by the GAN component, which incorporates unlabeled data into training.

The final model uses the trained discriminator and BERT to classify new requirement texts. This structure allows the model to learn from limited labeled data while leveraging unlabeled examples to improve generalization across domains. The implementation was carried out using the Transformers[5] library from Hugging Face with the DistilBERT-uncased[6] model, and the architecture was trained using PyTorch[7], with dataset splitting handled via scikit-learn[8]. The optimization was performed using the Adam optimizer [102].

For the evaluation of the model, four datasets were used: PROMISE, PROMISE-EXP (an extended version of the PROMISE dataset which includes software requirements that have been manually labeled with quality attributes, covering both F (functional) and NF (nonfunctional) requirements), WordPress documentation with 939 requirements annotated by the authors and software engineering experts, and project documents from various open source systems consisting of 5292 requirements annotated by the authors and software engineering experts. It is worth mentioning that the project documents requirements are written in informal language. All samples are manually labeled as either functional or nonfunctional by selecting in the latter case one of the five NFR categories.

The authors chose to compare the performance of GAN-BERT against two baseline classifiers, LSTM and standard BERT, given their widespread use and demonstrated stability in previous requirements classification studies.

The evaluation includes both in-domain (where training and testing data come from the same dataset) and cross-dataset evaluation scenarios (where a model trained on one dataset is tested on a different dataset). For each classifier, a separate model was trained on each of the four datasets, resulting in a total of 12 models per evaluation approach.

In binary classification, the models are evaluated using two labels, functional and nonfunctional requirements. All models are tested in both in-domain and cross-dataset settings using the aforementioned four datasets.

When trained and tested on the WordPress dataset (in-domain), GAN-BERT and LSTM both achieved an F1-score of 49%, slightly outperforming BERT (48%). In cross-dataset

[5] Hugging Face Transformers

[6] DistilBERT (uncased)

[7] PyTorch

[8] scikit-learn

evaluation with WordPress-trained models, BERT performed better, achieving 44% F1-score on PROMISE-EXP (9% higher than GAN-BERT) and 40% on Project Documents (10% higher).

When trained on PROMISE, LSTM performed best in the in-domain test with 48% F1-score, outperforming BERT (30%) and GAN-BERT (28%). In cross-dataset settings with PROMISE-trained models, BERT achieved higher scores, reaching 65% F1-score on WordPress and 88% on PROMISE-EXP while GAN-BERT achieved 58% and 49% respectively.

For models trained on PROMISE-EXP, GAN-BERT achieved the highest in-domain F1-score of 55%. However, in cross-dataset testing, BERT again performed better overall, reaching 56% F1-score, while GAN-BERT achieved 51% when evaluated on Project Documents and PROMISE.

When trained on the Project Documents dataset, BERT and GAN-BERT both achieved the highest in-domain F1-score of 77%. In cross-dataset evaluation, where models trained on Project Documents were tested on other datasets, BERT performed best on WordPress (77% F1-score), while GAN-BERT achieved 72%, and GAN-BERT slightly outperformed BERT on PROMISE (46% compared to 42%).

Overall, BERT demonstrated stronger cross-dataset evaluation performance in binary classification, particularly when trained on PROMISE or Project Documents. For example, when trained on PROMISE, BERT achieved an F1-score of 88% when tested on PROMISE-EXP and 65% on WordPress, both substantially higher than GAN-BERT's scores of 55% and 58% in the respective tests. Similarly, when trained on Project Documents and tested on WordPress, BERT reached 77% F1-score compared to GAN-BERT's 72%.

In multiclass classification, GAN-BERT demonstrated strong overall performance. It achieved the highest results in most scenarios, particularly when trained on PROMISE-EXP, where it reached F1-scores of 93% (PROMISE-EXP) and 92% (PROMISE). However, the model achieved only 28% F1-score in WordPress, and 37% in Project Documents, results that indicate the model performance degrades substantially when tested on datasets with different characteristics. Across 16 multiclass experiments, GAN-BERT outperformed the other models in 13 cases, confirming its consistency across different training and testing combinations. However, in the Project Document in-domain multiclass scenario, BERT slightly outperformed GAN-BERT, achieving an F1-score of 39% compared to 32%.

The authors highlight that in binary classification, BERT sometimes achieved higher F1-scores in cross-dataset evaluations than in in-domain testing, notably when trained on PROMISE and tested on PROMISE-EXP, where the F1-score reached 88%. This phenomenon, described as a classification anomaly, suggests strong transfer learning capabilities when datasets share similar characteristics. In contrast, multiclass classification generally performed better in in-domain scenarios, with GAN-BERT achieving the highest results, including F1-scores of 92% and 93% on PROMISE and PROMISE-EXP. The study also found that performance tended to decline on informal text datasets like WordPress, especially in multiclass settings. According to the authors, binary classification tends to degrade less than multiclass classification on informal texts, such as WordPress, while

multiclass classification achieves higher scores when training and testing data are similar and structured, as in PROMISE and PROMISE-EXP. Here, formal text refers to well-structured and standardized requirements written in conventional natural language, not formal specification languages, while informal text refers to requirements written in less standardized and less structured language, often derived from community forums or documentation that does not follow established templates or guidelines. Overall, the results indicate that models achieved higher accuracy on these structured datasets compared to informal datasets. BERT demonstrated superior binary cross-dataset performance, while GAN-BERT showed clear advantages in multiclass classification tasks when trained and tested on similar datasets.

This study introduces a method for the classification of software requirements. Its ability to leverage unlabeled data and adapt to different domains makes it a promising solution for software requirements classification tasks and adaptation to real world application. It improves generalization and reduces the need for expert annotation, while its handling of informal texts can be practical although not so precise, in particular on multiclass classification. The evaluation of the methodology is very thorough and detailed, including multiple domains. GAN-BERT demonstrated strong multiclass performance and robustness in similar dataset scenarios.

However, in binary classification, especially cross-dataset, it was frequently outperformed by BERT, which achieved higher F1-scores in most settings, except a limited number of scenarios, such as PROMISE-EXP trained models tested on PROMISE data (55% vs. BERT's 39%) and PROMISE trained models tested on Project Documents (50% vs. BERT's 46%). This suggests GAN-BERT's semi-supervised architecture may be less effective when the problem is a simpler binary classification. Another limitation is that the paper does not evaluate training time or computational costs, important factors in real applications. Last, although the authors compared BERT and LSTM, other architectures like CNNs or hybrid models could have been evaluated for completeness reasons.

*3. Classification of Non-functional Requirements Using Convolutional Neural Networks*

The article [35] investigates the problem of accurately classifying non-functional requirements, which are often overlooked in software development projects. Misclassification or omission of NFRs during the early phases of development can lead to incomplete software functionality, cost increase, and project delays. The authors propose an approach based on Convolutional Neural Networks (CNNs) to automate and improve the classification process, treating the task as a multiclass classification problem. The model is trained to recognize specific NFR categories (such as usability, security, performance, reliability, maintainability, and portability), leveraging the labeling of the PROMISE dataset.

The study's purpose is to enhance the performance of NFR classification. To achieve this, the authors investigate which strategies in the state of the art previously led to better results, with the intention of unifying them by proposing a methodology of designing a CNN model that leverages preprocessing, data balancing, and optimized vectorization. The CNN

architecture is incrementally enhanced through various experiments, each adding techniques to improve classification.

Describing the methodology followed in this study, initially, a CNN model is implemented using three text vectorization techniques: TF-IDF[9], Tokenizer[10], and Count Vectorizer[11] for training. These vectorizers are tested to assess their effectiveness, with Count Vectorizer producing the best results. Next, the authors incorporate preprocessing techniques, such as cleaning and lemmatization. To mitigate the dataset's class imbalance, they apply Random Oversampling (ROS)[12] to increase the number of minority class samples through duplication so that all classes have a more balanced representation in the training data.

After preprocessing and improving class balance, the authors tested three types of embedding matrices in the CNN: Word2Vec[13], FastText[14], and GloVe[15]. The proposed architecture was further enhanced by adding a dropout layer and adding a second Conv1D [16]and MaxPooling1D [17]block. Also, during testing CountVectorizer was configured with different n-gram ranges. The n-gram enhancement was highlighted as an important factor for performance improvement, because it allows the model to capture contextual meaning by considering single, two, three or more words.

The performance was measured using precision, recall, and F1-score. Across a series of incremental experiments with different preprocessing, vectorization, and embedding strategies, the final configuration combined CountVectorizer with n-gram range (1,4), Word2Vec embeddings, ROS, and an enhanced CNN architecture through the aforementioned additions of a dropout layer and a second Conv1D–MaxPooling1D block which together achieved the best performance under 10-fold cross validation.

All results were obtained using the PROMISE dataset. The final model resulted in the highest performance: Precision: 90%, Recall: 88% and F1-score: 88% under 10-fold cross-validation, resulting in a significant improvement over earlier relative studies mentioned in the studied article. Furthermore, Analysis of Variance (ANOVA[18]) was applied to the average F1-scores from cross-validation to assess whether the performance differences between models using different pre-trained embeddings were statistically significant. It was confirmed that differences between models were statistically significant.

[9] Understanding TF-IDF (Term Frequency-Inverse Document Frequency)

[10] Tokenizing with TF Text

[11] Using CountVectorizer for NLP feature extraction

[12] Oversampling and Undersampling, Explained: A Visual Guide with Mini 2D Dataset

[13] Word Embedding using Word2Vec

[14] FastText Working and Implementation

[15] GloVe: Global Vectors for Word Representation

[16] Understanding the 1D Convolutional Layer in Deep Learning

[17] Max Pooling in CNNs: Why It Matters and How It Works

[18] ANOVA (*Analysis of Variance*)

The innovation of this study is that even though the authors do not introduce an entirely new methodology, through the systematic evaluation of multiple techniques/models previously used, they unify and optimize the existing ones through a statistically validated framework. This approach leads to a high classification performance (Precision 90%, Recall 88%, F1-score 88%), showing clear improvements over related studies. On the other hand, the study does not assess its approach on external datasets, which would be necessary to confirm the model's generalizability beyond the PROMISE dataset, nor does it compare its performance on such datasets against other state-of-the-art methods.

4. *NoRBERT: Transfer Learning for Requirements Classification*

The work [72] presents NoRBERT, a transfer learning method that fine-tunes BERT models for classifying software requirements. The goal of this study is to support automation in requirements analysis by focusing on binary classification of requirements as functional or non-functional, as well as classification of non-functional requirements to their quality attributes and functional requirements to concern types (Function, Behavior, Data). As well as to achieve a generalized model with minimum training data such that classification accuracy is preserved even on unseen requirements.

Two datasets are used in this study. The first is the original PROMISE NFR dataset, and the second dataset is a relabeled version of PROMISE created by Dalpiaz [65], where the requirements are annotated into three broader categories: functional aspects, quality aspects, and requirements that express both. In addition to these, the authors annotate functional requirements from PROMISE NFR dataset into three functional concerns: Function, describing what the system should do (e.g., specific capabilities or features), Behavior, describing how the system should react under certain conditions or in response to events and Data, describing the information the system must handle, store, or process. This enables a new classification for functional requirements according to the aforementioned concerns.

The authors experiment with two pretrained transformer models for finetuning, BERT-base and BERT-large. BERT-base consists of 12 layers and 110 million parameters, while BERT-large contains 24 layers and 340 million parameters.

No preprocessing is performed, the only step before training is tokenization using the BERT tokenizer. The output vector (classification tokens) is passed to a linear classification layer in a feed forward neural network. Then the softmax function is applied to convert the output into probabilities. The model is trained using the AdamW optimizer (Loshchilov & Hutter, 2019), which updates the various weights within the network and includes a weight decay correction that helps achieve better generalization [51]. The authors finetune the number of training epochs. The epoch number defines the number of times the entire training set is processed during training. For binary classification, the best results are observed between a 10 and 32 epochs setup, and for multiclass a 10 to 64 epochs setup.

The study includes four evaluation experiments described next. In each experiment, the authors compare the performance of BERT-base and BERT-large, with and without oversampling to deal with class imbalance in the datasets, particularly in the non-functional

requirement subcategories. Early stopping is applied during training. All tasks are evaluated using 10-fold, p-fold, and leave-one-project-out setups.

In the first experiment, the model is evaluated with stratified 10-fold cross-validation setting, for binary classification of the requirements as functional or non-functional on the original Promise dataset. NoRBERT achieves an F1-score of 90% for functional and 93% for non-functional requirements. These results are compared with three previous state of the art approaches [64,66,104]. On non-functional requirements, NoRBERT outperforms all previous models except [104] with 95%, but the authors argue that this competitive approach uses manually defined dictionaries and rules while NoRBERT requires no such preprocessing. On functional requirements, the model by [64] shows higher F1 (93%) using a variant without feature selection that relies on all word features. However, the aforementioned authors acknowledge that this model overfits to the dataset and may not generalize to other projects, whereas NoRBERT achieves competitive results.

To assess the model's ability to handle unseen project requirements, the authors employ two project-specific cross-validation strategies: leave-one-project-out (loPo) and p-fold project-level cross-validation. In the loPo setting, the model is trained on n-1 projects and tested on the single remaining project, repeating this process until each project has been used once as the test set. In the p-fold setting, the dataset is split into 10 folds, each containing 3 projects for testing and 12 projects for training, with the folds constructed to maintain an even distribution of functional and quality aspects. In both settings, the model is always trained on the projects not included in the test fold.

Specifically, for the original PROMISE F/NF binary classification task, the best NoRBERT configuration in p-fold achieved an F1-score of 91% for functional requirements and 93% for non-functional requirements. In loPo, it achieved 90% for functional and 94% for non-functional requirements. Comparable p-fold and loPo results for the baselines were not reported in their respective studies, so direct comparison is not possible in this setting.

However, these results are close to the 10-fold scores, indicating that NoRBERT maintains strong performance when tested on entirely unseen projects, supporting its generalization capability.

The second experiment evaluates NoRBERT on the classification of the most frequent NFR types. Three approaches are applied: first binary classification by training one classifier for each of the four most frequent NFR types (Usability, Security, Operational, and Performance). Second, multiclass classification limited to four types plus an "Other" category and third, multiclass classification across all ten NFR types from the dataset. Evaluation is again performed using stratified 10-fold cross-validation.

In binary classification of the most frequent NFR classes on NFR dataset, NoRBERT achieves a weighted average F1-score of up to 83%, while in multiclass classification the best results reach 87%. Overall, multiclass classification performs better than binary classification. Training on all NFR classes leads to the best overall and individual class performance in 10-fold cross-validation.

Compared to [64], NoRBERT performs better on three out of four NFR types: 88% F1-score for Usability, 91% for Security, and 83% for Operational. For Performance, the results

are identical, with both approaches reaching about 88–90% F1-score. On average, NoRBERT outperforms the results of previous work from [64] by over three percentage points. In both binary and multiclass settings, NoRBERT outperforms all three feature selection models, which used subsets of 50 or 500 best features as well as a model without feature selection. The authors, however, note that some results reported in the compared work appear inconsistent, specifically one previous work reported values for precision and recall that did not add up to the reported value for F1-score, so final conclusions could not be met for the specific comparison.

To test generalization to unseen projects under the second experiment settings, the model is evaluated using the same project-specific folds as in the first experiment. In the p-fold setting, the F1-score decreases slightly. In the loPo setting, results are similar or better. In these settings, the large model trained on only the four most frequent classes performs better than the one trained on all. The best model in these settings reaches 83% weighted F1-score in p-fold and 84% in loPo. The authors conclude that NoRBERT generalizes well and that transfer learning helps overcome limitations due to lack of training data in requirements classification.

In the third experiment, the model is evaluated on multiclass classification of non-functional requirements into all original NFR subclasses. The PROMISE dataset is used, with one class (Portability) removed due to having only one example. Binary classifiers are trained separately for each of the ten remaining classes. A five-times 5-fold cross-validation is used for evaluation, as in [104]. The results show that multiclass models perform better than binary ones. The best multiclass model outperforms all binary classifiers on all but one class.

The highest performance is achieved with BERT-large models trained for 32 and 50 epochs. The best result is 82% weighted F1-score, and compared to [104], NoRBERT performs better than all of the models in the latter approach except the Naïve Bayes classifier. However, that model requires manual preprocessing, while NoRBERT does not.

To evaluate performance on unseen projects, the same project-specific folding strategies were used as in the first experiment. In p-fold, the model achieved a weighted average F1-score of 76%, six points lower than the best 10-fold model. In the loPo setting, the model performed slightly worse than in p-fold (73%). A possible reason is that some easier-to-predict projects are tested more than once in p-fold. In all evaluation settings, multiclass models and large BERT configurations outperform binary models.

In the fourth experiment, the model is evaluated using the relabeled version of the PROMISE NFR dataset [65]. This dataset includes requirements labeled as functional aspects, quality aspects, or both. The goal is to evaluate NoRBERT when requirements include overlapping properties. Dalpiaz in [65] also reimplemented the approach by [64] on the same dataset, enabling a comparison between them. The evaluation is done using the same setups as in [65]: a 75% split, 10-fold cross-validation, p-fold, and a fixed random seed. NoRBERT is additionally evaluated using the p-fold and loPo settings. The results show that NoRBERT outperforms all previous approaches in every evaluation setting.

On the 75% split, both BERT-base and BERT-large perform best on different classes. The base model gives better results on classes containing only functional or only quality

aspects. The large model performs better on mixed classes. On 10-fold cross-validation, the best NoRBERT model outperforms the best model [65] by an average of ten percentage points. The highest improvement is on functional requirements, where NoRBERT achieves 91% F1-score, compared to 73% by the baseline.

Evaluation on unseen projects using p-fold and loPo shows that in the p-fold setting, NoRBERT outperforms the reimplemented [64] model by 15 percentage points on average. In contrast, the baseline model drops five points when tested on unseen projects. NoRBERT's performance remains stable between 10-fold and p-fold, which supports the hypothesis that it can be used without retraining.

Additionally, the authors annotate 214 functional requirements into three concern types: Function, Behavior, and Data. A separate classification task is performed using this new labeling. The results show 92% F1-score for Function, 80% for Behavior, and 62% for Data. The low performance for Data is attributed to class imbalance.

The main novelty of the study is that it applies transfer learning to requirements classification across different levels, covering binary (i.e., functional and non-functional) requirement classification at the global level plus functional and non-functional classification at a second level with functional concerns and non-functional subtypes, respectively. In parallel it does not require any preprocessing. It is evaluated on multiple classification tasks, each with different class configurations, and performs consistently well. Project-specific evaluation settings (p-fold and loPo) show that the model generalizes to unseen project domains.

Although NoRBERT performs well across all evaluated classification tasks, the study has some limitations. All experiments were conducted exclusively on the PROMISE NFR dataset and its relabeled variant by [65], which limits evidence of generalizability to other domains and industrial contexts. Also, NoRBERT does not include an analysis of the computational cost or training time, which could be relevant when considering practical adoption in industry. Finally, the experimental comparison is restrained to three state-of-the-art baselines [64, 66, 104]. Altrough these baselines are representative, the comparison can be considered as quite limited.

5. *Automatically classifying non-functional requirements using a deep neural network*

This study [73] investigates the use of deep learning to support requirements analysis, focusing on the automatic classification of non-functional requirements into certain subtypes. The authors propose the NFRNet model, which combines BERT embeddings with N-gram masking, a Bi-LSTM RNN and a Softmax output layer. To enhance generalization and training efficiency, the authors integrate a regularization technique known as multi-sample dropout. The goal is to categorize NFRs into quality attribute classes to assist with early-stage requirements understanding and system design. The evaluation compares the model against

17 state-of-the-art baselines approaches and the proposed methodology is implemented based on the PyTorch[19] framework.

For the preprocessing phase of the approach text cleaning and normalization are performed utilizing the regex library[20] and NLTK toolkit[21]. The proposed model consists of two main components. First a deep neural network model called NFRNet, which consists of a BERT word embedding layer to capture semantics and context, based on N-gram masking, where the latter assists in handling of short sentences and in mitigating the loss of important semantic information in large texts. Second, BERT outputs are passed into a bidirectional LSTM (Bi-LSTM) classification network. The latter considers the textual context both in forward and backward directions for a deeper understanding of the context of requirements, followed by a multi-sample dropout regularization technique, added to enhance generalization of the model. The final layer uses a Softmax classifier to provide an output in the form of a predictive label for each requirement.

For evaluation the authors used the SOFTWARE NFR dataset, an expansion of the PROMISE dataset, crafted by the authors, containing 6222 NFRs taken from Wikipedia articles and annotated into 32 NFR categories. Annotation was performed by software engineering students.

To ensure fairness, the authors applied the same training settings across all models evaluated, including early stopping[22], the Lookahead optimizer [103], and hyperparameter tuning. The model was trained using 10-fold cross-validation, with requirement texts initialized to a length of 64 words. All experiments were run on the same data folds to avoid the impact of different data partitions.

Evaluation was conducted for NFRNet and 17 baseline models found in previous work for comparison, all evaluated under identical training conditions. The precision, recall, and F1-score were used as metrics. NFRNet achieved the best performance on the SOFTWARE NFR dataset, with precision of 91%, recall of 92.65%, and F1-score of 91.47%. It is worth mentioning that NoRBERT [41], assessed in the present review, was one of the baseline models reevaluated with the new dataset, and even though this model achieved good results (87% precision, 87% recall, 87% F1-score), NFRNet outperformed it in the current specific task.

Furthermore, comparing the accuracy values between all the models investigated and the different 32 NFR types available in the SOFTWARE NFR dataset, NFRNet achieves in many cases the best accuracy and the lowest standard deviation ($\sigma = 4.82$) compared to the rest of the models, showing consistency and robustness in comparison to the others.

[19] PyTorch – An open-source machine learning framework for deep learning and tensor computation. https://pytorch.org/

[20] Regex – A Python module that provides advanced regular expression matching operations beyond those of the built-in library. https://pypi.org/project/regex/

[21] NLTK – The Natural Language Toolkit, a suite of Python libraries and programs for natural language processing. https://www.nltk.org/

[22] Early stopping – A regularization technique that halts training when the validation performance stops improving, preventing overfitting

An ablation study showed that N-gram masking improved BERT's output. Specifically, by comparing the original BERT and N-gram masking BERT showed that N-gram masking increased precision from 85.62% to 91%, recall from 85.86% to 92.65%, and F1-score from 85.64% to 91.47%. This confirms the positive contribution of the masking technique in enhancing BERT's output for NFR classification.

The novelty of this study lies in its introduction of a new high-quality dataset specifically designed for non-functional requirements (NFR) classification, covering a wide range of requirement categories that can support future research in this area. Additionally, it implements architectural innovations, including N-gram masking to enhance contextual representation and multi-sample dropout to improve generalization and training stability. The proposed method demonstrates superior performance compared to 17 baseline models, including NoRBERT analysed previously in this thesis, highlighting its ability to generalize effectively across diverse NFR classes.

However, the study does not include functional requirements classification which would be a beneficial task for integration into practical RE workflows. Another limitation is that Software NFR dataset was constructed by the authors from Wikipedia text rather than real-world requirements documents and annotated by students rather than industry practitioners, potentially affecting the reliability of the labels. Further, the experiments did not test whether the model works well on new domains. All training and testing used mixed data from all domains, so it is unclear if the model can generalize to unseen domains, unlike studies that train on some domains and test on others (like the one proposing NoRBERT).

6. *Improving BERT model for requirements classification by bidirectional LSTM-CNN deep model*

This study [74] proposes a deep learning and transfer learning approach named BERT-BiCNN for binary requirements classification and non-functional requirements classification in particular types, namely availability, operability, usability and security.

The BERT-BiCNN model consists of three components. First, the BERT base embedding layer is used for the extraction of context from requirements. It captures rich semantic and syntactic relationships using the transformer architecture.

Second, a Bidirectional LSTM (BiLSTM) layer processes these embeddings to model long-term dependencies (i.e., if the previous state that is influencing the current prediction is not in the recent past, the RNN model might not be able to accurately predict the current state) in both forward and backward directions. The authors mention that the BiLSTM manages to keep the grammatical and semantic meaning of the requirements.

Third, a Convolutional Neural Network (CNN) is applied to the BiLSTM output to select important features and reduce the model complexity. The CNN filters to capture the most important sequence information and reduce the dimensionality of input data. Then the data is pooled and given to the classification layer. The output is passed to sigmoid (binary classification) and softmax (multi-class classification) functions to estimate the predicted probability for classification. The BERT-BiCNN architecture was applied separately to

different classification settings rather than through a single multi-output model or a cascaded pipeline.

The model was evaluated using the PROMISE dataset and no preprocessing was applied, since the model is designed to process raw requirements. The training settings were taken from past BERT-based research studies by adopting the same hyperparameter configuration, using the BERT-Base model (12 attention layers, 12 hidden layers, and 728 hidden dimensions) with a fixed input sequence length of 30 tokens determined from the average requirement length.

All experiments were conducted in a Python environment using Jupyter Notebook[23], with model development implemented in TensorFlow[24] and Keras[25], and contextual embeddings provided via the Hugging Face Transformers[26] library. The Adam optimizer was used for training, and dropout regularization was applied to reduce overfitting.

The proposed model was evaluated against six baseline models previously proposed in the literature. Binary and multi-class classification tasks were performed, with comparisons to re-implemented baseline models and one ablation analysis. The authors applied 10-fold stratified cross-validation and evaluation metrics included precision, recall, and F1-score.

The baselines compared against BERT-BiCNN include several customized combinations of deep learning models proposed in earlier studies (also analysed in this thesis). For example, in [78] the same paper proposes both a CNN [78] that applies convolutional layers over word embeddings to capture local textual features and an ANN that uses word2vec representations with dense layers. [79] proposes an LSTM-based model that feeds LSTM outputs to a fully connected classifier. BiGRU [77] similarly models context with bidirectional gated recurrent units and GloVe embeddings. Att-BiLSTM [81] enhances BiLSTM outputs with an attention mechanism. DBGAT [80] combines BERT embeddings with a graph attention network to capture word relations. Finally, BERT in [72] uses the transformer architecture to produce contextual embeddings directly for classification.

*Binary Classification (Functional vs. Non-Functional Requirements)*. The model was first evaluated on binary classification to distinguish functional and non-functional requirements using the PROMISE dataset. According to the reported results, BERT-BiCNN achieved 91% precision and 97% recall for functional requirements (F1-score of 94%) and 98% precision and 94% recall for non-functional requirements (F1-score of 96%). The authors reported an average F1 of 95%, which outperformed all compared baseline methods, all of which were re-implemented from prior studies for this evaluation.

*Binary Classification of Four Frequent NFR Classes.* A second binary classification task was performed individually on the four most frequent non-functional requirement classes: Operability, Performance, Security, and Usability. For Operability, BERT-BiCNN achieved 100% precision, 93% recall, and 96% F1-score. For Performance, it reached 94% precision,

[23] Jupyter Notebook – https://jupyter.org/

[24] TensorFlow – https://www.tensorflow.org/

[25] Keras – https://www.databricks.com/glossary/keras-model

[26] Hugging Face Transformers – https://huggingface.co/docs/transformers/index

83% recall, and 88% F1-score. For Security, the results mapped to 83% precision, 91% recall, and 87% F1-score. For Usability, the model obtained 100% precision, 73% recall, and 84% F1-score. The authors reported an average F1-score of 89% across these classes. These results exceeded those of other models, including BERT [72], which reached only 77% average F1-score across the same classes. This signifies that the added layers increase the model's performance for each NFR classification.

*Multi-Class Classification (4 NFRs).* In the multi-class classification task involving the four frequent NFR categories, BERT-BiCNN again outperformed all baselines. It achieved per class F1-scores between 92% and 97%, with precision ranging from 88% to 100% and recall from 86% to 100%. The authors reported an average F1-score of 94%, which was higher than the 89% obtained in the binary classification of the same classes in the best compared baseline DBGAT, demonstrating that the multi-class classifier performed better in this scenario.

*Multi-Class Classification (4 NFRs + "Others")*. A five-class version of the classification task was conducted by grouping all minority NFR classes into one "Others" category. BERT-BiCNN achieved per class F1-scores ranging from 84% to 89%, with precision between 78% and 97% and recall between 74% and 93%. The authors reported an average F1-score of 86%, outperforming all other models evaluated.

*Multi-Class Classification (All 10 NFRs)*. Finally, the most complex evaluation involved classifying all 10 NFR subtypes. The paper reported per class F1-scores and average results, with the model performing particularly well in categories, such as Security (Precision 94%, Recall 83%, F1-score 88%), Usability (Precision 90%, Recall 91%, F1-score 90%), Look and Feel (Precision 87%, Recall 81%, F1-score 84%), and Legal, where it achieved perfect results (Precision 100%, Recall 100%, F1-score 100%). Performance was lower in minority classes, such as Fault Tolerance (Precision 17%, Recall 33%, F1-score 22%) and Maintainability (Precision 70%, Recall 65%, F1-score 67%). The model achieved for all the ten NFRs an average precision, recall and F1-score of 79%.

Additionally, the authors performed a student's T-test comparing BERT-BiCNN with each baseline model and reported p-values $< 0.01$ ($\alpha = 0.05$), which indicates that the performance improvements of BERT-BiCNN over the compared techniques are statistically significant. The paper notes that as task complexity increased, from binary to 10-class classification, the performance declined correspondingly.

Further, an ablation analysis confirmed that the performance of the BERT-BiCNN model depends on the collective contribution of all three layers. Removing any single component resulted in a noticeable drop in F1-score. Specifically, removing the BERT or BiLSTM layer reduced F1-score from 94% to 87% (a decline of 7%), removing CNN reduced it to 89% (a decline of 5%), and removing the entire BiCNN module resulted in the most severe decline to 68% F1-score. These results indicate that while BERT provides the strongest individual contribution by capturing contextual semantics, the BiLSTM and CNN layers further enhance the model by learning sequential dependencies and selecting key features. The full

architecture proved more effective than any partial variant, justifying the use of all three components in combination.

What makes the BERT-BiCNN model novel is its combination of BERT with BiLSTM and CNN layers for classifying requirements. The model consistently showed high performance in all tasks (F1 up to 96%). According to the authors this work represents the first instance of integrating these three components into a coherent DL architecture for the specific task. The model's layered architecture enhances classification results for both binary and multi-class tasks, without needing manual preprocessing. Last, processing raw requirements is another strength of this approach.

However, the approach has certain limitations. Only the PROMISE dataset was used, which includes a small number of samples that are unbalanced, which affects the model's ability to generalize. Another limitation is that the study did not evaluate cross-domain generalization. Consequently, it is unclear how the model would perform when applied to requirements from entirely new projects or domains not represented in the training data. Last, FR multiclass classification task was not considered in the approach.

### 7. *One and Two Phase Software Requirement Classification Using Ensemble Deep Learning*

The study [75] proposes two classification methodologies for software requirements: a one-phase model and a two-phase models ensemble. The one-phase model implements directly a multi-class classification into both functional and non-functional subtypes (17 classes), while the two-phase ensemble first classifies requirements as functional or non-functional (binary classification) and then performs a multi-class classification. The goal is to improve classification accuracy using deep learning and ensemble methods.

The functional requirements (6) subtypes used in this paper are the following: Solution, Enablement, Action Constraint, Attribute Constraint, Definition, Policy. Solution is used to describe a requirement that specifies how a problem should be solved, often describing the implementation approach rather than the need itself. Enablement is defined as a requirement that specifies capabilities or conditions needed to allow other functionalities or processes to work. Action Constraint refers to a requirement that limits or regulates what actions the system or its users can perform. Attribute Constraint refers to restrictions on data attributes or entities (e.g., allowed values, ranges, or formats such as fixed option lists or input length). The Definition refers to a requirement that clarifies or formally defines terms, concepts, or entities used in the specification. Last, Policy refers to business or organizational rules that constrain system behavior.

The non-functional requirements (11) subtypes used in the paper are the following: Availability refers to when the system must be accessible, Fault Tolerance addresses the ability to keep operating despite faults, Legal & Licensing refers to the required licenses/records the system must maintain, Look & Feel addresses the visual style and appearance, Maintainability relates to ease or constraints of modifying the system, Operability describes how easy the system is to run across environments, e.g. on existing

hardware, Performance describes resources required under defined conditions, Portability addresses the ease of moving to other platforms/environments, Scalability the ability to handle growth in load or resources, Security the protection, authentication or authorization, and Usability the effectiveness, efficiency for the users. Even though these requirement types are not all standardized or formally endorsed by ISO, IEEE, or any standardised or widely adopted framework, they were first proposed in a research [90] to support automated quality checking in requirements and adopted by later papers such as [75].

Four deep learning models were used as base classifiers: BiLSTM, LSTM, GRU, and CNN. These models were trained independently, and their outputs were combined using three ensemble strategies: (a) mean ensemble, which averages the predicted probabilities from all models, (b) accuracy as a weight assigning each model a weight based on its overall validation accuracy, and (c) accuracy per class as a weight, this uses the model that performed best on each specific class to make the final prediction. These strategies were designed to aggregate the predictions of the four models to increase robustness and overall performance.

The two-phase architecture first processes input through a binary classifier (FR vs. NFR), then classifies the result using the appropriate multi-class classifier. The best performance in the two-phase classification setting was achieved when using CNN as the binary classifier in Phase 1 (functional vs. non-functional) combined with the accuracy per class as a weight ensemble strategy in Phase 2 for NFR subtype classification, which assigns higher weights to classifiers that perform better on specific classes.

The PROMISE dataset was used, where FRs were divided into six subcategories, and NFRs into eleven. Preprocessing included lowercasing, tokenization, lemmatization, and padding. Oversampling was applied to minimize class imbalance. The dataset was split into 35% training, 35% validation, and 30% testing. The models were implemented in Python using the PyCharm IDE[27], with machine learning functionality provided by the Scikit-learn, TensorFlow, and Keras libraries. Training was performed using the default hyperparameters and the Adam optimizer. Evaluation was conducted using standard accuracy metrics: accuracy, precision, recall, and F1-score.

In the one-phase model, the best performance was achieved by the "accuracy per class as a weight" ensemble method, which reached 92.56% accuracy, 93% precision, 93% recall, and 93% F1-score when directly classifying requirements into one of 17 FR or NFR classes. The best performance overall came from the two-phase model using the same ensemble technique, which achieved 93.40% accuracy, 94% precision, 93% recall, and 93% F1-score. According to the authors, the binary phase helped the system focus by filtering input and allowing the subsequent classifier to operate only within the relevant requirement type (FR or NFR). Furthermore, assigning class-specific prediction to the most accurate model in the ensemble increased overall robustness and performance. In the first step of the two-phase approach, the binary classifier for FR/NFR achieved up to 96% accuracy and F1-score, while in Phase 2, FR subcategories reached up to 98% accuracy/F1-score and NFR subcategories

[27] PyCharm IDE

achieved 86.5% accuracy with an 87% F1-score. When evaluated as a complete two-phase classification system, the best-performing configuration "accuracy per class as a weight" ensemble recorded 93.4% accuracy, 94% precision, 94% recall, and 93% F1-score, outperforming all other models and methods, including its own one-phase mentioned earlier.

The authors also performed a comparative analysis based on other relative work, including NoRBERT (2020). For the latter, it is stated that a 76% F1-score was achieved for classifying according to four non-functional requirement categories. However, this result is based on NoRBERT's 10-class multiclass classification using p-fold cross-validation, not a four-class task. Additionally, while NoRBERT's results were averaged across multiple runs using cross-validation techniques (10-fold, p-fold, loPo), [75] evaluate their model using a 35/35/30 train validation test split. These differences in evaluation methodology make this direct performance comparison unreliable, therefore, a safe outcome cannot be derived from this comparison.

This ensembled approach combining multiple deep learning models (BiLSTM, LSTM, GRU, CNN) is novel as it is not introduced in any other study, considering the studies which are hereby reviewed. Also, the authors created a single integrated process, combining the binary and multi-class classification tasks into one pipeline. Furthermore, the authors provide thorough performance metrics of accuracy, precision, recall and F1-score for multiple tasks and detailed insights into model performance.

However, the evaluation lacks rigor due to reliance on a single train/validation/test split without repeated trials or cross-validation. Further, the comparison with relative work is limited due to differences in evaluations performed in each study. Additionally, no cross-domain or cross-project assessments were conducted, so it is difficult to conclude that the approach would generalize to different domains. Finally, the FR subtype model introduced in this article (e.g., Solution, Enablement, Policy, etc.) as mentioned earlier, is not standardized and partially overlaps conceptually with common NFR subtypes (e.g., Policy vs. Legal/Compliance). This non-standard taxonomy may introduce annotation ambiguity and limits comparability with other studies that used ISO/IEEE-aligned categories.

8. *An end-to-end deep learning system for requirements classification using recurrent neural networks*

Study [77] proposed a system for classifying software requirements into functional and non-functional categories, as well as into their respective subtypes, employing a deep learning RNN Bidirectional Gated Recurrent Units (BiGRU) architecture that processes text in both forward and backward directions. The study attempts to contribute to creating a solution without the need of extensive preprocessing and establishing an approach able to handle besides binary and multiclass, multilabel classification as well, which allows a requirement to belong to more than one nonfunctional subtype classes.

The approach uses BiGRU models that receive raw words or raw characters sequences as input for classification. The input is processed using a minimal pipeline for data cleaning and normalization, with no stemming, lemmatization, or feature engineering. By eliminating

common preprocessing steps the authors attempt to create a solution that will be used easier in real scenarios and generalizes well.

The proposed BiGRU architecture includes an embedding layer, BiGRU layers, and fully connected layers. The embedding layer turns input tokens into dense vectors, then, dense layers with ReLU activation function are utilized, which help the model learn non-linear patterns. A dropout layer is added to prevent overfitting by randomly deactivating some neurons during training. The output layer uses sigmoid activation for binary and multilabel classification and softmax activation for multiclass tasks (to select one class with the highest probability).

The model was trained using a supervised learning approach with the Adam optimizer and appropriate cross-entropy losses (binary or categorical).

The system was evaluated on three publicly available datasets: PROMISE, used for binary and multi-class classification (4, 5, or 10 NFR classes), PROMISE-ML [82] and electronic health records (EHR) [82]. PROMISE-ML is a fine-grained, multilabel extension of the PROMISE dataset. It contains 792 shorter requirement sentences, each annotated with one functional or up to 14 non-functional categories (e.g., Availability, Security, Privacy), allowing some sentences to carry multiple labels. Many of these classes remain small, with fewer than 50 examples each. The EHR is a curated dataset that includes 12 projects from the health field and is consisted of functional and 14 NFR labels, some of which are highly imbalanced (e.g., Recoverability with only 50 samples), this underlines the challenge of class sparsity in multilabel classification. All datasets were preprocessed with minimal cleaning, including punctuation removal and case normalization.

Word sequences were mapped to a vocabulary (up to 5579 unique words), while character sequences used a fixed 38-character vocabulary. The models were implemented using Keras with TensorFlow, and experiments were run on Google Colab[28]. All models used tuned hyperparameters. Word-based models used a single BiGRU layer, while character-based models required deeper architectures (up to 3 BiGRU layers) due to reduced semantic richness.

The evaluation consisted of four tasks: binary classification, multiclass classification, multilabel binary classification, and multilabel multiclass classification.

Results are provided for binary, multiclass, and multilabel classification tasks. The proposed BiGRU word-based model achieves competitive or higher F1-scores in all settings. However, the authors do not make direct claims of best performance. Instead, they emphasize that their model performs well without requiring pre-trained embeddings, manual features, or complex preprocessing, unlike many of the models it is compared to. All tasks were evaluated using standard metrics: accuracy, precision, recall, F1-score, and Hamming loss (for multilabel tasks). Macro, micro, sample, and weighted averaging methods were used where appropriate.

[28] Google Colab

- *Binary classification results*: In binary classification, the word-based BiGRU model achieved accuracy 93%, an F1-score of 94%, precision of 93%, and recall of 95%. The character model achieved lower performance, with accuracy 85%, an F1-score of 88%, precision of 86%, and recall of 91%. These results confirm that word-based input provides stronger semantic features for this task.

- *Multiclass classification results*: In multiclass classification, the word-based models consistently outperformed the character-based ones. With 4-class classification (Usability, Security, Operational, Performance), the word model achieved 87% F1-score and the character model 70%. With 5-class classification (including an "Other" class), the word model achieved 77% F1-score, while the character model dropped to 63%. For 10-class classification (removing only the Portability class due to a single sample), the F1-scores were 75% (word) and 54% (character). As the number of classes increased, performance dropped, especially for the character-based models. The confusion matrices show frequent misclassifications between overlapping classes, such as Usability and Look and Feel, or Security and Availability.

- *Multilabel classification results*: In multilabel multiclass classification on the PROMISE ML dataset, the word model achieved a weighted F1-score of 69% and a macro-F1 of 44%, while the character model scored 52% weighted and 31% macro. The Hamming loss was 0.05 for the word model and 0.07 for the character model. Most samples had a single label, while a smaller number had two or three labels. The system struggled with rare classes like Fault Tolerance or Privacy, where there were fewer than 15 examples. According to the authors, the low accuracy on rare classes is mainly due to the small number of training examples.

On the EHR dataset, which has a broader label distribution and more samples overall, the performance improved. In multilabel binary classification, the word model achieved an F1-score of 89%, and the character model achieved 86%. In multilabel multiclass classification, the word model achieved an F1-score of 81%, while the character model trailed slightly at 75%. Hamming loss values in these experiments ranged between 0.13 and 0.17.

The authors also report standard deviations across the 10 folds in binary classification (e.g., ±2.2 for binary word model), a result that shows stability in word models and higher variance in character models (±4.1). Overall, word-based models were more consistent across tasks and less affected by label size.

Regarding the external evaluation, the authors also performed a literature comparison with previously published approaches, using reported metrics (accuracy, precision, recall, F1-score) from studies that evaluated classifiers on the same datasets. In particular, the authors compared their approach to several traditional and deep learning methods, including SVM[29], Decision Trees, Naïve Bayes, CNNs with Word2Vec and fastText embeddings, and BERT-

[29] Support Vector Machine (SVM)

based models like NoRBERT, reporting comparable or higher F1-scores (for example, 94% in binary classification and 69% in multilabel classification) across tasks. Although these comparisons were not conducted under the same experimental conditions, they indicate that the BiGRU models achieved competitive or better performance without requiring pretrained embeddings or extensive preprocessing.

A notable strength of this study is that, unlike many approaches that focus primarily on non-functional requirements classification or only binary classification, the proposed system also performs multi-class classification of functional requirements into six subcategories. The approach also supports several types of classification, including multilabel classification. Furthermore, the proposed approach avoids the complexities of feature engineering and complex preprocessing. This can be particularly useful for practitioners without NLP expertise while the system's architecture is publicly available for reproduction.

However, the proposed approach shows lower performance on classes with very few samples, especially in multiclass and multilabel classification tasks. Another drawback is that even though the study shows that the model can work on different datasets, it does not test whether a model trained on one domain can predict well on another. As a result, the generalization to new domains is not demonstrated. Additionally, computational performance metrics were not reported. Furthermore, the study includes only an indirect comparison with state-of-the-art approaches, using previously reported results instead of experimenting under the same conditions. There was no direct experimental comparison provided, particularly with approaches that also operate without complex preprocessing of input requirements like the current approach.

9. *Extracting and Classifying Requirements from Software Engineering Contracts*

Authors of [63] propose a method to extract and classify requirements from software engineering contracts. The motivation came from the observation of the authors that contracts often contain obligations, which are relevant to requirements and can be potentially detected even before formal RE begins. The study aims to automatically identify these requirements and classify them into distinct types, which the authors later group into two main domains: architectural and governance requirements.

The dataset used in this study included 20 expired SE contracts from 9 domains (e.g., healthcare, finance, telecom), with a total of 18.614 sentences. Each contract ranged from 100 to 500 pages. A group of five participants, one researcher, two software engineers, and two members of the contracts governance team, manually analyzed the contracts and labeled 5,472 sentences as obligations and 13,142 as non-obligations. This labeled dataset was then used as the ground truth for training and evaluating the models.

The system was built in two main steps. First, extracting obligation sentences from contracts, and second, classifying these into specific requirement types. The contracts were converted from scanned documents to an editable electronic format using an Optical Character Recognition (OCR) based in-house tool and structured in HTML for easier processing. Sentences were extracted using a tokenizer. A binary classifier was used to

classify the contract sentences as obligation or non-obligation. The obligations deriving from this process, referred below as extraction, are the potential requirements.

For the extraction step, both traditional ML models (Naïve Bayes, SVM, Random Forest) and deep learning (BiLSTM and BERT) were assessed. All models were implemented using Python libraries (Scikit-learn, Keras, BERT API) and evaluated using nested cross-validation.

The traditional ML pipeline included Naïve Bayes, Support Vector Machine, and Random Forest. As part of data preparation, the sentences were converted to lowercase, stop words and special characters were removed, and lemmatization was applied. Bigrams and trigrams appearing at least three times in the dataset were added to capture useful patterns. The cleaned data was then transformed into TF-IDF matrices as representations of sentences in the dataset, to be used as input in ML models for training. Requirement detection was treated as a first, binary classification task aimed to identify whether a sentence expressed an obligation or not. In the second task, the authors classified the obligation sentences into specific requirement types. A sentence could belong to more than one requirement type, so this was treated as a multi-label classification problem. The binary relevance technique was used by training 14 separate binary classifiers, one for each requirement type. Each classifier determined whether a sentence belonged to that particular class. The same features and training approach were used for all classifiers. Grid search and nested five-fold cross-validation were used to tune the models. These 14 classifiers correspond to the requirement types identified in the exploratory study, which include both governance related obligations (such as Project Delivery, Legal Process, and HR Laws) and architectural concerns (for instance Information Security, Data Privacy, and Export Laws).

The BiLSTM model followed a different architecture and was implemented using the Keras library. A Word2Vec embedding model was first trained on the contract dataset to capture domain word meanings. These embeddings converted the tokenized sentences to vector sequences. Then a BiLSTM layer was applied to capture the contextual relationships in forward and backward directions. An attention layer was applied to understand the most relevant words in each sentence and last the output layer used a sigmoid function to convert the final value to 1 or 0. For requirement detection treated as a binary classification task, the output layer contained one node. For requirement classification, which was treated as a multi-label problem, the output layer was expanded to 14 sigmoid nodes, one for each requirement type. A weighted binary cross-entropy loss function was used to address class imbalance, with weights derived from class frequencies. Model tuning was performed using nested cross-validation with grid search. The final configuration included the Adam optimizer.

The third model tested was BERT. Unlike BiLSTM, which learns from shallow sequences of word embeddings, BERT captures full bidirectional context using a transformer architecture. The model was pretrained using 13,142 unlabeled non-obligation sentences from the contracts. This step helped the model adapt to the domain-specific vocabulary and sentence structure. Pre-training involved two tasks: masked word prediction and next sentence prediction, which achieved 98.96% and 100% accuracy, respectively. After pretraining, the model was fine-tuned using the labeled dataset. Each sentence was formatted for BERT input, and training was performed using five-fold cross-validation. The output

layer used 14 sigmoid nodes, one for each requirement type. Each node produced a probability value, which was converted to 1 or 0 using a threshold of 0.5 to indicate whether the sentence belonged to that type. BERT was able to capture subtle linguistic patterns and differentiate overlapping categories more effectively than the other models.

During evaluation of the assessed models, binary classification of obligation versus non-obligation showed that BiLSTM outperformed all models. It achieved an F1-score of 93.0% for obligations (precision: 91.0%, recall: 94.0%) and 86.0% for non-obligations (precision: 89.0%, recall: 84.0%). Random Forest followed with F1-scores of 91.0% for obligations and 83.0% for non-obligations. SVM achieved 88.0% and 81.0%, respectively, while Naïve Bayes was less effective, with F1-scores of 80.0% and 74.0%. SVM and Random Forest frequently misclassified complex sentences containing obligation-like phrases. BERT was not evaluated for this task, as it was only used in the multi-label classification of requirement types.

In the requirement classification task, modeled as a multi-label problem, BERT achieved the highest performance across both frequent and rare categories. For example, in least represented classes, such as "Standards" and "Export Laws", BERT achieved F1-scores of 83% and 96%, respectively, outperforming all other models. In frequent categories, such as "Project Delivery", BERT reached an F1-score of 89%, slightly higher than BiLSTM, which achieved 85%. BERT also scored 90% for "Screening/Onboarding" and 86% for "Data Privacy". Other models, including BiLSTM, performed well in frequent categories but were less consistent in minority classes and tended to confuse overlapping types, such as "Information Security" and "Export Laws." BERT handled such cases more accurately due to its use of embeddings and bidirectional attention, which allowed it to model subtle semantic differences.

A validation was also conducted using a new contract containing 1608 sentences from the finance domain. Two domain experts, one from the RE team and one from the contracts governance team, manually annotated obligation status and requirement types. The automated system processed the document in 3 minutes and achieved an overall F1-score of 85.8%, whereas the manual annotation process required 35 person-hours. Both experts confirmed that the manually annotated dataset correctly captured all relevant requirement types (compared to the automated system) but noted that fatigue could compromise manual quality when dealing with large volumes of contractual data.

The authors demonstrate through their research that contracts have essential SE requirements especially related to governance that tend to be neglected. It is a unique point of view regarding requirements source and analysis. The paper also presented an entirely automated workflow process with relatively low execution time (3 minutes for 1600 sentences) while utilizing different models and validating results through real-world testing. This shows promising insights regarding practical adaptation. Another important positive point of this paper is that the approach proposed was able to handle multilabel classification. In particular, the deep learning approach with BERT effectively processed domain-specific language and showed strong performance with imbalanced and overlapping labels.

However, there are several drawbacks to this study. The analysis was based on 20 contracts of 9 domains and requirement types were not tested in additional industries. Further, the researchers acknowledge that their identified non-functional types might lack universal applicability. In addition, the study did not explore functional requirements classification in detail, nor it was experimentally compared against established state-of-the-art approaches. Finally, BERT's domain-adaptive pretraining relies on a vast in-domain dataset, and the paper does not assess performance without such data, so results may not perform the same way when in-domain data are unavailable. Also, BERT was evaluated only for requirement-type classification, not for the binary obligation detection step.

10. *Requirements and GitHub Issues: An Automated Approach for Quality Requirements Classification*

The study [58] proposed a machine learning method for classifying software quality requirements. The focus is on improving automation in classifying non-functional requirements, specifically quality attributes defined in ISO 25010. The motivation arises from the extensive time consumed and the high potential for errors in manual classification of large-scale projects requirements. The study included a systematic literature review to identify suitable models and datasets. Based on this, the authors developed and optimized several supervised learning classifiers. The main goal was to evaluate whether the quality requirements knowledge can be reused to classify issue reports from GitHub. The study addressed two problems: training models on well formed quality requirements and testing them on informal issue reports from user feedback.

The first step is the creation of the training dataset. Quality requirements were collected from three sources: the PROMISE dataset, additional examples from existing literature, and the PURE dataset. From these sources, 630 requirements were extracted and labeled with one of seven ISO 25010 quality attributes: Availability, Fault Tolerance, Maintainability, Performance, Scalability, Security, or Usability.

Domain-specific content was removed from the requirements through generalization before training. This was necessary to reduce noise and improve the generality of learned patterns. The generalization process included three natural language processing steps. First, time and percentage expressions were normalized using the SUTime[30] library and rule-based substitution. For example, "99% uptime" was replaced with "alltimes," and "less than 60 seconds" became "fast." Second, named entities, such as organization names, URLs, numbers, and geographic references were removed using Stanford CoreNLP's NER[31] model. Third, system and user roles were masked. This was done by tagging nouns using POS tagging and then manually replacing all user related terms with "user" and terms related with system with "system" using the Doccano [32] annotation tool. These changes reduced vocabulary size and removed irrelevant variation.

[30] SUTime: https://stanfordnlp.github.io/CoreNLP/sutime.html

[31] Named Entity Recognition: https://stanfordnlp.github.io/CoreNLP/ner.html

[32] Doccano: https://doccano.github.io/doccano/

Standard preprocessing steps were applied as well, including lowercasing, punctuation removal, lemmatization, and stop word filtering. The preprocessed text was then converted into TF-IDF representation vectors. These vectors served as input to the machine learning models.

Four supervised learning models were used: Multinomial Naïve Bayes, Decision Tree, Support Vector Classifier (SVC), and Random Forest. Each model was first trained via Skikit learn using default hyperparameters. Then, two optimization processes were applied. The first used Differential Evolution (DE)[33] from the SciPy library. This process searched for the best hyperparameter settings to address class imbalance in the dataset and achieved to increase the model's GMean[34]. G-Mean is an evaluation metric used during the training to compare the model's efficiency. This metric evaluates the relationship between the True Positives and True Negatives.

The second optimization method used TPOT[35], an AutoML tool that applies genetic algorithms to evolve full pipelines, including feature selection and classifier configuration. TPOT was run on three separate virtual machines due to its high computational cost.

After optimization, both default and optimized models were used to classify GitHub issue reports. The classification task was multiclass, with each issue assigned to one of the seven ISO defined quality attributes. Models were evaluated using weighted Precision, Recall, and F1-score.

Evaluation was conducted in two stages. In the first stage, models were trained and optimized using the previous mentioned structure dataset of 630 quality requirements. The dataset was split into 85% for training and 15% for testing. G-Mean was used during hyperparameter tuning to address class imbalance. TPOT achieved the highest G-Mean during training (83.63%), followed by Random Forest (82.51%) after DE optimization.

In the second stage, models were evaluated on 500 GitHub issue reports collected from five popular open-source repositories: ansible[36], flutter[37], kubernetes[38], vscode[39], and tensorflow[40]. Two requirements engineering experts manually annotated the data, identifying 185 issue reports as relevant to at least one quality attribute.

On this informal data, all models performed poorly, highlighting the difficulty of transferring models trained on formal requirements. Among the optimized models, the DE Naïve Bayes achieved the highest F1-score (28.6%), followed by the DE optimized Random Forest (22.9%), Decision Tree (18.0%), and SVC (21.4%), while the AutoML-based TPOT

[33] Differential evolution

[34] Geometric Mean (Gmean)

[35] TPOT

[36] Ansible repository

[37] Flutter repository

[38] Kubernetes repository

[39] Vscode repository

[40] Tensorflow repository

pipeline reached 24.1%. It is observed that while TPOT achieved the highest G-Mean during training on formal requirements (83.63%), its transfer performance on GitHub issues dropped below the DE-optimized Naïve Bayes, confirming the models' limited generalization to informal text. Results were similar for the default models. This drop in performance highlights the difficulty of applying models trained on formal requirements to informal issue descriptions. The authors attribute this gap to major differences in vocabulary, sentence structure, and terminology between requirement documents and user generated feedback. The results suggest that TF-IDF based models are not well suited to transfer learning across these contexts.

This study presents an innovative technique to classify non-functional requirements from informal text sources like user feedback without requiring formal requirement documentation. The method enables early classification of requirements by training machine learning models on structured quality requirements and applying them to user generated feedback right after elicitation. Analysts can detect quality related issues like performance or usability before requirements evolve into formal specifications. The approach has the potential to enable partial automation of certain tasks in RE.

However, the F1-scores were below 30% showing a significant discrepancy when compared to the high-performance results seen during clean training data experiments. The training of models on formal requirements fails to achieve effective generalization on informal and unstructured text because of discrepancies in vocabulary, structure, and tone. As the authors note in their conclusions, replacing TF-IDF with semantically richer representations, such as word embeddings or deep learning architectures (e.g., BERT), could substantially improve transferability and classification. Another limitation is that this method can only be utilized with existing systems that maintain ongoing user feedback. The approach cannot be applied during the initial engineering stages of completely new systems because any user feedback in this case would be impossible. However, it could still be employed on the formally specified requirements produced in these stages.

### *11. SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects*

Authors in [29] propose SRPTackle, a semi-automated requirements prioritisation (RP) technique designed to address limitations in existing RP approaches, such as poor scalability, time inefficiency, overreliance on expert input, lack of stakeholder quantification, and low automation. SRPTackle implemented in C#, aims to support software projects by enabling accurate and time-efficient prioritisation with low expert involvement. It includes a supporting tool, which is evaluated by seven experiments using a real-world large software project dataset. SRPTackle operates in two phases, pre-prioritisation and post-prioritisation.

In the pre-prioritisation phase, stakeholders assign weights on a scale from 1 (lowest weight) to 5 (highest weight) to requirements based on one out of two criteria according to their (stakeholder) type. The first criterion is importance (akin to functional stakeholders) and the second one is cost (akin to technical/commercial stakeholders). The two criteria aim to

create a balanced list of prioritised requirements based on the stakeholders' preferences. Stakeholders are divided in three types: functional beneficiaries (i.e., users and customers), technical stakeholders (i.e., development teams) and commercial stakeholders (i.e., business analysts and marketing managers).

Functional stakeholders assign the weights based on the importance criterion, and the resulting value is denoted as the requirements importance weight value (RIWV). Technical and commercial stakeholders assign the weights based on the cost criterion, and the resulting value is denoted as the requirements cost weight value (RCWV). These weights are calculated by taking the average of the weights assigned by each stakeholder group and are then used as inputs in the next phase.

The post-prioritisation phase comprises three main steps:

- *Calculation of Stakeholder Priority Value (SPV) and Requirement Priority Value (RPV)*. The first is calculated using the StakeQP tool [30]. This tool combines four attributes: role, power, interest, and experience, into a value reflecting the stakeholder's influence. The latter (RPV) is calculated using the multi-criteria decision-making method Weighted Sum Model (WSM)[41]. WSM is used to calculate an overall score for each software requirement by considering multiple evaluation criteria. It considers three inputs, namely RIWV, RCWV and SPV, to calculate the RPV of each requirement.
- The next step involves *clustering*. After computing the RPVs, requirements are grouped into three priority groups (High, Medium, Low). The K-means++ algorithm is used to initialise K cluster centroids (cluster center) and then the K-means algorithm is used to assign each requirement to a cluster (closest centroid) based on its RPV. The centroids are recomputed as the average RPV of the requirements in each cluster. This process is iterative and continues until the assignments no longer change, thus achieving convergence. This unsupervised approach allows the proposed tool to categorise requirements without predefined labels.
- *Application of Binary Search Tree (BST)*. SRPTackle uses a BST to rank requirements in each priority cluster based on their RPV. Each requirement is inserted into the BST and according to its RPV, if a requirement has a lower RPV than the previous one, it becomes the left child on the tree and if it has a higher RPV, it becomes the right child. In this way, requirements are ordered to produce an ascending priority sorted list.

Upon the completion of the process, the SRPTackle tool displays the final prioritised list with RPV scores and priority levels (H/M/L). It should also be mentioned that the tool user can upload files in Excel format so as to provide all the input needed by this tool, including the list of the requirements, the initial requirements' weighting values, and the SPV of participating stakeholders.

[41] Weighted Sum Model (WSM)

The evaluation of the proposed tool relied on the RALIC dataset [92], which contains 122 requirements (49 general, 73 specific), the input from 87 stakeholders, and a pre-established ground truth ranking for validation.

The authors evaluate SRPTackle tool on seven subsets of the RALIC dataset and in parallel they also compare the results against four baseline prioritisation methods from previous relevant work, each representing a distinct methodological category. Each of the previous baseline methods was also evaluated on the same RALIC subsets. These methods include the following: (a) StakeRare [92] is a frequency-based approach that ranks requirements based on how often they are selected by stakeholders, (b) GA [92], a search-based optimisation approach to evolve requirement rankings that balance cost and benefit, (c) Saffron [67], a semi-automated approach combining clustering and stakeholder preferences and (d) OSA [69], which ranks requirements by calculating scores based on ordered stakeholder preferences.

The metrics calculated in the evaluation were accuracy and execution time (measured in seconds for the full prioritisation process). We report their values over the examined methods/tools per each experiment, where each experiment focuses on different subset of the RALIC dataset:

- *Experiment 1*: This experiment compared the tool with three prior techniques: StakeRare, GA and OSA. It used a medium (in size) set of 49 general requirements from the RALIC dataset, which was the same dataset size used in previous work assessments. SRPTackle outperformed the other methods with 94.65% accuracy, while GA scored 92.28% and StakeRare 50%. OSA appears only in the time consumption comparison in the article (see relevant details later on) but was omitted in the accuracy tables of the report.

- *Experiments 2, 3, and 4* were conducted to evaluate SRPTackle against the Saffron technique under three different dataset sizes (50, 59, and 65 specific requirements), respectively, replicating the experimental procedures used in the original Saffron study. In Experiment 2 SRPTackle achieved 93.99% accuracy versus Saffron's 92.31%. In Experiment 3 SRPTackle scored 93.81% comparing to Saffron's 81.56%. In Experiment 4 SRPTackle achieved 93.00% while Saffron achieved 76.69%.

- *Experiment 5*: In this experiment 73 specific requirements were used, reproducing the same setting of GA's experiment. SRPTackle achieved a 93.71% accuracy, compared to GA's 91.35% and StakeRare's 71% accuracy.

- *Experiment 6*: A subset of 80 general and their specific requirements was used, in accordance with the experimental procedure in Saffron and OSA. SRPTackle scored 93.92%, while Saffron and OSA achieved a 67.56% and 88.04% accuracy, respectively.

- *Experiment 7*: The complete 122 requirements (49 general + 73 specific) within the RALIC dataset was used, in accordance to the evaluation setup of the StakeRare work. SRPTackle achieved 94.49% and StakeRare 60.50% accuracy.

Moreover, the time consumption in the SRPTackle was assessed by comparing the execution time (in seconds) needed to produce the final prioritised list of requirements according to three from the above experiments against two from the four alternative techniques considered: GA and OSA. StakeRare and Saffron were excluded from the timing evaluation due to their lower degree of automation. The respective performance results per experiment are the following:

- *Experiment 1*: SRPTackle required 5.02 seconds to prioritise 49 requirements, whereas OSA took 25.89s and GA took 200s.
- *Experiment 5*: SRPTackle took 5.48 seconds for 73 requirements, while GA took 200s.
- *Experiment 6*: For 80 requirements, SRPTackle took 5.52s, compared to OSA 40.64s.

Across all experiments, SRPTackle consistently maintained accuracy above 93% while keeping execution time under 6 seconds. Although the authors state that the execution time reduction over both GA and OSA was statistically significant, the *t*-test results reported show that this is only true for GA ($p$ = 0.03). For OSA, the $p$-value of 0.139 exceeds the 0.05 significance threshold average.

The novelty of the approach proposed is that the tool is designed to support large scale projects, producing results in a few seconds and can offer a major time saving because classical requirements prioritisation performed by humans can be very time consuming since it requires a growing number of stakeholder interactions, comparisons and effort. It also uses influence (SPV) based on role, power, interest, and experience which is difficult to be judged consistently by humans in classical RE. However, by automating this calculation, SRPTackle captures the influence of each stakeholder more accurately and reduces bias and manual effort in prioritizing requirements. The authors argue that this helps reduce bias and improves scalability.

However, all requirements are treated as independent, which may not reflect real-world dependencies, because frequently, requirements are related to each other and this factor must be taken into consideration. More specifically, in practice, dependencies often require that related requirements are prioritized consistently or that implementation order constraints are respected. Incorporating this capability would improve the tool's applicability to real-world scenarios. Additionally, the approach depends on accurate input values from stakeholders to calculate priorities. This can be seen as a limitation, since collecting and maintaining this information requires effort and careful management. A further limitation is that SRPTackle was evaluated only on the RALIC dataset, which restricts the judgement or assessment in terms of the generalizability of the model.

### 12. *CDBR: A Semi-Automated Collaborative Execute-Before-After Dependency-Based Requirement Prioritization Approach*

The study by [53] proposes CDBR, a semi-automated approach for the requirement prioritization activity in requirements analysis. This approach aims to resolve the often-

observed issue of conflicting priorities while generating accurate rankings based on consensus.

The proposed Collaborative Dependency-Based Ranking (CDBR) approach takes input from two sources: the stakeholders' and developers' prioritization on requirements. Stakeholders give their priorities in linguistic values (e.g. low, medium, high) by rating each requirement on importance and urgency. These linguistic inputs are given as input later in the model. Developers provide their input by analysing the execute-before-after (EBA) dependency relation among requirements through dependency graphs. This relation is kept in a dependency matrix to be supplied as input in the model as well. Each entry in the matrix indicates if a requirement must be implemented before or after another.

Once these inputs are collected, the system searches for a ranking of the requirements that satisfies these priorities and dependencies. To do this, the method uses Particle Swarm Optimization (PSO), a population-based search algorithm. In PSO, multiple candidate rankings, called particles, are tested in parallel. These candidates are improving their positions based on their performance. Each ranking is scored using a fitness function, which adds penalties when stakeholder or developers prioritisations are not satisfied and the goal is to find a ranking with the lowest penalty possible. The algorithm updates the rankings over several iterations until there is a convergence towards a solution with a low penalty. The final output is thus an ordering of all requirements.

The method can also handle cases where multiple requirements receive the same computed priority (i.e., duplicate ranks). Between requirements without dependencies and requirements with dependencies, the ones without dependencies are ranked higher. If all requirements have the same dependencies, any order is acceptable by the algorithm. If the requirements have others depending on them, the one with more dependencies is ranked higher, in order to be resolved the earliest possible and to ensure that foundational requirements are implemented earlier. Further, the algorithm handles conflicts between stakeholder priorities and developer constraints. When a requirement is marked as more important by a stakeholder but must be implemented after another due to EBA dependencies, the optimization algorithm enforces the dependency constraint while minimizing the impact on the overall ranking.

The authors evaluated their method using nine synthetic datasets and one real-world case study. The synthetic data was constructed through the random generation of multiple stakeholder and dependency matrices to reflect noise and conflict levels at 5%, 10%, and 20% input error. Further, the execution dependencies were transformed as matrices with different density levels (which indicate the complexity of the system). Requirements were represented as nodes, and EBA dependencies were modeled as directed edges in the graph.

Two empirical studies were implemented to evaluate the performance of CDBR. In the first experiment, the nine aforementioned synthetic requirement sets were utilised. The PSO algorithm was executed using population sizes of 50 and 100 with a set iteration limit of 100. The CDBR algorithm demonstrated rapid and consistent convergence throughout the experiments. The included PSO algorithm achieved convergence with only a few iterations for small datasets while reaching convergence at the 141st iteration (although the algorithm

was initially configured for 100 iterations, the authors extended the run length for larger datasets) for the largest data set consisting of 100 requirements. This demonstrates the method's scalability.

The second evaluation used a real-world case study, a cargo booking management system for warehouse operations (CBMW), which contained 53 requirements structured into modules and submodules. Five participants were involved, supplying their preferences: three external stakeholders with a software engineering background and two authors acting as developers. Stakeholder and developer priority lists served as input data for AHP, Interactive Genetic Algorithm (IGA) proposed by [91] and CDBR. CDBR achieved the best performance in requirement rankings, with only 4 disagreements between its output and stakeholder/developer inputs, compared to AHP (7 disagreements) and IGA (5 disagreements). Disagreement refers to the consideration of a value of 1, representing conflict between the two priority values (developer and stakeholder).

The robustness of the system was evaluated by inserting errors of 5%, 10%, 15%, and 20% into stakeholder or developer inputs. Ten executions of the algorithm were performed at every level of error rate. Results showed that disagreement values (i.e. the discrepancy between two preferences listed as ranking orders) remained stable, indicating CDBR's resistance to noisy input (14 at 5%, 15 at 10%, 13 at 15%, and 13 at 20%), compared to a baseline of 4 (disagreements) without error demonstrating the model's resilience to noise.

The same error procedure was applied to AHP and IGA and the resulting priority lists were statistically compared using ANOVA. The p-values against AHP were all below 0.05, indicating that CDBR's disagreement values were significantly better than those of AHP at all noise levels. In contrast, the p-values against IGA ranged from 0.0678 to 0.2660, all above 0.05, showing that CDBR and IGA performed similarly under noisy conditions. However, the paper does not report raw disagreement values for AHP or IGA under noise, so it cannot be confirmed whether those methods (CDBR & IGA) were equally stable.

The CDBR method combines technical and business perspectives during prioritization and formally integrates requirement dependencies which can be very useful in practice, while the Execute-Before-After modeling ensures that the implementation feasibility is not compromised. Additionally, the use of swarm intelligence reduces reliance on manual tuning, enables fast execution time for large data sets and supplies resistance to varying levels of noise.

However, the method requires accurate dependency input from developers. Further, scalability was tested only on synthetic datasets, limiting confidence in its performance on larger real-world systems. In addition, the use of meta-heuristic optimization (PSO) does not guarantee finding the optimal solution, however, it offers a practical advantage in quickly obtaining near-optimal results in large and complex problems. Further, other metaheuristic or kinds of optimization techniques could also have been explored and empirically compared to identify the most effective technique for requirements prioritization.

### 4.2.2. Articles based on Requirements Validation & Verification

*1. Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-Solution Study*

Authors in [27] proposed an approach to handle ambiguities detection and resolution. More specifically, the authors compared six different solutions for detecting and resolving anaphoric ambiguities in natural-language software requirements, using Natural Language Processing (NLP), Deep Learning (DL) and Machine Learning (ML) approaches. In RE, anaphoric ambiguity occurs when pronouns refer to different potential / alternative entities (antecedents), leading to a misunderstanding among different stakeholders. An example of anaphoric ambiguity is: "The S&T component shall send all approval requests to the DBS. If the request contains storage parameters, it shall create a configuration record from the parameters." In this requirement, the pronoun "it" can refer either to "the S&T component" or "the DBS," leading to multiple possible interpretations, and therefore, it is considered ambiguous.

In comparison with relative work previously proposed for detecting and resolving anaphoric ambiguities, [27] conduct their multi-solution study by using a large dataset, the Dataset for Anaphoric Ambiguity in Requirements (DAMIR), produced by the authors, comprising 1251 requirements with 737 pronoun occurrences across eight industrial domains. Further, they take into consideration the most recent technological advances in NLP and ML for their proposed solutions and enhance a previously proposed approach [105] with new features.

As part of the preprocessing, the requirements were analyzed using an NLP pipeline to reform data and extract suitable features for training purposes. First, the occurring pronouns in each requirement (POS tagger marks) were identified. Then, the context was also identified which equals to the requirement sentence, where the pronoun occurs, with the preceding sentence. This combined context is used to search for possible antecedents. Candidate antecedents (possible noun references) are extracted using constituency parsing, including compound noun phrases. To facilitate training, each candidate pronoun was manually labeled with one of four classes: Correct, Incorrect, Inconclusive, or Invalid[42], depending on whether the candidate antecedent correctly referred to the pronoun or not. These pairs were used as inputs for the machine learning and SpanBERT-based models, to train them to detect if a pronoun is ambiguous, and resolve the pronoun by identifying the most likely antecedent.

The structured dataset obtained through this process serves as input for the proposed models.

The datasets used for evaluation of the solutions are the following:

[42] Correct: The annotator is confident that a triple links the pronoun to the right antecedent, so the triple is marked correct.

Incorrect: Other triples for the same pronoun are marked incorrect if one correct triple exists.

Inconclusive: If the annotator is unsure, all triples for that pronoun are marked inconclusive.

Invalid: If a triple has a technical error, it is marked invalid and discarded.

- DAMIR, which, for resolving anaphoric ambiguity, was manually labeled with labels (Correct, Incorrect, Inconclusive, and Invalid) to test how well a pronoun is linked to its noun in each requirement.
- An external validation dataset ReqEval [94], consisting of 98 software requirements and 109 pronoun occurrences labeled as ambiguous or unambiguous, was used to assess model performance. This dataset was exploited to evaluate the generalization potential of the proposed solutions beyond the DAMIR dataset.

The solutions proposed and evaluated in the study are the following:

The first two models are based on SpanBERT [43] and they take as input the pronoun and its context (tuples) and predict the most likely noun phrase antecedent that the pronoun refers to. It must be noted that in addition to evaluating SpanBERT-based models, the authors designed and implemented four original approaches for this study. Three supervised ML solutions, MLLF, MLFE and MLensemble plus an NLP-based hybrid solution, NLPcoref. Alternative models 3, 4 and 5 are based on the supervised ML solutions with triples as input and 6 is based on the NPL coreference resolver with context as input.

1. *SpanBERT based solution: SpanBERTNLP*

The study fine-tuned the pretrained SpanBERT model on the CoNLL2011 dataset [23], which contains 7,000 pronoun occurrences from generic text to improve SpanBERT's performance on the anaphora resolution activity. This step refined the model's ability to recognize and resolve pronoun references before further domain tuning.

2. *SpanBERT based solution: SpanBERTRE*

SpanBERTRE was a further fine-tuned version of SpanBERTNLP, on a subset of DAMIR dataset. This additional fine-tuning goal was to adapt the model to the RE domain with examples of ambiguous and unambiguous pronouns.

3. *Supervised Machine Learning (ML) solution: MLLF*

MLLF (Machine Learning Language Features) was trained on 45 language features (LFs) derived from NLP and RE literature, capturing grammatical and referential relationships.

4. *Supervised ML solution: MLFE (Machine Learning Feature Embeddings)*

This model uses 768-dimension feature vectors representing the feature embeddings from BERT and SBERT, which provide deep semantic and syntactic representations of text sequences.

5.*Supervised ML solution: MLensemble (Machine Learning Ensemble model)*

This is a hybrid solution that combines MLLF and MLFE. It assigns a final label based on agreement between the two models or, if they disagree, on the prediction with the highest probability.

*6.Solution based on NLPcoref (NLP coreference resolvers)*

NLPcoref, is a coreference resolution approach that uses two existing NLP tools to determine the exact reference of a pronoun. It works by combining the results from two different

[43] SpanBERT

coreference resolvers: one from Stanford CoreNLP and another from the SpaCy library. These tools analyze the context independently, and NLPcoref compares their results to reach a final decision through consensus.

After multiple configuration testing and fine-tuning, involving tuning hyperparameters, evaluating different model architectures, feature extraction methods, the best-performing settings were selected for each solution to achieve the best results of the solutions and then ambiguity detection accuracy was evaluated using precision, recall and F-score metrics. The evaluation of ambiguity resolution was conducted using the success rate metric, which is the ratio of correctly resolved pronoun occurrences to the total number of pronoun occurrences labeled as unambiguous.

Based on the results of the conducted evaluation, the authors finally propose a hybrid solution that consists of:

(a) MLensemble with Precision 60% and Recall 100% for detecting anaphoric ambiguity in requirements

(b) SpanBERTRE for ambiguity resolution as it achieved the highest Success Rate 98%, which makes it the most accurate solution for resolving anaphora in requirements.

Concerning the performance of the finally proposed solution, the overall execution time per pronoun for ambiguity detection and resolution was around 16 seconds. It could be split as follows:

- MLensemble took ~14.5s per pronoun for ambiguity detection.
- SpanBERTRE took 1.5s per pronoun for resolution.

This study is innovative because it systematically develops and compares six alternative solutions for detecting and resolving anaphoric ambiguity in software requirements, combining supervised machine learning, pre-trained language models, and existing NLP tools, coming up with the most important and significant contribution, the proposed hybrid solution with 100% recall on detecting ambiguity and ≈98% success on resolution. Further, the authors created a large-scale industrial requirement dataset DAMIR based on which the hybrid technique was empirically compared and benchmarked with previous baselines.

However, a challenge that should be taken into consideration is that the Precision (~60%) indicates that false positives (unambiguous pronouns marked as ambiguous in sentences) still exist in the results, which this leads to increased execution time and unnecessary resolutions, which can be also erroneous. Further, the execution times of the ML solutions are high, and optimizing them for a real-world application might be required. Another critical issue is that the DAMIR dataset, while larger than previous datasets, is author-created and not widely adopted, and the only external dataset (ReqEval) is small. This limits the generalizability of the findings. [25] Finally, the study did not compare its methods against newer LLMs or more diverse ML approaches, which could offer additional benchmarks for performance.

*2. Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements*

Authors [28] attempt to tackle the problem of syntactic ambiguity in NL requirements. Unlike prior work that detects predefined patterns of ambiguity, this study introduces the Score-Based Ambiguity Detector and Resolver (SBADR), a tool that aims to detect and resolve syntactic ambiguity without relying on static templates. It focuses on three types of syntactic ambiguity, the coordination, attachment, and analytical[44] ones, using a dynamic method based on scores. Besides alerting users for the occurrence of ambiguity, SBADR also suggests correct interpretations, helping to resolve these ambiguities. In this respect, this tool addresses a broader variety of ambiguity types and aims to improve recall and user support compared to existing tools like AmbiGO [70].

SBADR processes each sentence from a requirements specification document and parses it into multiple syntactic interpretations using Stanford CoreNLP. The tool applies a four-stage filtering pipeline to identify meaningful parse trees[45]**.**

First, the authors filter out parse trees that do not form grammatically complete sentences. Then, they remove structurally identical parse trees. Following these, they also filter out parse trees with redundant grammatical roles (typed dependencies), for instance, if two parse trees have the same set of grammatical relations between words (e.g. both identify "the boy" as the subject and "with the telescope" as a modifier of "saw"), the lower scoring one is discarded. Finally, they apply a score-based filter. From the remaining interpretations, only those whose scores fall within a 10% variance of the highest-scoring interpretation are kept. These final interpretations are then presented to the user with visualized bracketed structures to assist in their understanding.

SBADR was evaluated through two experiments. At the sentence level, the tool determines whether each requirement sentence is ambiguous or unambiguous (ambiguity detection). At the interpretation level, it evaluates the correctness of the multiple syntactic interpretations generated for each ambiguous sentence (ambiguity resolution).

In the first experiment, SBADR was applied on the AmbiGO dataset. The dataset included 50 cases of analytical ambiguity, 28 cases of coordination ambiguity, and 22 cases of attachment ambiguity. BADR achieved an average recall of 99%, precision of 65%, and F1 score of 77% at the sentence level ambiguity detection. At the generated interpretation level, SBADR achieved an average recall of 96%, precision of 62%, and F1: 74%.

These results were compared to the relative previous work implemented with the same dataset. SBADR outperforms AmbiGO in recall measures with around 80% or more improvement for all types of ambiguity. Further, SBADR has a better F-measure which reflects that the improvement in recall outweigh AmbiGo's lead in precision. On the other hand, AmbiGO outperforms SBADR in the precision for both attachment and analytical types

[44] Coordination ambiguity occurs when a conjunction allows different groupings of elements (e.g., "young men and women" can mean (young men) and (young women) or (young men) and women)

Attachment ambiguity happens when it is unclear which part of the sentence a phrase is linked to (e.g., "I saw the boy with the telescope" could mean the boy had the telescope or the speaker used it to see the boy.)

Analytical ambiguity arises when the role of words in a compound noun phrase is unclear (e.g., "general assistance program" could mean a program of general assistance or a general program of assistance.) [28]

[45] Parse Trees

of ambiguity (around 45%, and 35% improvement respectively), while providing around 10% improvement in coordination ambiguity.

In the second experiment, to assess SBADR's capability beyond predefined ambiguity patterns, the authors evaluated it on 26 complex cases of syntactic ambiguity that were previously curated from publicly available linguistic resources. These examples represent challenging ambiguity scenarios not handled by AmbiGO or other template-based tools and were therefore not used for comparison against other baselines.

The evaluated ambiguity types included are attachment and coordination types. SBADR successfully detected 100% of the coordination ambiguities and 90% of the attachment cases.

At the interpretation level, it achieved F1 score of 88% for coordination and 87% for attachment, confirming its effectiveness in both detection and resolution. At the sentence level for ambiguity detection, it achieved 100% precision, recall and F1-score for coordination cases, and 100% precision. 90% recall and 94% F1-score for attachment cases. At the interpretation level, SBADR achieved 82% precision, 95% recall and 88% F1-score for coordination cases, and 81% precision, 93% recall and 87% F1-score for attachment cases.

The method presented in this study called SBADR, introduces a score-based ambiguity detection and resolution framework that can work with real world NL requirements and does not require static templates for ambiguity detection. This tool also produces possible correct interpretations, to assist in ambiguity resolution. Its evaluation included both established datasets and complex cases beyond the scope of existing tools, demonstrating high recall.

The method's precision is limited by the statistical nature of the Stanford CoreNLP parser, which can assign high scores to semantically incorrect interpretations. Additionally, the tool currently lacks a semantic-level analysis, which leads to false positives in unambiguous cases. Lastly, performance may degrade when applied to longer documents or in the context of the use of domain-specific language in requirement documents that was not seen during parser training.

3. *Leveraging LLMs for the Quality Assurance of Software Requirements*

The study by [31] investigates how a Large Language Model (LLM) can support and improve the quality assurance of software requirements in the requirements validation and verification phase. It is evaluated whether an LLM can correctly assess requirements based on quality characteristics as they are defined in the ISO/IEC/IEEE 29148 standard. An empirical study was conducted to compare the LLM assessment with that of human participants.

First, it is shown how an LLM can be instructed to evaluate the quality of software requirements. Second, the authors analyze the LLM's ability to explain the assessment it performs in order to improve transparency. Third, the authors investigate the capability to suggest improved versions of faulty requirements, to resolve the quality issues identified.

To achieve the above, the authors used the Llama[46] Chat LLM, an open-source language model developed by Meta AI. The model was hosted by Replicate[47] with temperature = 0.01.

[46] Llama

The experiment was based on the issuance of prompts on the LLM. More specifically, each input prompt, which was a fixed prompt template, included a project description, a definition of the quality characteristic (e.g., unambiguous, verifiable), and the requirement. The quality characteristics assessed in this study for individual software requirements were nine: Appropriate, Complete, Conforming, Correct, Feasible, Necessary, Singular, Unambiguous and Verifiable. The LLM was instructed to answer if the requirement fulfilled the quality criteria, to explain with reasoning, and to provide a corrected version of the requirement. All evaluations, comparisons, and improvements were handled through these static prompts and reply interactions with the model.

Two project datasets were used for evaluation. The first one was from the Stopwatch Project, a small fictional Android app, consisting of ten requirements (seven functional, three non-functional) that were generated using Llama 2. The second dataset was from Digital Home Project, a real smart home project from the PURE dataset. It includes 63 requirements (42 functional, 21 non-functional).

Four participants were involved for the evaluation in this study with background in software engineering, who were given definitions of the ISO quality criteria and the project scope. To answer the question how accurately LLMs can identify quality issues, the participants judged whether each requirement fulfilled each quality criterion. Study participants evaluated requirements with and without seeing the LLM's response (independent and bound phase respectively). Then, the agreement between the LLM response and the human classification of the same requirement, either as satisfying a quality criterion or not, was measured using Cohen's kappa (κ)[48], that captures inter-rater agreement beyond chance.

The first evaluation phase was the independent phase (where participants were unaware of the LLM response). For Stopwatch project the result is $\kappa = 0.40$ (weak agreement) between human and AI agreement. For Digital Home project the result is $\kappa = 0.05$ (no agreement).

The second evaluation phase was the bound phase (where participants saw LLM response before evaluating). For Stopwatch project the result is $\kappa = 0.75$ (substantial agreement) and for Digital Home project the result is $\kappa = 0.22$ for Digital Home (fair agreement).

Further, precision and recall were measured. Precision was measured as how many of the requirements the LLM marked as quality faulty were actually faulty based on human agreement in each phase. Recall shows the percentage of the real flaws the LLM managed to identify.

The LLM reached a recall of 89% for Stopwatch and 74% for Digital Home, but precision was quite lower: 50% for Stopwatch and 12.6% for Digital Home in the independent assessment phase.

In the bound assessment, Stopwatch precision increased to 88%, and recall to 100%. For Digital Home, precision increased to 32% and recall also to 100%. This indicates that participants agreed in some cases with the LLM's reasoning.

[47] Replicate

[48] Cohen's kappa (κ)

Furthermore, the participants assessed whether the LLM can provide reasonable explanations for its decisions for each project and quality criterion. The results were for Stopwatch above 80% and for Digital Home above 70% (these are percentages of LLM-generated explanations judged as plausible by participants).

Last, it was examined whether the LLM can suggest improved versions of faulty requirements. In the Stopwatch project most of the LLM's suggestions were considered better by all participants in most criteria (100% agreement), while one criterion (singular) achieved slightly lower agreement (90%). For the Digital Home project the agreement among the assessed criteria varied between 50% and 100.

The novelty of this study is that it is one of the first to evaluate LLMs for requirement quality assessment based on ISO requirements quality standards. It utilizes a setup that can be relatively easily reproduced. The LLM feedback helped participants reconsider their evaluations, a methodology that could be applied to real requirements reviews by stakeholders during the RVV phase. In addition, this study provides useful insights about LLMs, such as their strengths in detecting missing or unclear information, forming textual improvements, and providing reasoning across certain criteria, as well as their weaknesses, such as low precision. Last, ML models can be easily accessible through public platforms and can be used without deep technical expertise to provide useful suggestions for improvement, which make them practical for integration into everyday RE activities.

However, only one LLM (Llama 2) and one prompt structure were tested, while other prompts or prompting patterns could be assessed to increase the results effectiveness. In addition, the data came from only two projects (one fictional and one real but small) with quite a limited size (in terms of the number of involved requirements). This greatly affects the evaluation results generabilizability. Furthermore, the attained low precision levels suggests that false positives are common and must be somehow resolved. Finally, the proposed method was not compared with other related methods/approaches.

*4. Can GPT-4 Aid in Detecting Ambiguities, Inconsistencies, and Incompleteness in Requirements Analysis? A Comprehensive Case Study*

The study [17] assesses OpenAI's GPT-4 in supporting RE in large-scale industrial software projects by investigating if in a zero-shot setting, it can detect specific defects in Software Requirements Specification (SRS) documents. Three defect categories were taken into consideration, namely ambiguity, inconsistency, and incompleteness. The study also explores GPT-4's ability to identify issues across different SRS versions. The performance results are provided through qualitative and quantitative evaluations.

The authors used the Mechanical Lung Ventilator (MLV) case study, a system that supports patients with breathing difficulties due to health issues. This case study SRS consists of 52 pages and five sections with textual requirements and diagrams. The sections were Introduction, System Requirements, GUI, Controller, and Alarms. Data preparation involved data cleaning and formatting and removing extraneous information to ensure accuracy. Diagrams were depicted in standard UML, presening system components and relevant aspects. To utilize them as well, GPT-4 Vision was used to extract the descriptions of the

figures, noting that its output required multiple corrections due to frequent misunderstanding of directions of arrows or missing transitions.

In this proposed approach no new tool was implemented. The authors implemented a pipeline where they fed the extracted text segments and figure descriptions from the SRS sections and sub sections into the GPT API for analysis by the GPT model. For the evaluation, each section of the SRS was analyzed using prompts dedicated for each defect type. The prompts were standardized. For instance, for ambiguity detection the following prompt was used: "Are there any ambiguities or unclear statements within the <sub-section>? (Evaluate any ambiguities or unclear statements within the <sub-section>.)". The outputs were manually reviewed by two annotators who marked them as true or false detections. Further, true detections were marked with a severity scale, i.e., how much the defect could affect (requirement) understanding or implementation if left unresolved, from 1 (minor) to 5 (critical). False positives were reviewed to identify whether they resulted from lack of domain knowledge by the model or from context limitations, for instance the inability to reference to different requirements across document sections. The model was also tested on five SRS versions, which correspond to successive revisions of the same Mechanical Lung Ventilator specification, each improving upon the previous by correcting issues occurred, to check its ability to detect defects or verify previously corrected ones were no longer flagged as defects. In the following paragraphs each evaluation is examined, consisting of a qualitative and a quantitative analysis including how detection results evolved across different SRS versions.

***Ambiguity Detection***

Qualitatively, GPT-4 managed to identify vague phrasing, missing system equations, and missing numerical calculations, especially in the System Requirements and GUI sections. Some flagged requirements lacked mathematical formulas or contained vague terms. However, many false positives occurred due to lack of domain knowledge or an isolated analysis of one requirement without linking to other details and context.

Quantitatively, GPT-4 achieved a precision from 17% to 60% for ambiguity detection in different SRS sections. For System Requirements 60% of ambiguities were correctly identified. For GUI section 41%, for Controller section 17% and for Alarm section 38%. Most correct detections were marked with minor importance (severity 1 or 2), while 4 were of moderate importance (severity 3) and one was rated high (severity 4).

In a secondary experiment, the authors tested whether providing GPT-4 with full-section or global context would improve ambiguity detection. When entire sections were analysed without breaking them into sub-sections, it flagged four potential ambiguities, of which two were correct resulting in a precision 50%. Additionally, when full SRS context was given in the original prompts, most of the previously observed false positives persisted. This showed that GPT-4 has limitations in using the broader context (due to its restricted context window size) and referring to information in different sections.

**Inconsistency Detection**

Qualitatively, GPT-4 identified inconsistencies linked with lack of clarity, i.e., conflicting requirements, or due to the terminology used in the requirements and correlated effectively

the UML diagrams with the requirements. For example, an inconsistency that arose from lack of clarity occurred in access-control requirements where the terms "authorized personnel" and "healthcare professional operator" were used interchangeably but never defined. Since their relationship was unclear, the requirements appeared contradictory about who would access the system function. Regarding terminology issues, the SRS stated that "users can set alarm thresholds", but also that "operators cannot save alarm presets". Since the terms "threshold" and "preset" were not explicitly differentiated, GPT-4 flagged a contradiction. However, some false positives came from misreading diagrams or misunderstanding how valid alternatives can exist in parallel in the system flows.

Quantitatively, GPT-4 achieved a precision from 33% to 57% for inconsistencies detection although it is unclear whether the model received full section text or subsections. Specifically, in System Requirements category the precision was 57%, in GUI 50%, Controller 50% and Alarm 33%. Out of the eight inconsistencies found, six were minor (severity 1–2) and two were rated high (severity 4 and 5).

**Completeness Detection**

Qualitatively, GPT-4 performed well in identifying missing information like undefined roles, absent default values, and incomplete rationale. In some cases, it even referenced relevant industry standards (like IEC or HIPAA), despite not being explicitly prompted to do so. However, some false positives were caused by its inability to infer implicit assumptions, or by failing to recognize that some values were provided in other sections.

Quantitatively, GPT-4 achieved a precision from 50% to 92% in completeness detection. Specifically, in System Requirements section the precision was 60%, in GUI 92%, in Controller section 50% and Alarm section 92%. Among the true positives, 78% were low-to-moderate in severity, and 22% were rated high (severity 4), particularly those related to safety and regulatory compliance.

**Diagram Ambiguity Detection Evaluation**

Using GPT-4 Vision, the authors evaluated diagram requirements for ambiguity. The model sometimes identified missing transitions or inconsistently labeled states but often misinterpreted flow directions, misread arrow behavior, or flagged valid representations as flawed. The precision in diagram analysis was 44% (7 true positives, 9 false positives). Most valid detections were minor, with only one rated as moderate (severity 3).

To summarize the above, the authors report average precision scores of 61% for completeness, 43% for inconsistency, and 39% for ambiguity detection.

Another evaluation the authors performed was to use five versions of the SRS to evaluate GPT-4's ability to detect defects across document versions. Through the publicly available discussions from the ABZ 2024 GitHub repository (containing issue reports and discussions for the same MLV software project) that provided known defects that led to new versions, GPT-4 was able to identify 22 out of 49 known versioned defects, resulting in a recall of 45%. Some issues were detected before they were revised by human reviewers, however some defects were found only in earlier versions and missed later. This indicates that rewording or restructuring a requirement can make the underlying defect harder for GPT-4 to

detect. Others, especially those related to diagrams or complex interactions, were never detected.

Furthermore, to compare GPT-4 identified defects against those found through expert analysis, the authors examined the ABZ 2024 conference proceedings, where participating teams had applied formal modelling to the same MLV SRS to identify requirement defects. GPT-4 detected 6 out of 10 known defects, resulting in 60% recall.

Finally, the authors examined the effectiveness of GPT-4 in automated assistance in requirements validation & verification activity through a question-answer approach. They tested GPT-4 on a validation dataset of 18 questions. Each question sourced from the GitHub issues section of the project official repository, where ABZ conference participants expressed their queries, which were answered by MLV system experts (ground truth answers). With section-specific context to answer a specific question fed to LLM, GPT-4 correctly answered 77% of the questions. With full global context given to LLM, its performance dropped to 62%.

It is suggested that LLMs excel at extracting specific technical details from comprehensive SRS documents through contextual understanding and utilizing external knowledge to synthesize answers. While responses often referenced relevant SRS content, they were significantly longer than expert answers and struggled with diagram-related logic. These results suggest that GPT-4 can assist in early defect detection and stakeholder communication but lacks the precision and conciseness required for replacing expert review processes.

A key strength of the study is that it evaluates GPT-4, a widely accessible LLM, using a realistic industrial SRS. The evaluation combined qualitative and quantitative analysis considering three requirements defect categories. In addition, by providing a beyond typical output evaluation by including a version history analysis and a question-answering activity evaluation, useful insights are offered about how GPT-4 might assist analysts in real-world settings. The study applies zero-shot prompting directly on a long requirements document without the need of fine-tuning, which makes this approach easily transferable.

On the other hand, the study does not compare GPT-4's performance with other LLMs, which would help position its results within the broader landscape of AI tools/LLMs for RE. It also uses only one prompt style and does not explore other prompts and prompting patterns. The authors do not address how to manage or restrict input size relative to the LLM's context window. Context window is likely to contribute to the drop in accuracy when the whole SRS document is provided to the LLM. While the authors propose few-shot prompting and improved context-reasoning as future research directions, they do not suggest practical strategies for handling long-document inputs.

*5. Improving requirements completeness: automated assistance through large language models*

This study [16] explores the utilization of a pre-trained BERT model to assist in detecting incompleteness in natural language software requirements and resolving it through a list of suggestions (predictions) of words absent from the requirements. In parallel, it aims to figure

a balance between the simulated prediction and errors to avoid generating unhelpful predictions or noise. In order to achieve this balance, BERT predictions are post processed by filtering with machine learning to reduce the noise occurrence. The methodology applied in the study consists of six steps.

The first step is parsing the requirements using an NLP pipeline, to create annotations including tokens, POS taggers, sentences and lemmas.

The second step is masking verbs/nouns to generate BERT predictions. This is implemented by masking verbs or nouns of each sentence based on POS tags. The output is provided as input to BERT to generate a configurable number of predictions. BERT generates a probability score for each prediction.

The third step involves filtering out insignificant predictions. The words that are already present in the requirement or are too common and vague are discarded.

The fourth step is generating a domain-specific corpus for the requirements. The authors use a tool called WikiDoMiner to automatically build a domain-specific text corpus based on the input requirements. They limit the tool to only fetch directly articles whose titles exactly matched the domain terms extracted from the requirements, rather than also including related articles from subcategories, in order to keep the corpus focused and reduce noise.

In the fifth step a feature matrix is built for ML-based filtering of nonrelevant terms. Each predicted word is turned into a feature vector and passed to a machine learning classifier to decide if it is relevant to the requirements document. The features are designed to be generic and normalized so the model works across domains and new documents. During this step, syntactic and semantic similarity measures are also computed for each prediction. Syntactic similarity is calculated based on the Levenshtein distance and a character-length ratio between the masked and predicted terms, identifying words that are syntactically close. Semantic similarity is evaluated through cosine similarity over GloVe word embeddings, applying an 85% threshold to capture semantically related terms for example "encryption" and "security".Some features involve ranking predictions by frequency and calculating term importance. Together, these metrics help represent both lexical and semantic relationships within the feature matrix, enabling the classifier to better distinguish relevant from nonrelevant suggestions. This configuration helps the system filter out irrelevant suggestions and keep only the useful ones.

In the sixth and last step predictions are classified as relevant/non-relevant using a classifier fed with the feature matrix. The output is a list of recommended terms missing from the requirement.

The approach was implemented primarily in Python. SpaCy was used to build the NLP pipeline while for word embeddings GloVe was utilized. BERT generated masked language model predictions using the Hugging Face Transformers library operated in PyTorch. The authors computed TF-IDF-based features with scikit-learn while WikiDoMiner constructed a domain-specific corpus from Wikipedia. The baseline synonym sets were expanded through the application of WordNet.

To evaluate how accurately BERT can predict relevant but missing terminology for an input requirement, the authors simulated incompleteness by masking terms in the PURE

dataset's requirements. They configured BERT to generate 15 predictions per masked term and assessed how many of the original terms were correctly recovered. This resulted in approximately 38% coverage (BERT's suggestions recovered about 4 out of every 10 hidden terms) and 12% accuracy (roughly 1 in 8 predictions was relevant) in the 50–50 split scenario without filtering. Although this demonstrates reasonable coverage, the accuracy remained too low for practical use without additional filtering.

To address the aforementioned issue, the authors evaluate two classifiers, Random Forest and SVM, for post-filtering BERT's predictions. They experiment with three filtering configurations: (1) strict filtering using the full training set, (2) moderate filtering with under-sampling, and (3) lenient filtering using under-sampling combined with Cost-Sensitive Learning (CSL). CSL can improve accuracy by assigning different wrong classification costs to different classes or types of errors, to minimize the prevalence of certain types of misclassified data. In the strict filtering setup, Random Forest achieved the highest accuracy (84.1%), followed by SVM (80.3%). In terms of precision, SVM performed better (83.6%) compared to Random Forest (67%). For lenient filtering, where the goal is to maximize recall and avoid discarding useful suggestions, SVM was preferred, achieving 76.8% accuracy and 92.4% recall.

The final evaluation tested how the system performs on unseen documents under both simulated scenarios. In S1 (50% masking), the accuracy without filtering was 12.1%. With lenient filtering, accuracy increased by approximately 13%, while coverage decreased by around 5%. In the more realistic S2 (10% masking), accuracy without filtering was 2.25%, and lenient filtering improved it by 1.76%, maintaining a relatively high coverage of 33%. Although the absolute accuracy values appear low, this is expected since only a small fraction of terms are masked in S2, making it a much harder setting. Overall, these results suggest that while filtering improves suggestion quality, the system still struggles to reliably recover missing content, especially under mild incompleteness.

This study innovation is that it eliminates the need for domain-specific resources by using pretrained BERT and Wikipedia as knowledge sources, making the approach domain-independent and applicable to unseen documents. Further, the study is thoroughly evaluated under two scenarios (small and large incompleteness), with statistically validated results.

However, the study also has limitations. The evaluation is based on simulated incompleteness instead of real incomplete requirements. Another important limitation of the study is the lack of an overall performance evaluation of the effectiveness of the complete approach towards requirements incompleteness tracing and resolution. While the paper presents detailed results for coverage and accuracy under many filtering levels and incompleteness scenarios, it does not provide a unified measure, which makes it difficult to compare that work with other approaches in requirements incompleteness detection. Additionally, the performance remained low in the evaluation (e.g., ~4% in the S2 scenario), indicating limited practical effectiveness in detecting incompleteness. Finally, the approach does not consider sentence-level contextual semantics, as similarity is computed only between individual words.

*6. AI-Based Question Answering Assistance for Analyzing Natural-Language Requirements*

The paper [12] introduces an AI-based question-answering (QA) tool called QAssist, designed to support the RVV phase, by helping analysts detect quality issues in natural language (NL) requirements. According to the paper, it is often observed that during analysis stakeholders have clarification questions and these questions cannot always be answered from the SRS alone.

QAssist aims to address this by retrieving relevant texts from the SRS and from an external domain-specific corpus, then extracting probable answers from this information. This setup allows QAssist to mitigate missing knowledge, incompleteness, or inconsistencies in requirements. The approach enables analysts to ask natural language questions and receive contextual answers, supporting faster and more accurate review of requirements. Besides QAssist implementation, the authors develop REQuestA, a dataset consisting of 387 question answer pairs, built from six industrial SRSs across three domains, combining human-authored and machine-generated questions. This is the first large-scale dataset of its kind tailored to clarification questions in RE.

QAssist is structured as a four-step pipeline that combines information retrieval and machine reading comprehension. It takes as input a natural-language question and an SRS document. In the first step it retrieves the most relevant document from a domain-specific corpus (using a document retriever). If no corpus exists, QAssist automatically builds one from Wikipedia based on the existing terminology in the SRS. In the second step, the SRS and the retrieved document are split into text passages of up to 512 tokens using sentence segmentation.
In the third step, a passage retriever selects the top-k passages relevant to the question from both sources. Recall@k, which measures the proportion of relevant passages retrieved in the top k results (i.e. the system's ability to retrieve all relevant items within the top-k results), was used to evaluate retrieval performance. The reranking retriever achieved Recall@1 of 78.9% and Recall@10 of 92.4% for SRS documents, and k=3 was chosen as a compromise between comprehensiveness and speed. In the fourth step, a reader model extracts a candidate answer from each retrieved passage, and the system selects the highest-confidence answer to return to the user.

To evaluate QAssist, the authors constructed the REQuestA dataset using six industrial SRSs from the aerospace, defence, and security domains. The dataset contains 387 question-answer pairs. Half of the questions were written manually by two expert annotators. The remaining half were generated using a text-to-text transfer transformer (T5)[49] based question generation model and then validated, corrected, and filtered by the same annotators.

Multiple retrievers and readers were tested. For document retrieval, BM25[50] was selected as the most efficient retriever with 100% recall. For passage retrieval, the reranking retriever

[49] Text to text transfer transformer (T5)

[50] The Probabilistic Relevance Framework: BM25 and Beyond

performed best with an average recall@3 of 90.1% and 96.5% for SRSs and external (corpus) documents, respectively.

For answer extraction, six reader models were evaluated, including BERT, RoBERTa[51], ALBERT[52], ELECTRA[53], DistilBERT[54], MiniLM[55]. ALBERT was chosen as the best-performing reader, achieving for answer extraction an average accuracy of 24% in the exact matching mode, 79% in the partial matching mode, and 84% in the semantic matching mode while maintaining low response time, with an average runtime per question 10.36 seconds. Exact match means the text is exactly the same, partial match means some words are the same, and semantic match means the meaning of words is similar.

All models were run in Python and Jupyter notebooks using HuggingFace transformers, Wikipedia and Scikit-learn libraries, while the NLTK toolkit was also utilized. The solution was tested on Google Colab for baseline evaluation. For this experiment, 50 questions were submitted to Google Search, which benefits from a massive web index and fast retrieval, but its general-purpose scope and lack of access to the project's SRS limited its performance to 32% correct answers on the test questions, highlighting the value of a domain-specific tool like QAssist.

The paper offers several novel contributions. QAssist provides real-time QA for requirements via the SRS and an automated external knowledge source that stands as a first in RE. REQuestA represents the first publicly available QA dataset designed for natural language requirement analysis. The combination of semantic matching with ALBERT-based answer extraction generated precise results while maintaining acceptable runtime performance. The system operates without any training or fine-tuning processes and depends on the presence of properly structured SRS documentation and domain-specific resources.

QAssist performs well at identifying missing or incomplete knowledge but the functionality to classify or rewrite requirements is not investigated. A drawback of the approach is that the average runtime was ~10 seconds per question. This latency might be acceptable for small batches but could become impractical for large-scale industrial reviews or real-time querying. Also, the comparative evaluation included only a single baseline (Google Search), lacking systematic comparison against other state-of-the-art domain-specific QA tools. Last, the authors do not analyze how performance changes when Wikipedia replaces a true domain corpus, leaving this trade-off unexplored. Future research could evaluate this effect or examine whether LLMs can leverage relevant semantic context to maintain accuracy as an alternative for absent domain documents.

## 4.3 Comparative Analysis

[51] Robustly optimized BERT pre-training approach (RoBERTa)

[52] ALBERT

[53] ELECTRA

[54] DistilBERT

a[55] MiniLM

This section presents a comparative analysis of the selected AI-based approaches in Requirements Analysis and Requirements Verification & Validation. To conduct a thorough and meaningful evaluation, it is essential to establish a clear objective and well-defined comparison criteria. The objective of the evaluation is to identify similarities among the approaches studied, display common AI techniques adopted in the field, discover methodologies with optimal or low performance results and to highlight the main strengths, innovations and weaknesses identified. The criteria chosen address both qualitative and quantitative characteristics derived from the articles considered, providing a structured basis for assessing each approach.

Although the comparison criteria are the same for the two different reviewed RE phases, the comparison will be separately implemented for each phase, due to the differentiation in the nature and purpose of each phase. It should be noted that RA systematically refines, classifies, organizes, prioritizes, and negotiates stakeholder inputs to produce clear and formal requirements. On the other hand, RV&V assess the qualities of requirements by ensuring they are correct, feasible, unambiguous, complete in syntactic point of view (verification) and that they are relevant, justified, traceable and complete in semantic point of view (validation).

### 4.3.1. Comparison Criteria

The first criterion is the *purpose*, selected to indicate the main goal of each approach. This criterion was selected due to the variety of the observed solutions intentions/goals, in order to highlight which RE problems included in the aforementioned two phases have been addressed using AI techniques the most.

The second criterion is *datasets*. The choice to record the datasets used to train and test the performance of the different solutions included in the review, was motivated by the very important role this aspect plays when it comes to evaluating results. This information is essential to ensure fair comparison, because it is fair to compare different approaches trained and tested in the same or similar datasets. Also, this information will give an idea of how often each dataset is used in the literature. Following this, the size of datasets is also reported. This information aims to reflect the extent of empirical testing. Approaches evaluated on multiple datasets or larger size of datasets demonstrate greater experimental robustness and higher confidence in their results and generalizability.

The third criterion maps to *tools*. It is intended to help record information about the tools used to develop each of the considered approaches. This information provides insights into the diversity of the supporting tools and software utilized to implement the approaches, and the commonality of usage in the studied activities.

The fourth criterion, *techniques*, refers to the core AI/ML algorithms and architectures implemented in the considered approaches. It highlights the types of methods leveraged, for example classical machine learning, deep learning architectures, natural language processing, or hybrid approaches. Identifying these elements will provide clarity on methodological innovations and trends in the field.

*Results* is the fifth criterion that provides a complete snapshot of the quantitative information reported by the authors of the different studies in terms of their solutions' performance. This includes information about quantitative metrics, such as precision, recall, F1-score, accuracy or execution time. It should be noted that the metrics are reported as they were originally published in order to avoid any misinterpretations of the authors' findings. The choice to include this aspect is straightforward as, even though a certain level of direct comparison is difficult due to differences in the datasets and the metrics reported, this criterion provides a view on the performance that each solution can be expected to achieve, and will enable to conclude which articles are standing out.

Next, in the sixth criterion, recording information about the *innovative aspects* of each of the surveyed approaches helps to better understand the way in which each of them contributes to the overall advancement of the RE phases assisted by AI. Innovations can be related to the choice of AI methods or architectures, the way they are methodologically combined, or a unique purpose or application scenario.

Finally, *limitations* is the last criterion that concerns the limitations and other potential issues that were identified during the review. Recording information about limitations is important from the perspective of analyzing the relative completeness and reliability of each of the proposed solutions. Acknowledging these limitations will ensure a more comprehensive analysis and point to areas needing further research.

### 4.3.2. Comparison Results of Reviewed Approaches

*Table 1. Comparison Table of RA Approaches*

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| Ronanki et al. 2023 | Binary Classification | Systematic prompt engineering with Zero-shot prompting with GPT-3.5 (5 prompting patterns) for binary FR/NFR classification and traceability. | PROMISE NFR dataset (classification): 621 requirements, PURE dataset (traceability): not reported | Python (for automation), OpenAI GPT-3.5 API (LLM access) | LLM | Performance metrics for the Question Refinement prompting pattern across both RE tasks:<br>Binary Classification:<br>Precision: 80.8% (0.4 temp) – 84.6% (0.0 temp)<br>Recall: 83.7% (1.0 temp) – 87.4% (0.0 temp)<br>F1: 81.0% (1.0 temp) – 85.2% (0.0 temp)<br>Accuracy: ~89% across all settings | Systematic prompt engineering with zero-shot prompting, reusable evaluation framework, tested across multiple temperature settings, no model training required | Only GPT-3.5 Turbo evaluated with no comparison to newer models, no human evaluation of practical output usability, prompt patterns assessed only individually without combinations, limited generalizability to other datasets, tasks, and domains |
| Airlangga et al. 2024 | Binary Class., Multiclass Class., NFR Class. | Semi-supervised GAN-BERT combining GAN with BERT for binary FR/NFR classification and multiclass classification in five NFR subtypes | PROMISE: 625 requirements, WordPress: 939 requirements, Project Documents: 5,292 requirements | Transformers library (Hugging Face), PyTorch, scikit-learn, Adam optimizer | Semi-supervised GAN-BERT | Binary: In-domain best F1-score 77% (Project Docs) and 55% (PROMISE-EXP), cross-dataset up to 58% (PROMISE→PROMISE-EXP), lower in other settings (e.g., 35% F1 in WordPress→PROMISE). Precision/Recall varied: max 76–77%, min 27–50%.<br>Multiclass: In-domain best F1 93% (PROMISE-EXP) and 92% (PROMISE), cross-dataset up to 91% (PROMISE- | Leverages unlabeled data, adapts to multiple domains, improves generalization, reduces need for expert annotation, handles informal texts, thorough multi-domain evaluation, robust in similar-domain scenarios. | Often outperformed by BERT in binary cross-dataset classification, precision/accuracy is not so good in case of informal text, semi-supervised architecture less effective for simpler tasks, training time |

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | | | | | EXP→PROMISE). Lower performance in WordPress-trained scenarios (e.g., 30% F1 in-domain). Precision/Recall ranged from 28–93%. | | and computational costs not evaluated, could use CNN as baseline for completeness |
| Martínez Garcia et al. 2023 | Multiclass Class., NFR Class. | Combine and optimize existing techniques into a single architecture (CNN with word embeddings and vectorization) for NFR classification. | PROMISE (625 requirements) | TF-IDF, Tokenizer, CountVectorizer | CNN | Precision 90%, Recall 88%, F1-score 88% | Systematic evaluation of multiple existing techniques within a single framework, high accuracy/performance | Promise small sample, did not classify functional requirements, generalizability beyond PROMISE dataset not evaluated |
| Hey et al. 2020 | Binary Class., Multiclass Class., F Class., NFR Class. | NoRBERT, a transfer learning approach that fine-tunes BERT models for requirements classification into FR and NFR, as well as into specific functional concerns and NFR subtypes with minimal preprocessing. | PROMISE and Relabeled PROMISE: 625 requirements | Word2Vec, HuggingFace transformers | BERT | Binary class. F/NFR (10-fold): F1 up to 93% (NFR) and 90% (Functional). Multiclass NFR (4 main): per-class F1 up to 91% (Security), weighted avg ~87%. All NFR subclasses: weighted F1 ~82% (10-fold), 76% (p-fold), 73% (loPo). Relabeled dataset class.: F1 up to 91% (Functional aspects), with ~86–87% average in unseen project tests. Functional concern class.: F1 up to 91% (Function), 80% (Behavior), 62% (Data). | Thorough evaluation, generalizes well to unseen projects through project hold out in testing, no manual preprocessing, supports classification in FR & NFR subtypes | Only PROMISE dataset no industrial evaluation, limited comparison to three state-of-the-art baselines, no computational cost analysis |
| Li and Nong 2022 | Multiclass Class., NFR Class. | NFRNet, a deep neural network that classified non-functional | SOFTWARE NFR (PROMISE extension | NLP Toolkit, Regex Library, PyTorch, Python, | BERT + BiLSTM NFRNet | Precision: 91%, Recall: 92.65%, F1-score: 91.47% (SOFTWARE NFR), N-gram masking impact (ablation): +5.38% precision, +6.79% recall, | Introduced a new dataset specifically designed for NFR classification, | Did not classify functional requirements, dataset crafted by |

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | requirements into 32 subcategories as defined in the SOFTWARE NFR dataset, built by authors by extending PROMISE with Wikipedia derived definitions | with Wikipedia): 6222 requirements | Softmax, Lookahead optimizer and early stopping | | Robustness: lowest st. deviation σ = 4.82 and highest accuracy in many NFR type | comparison with 17 baseline models, effective use of N-gram masking and multi-sample dropout leading to good generalization | Wikipedia text and annotated by SE students, generalizability not tested |
| Kaur et al. 2023 | Binary Class., Multiclass Class., NFR Class. | DL model combining BERT embeddings, Bidirectional LSTM, and CNN layers to perform binary classification and non-functional requirements classification, without manual preprocessing | PROMISE (625 requirements) | TensorFlow Keras HuggingFace Transformers Adam Optimizer | BERT, BiLSTM, CNN | F1-scores were 95% (binary functional vs. NFR), 89% (binary 4 NFRs), 94% (multiclass 4 NFRs), 86% (multiclass 5 classes including "Others"), and 79% (multiclass 10 NFRs), improvements over baselines were statistically significant ($p<0.01$) | Innovative multilayered architecture, no preprocessing needed, enable raw requirements input | only PROMISE dataset - no industrial evaluation, small/imbalanced data, no FR multiclass considered |
| Rahimi et al. 2021 | Binary Class., Multiclass Class., F Class., NFR Class. | Classification into FR and NFR and both subtypes (6 FR subtypes + 11 NFR subtypes) through sequential binary multi-class pipeline. | PROMISE: 625 requirements | Python 3.6, PyCharm, Scikit-learn, Keras, TensorFlow Adam | DL Ensemble Two-phase ensemble | One-phase: Accuracy=92.56%, Precision=93%, Recall=93%, F1=93%, Two-phase: Accuracy=93.40%, Precision=94%, Recall=93%, F1=93%, FR subcategories up to 98% accuracy/F1, NFR subcategories: Accuracy=86.5%, F1=87% | Combining the binary and multiclass classification in one pipeline, Ensemble of four DML models | Single train/validation/test split (no cross-validation), not standardized FR NFR subtypes, no cross domain evaluation, comparison with other work limited due to inconsistent |

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | | | | | | | evaluation setups, |
| AlDhafer et al. 2022 | Binary Class., Multiclass Class., F Class., NFR Class., Multilabel | BiGRU-based RNN to classify software requirements into FR, NFR, their subtypes, and multilabel classification, without extensive preprocessing. | PROMISE: 625 requirements, PROMISE-ML: 792 sentences, EHR: 6904 sentences | Python, Keras, Scikit-learn, Adam Optimizer | BiGRU | Binary classification: Word F1-score = 94%, Precision = 93%, Recall = 95%, Char F1-score = 88%<br>Multiclass (4-class): Word F1-score = 87%, Char F1-score = 70%<br>Multiclass (5-class): Word F1-score = 77%, Char F1-score = 63%<br>Multiclass (10-class): Word F1-score = 75%, Char F1-score = 54%<br>Multilabel (PROMISE ML): Word Weighted F1-score = 69%, Macro F1-score = 44%, Char Weighted F1-score = 52%, Macro F1-score = 31%, Hamming Loss = 0.05–0.07<br>Multilabel (EHR binary): Word F1-score = 89%, Char F1-score = 86%, Hamming Loss = ~0.13–0.17<br>Multilabel (EHR multiclass): Word F1-score = 81%, Char F1-score = 75%<br>Stability: Word model ±2.2%, Char model ±4.1% (binary classification), similar variance in multiclass and multilabel tasks | Functional Subtypes included, minimal preprocessing, multilabel classification | Low performance in rare classes, no generalizability, no computational cost evaluation, no experimental comparison with state-of-the-art |
| Sainani et al. 2020 | Binary Class., Multilabel | BiLSTM for requirements (obligation) extraction and BERT for multilabel | Expired SE contracts (9 domains): 18,614 labeled | Word2Vec embeddings, TF-IDF vectorization, | BERT, BiLSTM, SVM, RF, Naïve Bayes | Binary classification (BiLSTM):<br>Obligations – Precision 91%, Recall 94%, F1 93%,<br>Non-obligations – Precision 89%, Recall | Novel use of contracts as requirements source and analysis, real-world validation. | No cross-industry testing, no functional requirements classification, no |

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | classification into 14 architectural and governance requirement types | sentences from 20 contracts | Python, Scikit-learn, Keras, TensorFlow, Adam Optimizer | | 84%, F1 86%.<br>Multilabel classification (BERT):<br>Per-class F1 up to 96% (e.g., Export Laws Precision 100%, Recall 92%, F1 96%).<br>Validation contract overall F1score 85.8%, processes 1608 sentences in 3 min. | | baseline evaluation |
| Verdejo et al. 2021 | Multiclass Class., NFR Class. | Classify NF requirements into seven ISO 25010 quality attributes, train model on structured requirements and test it on unstructured text from Github comments. | PROMISE: 254 requirements, PURE: not reported, Miller(40 requirements), GitHub(500 issue reports) | TPOT<br>Stanford CoreNLP<br>Doccano<br>scikit-learn<br>SciPy<br>UMAP method | ML algorithms Naive Bayes, SVM, Random Forest | On structured requirements, TPOT achieved G-Mean 83.6%. On GitHub issues, Naïve Bayes achieved highest F1 (28.6%), while TPOT had highest precision (44.1%) but lower F1 (24.1%). Performance dropped significantly on informal text. | No need for formal documentation, enables requirements analysis before requirements evolve into formal specifications, supports partial automation of certain RE tasks. | F1 <30% on real issues, poor generalization to informal text, not usable for new systems without user feedback enabled, did not classify functional requirements |
| Hujainah et al. 2021 | Prioritisation | A semi-automated prioritization model StakeQP, that combines stakeholder weighting, influence modeling, clustering, and ranking to prioritize software requirements. They authors created the RALIC semi-simulated benchmark real requirements | RALIC: 122 requirements | C#, Excel | K-means++ | Accuracy values ranging from 93.0% to 94.65% among experiments, execution time 5.02–5.52s | Supports large-scale projects, combines stakeholder influence factors (role, power, interest, experience), semi-automated with low expert involvement, Unified framework combining WSM, clustering & BST sorting, outperforms multiple baseline techniques, | Relies on accurate stakeholder input, effort needed to prepare and maintain input data, treats requirements as independent - no modeling of dependencies, only tested on one dataset restricts generalization evaluation |

| **Requirements Analysis Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RA Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | dataset from industrial projects with synthetic stakeholder input. | | | | | reduces bias | |
| Gupta et al. 2022 | Prioritisation | Semi-automated requirements prioritization method that combines stakeholder ratings, using PSO to generate consensus-based ranked lists of requirements | CBMW: 53 requirements, 9 synthetic datasets | spaCy | PSO | Best result: 4 disagreements (vs. AHP = 7, IGA = 5), Robustness: disagreement remained stable under 5–20% noise (13–15),<br>Scalability: converges in up to 141 iterations (100 requirements), statistically outperforms AHP ($p < 0.05$), comparable to IGA ($p > 0.05$) | Combines business and technical priorities, incorporates execute-before-after dependencies, resolves conflicting priorities, | Needs accurate dependency input, scalability only tested synthetically, evaluation used only disagreement metric, no precision, recall, or F1score reported, only one technique assessed |

*Table 2. Comparison Table of RVV Approaches*

| **Requirements Verification & Validation Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RVV Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| Ezzini et al. 2022 | Anaphoric Ambiguities | Detect and resolve anaphoric ambiguity in requirements, proposing in a hybrid solution, created the DAMIR dataset | DAMIR: 1,251 industrial requirements, ReqEval: 98 requirements | SpaCy, NLTK, Stanza, CoreNLP, Scikit-learn, Hugging Face Transformers, PyTorch, Jupyter Notebooks | SpanBERT + ML ensemble | Best anaphoric ambiguity detection with supervised ML classification :<br>MLensemble (~60% precision, 100% recall, F1-score 75%), 14.5 sec per pronoun<br><br>Best anaphora resolution: SpanBERTRE 98% success rate and 1.5 sec per pronoun | Systematically develops and compares six solutions, combines supervised ML, pre-trained language models, and NLP tools, creates a large industrial dataset (DAMIR) | Precision is moderate (~60%) causing false positives, high execution times requiring optimization, DAMIR dataset is author-created and not widely adopted, ReqEval dataset is small, limited generalizability, no comparison with newer LLMs or diverse ML methods |
| Osama et al. 2020 | Attachment, Coordination, Analytical Ambiguities | Score-based tool to detect and resolve syntactic ambiguities in requirements, generating ranked parse interpretations to help users clarify ambiguous sentences. | Ambigo: 100 sentences,<br>Curated dataset: 26 sentences | Stanford CoreNLP | Rule—based NLP (SBADR) | Experiment 1 – Comparison with AmbiGO<br>Sentence-level results:<br>Average: Precision 65%, Recall 99%, $F_1$-score = 77%<br>Interpretation-level results:<br>Average: Precision 62%, Recall 96%, $F_1$ = 74%<br>Experiment 2 – Complex Scenarios<br>Sentence-level results:<br>Coordination ambiguity: P = 100%, R = 100%, $F_1$ = 100%<br>Attachment ambiguity: P = 100%, | Detects and resolves multiple types of syntactic ambiguity, does not rely on static templates, outperforms prior tools in recall, produces possible interpretations to aid resolution | Precision technically limited by Stanford CoreNLP parser scoring, no semantic-level analysis causing false positives, performance may degrade on longer or domain-specific texts |

| **Requirements Verification & Validation Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RVV Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | | | | | R = 90%, $F_1$ = 94%<br>Interpretation-level results:<br>Coordination (P = 82%, R = 95%, $F_1$ = 88%) and Attachment (P = 81%, R = 93%, $F_1$ = 87%).<br>P=100%, R=90%, F1=94% | | |
| Lubos et al. 2024 | Ambiguous, Inconsistent, Incomplete , Inappropriate , Non conforming, Incorrect, Infeasible, Unnecessary, Non Singular, Unverifiable (Requirements Detection & Resolution) | Evaluated Llama 2's ability to assess requirements against ISO 29148 quality criteria, explain its assessments, and suggest improved rewrites, comparing its output to human judgments on two datasets | Pure dataset subset: Stopwatch (10 requirements), Digital Home (63 requirements) | Llama 2 | LLM | Evaluation Results (Independent Phase):<br>Stopwatch: κ = 0.40, Precision = 50%, Recall = 89%<br>Digital Home: κ = 0.05, Precision = 12.6%, Recall = 74%<br>Evaluation Results (Bound Phase):<br>Stopwatch: κ = 0.75, Precision = 88%, Recall = 100%<br>Digital Home: κ = 0.22, Precision = 32%, Recall = 100% | First study on use of LLMs and ISO standards, identifies multiple types of quality issues and suggests improvements, reproducible prompt setup, easy to use without deep expertise | Only one LLM and prompt structure tested, datasets limited to two small projects, low precision indicates frequent false positives, not compared to related work |
| Mahbub et al. 2024 | Ambiguous, Inconsistent, Incomplete (Requirements Detection) | Zero-shot prompting with GPT-4 on industrial SRS to detects ambiguities, inconsistencies, incompleteness | Mechanical Lung Ventilator (MLV) case study SRS: 52 pages of | GPT-4 | LLM | Ambiguity: Precision 17–60% (avg 39%), Inconsistency: 33–57% (avg 43%), Completeness: 50–92% (avg 61%), Diagrams: 44% precision, Version recall: 45%, Formal modeling recall: 60%, QA success: 77% (sectional), 62% (global), | Evaluates GPT-4 on real industrial SRS, demonstrates practical zero-shot prompting without fine-tuning, combines qualitative and quantitative analysis, assesses multiple | No comparison with other LLMs, only one prompt style tested, no suggestions to limit the content so as to fit the LLM's context window size |

| **Requirements Verification & Validation Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RVV Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| | | issues | requirements | | | limitations in cross-referencing. | defect types including ambiguity, inconsistency, and incompleteness, includes version history and question-answer evaluations | |
| Luitel et al. 2024 | Incomplete Requirements (Detection & Resolution) | Proposed a BERT-based approach for identifying potentially missing content in natural language requirements by simulating incompleteness through word masking and generating word suggestions, with post-filtering via traditional ML classifiers. | PURE subset (24000 requirements) | Hugging Face, Transformers library, PyTorch, GloVe (word embeddings), Spacy, WikiDoMiner | BERT | Top-1 suggestion accuracy: 12.1% (50–50 split), 2.25% (90–10 split), improved to ~25.1% and ~4% with lenient filtering. Coverage: ~40% (50–50 split), ~33% (90–10 split) | Domain-independent approach using pretrained BERT and Wikipedia | Evaluation based on simulated incompleteness not real requirements, lacks unified performance measure for overall effectiveness, accuracy remains low, does not consider sentence level contextual semantics |

| **Requirements Verification & Validation Approaches Comparison Table** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Year** | **RVV Category** | **Purpose** | **Datasets** | **Tools** | **Techniques** | **Results** | **Innovation** | **Limitations** |
| Ezzini et al. 2023 | Clarifications on Requirements | AI-based question answering system combining retrieval and reading comprehension to help in requirements analysis, introduced REQuestA, the first large-scale QA dataset for requirements engineering | REQuestA semi synthetic dataset: 387 QA pairs | Python and Jupyter notebooks using HuggingFace transformers, Wikipedia and Scikit-learn libraries, NLTK toolkit | ALBERT | BM25 doc retriever (finds doc, Recall@1=100%),<br>rerank passage retriever (finds top passages, Recall@3=90.1% SRS /96.5% corpus),<br>ALBERT reader (extracts answer, semantic acc=84.2%, exact=24%, partial=79%),<br>avg runtime 10.36s/q , Google Search baseline accuracy: 32% | First publicly available RE QA dataset, achieves high semantic accuracy, no fine-tuning required, supports real-time analyst queries | No rewriting support, long exec. times |
| Ronanki et al. 2023 | Traceability | Systematic prompt engineering with Zero-shot prompting with GPT-3.5 (5 prompting patterns) for FR, NFR classification and traceability. | PROMISE NFR dataset (classification): 621 requirements, PURE dataset (traceability): not reported | Python, OpenAI GPT-3.5 API (LLM access) with Zero-shot prompting | LLM | Traceability:<br>Precision: 43.2% (0.0 temp) – 45.4% (0.4 temp)<br>Recall: 60.7% (1.0 temp) – 64.0% (0.4 temp)<br>F1: 51.0% (0.0 temp) – 53.2% (0.4 temp)<br>Accuracy: ~91% across all settings<br>Lowest standard deviation observed in all metrics. | Systematic prompt engineering with zero-shot prompting, sophisticated evaluation framework that enables testing across multiple temperature settings, no model training required - good binary class results | Only GPT-3.5 Turbo evaluated with no comparison to newer models, no human evaluation of practical output usability, prompt patterns assessed only individually without combinations, limited generalizability to other datasets, tasks, and domains |

### 4.3.3. Comparison of Requirements Analysis Approaches

The following section compares the reviewed approaches in RA across the selected criteria including purpose, datasets, tools, techniques, results, innovations and limitations and aims to provide insights of common patterns and differences among them.

#### 4.3.3.1 Requirements Analysis Category & Purpose

The majority of the RA reviewed approaches focused on the automated classification of requirements which accounted for 10 out of 12 studies (83,3%), while 2 studies (16,7%) focused on requirements prioritisation. Specifically, the main tasks of the approaches were binary requirements classification (58%) and multiclass classification found in 67% of approaches. Out of the approaches that assessed multiclass classification, all of them handled the non-functional subtypes classification and 3 out of 8 involved the functional subtypes classification task. Last, a few approaches included the multilabel classification task (2 of the total 8 handling classification.

In terms of task coverage, there were studies that considered multiple types of classification. [77] addressed the most classification tasks: binary, multiclass and multilabel classification. In parallel, they considered both functional and non-functional requirements for classification. [63] addressed the binary and multilabel classification although the classes chosen are not standardized. [75], [71,72,74] covered binary and multiclass classification. Authors of [73], [58]and [35] limited their focus to NFR subcategories classification and [53] and [29] to prioritisation task. Notably, a single effort to combine a broader range of activities that belong both in analysis and verification of requirements was observed by [21], which focused on the application of zero-shot prompting technique to support binary classification and traceability.

However, it should be noted that other significant requirements analysis activities such as conflicting requirements identification and resolution, measures of performance identification, or in general human in the loop architectures in which analysts could interact with the model, appear to remain under or uninvestigated.

#### 4.3.3.2 Datasets

A combination of curated benchmark datasets, semi-synthetic, and curated real project datasets(the data originated from actual projects or repositories and were annotated by experts for research use purposes) were employed to train and evaluate the approaches, although the overall scale and diversity remained limited. The most frequently used dataset is the PROMISE and its extensions, such as PROMISE-EXP and SOFTWARE NFR, which provide structured examples to support classification of functional and non-functional requirements. In fact, 9 out of 12 studies made use of PROMISE or one of its versions (75%). These resources were adopted in multiple studies due to their consistent annotation and widespread acceptance as reference benchmarks [21,58,71,74], [75] [73] , [77], [72], [63]). This reliance on PROMISE has an advantage, it enables comparability across the reviewed studies but also introduces a limitation. However, the generalizability of results may be compromised due to the dependence on a single and potentially outdated dataset, which covers some but not all possible domains.

Beyond PROMISE, the second mostly used dataset was PURE appearing in 2 out of 12 papers. Other approaches introduced curated real-world or semi-synthetic datasets with one occurrence per article. Notably, one work relied on expired software engineering contracts [63], others on real-world case-study datasets like EHR [77] or CBMW [53], or RALIC, a dataset based on industrial requirements [29]. Further, one study on NFR quality attributes relied on first training on structured requirements and then testing generalization using unstructured GitHub issue comments [58], thus demonstrating an effort to bridge formal and informal requirements and simulate practical conditions. These alternatives indicate an effort to differentiate on training corpora. These datasets in some cases was mentioned that they contained more heterogeneous and informal text, aiming to reflect real-world requirements with broader variation in phrasing and quality.

Across the 12 reviewed RA studies, 18 dataset instances were identified (several studies utilized more than one dataset). Benchmarks make up the majority (56%), including PROMISE and its variants (PROMISE, PROMISE NFR, PROMISE-ML, relabeled PROMISE, SOFTWARE NFR). Curated real-world datasets follow with 39%, including WordPress, project documents, EHR, GitHub issues, RALIC, SE contracts, and CBMW datasets. Semi-synthetic datasets are rare (6%), with only the synthetic sets used in [53]. The Miller dataset was not included as no details on the content were found.

The quantity of data used across the approaches varied significantly. Most studies used datasets containing fewer than 1000 requirements, with commonly used benchmark datasets, such as PROMISE, containing around 600 to 700 entries. Some approaches expanded their evaluation by utilizing multiple datasets by incorporating externally sourced data. A notable increase in data volume was achieved in those studies that aggregated project documentations, such as requirements collected from software projects [77] and other external sources like Wordpress [71], Wikipedia used in [73] or contracts [63], leading to a bigger dataset size.

Furthermore, in several cases it was reported that the dataset was imbalanced, complicating the interpretation of performance. These observations highlight the need for larger, more balanced datasets to support robust and generalizable evaluation.

#### 4.3.3.3 Tools

The tools utilized across the RA studies reflects a broad variety of deep learning, NLP, and Python technologies.

Regarding development infrastructure, Python appeared as the foundational programming language used to build almost all the models proposed (in the 10 out of 12 studies). Among DL frameworks, the most frequently used were Keras, a deep learning library used for designing and training neural networks, and TensorFlow open source platform, appearing in four studies. It is worth noting that Keras is nowadays used as a high-level API on top of TensorFlow, which provides the backend functionality, so both usually appear together. PyTorch, an open-source deep learning framework, was also adopted in three studies and stands as a competitive alternative to TensorFlow in modern deep learning practice.

Regarding the core AI models and algorithms, LLMs such as GPT-3.5 were utilized for prompt-based tasks. Word embeddings were mainly represented by BERT based embeddings for most DL architectures, while Word2Vec was the secondly often used model for this task.

Vectorizers like TF-IDF (in two studies) and CountVectorizer (in one study) were also applied for feature extraction. Common optimizers included Adam, which appeared in half of the reviewed studies, making it the most used tool in the RA studied approaches, while Lookahead and multi-sample dropout were used only once.

For processing pipelines and supporting libraries, the most frequently mentioned tool was scikit-learn, appearing in five studies, supporting dataset splitting and feature extraction. The Hugging Face Transformers library followed. Other NLP tools, including spaCy and Stanford CoreNLP, were used less, while SciPy, TPOT, and UMAP were each reported only once.

Finally, domain and task-specific tools were also utilized. WikiDoMiner was used for domain knowledge extraction, Regex libraries for rule-based text matching, and Doccano for annotation and dataset creation, supporting the preparation of curated corpora for model training and evaluation.

Considering the number and diversity of tools used, [71], [74], [75], and [63] stand out. These studies integrate the most frameworks, embeddings, optimizers, and supporting libraries. In contrast other approaches, such as [53], and [35], rely on a few tools utilized in their approach.

#### 4.3.3.4 Techniques

Regarding the technique's criterion, the analysis of AI and ML applied in the articles reveals a dominance of deep learning architectures, particularly those based on BERT. BERT-based models were mostly used across the reviewed approaches, a trend that can be attributed to their consistently strong performance in NLP tasks and their transfer learning capability. BERT-based models are frequently combined with additional layers, such as BiLSTM and CNN, and GAN, forming hybrid architectures that aim to capture contextual semantics and local features in requirements. Notably, [74] followed a hybrid transformer-based architecture with BERT in combination with BiLSTM and CNN. The authors of [71] proposed a technically complex architecture employing BERT with GAN in a semi-supervised architecture and [72] used a transfer learning BERT approach specialized for requirements datasets.

Besides BERT, a variety of DL models were explored. Some studies used RNN models like BiLSTM and BiGRU ([77]) to capture the meaning of longer requirement sentences, while CNN was employed as a standalone model as well ([75]).

Classic machine learning approaches were also seen but in fewer cases. Some include RF (Random Forest), SVM (Support Vector Machine), Naive Bayes, k-means clustering, and others. An optimization technique, PSO (Particle Swarm Optimization) [53], was used in a requirements prioritization model that combined user rankings for obtaining a consensus-based ranking of requirements.

Zero-shot prompting with GPT-3.5 was proposed in one paper [21]. This may be considered as a bridge between traditional model training and starting to rely on the LLM's capabilities to perform a task with no fine-tuning. In general, the field has moved towards transformer-based models, with hybrid deep learning being a major focus. There is potential to investigate further in LLMs.

#### 4.3.3.5 Results

The reported results across the papers vary considerably in terms of metrics, presentation formats, and performance levels, but certain patterns can still be identified. Most commonly, performance is evaluated using precision, recall, and F1-score. Accuracy is also occasionally reported, though less frequently.

In general, reported F1-scores for binary classification (e.g., FR vs. NFR) are high, often ranging from 88% to 96%. These values reflect a strong performance trend; however, these individual results are difficult to compare systematically, largely due to the absence of a common reporting practice across studies. Instead of following a unified experimental methodology, most studies report performance based on multiple task-specific experiments, often reporting multiple F1-scores or other metrics across experiments. While these results are valuable and informative individually, this practice makes it difficult to evaluate and compare models, as the absence of an overall model performance metric (more advanced than F1) limits direct comparison across the studies. The lack of standardized experimental setup, for example a consistent use of 10-fold or k-fold cross-validation and statistical testing across studies, further complicates the comparison. The introduction of a composite evaluation metric that aggregates results across the experiments could be a step forward, but it would require a consistent methodology to ensure a meaningful and fair comparison. Additionally, there is a lack of standardized large-scale datasets that include multiple domains, which would enable the assessment of a models generalizability. A few studies adopted cross-dataset or cross-domain validation, a very important factor to verify the performance of approaches beyond the dataset they were trained.

Finally, an important observation is that in most of the approaches evaluations, execution times reporting was omitted. Runtime performance is rarely measured or discussed, leaving an important gap in the assessment of practical applicability, especially for industrial applications where the real time execution and evaluation of requirements would be necessary.

Although as mentioned earlier the approaches results cannot be directly compared to each other due to differences in evaluation methodology, however from individual results point of view, some approaches stand out as they achieved consistently high performance.

Kaur et al. in [74] achieved the highest results with 95% F1-score in binary classification and 79–94% F1-score across multiclass NFR subtypes utilizing the PROMISE dataset, showing statistically significant improvements over baselines. [75] reached 93% F1-score in binary classification (PROMISE) and 87–98% F1-score in multiclass NFR classification (PROMISE), demonstrating very strong and consistent multiclass performance. Approach [73] reported robust performance in NFR subtypes classification, reaching 91.47% F1-score across 32 NFR subcategories (SOFTWARE NFR: PROMISE and Wikipedia). Approach [72] reached 90–94% F1-score for binary classification and 82–87% F1-score for multiclass NFR subtypes with the PROMISE, 91% F1-score on the relabeled PROMISE for quality aspects multiclass classification, and 62-92% F1-score for functional concerns multiclass classification with PROMISE. Authors in [77] conducted extensive experiments across binary, multiclass, and multilabel tasks, achieving 94% F1-score in binary classification (PROMISE), 75–87% F1-score in multiclass classification (PROMISE), and 52–89% F1-score in multilabel tasks (PROMISE-ML and EHR dataset respectively). The approach [63] demonstrated strong performance in multilabel contract requirements, reaching up to 96% F1-score and 85.8% overall F1-score (Expired software contracts).

On the other hand, some approaches involved with classification tasks achieved more modest results. Such as [71] which achieved 55–77% F1-score in binary classification and 91–93% F1-score in multiclass NFR classification (PROMISE, PROMISE-EXP, Project Docs), but cross-dataset performance dropped to 30–58%. Approach [35] achieved 88% F1-score in binary NFR classification (PROMISE). Approach [21] reached 81–85% F1-score in binary classification and was the single approach handling both RA and the traceability that belongs to RVV phase. [58]reported 28–44% F1-score on informal text and achieved 83.6% G-Mean on structured requirements (PROMISE, Miller, GitHub, PURE).

For prioritization tasks, [29] and [53] focused on semi-automated stakeholder-driven prioritization. Hujainah et al. reported a high 93–94.65% accuracy (RALIC) with execution time approximately 5 seconds. Gupta et al. achieved robust convergence with minimal disagreements, reaching a best case of 4 versus 7 for AHP (CBMW, synthetic datasets), demonstrating efficiency and reliability under noise and varying requirements.

#### 4.3.3.6 Innovations

The innovations presented across the RA studies span a variety of directions, including architectural novelties, task scope expansion, practicality improvement and minimizing dependency on labelled data. Several studies proposed combining deep learning architectures, notably those combining BERT with CNN or BiLSTM layers, as a way to boost classification performance and handle multiple requirement categories simultaneously. Others provided innovations in training, such as the GAN-BERT approach, which incorporated semi-supervised learning to make use of labeled and unlabeled data and to improve adaptability across datasets and domains. Another one was the systematic prompt engineering framework for GPT-based LLMs, evaluating different prompting strategies for requirements classification and traceability without any training. Transfer learning also appeared in some studies ([72], 2020, [74], allowing models to reuse pretrained embeddings. In the area of the requirements prioritization activity, an innovation came through the integration of stakeholder input with optimization algorithms, such as PSO, so as to automate decision-making. Other papers contributed by evaluating realistic document structures, supporting non-formalized inputs (e.g., GitHub issues), or enabling multi-stage pipelines that separate binary and multiclass tasks for better interpretability.

Considering innovation overall, [21] appears as the most innovative proposed approach by introducing a systematic prompt-engineering framework for zero-shot RA (binary FR/NFR) and RVV (traceability) within a single pipeline the only study spanning RA and RVV without training, shifting RE towards LLM-enabled workflows. There are several other approaches that contribute to important innovation as well. Article [71] by proposing the GAN-BERT methodological innovation to leverage unlabeled data and multiple datasets can be very useful in RE settings with scarce label settings. Additionally [74] demonstrated a no-preprocessing, raw-text BERT+BiLSTM+CNN hybrid approach, which is very innovative considering practical applicability and operational simplicity.

A few studies introduced more task-specific innovations, proposing some less common aspects of RA. [63] explored an alternative perspective of the data source of requirements, through software engineering contracts as a novel form of requirements documentation. Through this work it was demonstrated that legal artifacts can be exploited as alternative structured sources for

requirement extraction. Similarly, [58]explored the task of training models on structured requirements and testing them on unstructured issue reports from GitHub. This approach provided early evidence of how formality mismatches between datasets can degrade performance but also displayed a path to models capable of handling both formal and informal requirement expressions.

In contrast, a few approaches showed relatively limited innovation to the broader state of the art. [35] implemented a CNN pipeline with rather classical techniques such as TF-IDF and CountVectorizer. While the model was evaluated systematically and achieved good baseline performance, its contribution remains limited by relying on well-established deep learning methods without introducing novel mechanisms compared to others. Approach [75] presented a two-phase ensemble architecture that, despite its technical layering, follows a conventional supervised learning pipeline compared to the transformer-based, semi-supervised, or LLM-driven architectures that characterizes more recent research.

#### 4.3.3.7 Limitations

Regarding the last criterion for the the reviewed papers in RA, it is observed that the approaches present several recurring limitations, particularly in terms of generalizability, dataset constraints, evaluation practices, and model robustness. One of the most frequently observed limitation is the reliance on a single dataset, especially the PROMISE dataset, which constrains both the size and diversity of the evaluation corpus. This reliance hinders the ability to test models in cross-domain conditions and raises concerns about the applicability of findings to industrial environments.

Another common limitation, related to the previous one, is the lack of generalization testing in many studies. Several papers reported high performance on internal splits using the same dataset as in training, but did not include cross-validation or testing across different domains, making it difficult to figure whether the models would maintain their performance in new or different domain requirements.

Additionally, execution times and scalability were largely omitted or underreported, leaving an important gap in assessing the feasibility of deployment in practical settings. Finally, some limitations are tied to task scope or coverage. For instance, some classification models excluded functional requirements entirely by implementing solely NFR classification. In the LLM-based study, only one LLM, namely GPT-3.5 Turbo, was tested, and no comparisons with other LLMs or prompt strategies were made, limiting the broader applicability of findings.

Several approaches display limitations, that restrict their generalizability and methodological robustness. In [75] approach, despite reporting high accuracy and F1-scores, its evaluation depends on a single static dataset split (35/35/30), without any cross-validation or cross-domain evaluation. Additionally, [58] showed performance drop in the experiments with informal text. This highlights the model's lack of robustness in real environments.

The approach [21] with zero-shot prompting approach, evaluated only a single LLM (GPT-3.5 Turbo) and one set of prompt configurations which resulted in moderate results, but did not compare against baselines. Similar limitations faced by [71] although their approach adopted a complex architecture, it showed a performance decline in cross-dataset evaluation, while omitted runtime and computational-cost assessment. Finally, [73] showed some potential dataset-validity issues, since its SOFTWARE NFR corpus was constructed using Wikipedia text and annotated by

students, which may compromise external validity despite strong quantitative results. These examples show that many limitations stem less from algorithmic design than from evaluation practice, dataset utilization, and empirical breadth.

#### 4.3.3.8 Overall Comparison of Requirements Analysis Approaches

The following comparison aims to highlight which approaches demonstrate the highest and lowest overall balance and maturity, considering the selected cr iteria previously described and analyzed.

Across the selected criteria, the approaches proposed by [74] and [72] appeared as the most complete and balanced across all criteria, with [74] exceling in innovation and [72] in methodological depth.

Approach in [74] successfully fulfilled its purpose by addressing both binary and NFR multiclass classification in a single framework. Although it relied only on the PROMISE dataset, a limitation acknowledged by the authors, this choice allowed comparability with other studies and ensured a consistent evaluation basis. The authors utilized a modern DL pipeline built on TensorFlow, Keras, and the Hugging Face Transformers library, integrating BERT embeddings through a hybrid architecture that combines CNN and BiLSTM layers. The achieved results were among the strongest reported, reaching up to 95% F1-score for binary classification (the highest among the reviewed approaches for binary classification in RA) and 79–94% F1-score for NFR subtype classification, with statistically significant improvements over the five baselines that the model was evaluated against. Furthermore, the model's ability to operate on raw requirements for input without preprocessing represents one of the most important innovations seen that can lead to practical application and automation. Although specific execution times were not discussed, this approach overall demonstrated consistent strength across all criteria assessed in the current comparison analysis.

In [72] proposing the NoRBERT transfer learning methodology, successfully supported both binary and multiclass classification, covering both user defined functional and NFR subtypes classification. The approach was built and evaluated on the PROMISE and relabeled PROMISE datasets, which, although limited, allowed reliable comparison and reproducibility across studies. The use of a domain-adaptive transformer model, NoRBERT, built through the Hugging Face Transformers library and Python environment, demonstrated an effective utilization of tools and techniques, as the architecture fine-tunes BERT embeddings specifically for the requirements engineering context. The results achieved high performance (F1-scores up to 93% for binary classification, 82%-87% for NFR subtypes, and 86%-91% for functional subtypes classification), while the project hold-out evaluation indicated strong generalization to unseen projects, an uncommon methodological strength. From an innovation perspective, the study introduced domain adaptation in BERT training, advancing transfer learning for RE. Although execution times and computational costs were not discussed in neither of these studies, the works' rigorous methodology, high results, and balance across all criteria make them the most credible RA approaches.

Approach [73] ranks next due to its strong methodological design, tool diversity, consistent results and evaluation. The approaches' purpose focused on the detailed categorization of non-functional requirements, addressing thirty-two NFR subtypes. The study introduced the

SOFTWARE NFR dataset, extending the PROMISE corpus with Wikipedia examples and NFRNet, a BERT–BiLSTM model implemented with a wide variety of tools (PyTorch, NLTK, Regex, Lookahead optimizer. The model achieved 91.47 % F1, surpassing 17 baselines, including NoRBERT, and demonstrating statistically validated robustness. Although its task coverage is limited compared to the top performers, the dataset used is semi-synthetic and lacks cross-domain validation, the study combines methodological innovation, reproducibility, and high performance.

Following next, it is worth mentioning [77] that stood out as one of the most comprehensive and empirically diverse approaches, achieving strong results while addressing several limitations observed in earlier works. In terms of purpose, the study covered a broad task range than most others by handling binary, multiclass, and multilabel classification and by incorporating both FR and NFR categories. Its dataset strategy demonstrated an attempt to move beyond benchmark dependence, combining the PROMISE, PROMISE-ML, and EHR datasets. Regarding tools and techniques, the approach was implemented using Python, Keras, and scikit-learn, employing BiGRU architectures at both the word and character levels to capture semantic and morphological variation in the requirements. In terms of results, the model achieved descent F1-scores, for binary classification 94% (word level embeddings) and 88% (character level embeddings), while multiclass results ranged from 54–87 %, and multilabel tasks from 31–89 %, revealing a variation across data types and rare labels. The study's innovation lied in its broad task coverage with minimal preprocessing. Although some decline in performance was noted and no runtime measurements were reported, the paper demonstrated methodological transparency and credible multi-dataset evaluation.

Among the approaches focused on prioritization tasks, [29] stood out as the most complete and balanced across all criteria. The study used the RALIC industrial dataset, ensuring realistic data and dataset quality, while it leveraged commonly known tools, providing a high degree of usability and integration potential. The applied technique, based on the Weighted Sum Model (WSM), quantifying stakeholder importance and computing ranked priorities transparently with high 93–94.65% accuracy and short execution times (five seconds per run). Its innovation lies in transforming prioritization into an automated, tool-supported decision framework validated in a real industrial context. The limitations found involving necessity of human input and limited dataset utilization do not undermine its practical completeness.

Although some of the remaining approaches contribute valuable ideas and technical experimentation, they lack on completeness by falling short on certain criteria, therefore they could not be considered as most balanced approaches.

Authors of [71] introduced a methodological novelty through semi-supervised learning with GAN-BERT, which aimed to overcome dataset limitations by enabling unlabelled data usage across multiple dataset sources and a variety of tools. However, this approach was outperformed by other BERT-based models with simpler architectures in several scenarios, while the achieved results appeared inconsistent (F1-scores from 30% to 93%). This suggests that more complex architectures do not necessarily achieve better results and must be validated rigorously.

The work by [21] explored the innovative zero-shot prompting with GPT-3.5 and brought valuable insights into how prompting could be used to support classification and traceability without model training. This was a unique approach linking RA and RVV tasks, without model training. However, multiple and important limitations were found, including a limited evaluation

to a single model variant and a single prompt pattern, benchmarking absence against alternative methods or baselines and weak results especially for traceability task.

Finally, across all criteria, the weakest-performing approaches are [58] and [53]. The first [58], explored the combination of structured and informal requirements by training on formal datasets (PROMISE, Miller) and testing on informal ones (GitHub issues) to perform classification. While this purpose is conceptually valuable and innovative, there are certain limitations and drawbacks that degrade the outcomes of this approach. The datasets utilized were small and imbalanced. The study relied on traditional ML tools (TPOT, scikit-learn, CoreNLP) and basic classifiers (Naïve Bayes, SVM, Random Forest) without leveraging embeddings or deep architectures, which although acceptable, other approaches surpass this one on these criteria. The results were very poor with F1-score below 30% on informal text, while showing limited generalization. Despite innovation in cross-formality testing, the aforementioned limitations combined make it one of the least effective approaches.

Similarly, [53] introduced a conceptual variety by applying PSO for requirements prioritization. However, no tool variety or diversity were identified and most importantly no quantitative evaluation metrics such as accuracy or F1-score were used, offering only qualitative comparison with expert rankings.

### 4.3.4. Comparison of Requirements Validation & Verification Approaches

The following section compares the reviewed approaches in RVV across the selected criteria and synthesizes common patterns, differences and identifies the most and least balanced approaches.

#### 4.3.4.1 Requirements Validation & Verification Category and Purpose

Across the seven RVV studies, five focused on automated detection and resolution of requirement quality issues (71%), one targeted clarification and review support (14%), and one targeted traceability (14%) task. From the five quality-issue approaches, two covered multiple issue types (2/7, 29% overall). Also, [31] aligned ISO/IEC/IEEE 29148 across ambiguity, incompleteness, inconsistency and additional criteria, and [17] examined ambiguity, inconsistency and incompleteness on an industrial SRS. Three approaches focused on a single issue (3/7, 43% overall). [27] addressed anaphoric ambiguity, [28] targeted syntactic ambiguity, and [16] focused on incompleteness. Besides quality issues, clarification and review was represented in [12], while traceability was represented by [21].

Ambiguity detection was the most widely addressed quality concern appearing in four studies (57%), incompleteness followed appearing in three (43%), and inconsistency in two (29%), while the broader ISO-aligned attributes were covered by one LLM-based approach (14%). Overall, the literature remains dominated by defect detection and resolution with a clear emphasis on ambiguity, while multi-issue coverage, analyst-oriented clarification, and traceability appeared less frequently. The approaches that stood out for this criterion were [31] and [17] due to their coverage on multiple RVV issue types in a single workflow, [27] offering the best single-issue design because it defined and completed an end-to-end resolution objective, and [21] that combined a solution for classification and traceability.

#### 4.3.4.2 Datasets

Most RVV studies relied on datasets curated by the authors. Although they attempt to standardize domain relevant datasets, no standardized dataset was adopted. Cross-domain generalizability was rarely validated, which limited the approaches' cross evaluation and evaluation for real-world applicability. The volume of data varied significantly, from a few hundred sentences to thousands.

Paper [27] introduced DAMIR with 1251 industrial requirements and a smaller ReqEval set with 98 items. Together these provided the strongest dataset contribution, since they were released and structured for benchmarking purpose. The study [16] offered the largest scale evaluation by incorporating the PURE subset that contained 24000 requirements. The approach [17] also worked at larger scale, testing on a 52-page SRS, although this document was not released for reproducibility. Article [12] contributed REQuestA, a semi synthetic QA resource with 387 question answer pairs, which enabled reproducible experiments for review support.

The remaining approaches made more modest contributions for this criterion. Approach [28] evaluated on AmbiGO with 100 sentences and added a small curated set of 26 further sentences. LLM based quality assessment over ISO criteria in [31] used small PURE project subsets, in particular Stopwatch with 10 requirements and Digital Home with 63 requirements. Traceability in [21] also used PURE, although the exact subset size was not reported. Overall, these observations highlight a clear need for larger and more balanced standardized datasets that are specific to RVV.

#### 4.3.4.3 Tools

Development infrastructure was consistently Python-based in RVV approaches, typically within Jupyter environments. In deep learning frameworks, PyTorch was prevalent with which models were trained or fine-tuned. For core models, most studies used pretrained encoders from Hugging Face ([27], [16]). LLM APIs such as GPT-4 (Mahbub et al., 2024), GPT-3.5 [21], and Llama 2 in [31] were used for prompting. Pipelines commonly paired scikit-learn with NLP toolkits such as spaCy, NLTK, and Stanza. Where syntax mattered, Stanford CoreNLP was used ([28], [27]). Domain-specific tools were rare. WikiDoMiner supported vocabulary expansion in [16]. Overall, a small set of tools recurred across papers rather than a single dominant choice. Hugging Face Transformers appeared in 3 of 7 papers, and in 2 of 7 papers each PyTorch, spaCy, NLTK, scikit-learn, Stanford CoreNLP, Jupyter, and OpenAI GPT APIs. Considering tool variety and operational robustness for this criterion, [27] offered the strongest tool stack by integrating the largest set of components. [12] and [16] followed by using also a broad stack of tools. API-based setups in [17] and [31] were minimal in variety of tools, however they provided ease of deployment.

#### 4.3.4.4 Techniques

Across RVV there was a shift from rule based parsing techniques to transformers and LLM prompting. Early work focused on syntactic analysis. Later approaches used contextual embeddings and generative reasoning to capture semantic relationships within requirements. Transformer encoders from the family of BERT models appeared in three of the seven studies and formed the dominant technical paradigm among methods that did not use LLM prompting.

Approach [27] used SpanBERT, together with an ensemble of ML classifiers for anaphora detection and resolution. The model combined contextual token representations with a controlled reranking mechanism. This design enabled high recall and fine grained disambiguation. Study [16] used a BERT based masked language modeling strategy to simulate missing content and to predict incomplete requirements. The approach scaled well, yet its formulation at the token level limited semantic coverage and led to low accuracy. The approach in [12] extended transformer encoders to a retrieval augmented question answering setting. They combined BM25 retrieval, passage reranking, and an ALBERT reader so that clarification tasks depended on grounded textual evidence instead of pure classification.

LLMs appeared in three studies and introduced a separate methodological line that focused on zero shot evaluation instead of model training. Approach [31] used Llama 2 to assess requirements against ISO/IEC/IEEE 29148 quality criteria and to generate rewrites. Study [17] used GPT-4 to detect multiple quality issues, including ambiguity, incompleteness, and inconsistency, on industrial SRS data. The [21] used GPT 3.5 for traceability and for FR and NFR classification through systematic prompt engineering with variation in temperature. These approaches shaped the architecture design with prompt optimization.

The [28] remained the only study that used a purely rule based syntactic parser. Its scoring algorithm generated and ranked multiple parse trees to resolve structural ambiguities. The method was simpler but reached high precision on coordination and attachment cases and produced interpretable intermediate outputs.

Under this criterion, the most effective technical configuration was the hybrid SpanBERT with reranking proposed by [27]. This model effectively captured contextual meaning through SpanBERT embeddings while its reranking mechanism limited candidate antecedents to the most probable ones, resulting in consistently reliable performance in anaphora detection and resolution. Within its own task, [12] followed closely by grounding transformer reading in retrieval, which improved interpretability. The weakest technique appeared in study [16], where simulated masking with post-filtering via traditional ML classifiers resulted in low performance results. Also, in the single prompt zero shot frameworks of [17] and [31], although the approaches offered broad coverage, they achieved inconsistent precision.

#### 4.3.4.5 Results

The reported results across the RVV studies differed in both performance metrics and the way they were presented, yet some shared patterns still emerged. Precision, recall, and F1 score appeared most often, while a few works also reported measures such as F2 score, success rate, or coverage. In general, recall values were high and precision was sometimes lower, especially in LLM based approaches. This pattern shows that false positives remain a common issue. As in RA, execution times were rarely reported, since only two of the seven studies provided them, so scalability in RVV largely remains unassessed.

Looking at individual approaches, [27] achieved the most balanced results. Their hybrid method, which used a supervised ML ensemble for ambiguity detection and SpanBERT for resolution, reached 100% recall, 58–61% precision, and F2-score between 87% and 89% for detection, with a 98% success rate for resolution. Runtime was around 16 seconds per pronoun, including 1.5 seconds for resolution. Precision stayed at a moderate level, but recall was perfect

and ambiguity resolution accuracy was high. [28] also showed strong performance with the rule based syntactic parser. The model achieved F1 scores between 63% and 100% across all experiments, with results in the complex experiment ranging from 87% to 100%. Precision varied from 46% to 100% depending on the experiment, while recall stayed consistently above 88%. These results indicate that the model detected ambiguities reliably but produced false positives. Even so, its stable and high scores make it one of the most solid RVV approaches under this criterion.

The [31] reported good recall but very poor precision and could not be promoted under this criterion. The evaluation of Llama 2 on ISO quality criteria produced κ between 0.05 and 0.40, precision between 12.5% and 50%, and recall between 74% and 89% across different experiments and datasets. [17], also based on LLM prompting, showed fluctuating performance across quality dimensions. Ambiguity precision ranged from 17% to 60% with an average of 39%, inconsistency from 33% to 57% with an average of 43%, and completeness from 50% to 92% with an average of 61%. Recall and coverage were reasonable, yet unstable precision showed that prompting robustness had not been reached. The proposed approach in [12] achieved satisfactory performance within the QA setting, with 84.2% semantic accuracy, 79% partial match, 24% exact match, and Recall@1 equal to 100% for document retrieval. However, the use of metrics such as Recall@3 and semantic accuracy made direct comparison with other RVV studies difficult. Approach [21] reported 91% accuracy but only 43–45% precision, 60–64% recall, and 51–53% F1 for traceability, which means that the model struggled to identify correct traceability links consistently, despite the apparently high accuracy.

Finally, [16] achieved the weakest results overall. Their BERT based approach for incompleteness detection reached 12.1% top 1 accuracy on the 50–50 split and 2.25% on the 90–10 split, with only slight improvement under lenient filtering. Coverage reached 40% and 33% for these two configurations. The dataset was large, but the overall effectiveness remained low.

#### 4.3.4.6 Innovation

The most common innovations in the reviewed approaches focused on two main areas: the introduction of new datasets and the design of new architectures. For datasets, two new resources were curated specifically for the studies. Papers [27] introduced DAMIR, and [12] created REQuestA. On the architectural side, the use of LLMs which appeared often enabled zero-shot setup or systematic evaluation with multiple prompt techniques. Some work also proposed hybrid architectures that combined classical machine learning with transformer based models. Other notable directions included moving towards quality assessment aligned with ISO standards and supporting clarification to users through question answering.

Methods in [27] and [12] stand out as the most innovative approaches, each pushing the field forwards. The study from 2022 introduced the DAMIR industrial dataset, compared six distinct models, and addressed both ambiguity detection and anaphora resolution through a hybrid of ML ensembles and SpanBERT. The 2023 work delivered a clarification tool based on questions and answers and introduced REQuestA, a dedicated QA dataset for requirements engineering. The

study [31] is also among the most innovative contributions, since it enabled evaluation of requirements against ISO 29148 criteria and provided a reproducible prompt structure.

Other studies made useful contributions, but with an intermediate level of innovation compared to these. Approach [17] used GPT 4 to handle several tasks, including detection of quality issues, evaluation of diagrams for quality problems, and assessment of different SRS versions. [21] proposed a prompt engineering methodology with GPT 3.5 that tested multiple prompt patterns together with several temperature settings for the traceability task.

#### 4.3.4.7 Limitations

Across the RVV studies, several recurring limitations appeared, mainly related to dataset diversity, evaluation design, prompt robustness, and generalizability. A common issue was the use of small datasets defined by the authors themselves, which restricts both scalability and variation in domains. Most studies relied on limited or custom made corpora such as DAMIR, PURE subsets, or a single SRS case study, without evidence of cross domain or large scale validation. This dependence on custom or small datasets reduces the external validity of the reported results and limits their applicability in industrial settings.

A second major limitation concerns the lack of a consistent evaluation methodology. Metrics such as precision, recall, and F1 were widely used, but only a few studies applied any kind of statistical validation, and none reported standard cross validation or external replication. In the LLM based work, evaluation depended heavily on a single prompt configuration, without systematic exploration of prompt patterns, temperature settings, or few shot examples. This led to unstable precision and incomplete understanding of model sensitivity. Execution times were also rarely documented, apart from [27] and [12]. Three LLM based approaches were expected to respond quickly, yet none reported runtime or computational cost, which still matters for assessing scalability and latency under industrial workloads.

Other limitations identified in specific studies further reveal methodological gaps. The paper [31] evaluated only one LLM, Llama 2, with a single fixed prompt template and two small datasets. Together with low precision, this strongly restricts the potential for generalization. The [17] tested only one prompt structure on industrial data and did not examine how limits of the context window affected results, nor did they compare against other LLMs. Approach [21], while proposing a systematic prompting framework, still evaluated only GPT 3.5 Turbo and did not benchmark against baselines or alternative models. Approach [16] used simulated incompleteness instead of real incomplete requirements, which reduces the representativeness of the evaluation. Approach [28], although syntactically robust, did not consider semantic ambiguity or transfer across domains. Even the strongest studies, [27] and [12], relied on author created datasets and small external test sets, which still limits the scope of their generalization.

#### 4.3.4.8 Overall Comparison of RVV Approaches

Although a direct comparison between the reviewed approaches is not entirely fair due to differences in their objectives, evaluation settings, and datasets, the current section highlights approaches that demonstrate balanced performance among the selected criteria, while also pointing out those that, despite interesting ideas, cannot be considered optimal due to significant

limitations. Among the reviewed RV&V approaches, a few stand out for either their high performance, innovative design, or balanced methodology, though none are without limitations.

Across all criteria, [27] emerged as the most complete and balanced approach. The purpose was realised from end to end, moving from anaphoric ambiguity detection to its resolution. The dataset contribution was substantive through DAMIR together with the structured use of ReqEval. The tool stack was the most comprehensive and transparent, combining classical natural language processing, scikit learn, and Hugging Face models on top of PyTorch. The chosen technique, a SpanBERT pipeline coupled with a lightweight ensemble, matched the task well and produced the most credible result profile for RVV, with 100 percent recall and about 60 percent precision for detection and about 98 percent success for resolution, while also reporting latency. Innovation came from the careful comparison of six alternative solutions and the domain adaptation of SpanBERT. The main weakness concerned external validity, since the data was curated by the authors, yet overall this study satisfied the largest subset of criteria at a consistently high level.

The [28] followed as the next most balanced approach. The purpose focused on syntactic ambiguity with transparent rule-based resolution. The tools centred on Stanford CoreNLP and produced interpretable intermediate artefacts. The technique itself was simple but well aligned with structural phenomena. Results were strong and stable, with F1 scores ranging from 63 percent to 100 percent across experiments and from 87 percent to 100 percent in complex scenarios, although precision varied widely from 46 percent to 100 percent. Limitations included the focus on syntax without semantic modelling and the modest size of the datasets. Even so, methodological clarity, reproducibility, and reported performance placed this work among the most credible RVV baselines.

Approaches oriented towards analyst support and broader coverage ranked well on specific criteria but were less balanced overall. Paper [12] defined and fulfilled a distinct purpose, namely clarification and review, through a retrieval augmented question answering pipeline with an ALBERT reader and contributed the REQuestA dataset, which improved reproducibility. Tools and techniques were sound and modular, and results were satisfactory within the intended scope, although the metrics were not directly comparable to defect detection studies and runtime was only partly reported. [31] and [17] achieved breadth at the purpose level by covering multiple issue types and, in the case of Lubos et al., by aligning checks with ISO 29148 and generating rewrites. Both approaches, however, relied on single prompt or otherwise limited prompting setups over small datasets and showed unstable precision, which reduced their overall balance despite their innovative character.

At the opposite end, [16] proposed the least balanced approach overall. The dataset scale was the largest, the tools and encoders were appropriate, and the purpose of the approach involving incompleteness was valuable. Yet the technique depended on simulated masking with limited sentence level semantics, and the result profile was weak, with low top one accuracy and only partial gains in coverage, while external validity remained uncertain. The [21] provided a useful prompting framework for traceability and FR and NFR classification and reported high accuracy, but precision and recall remained at moderate levels, dataset reporting for traceability was incomplete, and the evaluation breadth was restricted to a single model, which constrained the overall maturity.

### 4.3.5. Global Comparison Conclusions

Across all eighteen approaches, the most common purpose was classification, covered in ten studies. Quality defect detection followed in five studies. The remaining three works addressed prioritization in two cases, clarification in one case, and traceability in one case. Important activities, such as conflict identification, richer human in the loop workflows, and end to end traceability received little attention.

From a dataset perspective, the PROMISE family and its extensions formed the most frequently used group, appearing in nine RA studies. This reliance supported direct comparison between models but concentrated evaluation on a relatively small and partly outdated corpus. The PURE dataset comes second in terms of usage appearing in four studies. Other datasets, including RALIC, CBMW, EHR, expired contracts, project documents, WordPress content, Wikipedia articles, and GitHub issues, appeared only once in individual studies and typically at limited scale. In RVV there was no shared benchmark comparable to PROMISE. Most datasets were curated by the authors for specific tasks. DAMIR and ReqEval supported ambiguity related work. REQuestA supported clarification and AmbiGO and a small-curated set supported syntactic ambiguity.

Concerning tools, the overwhelming majority of implementations used Python, often inside Jupyter environments. The most frequently reused libraries were scikit learn and Hugging Face Transformers. Scikit learn appeared in several RA works, including [58, 63, 71, 75, 77] and in at least two RVV studies [12,27] . Hugging Face Transformers was widely used in RA, for example in [71 - 74] and [63], and in several RVV approaches, including [12, 16, 27]. Deep learning frameworks were mainly split between TensorFlow with Keras, which was more common in RA classification, and PyTorch, which was common in transformer-based RA and RVV pipelines. With respec to LLM exploitation, OpenAI GPT interfaces and Llama 2 were the tools used for zero shot prompting. Task specific tools, such as Doccano, WikiDoMiner, or Excel based prioritization tooling were rare and usually unique to individual studies.

In terms of techniques, the reviewed literature was dominated by transformer based contextual encoders from the BERT family, used in eight studies. On the RA side, BERT or BERT like models were central in at least five approaches, including [72] with NoRBERT, [73] with NFRNet, [74] with a BERT plus BiLSTM plus CNN architecture, [71] with GAN BERT, and [63] with a BERT based multilabel setup. On the RVV side, SpanBERT in [27], BERT in [16], and ALBERT in [12] formed the core of several pipelines. The second most common technique group consisted of LLM based methods. One RA contribution, [21] for classification, and three RVV contributions, namely [31], [17], and again [21] for traceability, relied on GPT 3.5, GPT 4, or Llama 2 as their main method.

The most frequently used metrics to provide experimental results were precision, recall, and F1-score, with accuracy reported regularly but playing a less central role. Some studies additionally reported metrics tailored to their problem setting, such as F2-score and G-Mean. Others reported task-specific measures such as coverage and Hamming Loss, as well as agreement-based metrics such as kappa.

RA classification results were generally strong. In the best approaches for binary FR versus NFR, F1-scores typically ranged from about 80% up to 94%, with similarly high scores for many multiclass configurations. Differences in dataset splits however, baseline choices, and formulation details limited direct comparability and left generalization largely untested. RVV results were more uneven. Transformer based and rule-based approaches, such as [27] and [28], reached high

recall and credible F1-scores for specific issues. LLM based RVV studies often showed high recall but weak and unstable precision, which produced many false positives, especially in ISO aligned quality checks and industrial SRS evaluation. In some cases, for example [16], overall accuracy remained low despite substantial data. A critical global observation was that execution times and scalability were rarely documented. Only a few studies, such as [29] in RA and [12,27] in RVV, reported runtime information. For most approaches it remained unclear whether they could meet industrial latency or throughput requirements.

Viewed globally, the most common form of innovation concerned architectures built around transformer-based models. In RA, examples include hybrid BERT plus CNN plus BiLSTM architectures, semi supervised GAN BERT, domain adapted transformers, such as NoRBERT and NFRNet, and deep ensembles. In RVV, innovation was more closely tied to datasets and tasks. DAMIR, ReqEval, and REQuestA provided new resources for ambiguity handling and clarification. Other RVV approaches innovated by aligning LLM based evaluations with ISO 29148 criteria, introducing retrieval augmented QA for clarification, or assessing quality issues and diagrams in industrial SRS documents. Across both RA and RVV, an important emerging trend was the use of LLM enabled workflows for classification, quality assessment, rewriting, traceability, and interactive support. These approaches are still exploratory but indicate a shift towards embedding general purpose LLMs in Requirements Engineering processes.

The most widespread limitation was dependence on specific datasets and the resulting lack of generalizability. RA relied heavily on PROMISE and its derivatives. RVV often used small datasets defined by the authors, such as single SRS documents or small project subsets. A second common limitation concerned evaluation design. Many studies reported several experiment specific results, sometimes did not use cross validation, statistical testing, or consistent baselines. A third recurring limitation was underreported runtime and resource usage, which affected both RA and RVV.

Within these findings and considering the aforementioned comparisons and nominated approaches per phase, a few approaches appear the most mature overall because they combined a clear purpose, realistic datasets, appropriate techniques, strong and well reported results, innovation, and transparent limitations. These were [74] and [72] from RA and [27] from RVV.

Study [74] defined a clear purpose, binary FR versus NFR and NFR multiclass classification in a single framework. The study used a modern hybrid architecture with BERT, BiLSTM, and CNN on top of a standard benchmark and achieved some of the highest F1-scores in this systematic review, with statistically significant gains over several baselines. Its ability to operate directly on raw requirements without manual preprocessing further strengthened its practical relevance. Study [72] extended the scope/purpose, with FR versus NFR and functional and NFR subtypes, with a domain adaptive transformer, NoRBERT, achieving strong results across several evaluation schemes and importantly, plus project hold-out experiments that tested generalisation to unseen projects, a methodological strength that most other RA works lacked.

The study [27] defined an end-to-end objective from anaphoric ambiguity detection to resolution, contributed industrial dataset DAMIR, implemented a well matched hybrid technique with SpanBERT and ML ensembles, and reported detailed metrics for both detection and resolution, including recall, precision, F2-score and success rate. The study also included runtime

measurements and compared six different solutions, which provided rare methodological depth within RVV.

At the opposite end of the spectrum, and under the same criteria, some approaches were clearly less mature. In RA, [58] combined weak performance on informal issue reports, with F1 scores below thirty percent, and limited robustness when used beyond well structured specifications. In RVV, [16] produced the weakest results overall and relied on a single model despite having a large dataset, which led to low overall effectiveness and poor balance across the evaluated criteria. These cases show that in both RA and RVV, promising ideas can still fall short when results, dataset robustness, methodological completeness, and identified limitations are considered together.

Although the presented comparison showed substantial progress in AI-supported RA and RVV, it also exposed some persistent gaps and limitations that motivate the following section of future research directions.

# 5 *Gaps and Future Directions*

The overall review revealed that AI-based support for Requirements Engineering (RE) is promising but still fragmented. The most mature work concentrates on classification in Requirements Analysis (RA), while Requirements Validation & Verification (RVV) remains less explored and often limited to narrow quality issues. In addition, several cross-cutting limitations persist across both phases, including restricted task coverage, dependence on small and unbalanced datasets, limited use of semantics, incomplete evaluation practices, and method-specific issues in LLM-based approaches. The following subsections structure these gaps into concrete challenges and outline future research directions for each one.

## 5.1 Imbalance between Phases Coverage

A clear imbalance was observed between the volume and scope of AI-based approaches for RA compared to those addressing RVV activities. While the reviewed RA studies demonstrated diversity in terms of methods, datasets, and innovations, RVV remains relatively underexplored. The number of RVV contributions is smaller, and their scope is often limited on isolated quality concerns such as ambiguity or incompleteness without extending to broader validation and verification scenarios. This restricts the advancement of AI-supported RVV tools and leaves other important tasks to receive less AI assistance, despite being equally critical for quality requirements.

In parallel, it is worth mentioning that despite the methodological diversity seen in RA approaches, ranging from classical ML to hybrid deep learning, most approaches concentrated on requirements classification task. Specifically, the primary goal across most approaches was the accurate binary FR/NFR classification and multiclass NFR subtype prediction. While this is a valid and necessary task, it leaves other equally important RA activities underexplored and does not cover the full complexity of real requirements engineering practice. Activities such as multilabel classification and multiclass classification were mostly unexplored. Also, conflicting requirements detection and resolution (e.g., negotiation), and prioritisation were insufficiently addressed by the reviewed AI-based approaches.

To address this challenge, future research should extend to more complex RA and RVV activities. For RA, this includes analysis activities, supporting conflict detection by identifying incompatible requirements and negotiation, including multilabel and multiclass classification while considering standardized functional and non-functional requirements subtypes taxonomies. For RVV, future approaches should go beyond quality issues identification to improvements support, such as clearer and more precise wording or alignment with domain terminology. Models should be able to identify how requirements evolved over iterations (traceability) and detect when changes introduce quality issues. Addressing this imbalance requires not only technical

development but also improved problem formulations that reflect the complexity of end-to-end RE practice.

## 5.2 Dataset and Labelling Limitations

The next identified gap is the strong dependence on small number of datasets and the lack of balanced, well-structured data for both RA and RVV. In RA, many studies relied heavily on the PROMISE dataset and its extensions. Although they offer consistent labels and have supported significant progress in FR/NFR and NFR-subtype classification, they are generally small and imbalanced and represent relatively formal requirements. This restricts the evaluation of generalizability and may encourage models to overfit to PROMISE similar data. On the RVV side, almost all datasets are author-defined, often small, domain-specific and not always publicly available. There was no RVV dataset identified comparable to the PROMISE, and there is no shared dataset that covers multiple quality issues within the same requirement. This fragmentation made systematic comparison across approaches impossible and limited the assessment of generalizability, especially for models aiming to detect a broad range of quality problems aligned with ISO/IEC/IEEE 29148.

In addition to size and diversity limitations, the structure of existing datasets restricts the ability to train and evaluate models on tasks that better reflect real RE scenarios. Most RA datasets focus on formal SRS-style requirements, while practical requirements engineering may include preliminary descriptions, items that may appear in a more informal manner. On the RVV side, annotations typically capture only one issue type per requirement, even though real requirements could include multiple overlapping quality issues (e.g., ambiguous and incomplete at the same time). This under-representation of multiple issue problems limits the development of models capable of realistic quality assessment.

Future research should prioritize the development of large, balanced, publicly available benchmark datasets for both RA and RVV, as the current datasets in both areas are small, imbalanced and often not standardized. For RA, future benchmarks should go beyond simple FR/NFR labels and include standardized functional and non-functional subtypes, balanced class distributions and multilabel annotations to reflect the complexity of real requirements. For RVV, new datasets should annotate multiple quality issues per requirement such as ambiguity, incompleteness, inconsistency, so that models can be evaluated on realistic multi-issue scenarios rather than isolated cases. Importantly, datasets in both areas should incorporate formal and informal requirements to support cross-domain generalization. Creating such benchmarks would address current limitations in dataset size, diversity and coverage, and would enable more robust, comparable and practically relevant AI models.

## 5.3 Handling of Informal Requirements

A further challenge observed in the reviewed literature was the limited consideration of current approaches to handle informal requirements. Many models were trained exclusively on short, well-structured requirements, which lead to high results, but a performance degradation was

observed when applied to more realistic or informal requirements. For instance, [58] demonstrated this issue clearly with the proposed model performing well on PROMISE but dropping below 30% F1-scbut droppinged on GitHub issues.

Addressing this limitation requires approaches explicitly designed to manage informal requirements. Future work should develop training datasets that mix formal and informal requirements and evaluation must also include cross-domain testing to determine whether approaches generalise well beyond their training environment. Closing this gap is essential so that AI-supported RA and RVV tools can be useful in heterogeneous industrial environments and data.

## 5.4 Limitations in Evaluation Methodology

A further challenge identified across both RA and RVV literature is the lack of systematic, comparable, and comprehensive evaluation practices. Although the reviewed approaches reported a variety of performance metrics, the evaluation methodology was inconsistent across studies, which limited the comparability of results significantly.

Some studies relied on single train–validation–test splits without applying cross-validation, which limit the results to less representative score of how the model would perform on another random sample. Also, execution times and scalability were mostly omitted from the reviewed literature except a few studies e.g., [27], [12]. Given that industrial RE operate under large amount of requirements, adopting a model depends on factors like time and memory consumption, and the ability to handle large documents. The lack of execution time reporting limits the evaluation about the real-world practicality of the approaches. Another observation was that the current evaluation practice appeared fragmented regarding the performance metrics. Most RA works reported precision, recall and F1-score, sometimes over multiple task experiments without a single aggregated performance result, which made comparison difficult. Similarly, RVV studies used heterogeneous metrics such as κ agreement, accuracy, and coverage. Composite evaluation results that summarize the performance of an approach across multiple dimensions or tasks were rarely reported and their absence limits the ability to identify models that perform well not only in isolated scenarios but in a more general sense.

To address these methodological limitations, future research should develop complete, standardised evaluation frameworks to assess RA and RVV approaches. Such frameworks should include:

(a) Standardised evaluation protocols, including k-fold or stratified cross-validation and cross-domain validation to measure robustness.

(b) Systematic runtime and scalability reporting, covering execution times per requirement or SRS processing or output production, metrics such as memory use. A useful metric for practical applications could also be the performance of the model under increasing document size. These metrics are essential to determine whether an approach would be viable for industrial-scale documents.

(c) Composite metrics, capable of capturing multiple task performance. Such metrics would help identify models that are balanced and stable rather than narrowly optimised for a single task.

Developing such benchmarking frameworks will enable fair comparison between future approaches and provide a clearer view considering real-world requirements engineering practice and support the complexity, scale, and variability of modern RE processes.

## 5.5 Prompt and Context Engineering in LLMs

Prompt engineering represents a promising direction for applying LLMs in RA and RVV. Tools, such as ChatGPT and Copilot, are already being used in many organisations for general-purpose tasks, regardless of RE-specific needs. This widespread access creates a significant opportunity. Analysts could leverage these existing tools to support analysis, verification and validation of requirements without the need for custom model development, training or complex integration. This positions prompt-engineering in LLMs as a potentially low-effort and cost-effective option for industry adoption.

However, despite this potential, current studies showed unstable performance and low precision (17% in some cases) in RE-related prompting tasks. Future work should focus on developing systematic prompt engineering frameworks, such as the one proposed in [60], which assesses reusable prompt patterns (e.g., Cognitive Verifier, Clarifier, Persona) that could be leveraged across the RE tasks.

Moreover, these patterns should be tested with many publicly accessible state-of-the-art LLMs to evaluate their practicability in real requirements scenarios. Given the widespread organizational deployment of these tools, such validation is critical to determine their effectiveness. Additionally, recent research [61] has proposed prompting techniques that incorporate intermediate reasoning steps to achieve improvement in LLM complex reasoning capabilities through Chain-of-Thought prompting. Further, Yao et. al. (2023) assessed the idea of LLMs finding multiple solution paths before selecting an answer with the Tree-of-Thought framework. While not yet applied in RE, these approaches may provide an opportunity to enhance prompt reliability and reasoning quality.

Notably, one main architectural limitation of LLMs could be further improved to provide advancements in the performance of the LLM approaches for effective application to RE tasks. LLMs are limited in the number of tokens they can consider for each prompt by a fixed context window size and drop all additional input that exceeds these limits. The context window (or context length) of an LLM refers to the maximum number of tokens it can process in a single prompt, including both the input and the model's output. It represents the model's "working memory" and affects how much text the model can consider at once. This might be a preventing factor for LLMs from considering very long requirements documents. Increasing the context window brings other advantages besides the ability to handle longer and more complex information, like greater accuracy, reduced hallucinations and more coherent outputs. [38], [39]

Alternatively, context engineering could be treated as a core design concern. Instead of assuming that the entire SRS or project documentation fits inside a single prompt, future work should explore strategies that summarise, filter or segment content before it is sent to an LLM. Examples include retrieval-augmented generation, where only the most relevant requirement fragments are retrieved and passed to the model, or hierarchical prompting, where the model first summarises sections and then reasons over the summaries. Such approaches can help bypass context window limitations without waiting for arbitrarily large models, and they are likely to

reduce hallucinations and improve coherence by focusing the model on the most relevant parts of the input. [95]

Finally, it is important to evaluate whether prompting strategies generalize across domains. Since requirements can vary in structure, terminology or complexity depending on the application area, future studies should assess whether effective prompts retain their performance when applied to different types of requirements or tasks without re-engineering.

## 5.6 Ensemble Learning Techniques

A recurring pattern across the reviewed literature was the instability of models performance, particularly when models were evaluated outside their training dataset or in more complex tasks. These findings indicate that single-model pipelines, whether classical ML, deep learning, or LLM prompting, may not be sufficiently robust to handle the variability of real-world requirements.

Ensemble learning offers a promising direction to improve stability, accuracy and generalization in RA and RVV. The idea of ensemble learning is to combine two or more learners to leverage their complementary strengths, i.e., to increase the accuracy of predictions or lower the error rates [97]. Ensemble methods are widely recognised in machine learning for increasing predictive robustness, reducing variance and mitigating overfitting, particularly when dealing with small or noisy datasets. They can improve performance by aggregating outputs through methods, such as bagging, boosting, stacking or weighted voting, or by combining orthogonal reasoning strategies. [96], [101] From an RE perspective, ensembles would be particularly beneficial for tasks where different models perform well at different tasks.

Notably, ensembles were already used in some of the reviewed approaches. [75] employed a two-phase ensemble architecture for FR/NFR subtype classification, combining BiLSTM, LSTM, CNN and GRU models and achieving the highest performance. Authors [27] implemented an ensemble of ML classifiers for ambiguity detection before applying a SpanBERT-based resolver. These examples show that ensemble techniques are feasible and useful in RE, and future work could explore broader ensemble and hybrid methods for RA and RVV.

## 5.7 Semantics-Based Approaches for Deeper Understanding

It was observed in the reviewed literature that often semantics were neglected from the approaches in both RA and RVV which eventually affected their performance. Authors in [58] demonstrated that models trained on structured formal requirements performed poorly when applied to informal GitHub issues, indicating that the statistical learning (technique) chosen did not generalise well. Paper [28] relied on syntactic parsing for ambiguity detection but did not integrate semantic context as well, which led to false positives. Transformer-based RVV approaches, such as [27] and [16], lacked on semantic grounding, limiting their ability to identify semantic issues. This omission can become clearer in tasks that require a deeper understanding, including conflict identification, inconsistency detection, completeness verification between requirements.

Earlier RE work [98], [99], has shown that semantic representations can improve quality of requirements. For example, ontology-based approaches that combine domain ontologies with boilerplates (for example SENSE approach) can contribute to semantically correct requirements by helping detect contradictory constraints in text, ambiguous terms, or missing elements. [100]

Future research should therefore revisit and modernize semantics-based methods by integrating them into AI methodologies for automation. Semantic layers, such as domain ontologies, semantic role structures, or knowledge-graphs, could be incorporated into RA and RVV approaches to provide models with a more meaningful representation of requirements and help overcome several limitations aforementioned in this review, notably the difficulty in detecting complex quality issues or the inability of LLMs to provide semantically correct output.

## 5.8 User Feedback and Adaptive Mechanisms

Across both RA and RVV approaches, a clear gap is the absence of models that incorporate user feedback or enabling requirements corrections. The reviewed studies operated in a rather static configuration. The models mostly provide an output but mostly did not incorporate user feedback or adapt based on the acceptance or rejection of the outputs. This is partly contradicting with the nature of real RE processes which is iterative, includes negotiations, and is changed based on reviews and continuous clarifications. In several RA approaches, for example those handling NFR subtypes, it was observed that misclassifications occurred, which would have been necessary for an analyst to correct; however, the models did not provide such capabilities. Similarly, in RVV studies particularly ambiguity or incompleteness detection false positives occurred as well.

Future work should therefore investigate human-in-the-loop and improvement enabled approaches for RA and RVV tasks. Models should be able to adapt for example to domain terminology, writing patterns, organisational rules, based on the user feedback. This could involve reinforcing accepted suggestions, not considering repeatedly rejected ones, or understanding project or organization specific terminology. Such adaptive behavior would make AI-based RE tools more robust, personalised, and aligned with real analyst environment.

In parallel, AI support should include user actions, such as improvements, rewriting ambiguous texts, proposing domain-consistent terminology, or fixing completeness issues. Integrating these features within a single pipeline would offer more complete RE assistance for real world adaptation.

# 6 *Conclusions*

This thesis presents a comprehensive and structured review of AI-based approaches applied to RE, with a focus on both in RA and RVV. Its main contribution lies in offering an evaluation of relevant approaches proposed within the recent period 2020-2024, identifying trends, emerging innovations, and critical limitations that shall be addressed in order to transition from academic approaches to practical, scalable, and reliable tools for real-world AI use in RE. By employing clearly defined comparison criteria, the thesis enables a transparent comparison across approaches and provides various valuable insights that can assist researchers and practitioners build better models by following the proposed research future directions. In this context, approaches that stood out were highlighted, and assessed for their limitations or their applicability.

Both RA and RVV already contain solid and valuable approaches, yet the field has still several limitations to resolve. The main review limitations emphasize an imbalance between the number and depth of RA approaches compared to those supporting RVV. While RA studies have extensively explored classification tasks, other related RA tasks remain rather underexplored. Further, RVV seems to be often limited to ambiguity detection and compared to RA, relatively underexplored in terms of its main tasks. Therefore, broader task coverage is needed to reflect the complexity of real RE workflows. Additionally, the frequent use of benchmark datasets, such as PROMISE, restricts the assessment of generalizability, while many approaches also omitted a cross-domain evaluation or execution time analysis. These suggest the need for the creation of large, balanced, multi-domain benchmarks that integrate formal and informal requirements. Weak external generalisability highlight the importance of cross-domain evaluation. Methodological inconsistencies and performance metrics fragmentation emphasise the need for standardised evaluation protocols including runtime reporting. The instability of LLMs indicates the importance of reusable prompt frameworks and context-engineering techniques. Limited semantic grounding suggests integrating semantic techniques, such as ontologies and knowledge graphs. Finally, the absence of interactive or adaptive mechanisms points to the need for human-in-the-loop designs that enable users to refine and improve model outputs.

The review process provided valuable experience in systematically reviewing scientific literature, analysing and comparing machine learning, deep learning and natural language processing methods, and understanding how such technologies can be applied to RE problems. Through this work, a deeper understanding was developed of RE and its subsequent phases within SE frameworks plus of AI methodologies and techniques, including deep learning models, pretrained transformers and LLMs.

# *Bibliography*

*[1]* Méndez Fernández, D., Wagner, S., Kalinowski, M., et al. (2017), Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. In: *Empirical Software Engineering*, 2017, p 2298-2338. doi: 10.1007/s10664-016-9451-7.

*[2]* Sommerville, I. (2011), *Software Engineering* (9th ed.)

*[3]* Washizaki, H. et al. (2024), *Guide to the Software Engineering Body of Knowledge (SWEBOK v4.0).* IEEE Computer Society. doi: 10.1109/SWEBOK.2024.

*[4]* Chowdhary, K. R. (2020), *Fundamentals of Artificial Intelligence,* doi: 10.1007/978-81-322-3972-7.

*[5]* International Organization for Standardization. (2018), ISO/IEC/IEEE 29148:2018 – Systems and software engineering – Life cycle processes – Requirements engineering.

*[6]* International Institute of Business Analysis. (2009), *A Guide to the Business Analysis Body of Knowledge (BABOK Guide)* (Version 1.6).

*[7]* Romero, J. R., Medina-Bulo, I., & Chicano, F. (2023), *Optimising the Software Development Process with Artificial Intelligence.* doi: 10.1007/978-981-19-9948-2.

*[8]* Russell, S., & Norvig, P. (2021), *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

[9] Nuseibeh, B., & Easterbrook, S. (2000), Requirements engineering: A roadmap. In: *Proceedings of the Conference on the Future of Software Engineering (ICSE 2000)*. ACM, DOI:10.1145/336512.336523

*[10]* Hofmann, H. F., & Lehner, F. (2001), Requirements engineering as a success factor in software projects. *IEEE Software*. doi: 10.1109/MS.2001.936219

*[11]* Singh, S., & Sambhav, S. (2023), Application of artificial intelligence in software development life cycle: A systematic mapping study. In: *Proceedings of ICMETE 2022.* Springer. doi: 10.1007/978-981-19-9512-5_60.

*[12]* Ezzini, S., Abualhaija, S., Arora, C., and Sabetzadeh, M. (2023), AI-based question answering assistance for analyzing natural-language requirements. In: Proceedings of the 45th International Conference on Software Engineering (ICSE 2023), May 14–20, 2023, doi: https://doi.org/10.1109/ICSE48619.2023.00113

*[13]* Meyer, B. (2022), *Handbook of Requirements and Business Analysis.* Springer. doi: 10.1007/978-3-031-06739-6.

*[14]* Kitchenham, B., & Charters, S. (2007), *Guidelines for Performing Systematic Literature Reviews in Software Engineering.* Technical Report, Keele University.

*[15]* Powers, D. M. W. (2011), Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies,* doi :10.9735/2229-3981

*[16]* Luitel, D., Hassani, S., and Sabetzadeh, M. (2024), Improving requirements completeness: Automated assistance through large language models. Requirements Engineering, 29, 73–95. https://doi.org/10.1007/s00766-024-00416-3

*[17]* Mahbub, T., Dghaym, D., Shankarnarayanan, A., Syed, T., Shapsough, S., and Zualkernan, I. (2024), Can GPT-4 aid in detecting ambiguities, inconsistencies, and incompleteness in requirements analysis? A comprehensive case study. IEEE Access, 12, 171972–171988. https://doi.org/10.1109/ACCESS.2024.3464242

*[18]* Xie, T. (2013), The synergy of human and artificial intelligence in software engineering. In: *Proceedings of the 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2013)*, May 2013, doi: 10.1109/RAISE.2013.6615197.

*[19]* Liu K. Reddivari, S. & Reddivari K. (2022), Artificial intelligence in software requirements engineering: State-of-the-art. In: *Proceedings of the 23rd IEEE International Conference on Information Reuse and Integration for Data Science,* IEEE. doi: 10.1109/IRI54793.2022.00034.

*[20]* Cheng H., Husen J. H., Lu Y., Racharak T., Ubayashi N., Washizaki H. (2023), Generative AI for requirements engineering: A systematic literature review.

*[21]* K. Ronanki, B. Cabrero-Daniel, J. Horkoff, and C. Berger, *Requirements Engineering Using Generative AI: Prompts and Prompting Patterns*, In: Generative AI for Effective Software Development, Springer, 2024, pp. 109–127. doi: 10.1007/978-3-031-55642-5_5

*[22]* Arora C., Grundy J., Abdelrazek M. (2023), *Advancing requirements engineering through generative AI: Assessing the role of LLMs*. doi: 10.48550/arXiv.2310.13976.

*[23]* Pradhan S, Ramshaw L, Marcus M, Palmer M, Weischedel R, Xue N. 2011. *CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes.* In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2011 Shared Task)*.

*[24]* Necula S, Dumitriu F, Greavu-Șerban V. (2023), *A systematic literature review on using natural language processing in software requirements engineering.* doi: https://doi.org/10.3390/electronics13112055

[25] Kaur K., Kaur P. (2024), *The application of AI techniques in requirements classification: A systematic mapping*. doi: https://doi.org/10.1007/s10462-023-10667-1

[26] Zhao L, Alhoshan W, Ferrari A, Letsholo KJ, Ajagbe MA, Chioasca EV, Batista-Navarro RT. (2021). *Natural language processing for requirements engineering: A systematic mapping study.* ACM Computing Surveys, doi: https://doi.org/10.1145/3444689

[27] Ezzini S., Abualhaija S., Arora, C., Sabetzadeh M. (2022), *Automated handling of anaphoric ambiguity in requirements: A multi-solution study*. In *Proceedings of the 44th*

*International Conference on Software Engineering (ICSE '22)*, May 21–29, 2022, ACM, doi: https://doi.org/10.1145/3510003.3510157.

*[28]* Osama M., Zaki-Ismail A., Abdelrazek M., Grundy J., Ibrahim A. (2020), Score-based automatic detection and resolution of syntactic ambiguity in natural language requirements. In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2020)*. doi: 10.1109/ICSME46990.2020.00067.

*[29]* Hujainah F., Keung J., Grundy J., Ghani I., Bakar M., Hassan R. (2021), SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects. *Information and Software Technology*. doi: https://doi.org/10.1016/j.infsof.2020.106501

*[30]* Hujainah F., Bakar R. B. A., Abdulgabber M. A. (2019), *StakeQP: A semi-automated stakeholder quantification and prioritisation technique for requirement selection in software system projects*. *Decision Support Systems*, doi: 10.1016/j.dss.2019.04.009.

*[31]* Lubos, S., Felfernig, A., Tran, T. N. T., Garber, D., El Mansi, M., Erdeniz, S. P., and Le, V.-M. (2024), Leveraging LLMs for the quality assurance of software requirements, doi: 10.1109/RE59067.2024.00046

*[32]* Gusenbauer, M. (2022), *Comparing the disciplinary coverage of 56 bibliographic databases.* doi: 10.1007/s11192-022-04289-7.

*[33]* Shilpi Singh, & Sambhav, S. (2023), *Application of artificial intelligence in software development life cycle: A systematic mapping study*. In: *Proceedings of ICMETE 2022.* Springer. doi: 10.1007/978-981-19-9512-5_60.

*[34]* Meyer, B. (2022), *Handbook of Requirements and Business Analysis.* Springer. doi: 10.1007/978-3-031-06739-6.

*[35]* Martínez Garcia, S. E., Fernández-y-Fernández, C. A., & Ramos Pérez, E. G. (2023), Classification of non-functional requirements using convolutional neural networks. In: *Programming and Computer Software*, Springer, doi: 10.1134/S0361768823080133.

*[36]* Airlangga, G., Al-Baity, H. H., & Hussain, A. (2024), Enhancing software requirements classification with semisupervised GAN-BERT technique. In: *Journal of Electrical and Computer Engineering*. doi: 10.1155/2024/4955691.

*[37]* Russell, S., & Norvig, P. (2009), *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.

*[38]* IBM. (2024), What is a context window? *IBM Think Blog*. Available: https://www.ibm.com/think/topics/context-window.

*[39]* Tahir. (2025), Understanding LLM context windows: Tokens, attention, and challenges. *Medium*, Available: https://medium.com/@tahirbalarabe2/understanding-llm-context-windows-tokens-attention-and-challenges-c98e140f174d.

*[40]* Google. (2024), Accuracy, precision, and recall. *Machine Learning Crash Course*. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

*[41]* IBM. (n.d.), *What is Artificial Intelligence (AI)?* IBM Think. Available: https://www.ibm.com/think/topics/artificial-intelligence.

*[42]* IBM. (n.d.), *Machine learning.* IBM Think. Available: https://www.ibm.com/think/topics/machine-learning.

*[43]* IBM. (n.d.), AI vs machine learning vs deep learning vs neural networks. IBM Think. Available: https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

*[44]* Google Cloud. (n.d.), *What is supervised learning?* Available: https://cloud.google.com/discover/what-is-supervised-learning.

*[45]* IBM. (n.d.), *Self-supervised learning.* IBM Think. Available: https://www.ibm.com/think/topics/self-supervised-learning.

*[46]* IBM. (n.d.), *Natural language processing (NLP).* IBM Think. Available: https://www.ibm.com/think/topics/natural-language-processing.

*[47]* Amazon AWS. (n.d.), *What is natural language processing (NLP)?* AWS. Available: https://aws.amazon.com/what-is/nlp/. *(old: [17])*

*[48]* Educative.io. (2024), *Data science in 5 minutes: What is one-hot encoding?* Available: https://www.educative.io/blog/one-hot-encoding#what.

*[49]* Space Studies Institute. (2023), *Difference between a conference paper and a journal paper.* Available: https://spacestudies.co.uk/blog/difference-between-a-conference-paper-and-a-journal-paper.

*[50]* IBM. (2024), Analysis of variance (ANOVA). *IBM Documentation*. Available: https://www.ibm.com/docs/en/cognos-analytics/12.1.x?topic=tests-analysis-variance-anova

*[51]* Loshchilov, I., & Hutter, F. (2019), *Decoupled weight decay regularization. In: International Conference on ICLR 2019.* Available: https://arxiv.org/abs/1711.05101.

*[52]* ISO, What is artificial intelligence (AI)? International Organization for Standardization. Available: https://www.iso.org/ai.

*[53]* Gupta, A., and Gupta, C. (2022), CDBR: *A semi-automated collaborative execute-before-after dependency-based requirement prioritization approach*. In: Journal of King Saud University – Computer and Information Sciences, doi: https://doi.org/10.1016/j.jksuci.2018.10.004

*[54]* IBM. (2025), *Understanding embeddings in machine learning.* Amazon Web Services. Available: https://aws.amazon.com/what-is/embeddings-in-machine-learning.

*[55]* Singh S., & Sambhav S. (2023), *Application of AI in software development life cycle: A systematic mapping study.* In: Micro-Electronics and Telecommunication Engineering . Lecture Notes in Networks and Systems, vol 617. Springer, doi: https://doi.org/10.1007/978-981-19-9512-5_60

*[56]* H. Washizaki, eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*, Version 4.0, IEEE Computer Society, 2024, www.swebok.org.

*[57] Understanding precision, recall and F-score at K in recommender systems*. Available: https://krishnapullak.medium.com/understanding-precision-recall-and-f-score-at-k-in-recommender-systems-7146a0dce68e.

*[58]* Pérez-Verdejo J. M., Sánchez-Garcia Á. J., Ocharán-Hernández J. O., Mezura-Montes E., Cortés-Verdín K. (2021), *Requirements and GitHub issues: An automated approach for quality requirements classification.* Programming and Computer Software, doi: https://doi.org/10.1134/S0361768821080193

*[59]* U. K. Durrani, M. Akpinar M. F. Adak, A. T. Kabakus, M. M. Ozturk, and M. Saleh, *A Decade of Progress: A Systematic Literature Review on the Integration of AI in Software Engineering Phases and Activities (2013-2023)*, doi: 10.1109/ACCESS.2024.3488904

*[60]* 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*. doi: https://doi.org/10.48550/arXiv.2302.11382

*[61]* Wei J., Wang X., Schuurmans D., Bosma M., Ichter B., Xia F., Chi E. H., Le Q., Zhou D. (2022), *Chain-of-thought prompting elicits reasoning in large language models*. In: Advances in Neural Information Processing Systems (NeurIPS 2022). doi: https://doi.org/10.48550/arXiv.2201.11903

*[62]* Yao S., Yu D., Zhao J., Shafran I., Griffiths T. L., Cao Y., Narasimhan K. (2023). *Tree of Thoughts: Deliberate problem solving with large language models*. In: Advances in Neural Information Processing Systems (NeurIPS 2023). doi: https://doi.org/10.48550/arXiv.2305.10601

*[63]* Sainani A., Anish P. R., Joshi V., and Ghaisas S. (2020), *Extracting and classifying requirements from software engineering contracts.* In: Proceedings of the 28th IEEE International Requirements Engineering Conference (RE 2020), doi: https://doi.org/10.1109/RE48521.2020.00026

*[64]* Z. Kurtanović and W. Maalej, *Automatically classifying functional and non-functional requirements using supervised machine learning*, in *Proc. 25th IEEE Int. Requirements Eng. Conf. (RE)*, doi: 10.1109/RE.2017.82

*[65]* F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, *Requirements classification with interpretable machine learning and dependency parsing*, In: *Proc. 27th IEEE Int. Requirements Eng. Conf. (RE)*, doi: 10.1109/RE.2019.00025

*[66]* A. Dekhtyar and V. Fong, *RE data challenge: Requirements identification with Word2Vec and TensorFlow*, In: Proceedings 25th IEEE Int. Requirements Eng. Conf. (RE), doi: 10.1109/RE.2017.26

*[67]* S. A. Asif, Z. Masud, R. Easmin, and A. U. Gias (2017), *SAFFRON: A semi-automated framework for software requirements prioritization* In: International Journal of Advanced Computer Science and Applications (IJACSA), doi: https://doi.org/10.14569/IJACSA.2017.081265

*[68]* S. Lim, S. Lim, M. Harman, A. Susi (2012), *Using genetic algorithms to search for key stakeholders in large-scale software projects*, In: Aligning Enterp. Syst. Softw. Archit., 2012, doi: https://doi.org/10.4018/978-1-4666-2199-2.ch007.

*[69]* V. Veerappa (2012), *Clustering methods for requirements selection and optimisation*, Ph.D. dissertation, School of Computer Science and Engineering, Univ. College London, London, U.K., 2012.

*[70]* R. Khezri, *Automated detection of syntactic ambiguity*, Master's thesis, University of Gothenburg, Gothenburg, Sweden, 2017.

*[71]* Airlangga, G. (2024), *Enhancing software requirements classification with semisupervised GAN-BERT technique*. In: Journal of Electrical and Computer Engineering, vol. 2024, doi: https://doi.org/10.1155/2024/4955691

*[72]* Hey T., Keim J., Koziolek A., Tichy W. F. (2020), *NoRBERT: Transfer Learning for Requirements Classification.* In: Proceedings of the 28th IEEE International Requirements Engineering Conference (RE 2020), IEEE. doi: https://doi.org/10.1109/RE48521.2020.00028

*[73]* Li B., Nong X. (2022), *Automatically classifying non-functional requirements using deep neural network.* In: Pattern Recognition, vol.132, Elsevier. doi: https://doi.org/10.1016/j.patcog.2022.108948

*[74]* Kaur, K., and Kaur, P. (2023), *Improving BERT model for requirements classification by bidirectional LSTM-CNN deep model*. In: Computers and Electrical Engineering, vol. 108, Elsevier. doi: https://doi.org/10.1016/j.compeleceng.2023.108699

*[75]* Rahimi, N., Eassa, F., and Elrefaei, L. (2021), *One- and two-phase software requirement classification using ensemble deep learning*. doi: https://doi.org/10.3390/e23101264

[77] AlDhafer, O., Ahmad, I., and Mahmood, S. (2022), *An end-to-end deep learning system for requirements classification using recurrent neural networks.* In: Information and Software Technology, vol. 147, Elsevier. doi: https://doi.org/10.1016/j.infsof.2022.106877

*[78]* Baker C, Deng L, Chakraborty S, Dehlinger J. (2019), *Automatic multi-class non-functional software requirements classification using neural networks.* In: Proceedings of the 2019 IEEE 43rd annual computer software and applications conference (COMPSAC). IEEE doi:10.1109/COMPSAC.2019.10275.

*[79]* Kumar GR, Chakraborty S, Dehlinger J, Deng L. *Using recurrent neural networks for classification of natural language-based non-functional requirements*. In: Proceedings of the REFSQ Workshops

*[80]* Li G, Zheng C, Li M, Wang H. *Automatic requirements classification based on graph attention network*. IEEE, doi: 10.1109/ACCESS.2022.3159238

*[81]* Kaur K, Kaur P. *SABDM: a self-attention based bidirectional-RNN deep model for requirements classification*. In: J Softw Evol Process

*[82]* Slankas J, Williams L. (2013) Automated extraction of non-functional requirements in available documentation. In: *Proceedings of the 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, doi:10.1109/NAturaLiSE.2013.6611715

*[83]* Ferrari A, Spagnolo GO, Gnesi S. (2017) PURE: a dataset of public requirements documents. In: Proceedings of the 25th IEEE International Requirements Engineering Conference (RE), doi:10.1109/RE.2017.29.

*[84]* Cleland-Huang, J., Mazrouee, S., Liguo, H., Port, D.: NFR (2007). https://doi.org/10.5281/zenodo.268542

*[85] Software Engineering — Software Quality Assurance (SQA),* 2024, *GeeksforGeeks*, Available: https://www.geeksforgeeks.org/software-testing/software-engineering-software-quality-assurance/

*[86]* IBM. (2025). *What is Reinforcement Learning?* IBM Think. https://www.ibm.com/think/topics/reinforcement-learning

*[87] SARSA (Reinforcement Learning), GeeksforGeeks*, 2025, https://www.geeksforgeeks.org/machine-learning/sarsa-reinforcement-learning/

*[88] What is a Recurrent Neural Network (RNN)?,* 2025, IBM Think, https://www.ibm.com/think/topics/recurrent-neural-networks

*[89] Optimization algorithm*, Ultralytics, 2025. https://www.ultralytics.com/glossary/optimization-algorithm

*[90]* Alomari R., Elazhary H., *Implementation of a formal software requirements ambiguity prevention tool.* Int. J. Adv. Comput. Sci. Appl. 2018, doi: 10.14569/IJACSA.2018.090854

*[91]* Tonella, P., Susi, A., & Palma, F. (2013). *Interactive requirements prioritization using a genetic algorithm,* Information and Software Technology, doi: https://doi.org/10.1016/j.infsof.2012.05.010

*[92]* S. L. Lim and A. Finkelstein, StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation, *IEEE Transactions on Software Engineering*, 2012. https://doi.org/10.1109/TSE.2011.39

*[93]* GeeksforGeeks, 2024, *Intuition of SpanBERT,* https://www.geeksforgeeks.org/machine-learning/intuition-of-spanbert/

*[94]* Sallam Abualhaija, Davide Fucci, Fabiano Dalpiaz, Xavier Franch, and Alessio Ferrari, 2020*, ReqEval: The shared task on anaphora ambiguity detection and disambiguation,* https://github.com/frieden84/nlp4re-reqeval

*[95]* Laurent D, 2025, *What is context engineering? A guide for AI & LLMs,* https://intuitionlabs.ai/articles/what-is-context-engineering

*[96]* Olamendy J C, 2023, *Ensemble learning: Combining models for improved performance*, Medium, https://medium.com/@juanc.olamendy/ensemble-learning-combining-models-for-improved-performance-6be8297bbe08

*[97]* IBM, 2023, *What is ensemble learning?* IBM Think, https://www.ibm.com/think/topics/ensemble-learning

*[98]* Kravari K., Antoniou C., Bassiliades N.: *SENSE: a flow-down semantics-based requirements* engineering *framework*, 2021, doi: https://doi.org/10. 3390/a1410 0298

*[99]* Ahmad S., Anuar U. Emran N.A: *A tool-based boilerplate technique to improve SRS quality: an Evaluation,* 2018, *Journal of Telecommunication, Electronic and Computer Engineering, 10*, 111-114.

*[100]* C. Antoniou and N. Bassiliades, *A tool for requirements engineering using ontologies and boilerplates*, *Automated Software Engineering*, doi: 10.1007/s10515-023-00403-y

*[101]* LeewayHertz, 2022, *Ensemble Models Explained*, available: https://www.leewayhertz.com/ensemble-model/

*[102]* Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv, doi: https://arxiv.org/abs/1412.6980

*[103]* Zhang M. R., Lucas J., Hinton G., & Ba, J., 2019, *Lookahead optimizer: k steps forward, 1 step back*. arXiv, doi: https://arxiv.org/abs/1907.08610

*[104]* Abad Z. S. H., Karras O., Ghazi P., Glinz M., Ruhe G., & Schneider K., 2017. What works better? A study of classifying requirements. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 496–501.

*[105]* Yang H., de Roeck A., Gervasi V., Willis A., & Nuseibeh B., 2011, Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering, 16*(3).