Review

Artificial Intelligence Techniques for Requirements Engineering: A Comprehensive Literature Review

António M. Rosado da Cruz * and Estrela Ferreira Cruz *

Posted Date: 31 March 2025

doi: 10.20944/preprints202503.2259.v1

Keywords: Artificial Intelligence; Machine Learning; Natural Language Processing; Requirements Engineering; Software Engineering

Review

Artificial Intelligence Techniques for Requirements Engineering: A Comprehensive Literature Review

António M. Rosado da Cruz [1,*] and Estrela Ferreira Cruz [1,2,*]

[1] ADiT-Lab, Polytechnic University of Viana do Castelo, 4900-347, Viana do Castelo, Portugal
[2] Algoritmi Research Centre, Escola de Engenharia, Universidade do Minho, 4800-058, Guimarães, Portugal
[*] Correspondence: miguel.cruz@estg.ipvc.pt (A.M.R.d.C.) and estrela.cruz@estg.ipvc.pt (F.F.C.)

Abstract: Software requirements engineering is one of the most critical and time-consuming phases of the software development process. The lack of communication with stakeholders and the use of natural language for communicating leads to misunderstanding and misidentification of requirements or the creation of ambiguous requirements, which can jeopardize all subsequent steps in the software development process and can compromise the quality of the final software product. Natural Language Processing is an old area of research, however, it is currently undergoing strong and very positive impacts with recent advances in the area of Machine Learning (ML), namely with the emergence of Deep Learning and, more recently, with the so-called transformer models such as Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-trained Transformer (GPT). Software requirements engineering is also being strongly affected by the entire evolution of ML and other areas of Artificial Intelligence (AI). In this article we make a systematic review on how AI, ML and Natural Language Processing (NLP) are being used in the various stages of requirements engineering, including requirements elicitation, specification, classification, prioritization, requirements management, requirements traceability, etc. Furthermore, we identify which algorithms are most used in each of these stages.

Keywords: Artificial Intelligence; Machine Learning; Natural Language Processing; Requirements Engineering; Software Engineering

---

1. Introduction

Requirements engineering is one of the most critical phases of the Software development life-cycle (SDLC), or software development process, since an error in this phase can generate rework, additional costs and even compromise the success of the entire project. However, requirements engineering faces many challenges such as the difficulty in obtaining the correct and unambiguous information from stakeholders, the complexity in identifying all the actors involved, the software features they may access, domain relevant entities, the impact of a requirement's change, among many others. This difficulties arise from the ambiguity of natural language and misalignment of thought.

After collecting the requirements information, it is necessary to document, in an organized manner, all the information collected, that is, the Software Requirement Specification (SRS) must be created. SRS is a document that serves as a working basis for the rest of the software development process, including testing [1]. Ambiguous or incomplete descriptions of requirements can be problematic for the whole software project or solution.

Requirements can be classified into Functional Requirements (FR) and Non-Functional Requirements (NFR). FR describe what the system should do, that is, its functionalities and expected behaviors. NFR specify how the system should operate, encompassing aspects such as performance, scalability, security and usability.

Requirements Engineering (RE) tasks, being an important part of the software engineering discipline, have long been "promoted" to their own discipline. Requirements engineering deals with the elicitation, analysis, formalization, specification, documentation and validation of requirements, among other activities, in any software or system project. In this context, a system will always involve software, but may also have non-software components, such as sensors and actuators, in addition, of course, to different types of human users.

AI, despite having a long history, has undergone significant developments in recent times, especially in its machine learning aspect. AI can be defined as the use of technologies to create machines that are capable of imitating cognitive functions associated with human intelligence, such as the ability to see, understand and respond to spoken or written language, analyze data, make recommendations and much more. AI typically uses a set of technologies implemented to enable them to reason, learn, and act together to solve complex problems [2]. ML is the area of AI that allows a machine, or a system, to learn based on experience. To do this, ML uses algorithms to analyze large volumes of data, learn and make decisions based on that analysis.

The increasing number and diversity of algorithms used by ML has driven the evolution of this area. These new algorithms, especially Deep Learning algorithms, have also allowed for significant progress in the treatment of natural language. The study of NLP also has a long history. NLP has been studied since the 1940s, for example in automatic translation between languages. The evolution of NLP has followed the evolution of AI paradigms, symbolic, statistical, neural networks, etc. More recently, algorithms called Transformers, such as BERT or GPT, which are Deep Learning algorithms with the ability to process large volumes of data and capture complex relationships, have enabled even more surprising advances in the field of NLP [3].

Requirements collection, analysis and processing are closely related to NLP, so it is natural that the evolution felt in the area of NLP is also felt in the area of RE. However, requirements engineering involves other tasks, such as requirements classification and prioritization, etc., to which more traditional ML algorithms, such as classification and regression algorithms, can also contribute positively.

In this article we will carry out a comprehensive literature review of the ML strategies used in RE. We leverage all the ML algorithms currently used in each of the stages of the requirements engineering process. After summarizing the main requirements engineering activities and presenting some concepts and the evolution of ML algorithms, this article reviews the AI techniques used for RE. The analysis developed seeks to identify the techniques described in scientific literature in the last five years and the first months of 2024, between 2019 and 2024, to support Requirements Engineering tasks or solve problems that occur during these tasks.

1.1. Requirements Engineering Tasks and Issues

This subsection provides a summary of the main tasks of requirements engineering, and problems that exist when these tasks are developed by humans in large projects, or with a large number of requirements, and problems that arise when we try to automate these tasks, even partially.

Any systems project begins with a necessity from its users, typically a business need [4]. A systems project team needs to identify, contextualize and understand this necessity, before being able to formalize, prioritize and document requirements. This process involves some requirements engineering activities and tasks [4–6]:

- Inception: This first activity involves, basically, the identification of a necessity, which will trigger a new project to develop a system capable of provisioning the necessity.
- Requirements Elicitation: Involves identifying the sources of requirements, and gathering requirements using various available techniques, such as interviews, observation of the environment and work processes that the system will support, etc.
- Requirements Elaboration: Entails the analysis of the previously gathered requirements, their contextualization in the problem domain, and the identification of ambiguous, contradictory, or meaningless requirements. It also involves the classification of requirements into Functional and non-Functional, as well as in the latter case their classification into an NFR category.

- Requirements Negotiation: In this phase, candidate requirements, resulting from the previous activities, are negotiated, regulating divergences and adopting prioritization techniques [4].
- Requirements Documentation: Requirements documents, namely SRS documents, serve as the main reference for the subsequent software engineering phases. These documents must display a set of requirements with a formalized structure, and the respective quality and verifiability criteria. At this stage, requirements are typically organized according to two perspectives: user requirements, which describe users' needs; and, system requirements, which describe how the system should behave in different situations [4]. Both these perspectives may include FR and NFR.
- Requirements Validation: This activity includes examining the documented requirements and evaluate if they describe the system desired by the client. This may involve technical inspections and reviews and its main goal is to prevent defects in requirements from propagating to the following phases of the SDLC. Errors detected in this phase have much lower costs than errors detected in subsequent phases [4].
- Requirements Management: This activity runs throughout the whole system development process. Its main goal is to manage requirements and their changes. Requirements traceability is a tool for keeping track of requirements aspects. For example, it may be used for tracing requirements change (each requirement is linked back to its previous version), requirements dependability (each requirement is linked to the requirements on which it depends), system features and requirements (each feature is linked to a set of logically related requirements).

In Figure 1, the types and dimensions of requirements are illustrated, leveraging FR and NFR.

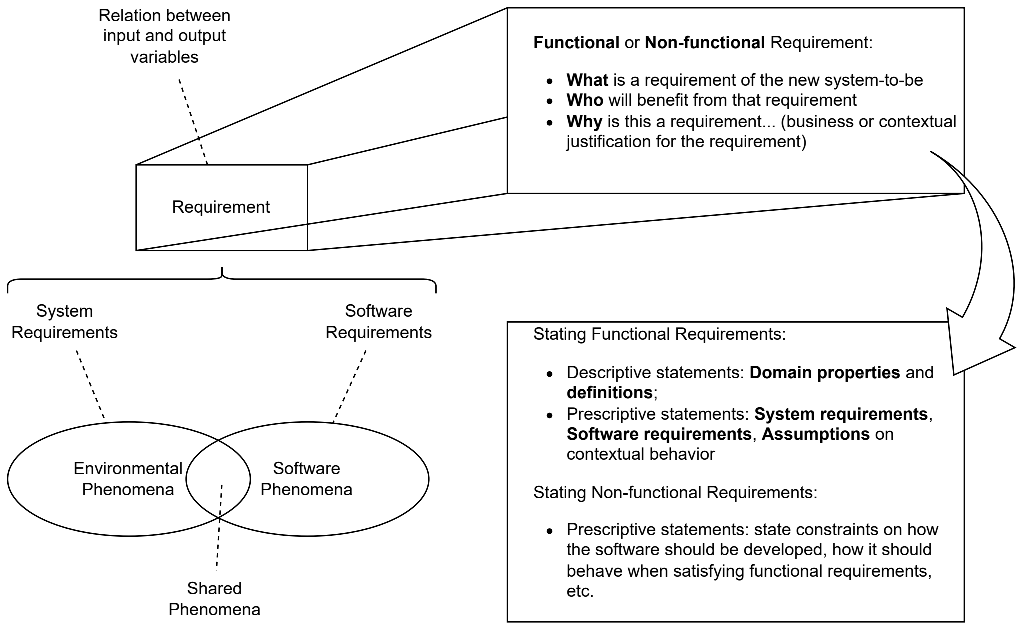

Relation between input and output variables
Functional or **Non-functional** Requirement:
What is a requirement of the new system-to-be
Who will benefit from that requirement
Why is this a requirement... (business or contextual justification for the requirement)
Requirement
System Requirements
Software Requirements
Environmental Phenomena
Software Phenomena
Shared Phenomena
Stating Functional Requirements:
Descriptive statements: **Domain properties** and **definitions**;
Prescriptive statements: **System requirements**, **Software requirements**, **Assumptions** on contextual behavior
Stating Non-functional Requirements:
Prescriptive statements: state constraints on how the software should be developed, how it should behave when satisfying functional requirements, etc.

Figure 1. Types and dimensions of Requirements (the figure aggregates and summarizes information taken from [4,6]) and reflects the authors' viewpoint.

1.2. Main Aim and Research Questions

This article's main contributions include: the identification of RE activities that can most benefit from the use of AI techniques; the identification of AI approaches to help in better concluding RE activities; determining which techniques are most used for each RE activity or group of activities; and, the identification of which AI techniques yield the best results in achieving each RE activity or group of activities.

The goal of this research is to answer the following research questions:

RQ1 Which Requirements Engineering activities take most advantage of the use of Artificial Intelligence techniques?

RQ2 Which Artificial Intelligence techniques are most used in each Requirements Engineering activity?

RQ3 Which Artificial Intelligence techniques have the best results in each Requirements Engineering activity?

1.3. Structure of the article

Te rest of this article is organized as follows. The next section presents the methodology used in the literature review. Section 3 summarizes the machine learning techniques, organizing them into different categories and methods. In section 4 a synthesis of previous literature reviews in this area is made. The results obtained in this work's literature review are detailed in Section 5. Section 6 presents a discussion on the results obtained, and conclusions are presented in Section 7.

2. Materials and Methods

The state-of-the-art literature of the last five years on AI approaches to RE tasks and issues is reviewed in this article. For answering the previously defined research questions, a search query has been defined and cast on Scopus and on Web of Science (WoS) on May 31st, 2024, having our research work been conducted between May and October 2024.

The research methodology used for this study follows the protocol defined in [7]. The search strategy (databases, search query, criteria for inclusion or exclusion) and the full screening method used for the literature review is depicted in Figure 2.

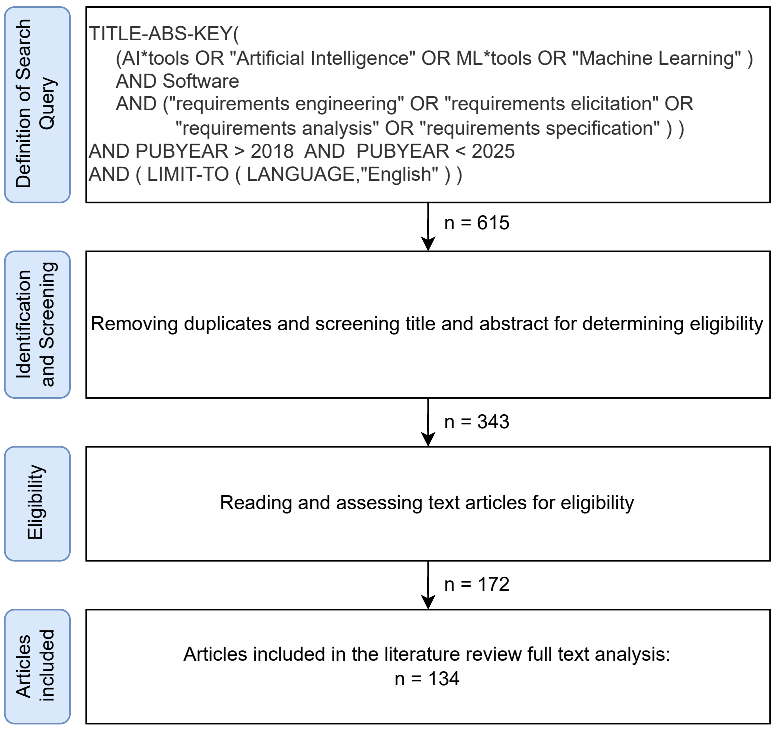

Definition of Search Query
TITLE-ABS-KEY(
(AI*tools OR "Artificial Intelligence" OR ML*tools OR "Machine Learning")
AND Software
AND ("requirements engineering" OR "requirements elicitation" OR "requirements analysis" OR "requirements specification"))
AND PUBYEAR > 2018 AND PUBYEAR < 2025
AND (LIMIT-TO (LANGUAGE,"English"))
n = 615
Identification and Screening
Removing duplicates and screening title and abstract for determining eligibility
n = 343
Eligibility
Reading and assessing text articles for eligibility
n = 172
Articles included
Articles included in the literature review full text analysis:
n = 134

Figure 2. Prisma Flow diagram for the methodoly used in this survey.

The search query has been defined to find indexed journal articles and conference papers, published between 2019 and 2024, on the usage of AI-based techniques and technologies to help with activities of the requirements engineering process. After removing duplicates, the articles were further selected by screening through title and abstract. In this phase, articles on software engineering techniques for AI applications have been removed, as the goal is to address AI techniques for software engineering focusing on requirements engineering. In this phase, previous surveys and literature reviews have also been removed. Further analysis has allowed to eliminate some more articles, which weren't also aligned with the defined goal. Finally 134 articles have been selected for a more thorough analysis.

3. Machine Learning Techniques

In this section, the main ML concepts and algorithms are summarized.

For decades, and especially in recent years, AI and its subfields of ML and NLP have received major investments worldwide, which has driven its rapid evolution. This evolution is reflected in the daily lives of each person and in almost all areas of business, in industry, commerce, finance, etc. [8]. Software development, including the areas of requirements engineering, is also positively influenced by this evolution.

Machine Learning is a branch of AI that allows computers to detect patterns in data and, based on that, make decisions to solve problems for which they were not explicitly programmed [9]. To do this, it is necessary to use complex algorithms that can be grouped into four broad categories, according to the way they learn from data [8–10]:

- Supervised Learning - The model learns from labeled data, where input features are already associated with the correct outputs. These types of algorithms can be used in predictions such as price prediction, correct/incorrect classification, medical diagnosis. Within this group we can further separate the algorithms between classification algorithms (used to classify yes/no) and regression algorithms (used to predict continuous values, such as prices or temperatures). In this group we have algorithms such as Linear Regression, Logistic Regression, Decision Tree (DT), Random Forest (RF), Support Vector Machines (SVM), Multilayer Perceptron (MLP), Artificial Neural Networks (ANN)s, K-Nearest Neighbors (KNN), Gradient Boosting (XGBoost, LightGBM, CatBoost), etc.
- Unsupervised Learning - The model learns without labeled data, identifying hidden patterns in the data. These algorithms can be used for clustering, anomaly detection, etc. In this group there are algorithms such as K-Means, which groups data into K clusters; Hierarchical Clustering, that creates a hierarchical structure of clusters; or, DBSCAN, which identifies dense groups of points, useful for unstructured data.
- Reinforcement Learning - in this group, algorithms learn through trial and error, receiving rewards or punishments. This type of algorithms are used in games, robotics, optimization of financial strategies. This group includes algorithms such as Q-Learning, a table-based algorithm for finding the best action, Deep Q-Networks (DQN), which uses neural networks for deep learning; Proximal Policy Optimization (PPO), an advanced algorithm used by OpenAI, etc.
- Deep Learning - Algorithms that use artificial neural networks with multiple layers to learn complex representations of data. These algorithms are especially used in image recognition, natural language processing, speech and audio processing [10]. In this group there are algoritms such as ANN, Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Pre-trained Language Models (PLM) algorithms, such as BERT and GPT, also belong to this group.

ML algorithms can be grouped into different categories depending on how they learn, the type of architecture they use, or the type of problem they solve. Figure 3 groups the most common ML algorithms based on the type of learning. The most commonly used algorithms in NLP are represented in bold letters in the figure.

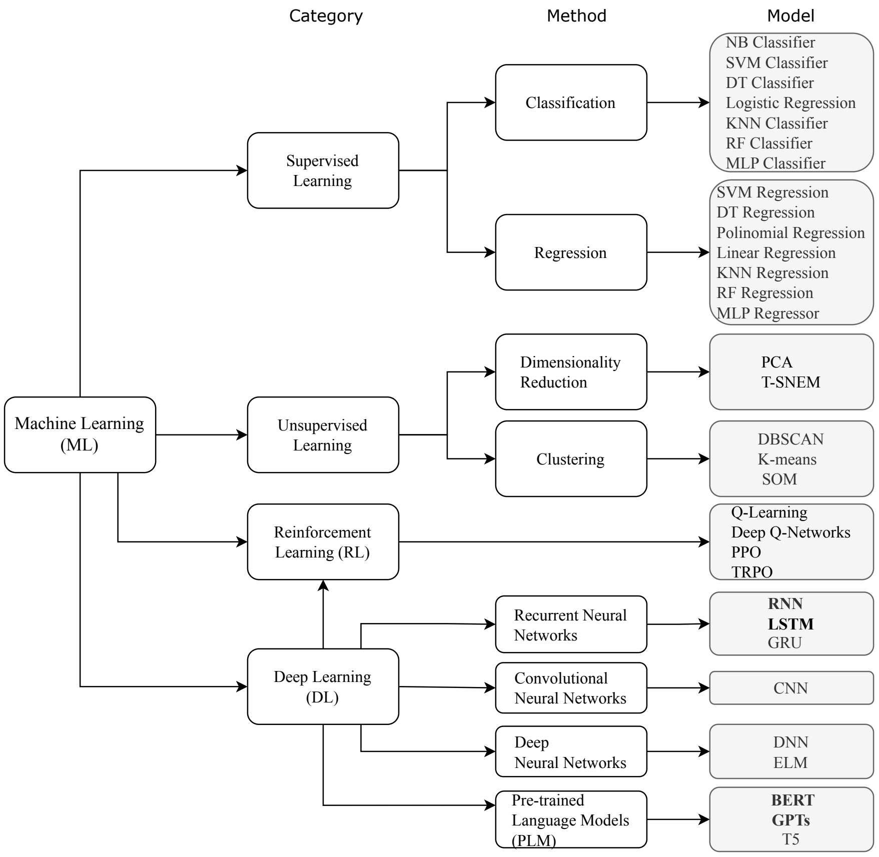


Figure 3. Summary of Machine Learning Algorithms (adapted from [9]).

The strong advances in ML are having a very positive impact in NLP, having revolutionized the area. Natural language is complex, subject to many rules and, at the same time, can be ambiguous, context-dependent, and involving inferences and intentions, such as irony, sarcasm and metaphors. Furthermore, natural language itself is constantly changing and evolving. This makes processing human language a difficult task.

NLP has a long history. Since 1980, researchers have been working to automate requirements engineering tasks using NLP techniques [11]. NLP uses algorithms to analyze, understand, and generate human language [11].

NLP includes tasks like document classification, paraphrase identification, text similarity identification, summarization, translation, etc. [10]. To do that, most of the NLP models involve steps like tokenization, where the text is broken down into words, and representation, where these words are represented in the form of vectors or n-grams (used to analyze sequences of n words to understand the context). Deep Learning (DL) algorithms are those that have been most successful in performing these tasks, which is why they are the most used in NLP [10]. DL has paved the way for the emergence of PLM. PLM such as BERT and GPT have given NLP a huge boost [3].

Thus, NLP may use models such as LSTM, for processing long texts, machine translation, Transformers, which are models that allow parallelization and deeper contextual understanding, among other models. The transformer model is a natural language processing model proposed by Google in 2017, which can include algorithms such as BERT, GPT, Text-To-Text Transfer Transformer (T5), and others [12].

NLP also makes use of more traditional algorithms such as Naive Bayes (NB), for Text Classification; SVM, for Sentiment Analysis and Text Categorization; Random Forest and DTs, for Classification and entity extraction; among others.

Most solutions do not just use one algorithm, but use several algorithms that complement each other and, together, present a better solution.

4. Previous Literature Reviews

There are several publications of literature reviews that address the topic of the use of AI in requirements engineering [13,14], and thus have appeared in the search results of this research work. Some reviews are more comprehensive and study the use of AI in the various stages of the software development process, from the modeling and design phase to the testing phase [15–18]. Others are more specific and study the use of AI in one of the requirements engineering stages, such as prioritization [19,20], classification [21–23], specification [1], or requirements traceability [24]. Others attempt to resolve the problem of ambiguity in identification and specification of requirements [25]. Others try to solve common problems for all projects and use AI to generate requirements that have to be verified in a large majority of projects, such as the case of [26], which deals with requirements that ensure compliance with the General Data Protection Regulation (GDPR).

Prioritizing requirements is an important task, especially in large projects with many requirements, where the work can be tedious for humans. So in [19] the authors have systematically reviewed articles that deal with the use of AI tools and algorithms in prioritizing requirements. They also survey the state of the art on approaches that use ML to prioritize requirements and identify the advantages of ML over other AI techniques.

The article [24] reviews articles that use ML techniques for requirements traceability. More precisely the authors carry out a study on which ML technologies are most frequently used in requirements analysis and traceability and which datasets are most frequently used. The authors conclude that different algorithms are used at different stages of the analysis and traceability process. The requirements traceability process was divided into three main stages: pre-processing stage, link generation stage, and link refinement stage. In the pre-processing stage the algorithms most frequently used are Word2vec and doc2vec. To predict new connections and implement requirements traceability, link generation stage, the most frequently used algorithms are algorithms such as Random Forest, DT and NB. In terms of the dataset used, the authors conclude that about 50% do not identify the dataset and that, among those who identify the dataset, open-source datasets are more frequently used than closed-source datasets.

In [15], the authors analyze articles published between 2015 and 2021 that use ML techniques in any of the stages of the software development process: from requirements elicitation, through design, implementation, testing and maintenance. They conclude that ML algorithms are useful in all stages of the software development process. For each phase, there are algorithms that best adapt (or are more suitable). Regarding the requirements elicitation phase, the authors conclude that the most used algorithms were Supervised ML algorithms, such as SVM, NB, DT, KNN, or RF.

In [27], the authors conducted a systematic review with the aim of identifying the main existing problems in requirements engineering, as well as which techniques have been used to solve those problems. The authors conclude that one of the main problems is the lack of communication between stakeholders accompanied by insufficient knowledge about the problem to be solved. Thus, requirements are often classified as ambiguous, incomplete, inconsistent, or incorrect. Prioritizing requirements and difficulty in maintaining documentation are also some of the problems identified. The authors identify artificial intelligence as a very promising area to help solve known requirements engineering problems.

Many authors propose tools and methods for generating software models based on information contained in the requirements specification, usually specified in Natural Language (NL) [16]. The authors, in [28], conducted a systematic literature review on approaches to transform natural language text into Unified Modeling Language (UML) diagrams. There are approaches to generating practically all types of UML diagrams: class diagram, use case diagram, object diagram, sequence diagram, activity diagram and others. Several solutions are heuristic-rule-based and, according to the authors, it appears to be a solution with strong potential for future work in this area.

The ability to extract model elements from requirements, increases the responsibility of requirements engineering considering that some of the following steps in the SDLC can be automated. The authors in [16] analyze proposals for works and tools that generate software models in UML based on software requirements, namely class models. The authors conclude that the approaches studied are complex and have many limitations, being most of them able to identify classes, some of them redundant, but they are not able to identify the relationships between the classes.

In [22], the authors survey the state-of-the-art articles that use ANN to classify software requirements. They conclude that the most used algorithms in requirements classification are NB, SVM and ANN, and PROMISE was the most popular database for those studies. Some of the studies classify functional requirements, but most of them focus on non-functional requirements, namely security and usability requirements.

NLP is receiving significant attention and yielding considerable technological advances. Software requirements are typically documented in natural language, which can raise ambiguity problems. Requirements ambiguity is one of the main problems in requirements specification, as it can derail the following steps in software development. Therefore, NLP-based tools are also being used in RE to disambiguate requirements. In [25] the authors study and compare automated and semi-automated disambiguation tools. They conclude that the studied tools cannot completely eliminate ambiguities, but some show promising results.

In [13] the authors conduct a study on ML techniques and approaches for automating RE activities. The authors conclude that automating RE analysis tasks helps in reducing the cost and time for carrying out these activities. The study presented in [13] concludes that there are no defined standards or guidelines for selecting the most appropriate ML and NLP techniques, and that most approaches combine several ML techniques to achieve better results. The same applies to the selection of the dataset to use. There is no consensus on criteria for deciding which dataset to use.

In [17] the authors carry out a more comprehensive study on the use of AI techniques in the different phases of the SDLC, to identify the ML techniques that are most used in each of the phases, from requirements gathering, design, implementation and testing.

The work in [20] provides a systematic review of the ML algorithms used in the prioritization of requirements. The study attempts to identify the ML algorithms that are most efficient in classifying and prioritizing requirements. The authors conclude that the most used algorithms are SVM, DT, KNN, NB, Linear Regression (LR) and Multinomial Naive Bayes (MNB), in this order [20].

The study presented in [29] is more detailed and focuses on the representations used in NLP as input for ML techniques. The way requirements are represented influences the way NLP algorithms behave and can be decisive for their performance and success. The authors presented a survey of the state of the art on the representations used in the various stages of RE, and conclude that this area has undergone enormous changes in recent years with advanced embedding representations that have greatly improved the effectiveness of tasks such as requirements analysis and requirements extraction. However, more traditional representations based on lexical and syntactic features are still widely used in tasks such as modeling and quality tasks at the syntax level [29].

In [14] the authors present a state-of-the-art analysis of articles published between 2015 and 2021 on how AI is used in RE. The authors conclude that the application of NLP techniques and supervised learning in the requirements documentation stages, more specifically in the elicitation, specification and validation stages, is a growing trend [14].

In [18] the study is more generic and focuses general trends in AI techniques at all stages of software engineering. The authors conclude that RE and testing phases are those that present the most research proposals using AI techniques, but it is also very promising in other stages, such as software design [18].

Classifying requirements into their different categories can be a very tedious, time-consuming and error-prone phase. In [23] the authors review the literature on the application of ML techniques in the classification of software requirements. The results indicate that the most commonly used classification algorithms are: NB, DT and NLP.

The systematic review presented in [26] focuses on the GDPR and how NLP can be used to automate the process of identifying requirements to ensure GDPR compliance. The GDPR was created by the European Union (EU) with the aim of protecting the personal data and privacy of European citizens. Therefore, all software operating in the EU must take into account a set of requirements to achieve GDPR compliance. Several possibilities for using NLP and NLP-based ML techniques to perform RE tasks, and thus achieve GDPR compliance, have been identified.

The study presented in [30] systematically reviews the literature on tools and techniques used in requirements validation. The authors grouped validation techniques into six categories: prototyping, inspection, knowledge-oriented, test-oriented, modeling and evaluation, and formal models, and conclude that knowledge-oriented techniques such as ML methods are the most frequently mentioned. The authors also conclude that it is necessary to standardize the quality characteristics of requirements and that the most frequently mentioned quality characteristics were correctness, completeness, consistency, and ambiguity. Other characteristics also mentioned are understandability, reusability, unexpected dependencies, variability, and testability [30].

In [31] the authors study the use of NLP techniques in Crowd-Based Requirements Engineering (CrowdRE), i.e., in the analysis of online user feedback about software products. The authors conclude that ML is frequently used in CrowdRE and that the most widely used ML algorithms are NB followed by SVM.

In [32] the authors conduct a study to identify and classify the type of ML algorithms used to identify software requirements on Stack Overflow. The authors conclude that Latent Dirichlet Allocation (LDA) associated with Bag of Words (BoW) are the most widely used ML algorithms. The authors also conclude that ML algorithms still face some problems in identifying requirements.

5. Results

For the purpose of this study, the surveyed literature has been categorized in the following five categories of RE tasks:

- Classification of Requirements according to their Functional/Non-Functional nature
- Supporting Requirements Elicitation
- Improving the Quality of Requirements and Software
- Extracting Knowledge from Requirements
- Supporting Requirements Management and Validation, and Project Management

This section presents the results of the developed literature review, and is divided into two subsections. In subsection 5.1, the five categories of RE's tasks are further explained. In subsection 5.2, references that address each of the task categories are examined. The aim is to later, in section 6, be able to identify common ML approaches and techniques used in the development of the task in question.

5.1. RE Categories of Tasks

The five identified RE categories of tasks, used to categorize the surveyed literature, are further explained in the following subsections.

5.1.1. Classification of Requirements according to their Functional/Non-Functional nature

A requirement expresses a user need or some constraint imposed on a system. According to the IEEE 610.12-1990 standard, a requirement is a condition or capability that must be verified or possessed by a system to satisfy a contract, standard, or specification; a documented representation of a condition or capacity, within the scope of the previous point [4].

Each requirement must be written in a form that:

- is clear, unambiguous and easy to interpret;
- expresses objective intentions and not subjective opinions.

Requirements may be divided into Functional (FR) and Non-functional (NFR) requirements. FR express functionalities that the system should exhibit to its users, whilst NFR impose restrictions to the system as a whole.

NFR may be further classified into several subcategories, such as [4]:

- Appearance, which is about the visual aspect of the system's graphical user interface;
- Usability or User Experience (XP), which has to do with the system's easiness of use and the friendliness of the user experience;
- Performance, related to characteristics of speed, storage capacity, ability to scale to greater numbers of simultaneous users, among other aspects;
- Security, having to do with authentication and authorization access to the system and to the data, data protection and integrity, etc.;
- Legal, namely standards, laws and rules that apply to the system or to its domain of application.

Many other NFR categories may be considered, and these are easily found in any Software Engineering book, such as [4,5].

5.1.2. Supporting Requirements Elicitation

Requirements elicitation involves identifying the sources for requirements and the actual gathering of the requirements, to form a set of candidate requirements. This can be done by recurring to a range of requirements elicitation techniques. This range includes techniques such as interviews, focus groups, surveys, introspection, observation of workers while doing their work, to understand the way they work and where and how the system can improve it [4,5].

5.1.3. Improving the Quality of Requirements and Software

Elicited requirements are considered as candidate requirements because they lack further analysis, elaboration, negotiation and acceptance by the stakeholders. For this, a requirements engineer must further elaborate the requirements to ensure that there are no [4,5]:

- contradictory requirements;
- ambiguous requirements;
- incoherent or senseless requirements;
- complex requirements, or requirements that need to be further divided into several requirements.

After elaboration and negotiation, requirements accepted by the stakeholders form the body of the SRS document.

5.1.4. Extracting Knowledge from Requirements

After establishing a set of accepted stable requirements, the system analysis modeling follows. This RE activity yields a set of models, each representing a different perspective of the system being conceptualized and, together, comprising a technology free model of the system. These different model perspectives are based on the requirements in the SRS. Examples of knowledge that may be extracted from the requirements are, among others, the following:

- Rewriting requirements in a standard form;
- System features;
- Types of system users;
- System Structural Entities;
- Dependency between Requirements;
- Related requirements, enabling requirements traceability.

5.1.5. Supporting Requirements Management and Validation, and Project Management

Requirements management is an activity that spans the entire SDLC and includes tasks such as assessing the impact of changing requirements.

Requirements validation helps ensuring that the established requirements define the system desired by the client [4].

5.2. ML Techniques used in RE Tasks

As mentioned before, the reviewed literature has been categorized into the five RE task categories presented in the previous subsection. The specific references found in the reviewed literature, for each category, have been organized in tabular form and are depicted in Table 1. Some references may appear in more than one category, in the cases they address several RE tasks.

In this section the main ML techniques used in the reviewed literature, for each of the RE categories of tasks, are analyzed.

5.2.1. Classification of Requirements according to their Functional/Non-Functional nature

Several ML techniques may be used for classifying Requirements, from natural language text in SRS documents, according to their Functional/Non-Functional nature. In Table 1, we have separated references that address a binary classification of each requirement as a "Functional Requirement" or a "Non-Functional Requirement", or a ternary classification, where non-requirements are also identified, from the ones that look for a more detailed classification of NFR into their subcategories, or also try to identify Architecturally Significant Functional Requirements (ASFR), which are requirements with important information for taking architectural decisions.

Classifying requirements into functional, non-functional and other categories involves several steps, from preprocessing and feature extraction to model training and evaluation. At its early stage, NLP is the main issue. For this, techniques such as tokenization and lemmatization may be used to preprocess the requirements text and transform it to numerical feature values. Once the requirements text is preprocessed and transformed into numerical feature vectors, a classification algorithm can be applied.

In Table 2, the main approaches for the early NLP phase, and the main ML-based approaches to the classification of Requirements according to their Functional/Non-Functional nature, are presented. As depicted in the table, several techniques are reported in the reviewed literature. In this section, the techniques used in each reference reviewed, which are present in Table 2, are summarized.

The authors in [33] studied several ML approaches to distinguish between FR and NFR. Their approach included data cleansing, normalization and text preprocessing and vectorization steps, in which BoW, Term frequency-inverse document frequency (TF-IDF), Featurization and Machine Learning Models, ROC and AUC curves, Bi-Grams and n-Grams in Python, Word2Vec, and confusion matrix were used. According to the authors, the combination of BoW and MNB provided the best performance for binary classification.

The authors in [34] argue that existing techniques for classifying FR and NFR consider only one feature at a time, thus not being able to consider the correlation of two features, and so they are biased. In their study, they compare and extend ML algorithms to classify requirements, in terms of precision and accuracy, and have observed that DT algorithm can identify different requirements and outperform existing ML algorithms. As the number of features increases, the accuracy using the DT is improved by 1.65%. To address DT's limitations, they propose a multiple correlation coefficient based DT algorithm. This approach, when compared to existing ML approaches, improves classification performance. The accuracy of the proposed algorithm is improved by 5.49% compared to the DT algorithm.

Table 1. Categorization of the reviewed literature, according to the Requirements Engineering Phase and Activity addressed.

RE Category	RE Activity	References
Classification of Requirements according to their Functional/ Non-Functional nature	Binary FR/NFR Classification, and ternary Classif. FR/NFR/Non-Req	[33–58]
	Classification of NFR, in further subcategories, and Classification of ASFR	[36,39–41,43,45,47–49,52,57,59–61]
Supporting Requirements Elicitation	Categorize and Classify Business Rules (as a source for RE)	[62]
	Predicting/recommending Techniques for Req. elicitation	[63]
	Generating questions for Requirements elicitation	[56,64]
	Identifying/Generating new Requirements from existing Requirements or from User Feedback; Identification of ASFR; Classification of User Feedback	[35,51,55,65–79]
Improving Quality of Requirements and Software	Requirements itemization, simplification, disambiguation; Detecting incompleteness, ambiguity, inaccuracy, implicit requirements, semantic similarity, and other risks (smells) in Requirements Specification; Coping with ambiguity through the use of controlled vocabulary and Ontologies	[35,58,80–93]
	Verification of Quality Characteristics in NFR (e.g., usability, UX, security, explainability) and user feedback; Ensuring security of Requirements; GDPR; Assessing the SRS quality by ISO 25010; Test case generation / Automate the quality checking/analysis of a Req./user story	[94–100]
	Identify potential effects on Sustainability; Assess Transparency and Sustainability as NFR	[101–103]
	Verification of pre-/post-conditions of Requirements; Requirements to Test Cases Traceability; Predicting probability of defects using design-level attributes	[96,104–110]
	Classification of requirements in two classes: “Integration Test” and “Software Test” using Machine Learning approaches	[111]
Extracting Knowledge from Requirements	Requirements Formalization; Extracting/Associating Features (Feature Extraction) or Model Elements from/to Requirements; Extract domain vocabulary from requirements for Feature Modeling or ontology construction	[43,58,81,83,112–134]
	Detecting or Extracting Requirements Dependencies / Requirements Traceability (forward and backward)	[105,120,121,135–143]
Supporting Requirements Management and Validation, and Project Management	Requirements Prioritization	[77,144–149]
	Requirements allocation to software versions (considering as criteria: requirement development time, priority levels, and dependency relationships); Predicting if a software feature will be completed in its planned iteration	[150–152]
	Project Management Risks Assessment / Req. Change Impact analysis on other requirements and on planned test cases	[153–155]

Table 2. Main ML-based approaches to the classification of Requirements according to their Functional/Non-Functional nature.

RE Category	RE Activity	Main approaches used for NLP and feature extraction from NL Text, and for dataset preparation, and Main ML approaches used for achieving the RE activity

Classification of Requirements according to their Functional/ Non-Functional nature	Binary FR/NFR Classification, and ternary Classif. FR/NFR/Non-Req	TF; BoW; TF-IDF; Word2Vec; FastText; Doc2Vec; SMOTE; PCA; RST; DT; SVM; KNN; NB; MNB; NN; RF;
		K-means; Hierarchical Clustering; SVC; Bagged KNN; Bagged DT; Bagged NB; ExtraTree; GNB; SGD;
		GB; XGBoost; AdaBoost;
		ANN; RNN; LSTM; Bi-LSTM; MLP; CNN;
		BERT; PRCBERT; NoRBERT; MLM-BERT; GPT;
		BoW + MNB; TF-IDF + SVM; Doc2Vec + MLP + CNN; BoW + SVM; BoW + KNN; TF-IDF + SGD;
		Multiple correlation coefficient-based DT; Bi-LSTM-Att (Bi-LSTM + Attention Model);
		Ensemble Grid Search classifier using 5 models (RF, MNB, GB, XGBoost, AdaBoost);
		Self-attention Bidirectional-RNN Deep Model (SABDM);
		Bidirectional Gated Recurrent Neural Networks (BiGRU);
		Bidirectional Encoder-Decoder Transformer Convolutional Neural Network (BERT-CNN);
		Trans_PRCBERT (PRCBERT fine-tuned on PROMISE); TPOT;
		Ensemble classifier combining 5 models (NB, SVM, DT, LR, SVC).

	Classification of NFR, in further subcategories, and Classification of ASFR	

In [36], the authors have used a zero-shot learning (ZSL) approach for classifying requirements into Functional and Non-Functional requirements, and to identify NFR categories, including security non-functional requirements, without using any labeled training data. The study shows that the ZSL approach achieves an F1 score of 0.66 for the FR/NFR classification task. For the NFR task, the approach yields a F1-score between 0.72 and 0.80, considering the most frequent classes.

In [37], TF-IDF and Word2Vec were the feature extraction techniques used after the NL text pre-processing phase. The study then compared different ML algorithms to assess their precision and accuracy in classifying software requirements, namely DT, RF, LR, Neural Networks (NN), KNN and SVM. The results showed that the TF-IDF feature selection algorithm performed better than the Word2Vec algorithm, in subsequent classification algorithms.

The study in [38] implemented an ensemble technique using Grid Search classifier that can automatically tune the best parameters of the low performed classifier. The objective was to use a fine-tuned Ensemble technique combining five different models, namely RF, MNB, Gradient Boosting, XGBoost, and AdaBoost, to classify software requirements into FR or NFR.

The study in [40] used a RNN based model, namely Bidirectional Long Short-Term Memory (Bi-LSTM). This algorithm combines the forward and backward hidden layers to solve the sequential task better than LSTM. By combining Bi-LSTM and self-attention mechanism, the authors noticed an improved requirements classification accuracy. The Bi-LSTM model has been trained with the GloVe model. The architecture proposed in [40], named Self-attention based Bidirectional-RNN Deep Model (SABDM), integrates NLP, Bi-LSTM, and self-attention mechanism, and has been developed for improving the performance of deep learning in classifying requirements, both FR/NFR categorizations and within NFR categories.

Most studies deal with requirements classification as binary or multiclass classification problems and not as multilabel classification, which would allow a requirement to belong to multiple classes at the same time. As a way of minimizing preprocessing and to enable multilabel classification of requirements, in [41] a recurrent neural networks-based deep learning system has been used, namely Bidirectional Gated Recurrent Neural Networks (BiGRU). The authors have investigated the usage of word sequences and character sequences as tokens. Using word sequences as tokens has achieved results similar to the state-of-the-art, effectively classifying requirements into functional and different non-functional categories with minimal text prepossessing and no feature engineering.

The authors in [43] propose an automated non-exclusive approach for classification of functional requirements from the SRS, using a deep learning framework. They found that domain-specific terms, used in requirements specification, cause a number of issues with requirements engineering methods. An NLP pipeline is proposed, for categorizing functional requirements from the SRS into several types.

They have used MLP and CNN in the classification model's development, after using Word2Vec and FastText word embeddings from SRS documentation. Along with the word vectors inferred from the pre-trained Word2Vec and FastText word vectorizers, the Doc2Vec model was used to vectorize the sentences in the used SRS documents. The word vectors were produced using pre-trained online embedding and re-training the current embedding model using internal data. The impact of data trained with Word2Vec and FastText was compared to pre-trained word embeddings models, available online. The retrained vector classifier models outperformed an initial vector model in terms of accuracy. The best accuracy was achieved by the retrained vector CNN classifier model (77%).

In [44], the authors have used Term Frequency, BoW and TF-IDF, together with four supervised and two unsupervised machine learning algorithms for classifying requirements specifications into FR and NFR. When using BoW, the authors observed an accuracy of 0.725 with K-Nearest Neighbors (K-NN), 0.835 with Support vector machines (SVM), 0.849 with Logistic Regression (LR), 0.543 with K-means, 0.839 with multi-naive bayes, and 0.560 with Hierarchical clustering. Accuracy achieved with agglomerative clustering using TF-IDF was 0.797 with K-NN, 0.876 with SVM, 0.845 with LR, 0.470 with K-means, 0.856 with Multinominal Naive bayes. The authors conclude that, for better results, it is best to combine SVM algorithm with TF-IDF. The authors also conclude that ML algorithms are suitable for classifying requirements on simple problems, but that for addressing larger problems it is necessary to apply rules-based AI models.

The research reported in [45] presents a Bidirectional Encoder-Decoder Transformer-Convolutional Neural Network (BERT-CNN) model for requirements classification. The convolutional layer is stacked over the BERT layer for performance enhancement. In order to extract features from requirement statements the study employs CNN in task-specific layers of BERT. Experiments using the PROMISE dataset evaluated the solution's performance through multi-class classification of four key classes: Operability, Performance, Security, and Usability. Results showed that the BERT-CNN model outperformed the standard BERT approach when compared to existing baseline methods.

The studies in [46,53] use five distinct word embedding techniques for classifying FR and NFR (quality) requirements. Synthetic Minority Oversampling technique (SMOTE) is used to balance classes in the dataset used. Some dimensionality reduction techniques are also used, namely Principal Component Analysis (PCA), which is used for reducing dimension, and Rank-Sum test (RST), which is used for feature selection, to eliminate redundant and irrelevant features. Then, the vectors resulting from the word embedding techniques used have been provided as inputs to eight different classifiers for requirements categorization: Bagged k-Nearest Neighbors, Bagged Decision Tree, Bagged Naive-Bayes, Random Forest, Extra Tree, Adaptive Boost, Gradient Boosting, and a Majority Voting ensemble classifier, with DT, KNN, and Gaussian Naive Bayes (GNB). The authors conclude that the combination of word embedding and feature selection techniques with the various classifiers are successful in accurately classifying functional and quality software requirements [46,53].

In [47], the use of PRCBERT, or Prompt learning for Requirement Classification using BERT, is proposed. This approach applies flexible prompt templates to classify software requirements from small-sized requirements datasets (PROMISE and NFR-Review), and then adopts it to auto-label unseen requirements' categories of their collected large-scale requirement dataset NFR-SO. Experiments conducted on PROMISE and NFR-Review datasets and on a large-scale requirement dataset collected by the authors, enables to conclude that PRCBERT exhibits moderately better classification performance than NoRBERT and MLM-BERT (BERT with the standard prompt template). On the de-labeled datasets, Trans_PRCBERT (a PRCBERT version fine-tuned on PROMISE) has a zero-shot performance with 53.27% and F1-score of 72.96%, when enabling a self-learning strategy.

In [48], the authors propose applying ML and active learning (AL) to classify requirements in a given dataset, introducing the MARE process, which utilizes Naïve Bayes as the classifier. AL employs uncertainty sampling strategies to determine which data points should be labeled by the "oracle". Three AL strategies are explored: Least Confident (LC), Margin Sampling (MS), and Entropy Measure (EM). Experiments using two datasets were conducted to evaluate the performance of the MARE process. The findings suggest that better organization and documentation of requirements improve classification results. However, significant progress is still needed to develop a system capable of categorizing requirements with minimal human intervention at different levels of abstraction.

The work in [49] presents a proposal for the automated classification of quality requirements. The study involved the training and hyperparameter optimization of different ML models, with the user feedback classification. The study leverages the inherent knowledge of software requirements to train various ML algorithms using NLP techniques for information reuse, extraction, and representation. The Tree-based Pipeline Optimization Tool (TPOT), an AutoML library developed by Olson et al. [156], which uses genetic algorithms, was employed to optimize ML models, improving fitness scores by up to 14%. TPOT achieved the highest weighted geometric mean (0.8363), followed by Random Forest (0.82). However, applying these models to informal text requirements proved challenging, as automated classifiers struggled to achieve results above 0.3, highlighting the gap between machine and human classification performance.

The authors in [50] propose a technique to automatically classify software requirements using ML to represent text data from software requirements text and classify them as FR or NFR, based on BoW followed by SVM or KNN algorithm for classification. They experimented with the PROMISE_exp dataset, which includes labeled requirements, and observed that the use of BoW with SVM is better than to use KNN algorithms with an average F-measure of all cases of 0.74.

The study in [51] looks for the automatic categorization of user feedback reviews into functional requirements and non-requirements. The study evaluates ML based models to identify and classify requirements from both formally written SRS documents and free text App Reviews written by users. Similarly to other approaches, the work uses ML algorithms (SVM, SGD, and RF) to identify and classify requirements, combined with NLP techniques, namely TF-IDF, to pre-process the requirements text.

In [52], an analysis of supervised ML models combined with NLP techniques is proposed to classify FRs and NFRs from large SRS. Experiments were conducted on the PROMISE dataset, in two phases: first, the focus was on distinguishing between FRs and NFRs; then, the aim was to classify NFRs into nine specific subcategories. The results show that SVM with TF-IDF achieved the best performance for FR classification, while SGD with TF-IDF was most effective for NFR classification. For subclassifying NFRs, SVM with TF-IDF yielded the best results for Availability, Look & Feel, Maintainability, Operational, and Scalability. Meanwhile, SGD with TF-IDF performed best for Security, Legal, and Usability, whereas RF with TF-IDF excelled in classifying Performance-related NFRs.

In [54], a new ensemble ML technique is proposed, combining different ML models and using enhanced accuracy as a weight in the weighted ensemble voting approach. The five combined models were NB, SVM, DT, LR, and Support Vector Classification (SVC). When using the ML based classifiers with the highest accuracies (SVM, SVC, and LR) these yielded the same accuracy of 99.45% with the proposed ensemble, only the time improved when using a smaller number of classifiers.

In [55], the authors propose Requirements-Collector, a tool for automating the identification and classification of FRs from requirements specification and user feedback analysis. The Requirements-Collector approach involves ML and DL computational mechanisms. These components are intended to extract and pre-process text data from datasets of previous works, containing requirements, and then classify FR and NFR requirements. Preliminary results have shown that the proposed tool is able to classify RE specifications and user review feedback with reliable accuracy.

The work in [60], proposes an approach for classifying ASFRs, which are FR that contain comprehensive information to aid architectural decisions, and thus have a significant impact on the system's architecture. ASFRs are hard to detect, and if missed, can result in expensive refactoring efforts in later stages of software development. The work presents experiments with a deep learning-based model for identifying and classifying ASFRs. The approach (Bi-LSTM-Att) applies a Bi-LSTM to capture the context information for each word in the software requirements text, followed by an Attention model to aggregate useful information from these words in order to get the final classification. For ASFR identification, the Bi-LSTM-Att model yielded an f-score of 0.86, and for ASFR classification an f-score of 0.83, on average [60]. The authors also noted that Bi-LSTM-Att outperformed the baseline RAkEL NB classifier for all the labels, with industrial size datasets, although RAkEL NB seems to perform well on less data.

In [56], the authors propose an intelligent chatbot for keeping a conversation with stakeholders in NL and automating the requirements elicitation and classification yielding formal system requirements from the interaction. Afterwards, a classifier classifies the elicited requirements into FR and NFR. The collected requirements are written in unstructured free flow English sentences, which are pre-processed to identify requirements, through the use of NLP and Dialogue Management, Rasa-NLU and Rasa-Core opensource frameworks. After requirements elicitation by the Chatbot, two classifiers have been implemented, MNB and SVM, to categorize the elicited requirements into FR and NFR. The results show that MNB has better Accuracy, Precision, Recall and F1-Score than SVM (0.91 vs 0.88 in all performance indicators) [56].

The authors in [61] research the application of two types of neural network models, an ANN and a CNN, to classify NFRs into five categories: maintainability, operability, performance, security and usability. The authors have evaluated their work on two widely used datasets with approximately 1,000 NFRs. The results show that the implemented CNN model can classify NFR categories with a precision ranging between 82% and 94%, a recall indicator between 76% and 97%, and an F-score between 82% and 92%.

In [57], RF and gradient boosting algorithms are explored and compared, to determine their accuracy in classifying functional and non-functional requirements. RF and gradient boosting are ensemble algorithms in ML. These combine the results from multiple base or weak learners to produce a final prediction, enabling to improve accuracy and other indicators of prediction performance. Experimental results show that the gradient boosting algorithm has improved prediction performance better than random forest, when classifying NFR. However, the random forest algorithm is more accurate in classifying FR.

In [58], the efficacy of ChatGPT in several aspects of software development is assessed. For requirements analysis, the ChatGPT's proficiency in identifying ambiguities, distinguishing between FR and NFR, and generating use case specifications, has been evaluated. The assessment, which has been qualitative and subjected to the authors' opinion, revealed that ChatGPT has potential in assisting various activities throughout the SDLC, including requirements analysis, domain modeling, design modeling, and implementation. The study also identified non-trivial limitations, such as a lack of traceability and inconsistencies among produced artifacts, which require human involvement. Overall, the results suggest that, when combined with human developers to mitigate the limitations, ChatGPT can serve as a valuable tool in software development [58].

5.2.2. Supporting Requirements Elicitation

Requirements Elicitation is a Requirements Engineering Phase, or set of activities, that deals with capturing, identifying and registering requirements. It helps to derive and extract information from stakeholders or other sources. It is an essential phase in building commercial software.

Table 3, presents the main approaches for the NLP phase, along with the main ML-based approaches for supporting Requirements Elicitation. The table illustrates the techniques reported in the reviewed literature. Each literature reference reviewed, for the "Supporting Requirements Elicitation" category, is summarized in this section.

Table 3. Main ML-based approaches to Supporting Requirements Elicitation.

RE Category	RE Activity	Main approaches used for NLP and feature extraction from NL Text, and for dataset preparation, and Main ML approaches used for achieving the RE activity

Supporting Requirements Elicitation	Categorize and Classify Business Rules (as a source for RE)	TF; BoW; TF-IDF; Word2Vec; FastText; Doc2Vec; Rasa-NLU; Rasa-Core;
		MNB; SVM; Multi-dataset training; zero-shot approaches;
		Having a perturbator and a classifier positively influencing each other;
		Multimodal Autoencoder and Multimodal Variational Autoencoder methods;
		Supervised ML models;
		BERT; GPT; DL; NN; Transformer-based DL models;
		Neural Language Models; Hierarchical cluster labeling;
		Speech-acts based analysis technique; Part-of-Speech (POS) Tagging;
		NLP + ML techniques leveraging the concept of requirement boilerplate;
		Trained LSTM RNN model (based on Rasa) + MNB or SVM;
		NLP4ReF; NLP4ReF-NLTK; NLP4ReF-GPT;
		BERT-based transformer + static preference linker.

	Predicting/recommending Techniques for Req. Elicitation	
	Generating questions for Requirements elicitation	
	Identifying/Generating new Requirements from existing Requirements or from User Feedback; —-Identification of ASFR; Classification of User Feedback	

Business rules, which give body to the description of the business processes, can be an important source of software requirements specifications. The authors in [62] propose an approach to categorize and classify business rules based on Witt's approach, which classifies business rules into four main categories: definitional (or structural) rules, data rules, activity rules, and party rules [62]. They conclude that the proposed approach showed good accuracy, recall, and F1-scores values, when compared to the state-of-art approaches [62].

The elicited requirements list resulting from the requirements elicitation phase is used as input for requirements analysis and management activities. Multiple elicitation techniques may be applied alternatively or in conjunction with other techniques to accomplish the elicitation. The prediction or recommendation of the best technique for requirements elicitation influences the requirements engineering approach. The authors in [63] analyze the current practices of requirements elicitation techniques application in practical software development projects, and define factors influencing the technique selection based on the two-classification ML model, and predict the usage of a particular elicitation technique depending on the project attributes and business analyst background. They conducted a survey study involving 328 specialists from Ukrainian Information Technology (IT) companies. Gathered data was used to build and evaluate the prediction models.

According to the authors in [64], integrating advanced models like GPT-3.5 into RE remains largely unexplored. With the goal of exploring the capabilities and limitations of GPT-3.5 in software requirements engineering, the research presented in [64] investigates the effectiveness of GPT-3.5 in automating key tasks within RE. The authors identify the limitations of using GPT-3.5 in the requirement-gathering process and conclude that GPT-3.5 demonstrates proficiency in aspects like creative prototyping and question generation, but has limitations in areas like domain understanding and context awareness. The authors offer recommendations for future research focusing on the seamless integration of GPT-3.5 and similar models into the broader framework of software requirements engineering [64].

As previously mentioned in [56], an approach to automate requirements elicitation and classification is proposed. The idea is to use an intelligent conversational chatbot. The chatbot converses with stakeholders in Natural Language and elicits formal system requirements from the interaction, and then a classifier classifies the elicited requirements into FR and NFR. Rasa-NLU and Rasa-Core opensource frameworks are used in the chatbot for natural language processing. For the dialogue management, a Rasa Core model is trained with a training data file consisting of several sample user and bot conversations, where the user response is represented by its intent, and the bot response is represented by the bot's utterance and actions taken [56]. The result is a trained LSTM RNN model capable of interpreting dialogue history and converting raw dialogue data into a probability distribution over system actions. These actions are defined either as the bot's textual responses to the user or as code that identifies system requirements from the user's input, extracts relevant entities, requests additional information if needed, writes the requirement to file, and maintains a natural language conversation with the user. The elicited system requirements are subsequently classified into FR and NFR categories using a text classification model trained on over eight hundred labeled samples from multiple domains. The input data is represented as feature vectors using the BoW method and TF-IDF frequency, which are then employed by text classification models developed using MNB and SVM algorithms [56]. The authors note that the chatbot has been trained to capture a limited set of requirements within a single domain and requires further extensive training data to recognize a complete set of system requirements. In what respects the MNB and SVM classifiers, the authors conclude that the first has better performance (Accuracy, Precision, Recall, and F1-Score) than the second in classifying FR and NFR [56].

Addressing requirements defects during the RE phase is more cost-effective than during development of after project delivery. In [35], the use of Natural Language Processing for Requirements Forecasting (NLP4ReF) is introduced. The authors' goal is to reduce missing and incorrectly expressed requirements, in order to minimize the number of requirement changes during the SDLC, and ensuring that requirements accurately reflect stakeholder needs. The NLP4ReF approach enables enhancing the process of requirements elaboration using ML and NLP, including the initial requirements organization, their classification as FR or NFR, the identification of system classes, and the generation of forgotten or unforeseen requirements [35]. The paper explores using NLP4ReF algorithms, namely NLP4ReF-NLTK and NLP4ReF-GPT, in the RE process, to evaluate their efficacy in requirements generation and classification, and to analyze their practical application. The algorithms were able to generate many new relevant requirements, and to effectively classify requirements into FR/NFR. The research highlights the importance of Model-Based Systems Engineering (MBSE) in guiding the development and optimization of algorithms, providing a logical framework for future research in the field of Natural Language Processing for Requirements Engineering (NLP4RE). The authors conclude that the systematic integration of MBSE, through the incorporation of various diagrams that underpin the development of the algorithms, provides comprehensive insights into the study. MBSE contributed to a deeper understanding of the capabilities and implications of NLP4ReF and NLP4RE tools and techniques in RE.

Besides addressing requirements classification between FR/NFR, as seen in section 5.2.1, the authors in [51] also seek to automate the process of extracting functional requirements and filtering out non-requirements from user app reviews. Their proposal evaluates ML-based models to identify and classify software requirements from both, formal Software SRS documents and Mobile App User Reviews. Initial evaluation of the ML-based models show that they can help classify user app reviews and software requirements as FR, NFR, or Non-Requirements.

The research in [55], already mentioned before when addressing requirements classification, also intends to automatically identify, extract and pre-process text containing requirements and user feedback, to generate requirements specification. The proposed Requirements-Collector tool uses ML and DL based approaches to automatically classify requirements discussed in RE meetings (stored in the form of audio recordings) and textual feedback in the form of user reviews. The authors argue that the Requirements-Collector tool has the potential to renovate the role of software analysts, which can experience a substantial reduction of manual tasks, more efficient communication, dedication to more analytical tasks, and assurance of software quality from conception phases [55].

Developers frequently elicit requirements from user feedback, such as bug reports and feature requests, to help guide the maintenance and evolution of their products [65]. By linking feedback to their existing documentation, development teams enhance their understanding of known issues, and direct their users to known solutions. The authors in [65] apply deep-learning techniques to automatically match forum posts with related issue tracker entries, using an innovative clustering technique. Strong links between product forums, issue trackers, and product documentation, have been observed, forming a requirements ecosystem that can be enhanced with state-of-the-art techniques to support users and help developers elicit and document the most critical requirements [65].

Studies to elicit stakeholder preferences have been developed in [66], using scenarios where users describe their goals for using directory services to find entities of interest, such as apartments, hiking trails, etc. The article's results reveal that feature support for preferences varies widely among directory services, with around 50% of identified preferences unmet. The study also explored automatic preference extraction from scenarios using named entity recognition across three approaches, with a BERT-based transformer achieving the best results (81.1% precision, 84.4% recall, and 82.6% F1-score on unseen domains). Additionally, a static preference linker was introduced, linking extracted entities into preference phrases with 90.1% accuracy. This pipeline enables developers to use the BERT model and linker to identify stakeholder preferences, which can then inform improvements and new features to better address gaps in service.

In [67], ML classifiers are used to classify bug reports and feature requests using seven datasets from previous studies. The authors evaluate classifiers' performance on users' feedback from unseen apps and entirely different datasets, and they assess the impact of channel-specific metadata. They find that using metadata as features in classifying bug reports and feature requests rarely improves performance, and while classification is similar for seen and unseen apps, classifiers struggle with unseen datasets. Multi-dataset training or zero-shot approaches can somewhat alleviate this issue, with implications on user feedback classification models for extracting software requirements.

In [68], the authors propose an approach to creatively generate requirements candidates via the adversarial examples resulted from applying perturbations to the original requirements descriptions. In the presented architecture, the perturbator and the classifier positively influence each other. Each adversarial example is uniquely traceable to an existing feature of the software, instrumenting explainability. The experimental evaluation has used six datasets, and shows that around 20% adversarial shift rate is achievable [68].

Several works investigate which techniques and ML models are most appropriate for detecting relevant users feedback and reviews, and for classifying, embedding, clustering, and characterizing those reviews for generating requirements across multiple feedback platforms and data domains [69–74,76].

The study in [69] explores unimodal and multimodal representations across various labeling levels, domains and languages to detect relevant app reviews using limited labeled data. It introduces a one-class multimodal learning method requiring labeling only relevant reviews, thus reducing the labeling effort. To enhance feature extraction and review representation with fewer labels, the authors propose the Multimodal Autoencoder and the Multimodal Variational Autoencoder methods, which learn representations that combine textual and visual information based on reviews' density. Density information can be interpreted as a summary of the main topics or clusters extracted from the reviews [69]. The studied methods achieved competitive results using just 25% of labeled reviews compared to models trained on complete datasets, with multimodal approaches reaching the highest F1-Score and AUC-ROC in twenty-three out of twenty-four scenarios.

In [70], the authors investigate whether enterprise software vendors can elicit requirements from their sponsored developer communities through data-driven techniques. The authors collected data from the SAP Community and developed a supervised machine learning classifier for automatically detecting feature requests of third-party developers. Based on a manually labeled data set of 1,500 questions, the proposed classifier reached a high accuracy of 0.819. Their findings reveal that supervised machine learning models may be an effective means for the identification of feature requests.

In [71], the authors propose using the state-of-the-art transformer-based DL models for automatically classifying sentences in a discussion thread. The authors propose a benchmark to ensure standardized inputs for training and testing for this problem. They conclude that their transformer-based classification proposal significantly outperforms the state-of-the-art [71].

The approach presented in [73] proposes a hierarchical cluster labeling method for software requirements that leverages contextual word embeddings. This method addresses previous issues such as duplicate requirements from user reviews and the challenges of handling different granularity levels that obscure hierarchical relationships between software requirements. The authors use neural language models to create semantically rich representations of software requirements, clustering them into groups and subgroups based on similarity in the embedding space. Representative requirements are then selected to label each cluster and sub-cluster, effectively managing duplicate entries and different granularity levels [73].

Within the RE process of defining, documenting, and maintaining software requirements, the authors in [74] focus on the problem of automatic classification of CrowdRE into sectors. CrowdRE involves large scale user participation in RE tasks. The authors proposal involves three different approaches for sector classification of CrowdRE, based on supervised ML models, NN, and BERT, respectively. Classification approaches have been applied to a CrowdRE dataset, comprising around 3000 crowd-generated requirements for smart home applications. The obtained performance is similar to several other classification algorithms, indicating that the proposed algorithms can be very useful for categorizing crowd-based requirements into sectors [74].

Although initial progress has been made in using mining techniques for requirements elicitation, it remains unclear how to extract requirements for new apps based on similar existing solutions and how practitioners would specifically benefit from such an approach.

In [76], the authors focus on exploring information provided by the crowd about existing solutions to identify key features of applications in a particular domain. The discovered features and other related influential aspects (e.g. ratings) can help practitioners to identify potential key features for new applications [76]. The authors present an early conceptual solution to discuss the feasibility of their approach.

User reviews from tweets, app forums, etc. are processed by applying NL techniques to filter out irrelevant data, followed by text mining and ML algorithms to classify them into categories like bug reports and feature requests. The research in [77] explores a linguistic technique based on speech-acts for the analysis of online discussions with the goal of discovering requirements-relevant information. A revised version of the speech-acts based analysis technique, previously presented by the same authors, is proposed together with a detailed experimental characterization of its properties. Datasets used in the experimental evaluation were taken from an open source software project (161120 textual comments) and from an industrial project in the home energy management domain. On these datasets, the proposed approach is able to successfully classify messages into Feature/Enhancement and Other, with F-score of 0.81 and 0.84 respectively. The authors conclude that evidence of an association between types of speech-acts and categories of issues, has been found, and that there is correlation between some of the speech-acts and issue priority.

To advance software creativity, several techniques have been proposed, such as multi-day workshops with experienced requirements analysts and semi-automated tools that support focused creative thinking. The authors in [75,78] propose a novel framework for providing an end-to-end automation to support creativity in both new and existing systems. The framework reuses requirements from similar software freely available online, uses advanced NLP and ML techniques, and leverages the concept of requirement boilerplate to generate candidate creative requirements. The framework has been applied on three application domains: Antivirus, Web Browser, and File Sharing, and further report a human subject evaluation. The results exhibit the framework's ability to generate creative features even for a relatively matured application domain, such as Web Browser, and provoke creative thinking among developers irrespective of their experience levels.

Software companies need to quickly fix reported bugs and release requested new features, or they risk negative reviews and reduced market share. The sheer volume of online user feedback renders manual analysis impractical. The authors in [79] note that online product forums are a rich source of user feedback that may be used to elicit product requirements. The information contained in these forums often include detailed context to specific problems that users encounter with a software product. By analyzing two large forums, the study in [79] identifies 18 distinct types of information (classifications) relevant to maintenance and evolution tasks. The authors found that a state-of-the-art App Store tool cannot accurately classify forum data, underlining the need for specialized techniques to extract requirements from product forums. In an exploratory study, they developed classifiers incorporating forum-specific features, achieving promising results across all classifiers, with f-scores ranging from 70.3% to 89.8%.

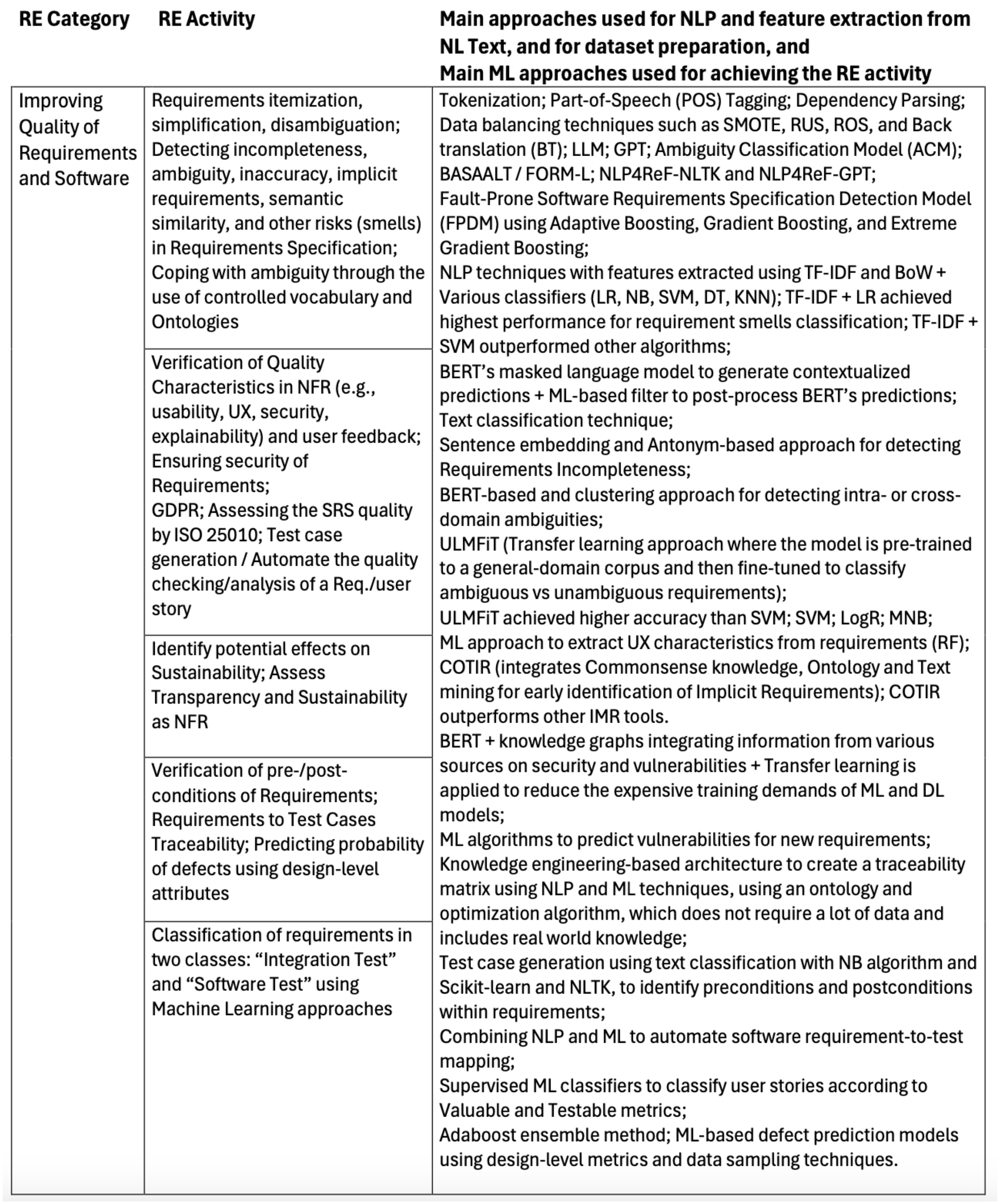
5.2.3. Improving Quality of Requirements and Software

Most software requirements are written using natural language, which has no formal semantics and has a high risk of being misunderstood due to its natural tendency towards ambiguity and vagueness. Improving the quality of requirements and software involves reducing the ambiguity, incompleteness, non-uniqueness and coverage (in terms of users' needs) of the requirements specification. It also involves, identifying requirements that may have effects of sustainability, security and usability of the future system, besides other quality characteristics. Another way of improving the requirements quality is registering and monitoring their inter-dependencies and pre- and post-conditions.

The quality of the software, and the adherence of planned or developed software features to the stated requirements, may also be addressed through validation tests. These may be, at least partially, drawn from the SRS. And, the probability of defects can also be predicted.

Table 4, presents the main approaches for the NLP phase, along with the main ML-based approaches for improving the quality of requirements and software. The table shown the main techniques reported in the reviewed literature. Each of the references reviewed is summarized in this section.

Table 4. Main ML-based approaches to Improving the Quality of Requirements and Software.



RE Category	RE Activity	Main approaches used for NLP and feature extraction from NL Text, and for dataset preparation, and Main ML approaches used for achieving the RE activity
Improving Quality of Requirements and Software	Requirements itemization, simplification, disambiguation; Detecting incompleteness, ambiguity, inaccuracy, implicit requirements, semantic similarity, and other risks (smells) in Requirements Specification; Coping with ambiguity through the use of controlled vocabulary and Ontologies	Tokenization; Part-of-Speech (POS) Tagging; Dependency Parsing; Data balancing techniques such as SMOTE, RUS, ROS, and Back translation (BT); LLM; GPT; Ambiguity Classification Model (ACM); BASAALT / FORM-L; NLP4ReF-NLTK and NLP4ReF-GPT; Fault-Prone Software Requirements Specification Detection Model (FPDM) using Adaptive Boosting, Gradient Boosting, and Extreme Gradient Boosting; NLP techniques with features extracted using TF-IDF and BoW + Various classifiers (LR, NB, SVM, DT, KNN); TF-IDF + LR achieved highest performance for requirement smells classification; TF-IDF + SVM outperformed other algorithms; BERT's masked language model to generate contextualized predictions + ML-based filter to post-process BERT's predictions; Text classification technique; Sentence embedding and Antonym-based approach for detecting Requirements Incompleteness; BERT-based and clustering approach for detecting intra- or cross-domain ambiguities; ULMFiT (Transfer learning approach where the model is pre-trained to a general-domain corpus and then fine-tuned to classify ambiguous vs unambiguous requirements); ULMFiT achieved higher accuracy than SVM; SVM; LogR; MNB; ML approach to extract UX characteristics from requirements (RF); COTIR (integrates Commonsense knowledge, Ontology and Text mining for early identification of Implicit Requirements); COTIR outperforms other IMR tools. BERT + knowledge graphs integrating information from various sources on security and vulnerabilities + Transfer learning is applied to reduce the expensive training demands of ML and DL models; ML algorithms to predict vulnerabilities for new requirements; Knowledge engineering-based architecture to create a traceability matrix using NLP and ML techniques, using an ontology and optimization algorithm, which does not require a lot of data and includes real world knowledge; Test case generation using text classification with NB algorithm and Scikit-learn and NLTK, to identify preconditions and postconditions within requirements; Combining NLP and ML to automate software requirement-to-test mapping; Supervised ML classifiers to classify user stories according to Valuable and Testable metrics; Adaboost ensemble method; ML-based defect prediction models using design-level metrics and data sampling techniques.
	Verification of Quality Characteristics in NFR (e.g., usability, UX, security, explainability) and user feedback; Ensuring security of Requirements; GDPR; Assessing the SRS quality by ISO 25010; Test case generation / Automate the quality checking/analysis of a Req./user story	
	Identify potential effects on Sustainability; Assess Transparency and Sustainability as NFR	
	Verification of pre-/post-conditions of Requirements; Requirements to Test Cases Traceability; Predicting probability of defects using design-level attributes	
	Classification of requirements in two classes: "Integration Test" and "Software Test" using Machine Learning approaches	

The work in [58], mentioned above, also targets the quality improvement of requirements and software. The efficacy of ChatGPT on identifying ambiguities and generating accurate use case specifications, and on fixing errors encountered during the software implementation, has been assessed. As mentioned before, the study also identified a lack of traceability and inconsistencies among produced artifacts, which to not dispense human involvement.

One recurrent difficulty in requirements analysis is requirements itemization, or simplifying/subdividing requirements. This difficulty arises due to the inherent ambiguity and redundancy of requirements described in natural language. It is very important to determine the list of itemized requirements from the requirements document. The work in [80] proposes a method to automatically extract requirement entries from requirement text by leveraging a set of NLP techniques and machine learning models. The approach tries to imitate the human expert process of extraction of itemized requirements, which consists of three main processes: identifying requirement locations and boundaries, building models, and extracting fine-grained requirement semantics. The authors performed evaluations in the field of military arguments, and the results showed a nearly 80 percent accuracy. The approach can be used to industry practitioners extract requirements items faster and easier.

In [81], a ML-based approach to formalize requirements written in natural language text is proposed. The approach targets critical embedded systems, and extracts information to assess the quality and complexity of requirements. The authors have used the open source NLP framework spaCy for tokenization, Part-of-Speech (PoS) tagging and dependency parsing based on a pre-trained English language model. Then, a phase for identifying text chunks follows, whuch uses a rule-based exploration approach in contrast to domain-specific training alternative. According to the authors, this is to ensure independence of a specific engineering domain. Text chunks are then put into a normalized order. If this is not possible, a quality issue may be detected. The normalized sequence of chunks can be used for "building test oracles, for the implementation of software, and for applying metrics so that requirements may be compared for similarity or be evaluated" [81].

In [82], the authors propose a model to detect fault-prone software requirements specifications, consisting of two main components: (1) an Ambiguity Classification Module (ACM); and, (2) a Fault-Prone Detection Module (FPDM). The ACM selects the best deep learning algorithm to classify requirements as ambiguous or clean, identifying various types of ambiguity (lexical, syntactic, semantic, and pragmatic). Then, the FPDM uses key SRS components, such as title clarity, description, intended users, and the ambiguity classification, to detect fault-prone requirements. The ACM achieved an accuracy of 0.9907 and the FPDM 0.9750. To further enhance detection, particularly for edge/cloud applications, the authors applied boosting algorithms (Adaptive Boosting, Gradient Boosting, and Extreme Gradient Boosting), improving accuracy by leveraging SRS features. They also propose a fault-prone severity scale that categorizes ambiguity as low, moderate, or high based on a calculated score from key SRS elements [82].

The use of NLP4ReF, proposed in [35], also targets the disambiguation of requirements specifications. It uses ML and NLP to identify duplicate, incomplete and hidden requirements. NLP4ReF-NLTK and NLP4ReF-GPT algorithms are used to classify requirements and generate new relevant requirements.

Software programs require rigorous testing and verification to prevent defects, and the same applies to requirements, whether elicited or assumed. This involves analyzing both form and semantics. The study in [83] highlights that fully automating requirement disambiguation is impossible, as human intervention remains crucial. Using the BASAALT method and FORM-L language, the authors formalized requirements to support behavioral simulation, which helps detect issues such as inadequacy (unsuitable requirements), over-ambition (unnecessary requirements with undue complexity and risks), and contradiction (conflicting requirements). This approach involves creating semantically precise and simulable models while integrating missing contextual details. Stakeholders can review these models and simulation results to ensure alignment with intended requirements. BASAALT/FORM-L models represent system requirements, environmental assumptions, and proposed solutions, with verification ensuring that these solutions satisfy specified requirements. However, defects in requirements and assumptions are a common reason for system failures. Beyond ambiguity, inadequacy is a key concern, as unsuitable requirements can lead to undesirable outcomes. FORM-L models support verification through modeling and simulation, leveraging tools like Stimulus for test case generation. A noted limitation of the study is the manual application of BASAALT and FORM-L for identifying ambiguities and formalizing corrected requirements.

To identify and correct poor quality software requirements, the work in [84] proposes a set of ML models for detecting different kinds of requirement smells and prioritizing them. Requirements smells are characteristics identified in requirements that function as an indicator of ambiguity and vagueness problems in requirements [84]. The authors discovered that previous approaches to detecting requirement smells face scalability and flexibility issues, and they do not prioritize smells for ordered action. The proposed method identifies ten classes of requirement smells and ranks them based on severity and the importance of the requirements. They used a dataset of 3,100 expert-labeled requirements for both classification and prioritization. Textual requirements were preprocessed with NLP techniques, with features extracted using TF-IDF and BoW. Various classifiers, namely LR, NB, SVM, DT, and KNN, were evaluated, and an additional sorting method was applied for project-specific smell prioritization. As a result, LR with TF-IDF achieved highest performance with 94% accuracy for requirement smells classification. For requirement smells prioritization, SVM outperformed other algorithms with 99% accuracy.

Detecting incompleteness in NL requirements is a major challenge. One approach is to compare requirements with external sources. Given the rise of Large Language Model (LLM)s, the work in [85] has addressed the question: Are LLMs useful external sources of knowledge for detecting potential incompleteness in NL requirements? The authors explore this question by using BERT's masked language model to generate contextualized predictions for filling masked slots in requirements. To simulate incompleteness, some content from requirements has been withhold, and BERT's ability to predict terminology that is present in the withheld content but absent in the disclosed content, has been assessed. BERT can produce multiple predictions per mask. The work contributes to determine the optimal number of predictions per mask, balancing effective identification of omissions in requirements with noise reduction. It also contributes with a a ML-based filter to post-process BERT's predictions and further decrease noise. An empirical evaluation on 40 requirements specifications from the PURE dataset shows that BERT's predictions successfully highlight missing terminology, outperforming simpler baselines, while the proposed filter enhances its effectiveness for completeness checking of requirements [85].

The problem with natural language is that it can easily lead to different understandings if it is not expressed precisely by the stakeholders involved [86]. This may result in building a product that is not aligned with the stakeholders expectations.

The work in [86] tries to improve the quality of the software requirements by detecting language errors based on International Standards Organizations (ISO) 29148 requirements language criteria. The proposed solution is based on previous existing solutions, which apply classical NLP approaches to detect requirements' language errors. In [86], the authors seek to improve the previous work by creating a manually labeled dataset and using ensemble learning, DL and other techniques, such as word embeddings and transfer learning to overcome the generalization problem that is tied with classical NLP and improve precision and recall metrics using a manually labeled dataset.

The work in [87] also addresses duality and incompleteness in NL software requirements specification. Different from previous approaches, in [87] the authors focus on the requirements incompleteness implied by the conditional statements, and propose a sentence embedding and antonym-based approach for detecting the requirements incompleteness. The guiding idea is that when one condition is stated, its opposite condition should also be there, or else the requirements specification is incomplete. Hence, the proposed approach starts by extracting the conditional sentences from the requirements specification, and eliciting the conditional statements which contain one or more conditional expressions. Then, conditional statements are clustered using the sentence embedding technique, and the conditional statements in each cluster are further analyzed to detect potential incompleteness, by using negative particles and antonyms [87]. The results of the proposed approach have shown a recall of 68.75%, and a F1-measure of 52.38%.

Ambiguity in Requirement Engineering document may lead to disastrous results thereby hampering the entire development process and ending up compromising on the quality of a system. In [88], the authors discuss the types of ambiguity found in the RE document, and approaches to handle and providing a level of automatic assistance in reducing ambiguity and improving requirements. The study also confirms the use of text classification technique to classify a text as "ambiguous" or

"Unambiguous" at the syntax level. The objectives of the work were mainly to identify the presence of ambiguity in any RE document with the help of ML Techniques and finally minimizing or reducing it.

The application of neural word embeddings for detecting cross-domain ambiguities in software requirements has recently gained significant attention. Several methods in the literature estimate how meaning varies for common terms across domains, but they struggle to detect terms used in different contexts within the same domain, i.e., intra-domain ambiguities or those in a requirements document of an interdisciplinary project. The work in [89] introduces a BERT-based and clustering approach to identify such ambiguities. For each context in which a term appears, the approach provides a list of similar words and example sentences illustrating its context-specific meaning. Applied to both a computer science corpus and a multi-domain dataset covering eight application areas, the approach has proven highly effective in detecting intra-domain ambiguities [89].

In [90], the authors have used transfer learning by using ULMFiT, where the model has been pre-trained to a general-domain corpus and then fine-tuned to classify ambiguous vs unambiguous requirements (target task). Back translation (BT) has also been used as a text augmentation technique, to see if it improved the classification accuracy. The proposed model has then been compared with machine learning classifiers like SVM, Logistic Regression (LogR) and MNB, and the results showed that ULMFiT achieved higher accuracy those classifiers, improving the initial performance by 5.371%. The authors conclude that the proposed approach provides promising insights on how transfer learning and text augmentation can be applied to small data sets in requirements engineering.

The work in [91] addresses identification of implicit requirements (IMRs) in SRS. Implicit requirements are not specified by users, but may be crucial to the success of a software project. A software tool has been developed, called COTIR, which integrates Commonsense knowledge, Ontology and Text mining for early identification of Implicit Requirements. In [91] the authors demonstrate the tool and conclude that it relieves human software engineers from the tedious task of manually identifying IMRs in huge SRS documents. Performed evaluation shows that COTIR outperforms existing IMR tools [91].

Semantic similarity information supports requirements tracing and helps to reveal important requirements quality defects such as redundancies and inconsistencies [92]. The authors in [92] created a large dataset for analyzing the similarity of requirements, through the use of Amazon Mechanical Turk, a crowd-sourcing marketplace for micro-tasks. Based on this dataset, they investigate and compare different types of algorithms for estimating semantic similarities of requirements, covering both relatively simple bag-of-words and machine learning models. After experiments on their dataset, they conclude that the best performances were obtained by a model which relies on averaging trained word and character embeddings as well as an approach based on character sequence occurrences and overlaps, achieve the best performances.

In [93], the authors present techniques of NLP which work out greatly to extract information properly and minimizing the bugs that may generate in later parts of Software Development. Using techniques of NL Interpretation, Software Engineers can outline the most accurate requirements of customers, which can improve the quality of requirements, and ultimately of the resulting software product.

There are also several works in the certification of quality characteristics, such as usability, user experience or security. In [94], a set of system UX Key Performance Indicator (KPI) is predicted based on a list of initial textual requirements. This helps assessing an application's UX KPI in the first phases of software requirements engineering, without having to develop a UX-oriented prototype at early software requirements elicitation phase. User Experience (UX) reveals users' product impressions when using or planning to use a software product. The suggested ML approach extracts UX characteristics from textual requirements to categorize them as UX scales, which are then used to forecast the total UX KPI of any software application. The UX predictions should be as reliable as a real case study of different application prototypes. Several machine learning models have been trained on a benchmark dataset of software requirements, showing a f1-measure performance of 0.91, for the RF Algorithm.

The authors also observed that the UX KPIs calculated based on outputs from ML models were highly interrelated with those calculated based on developed UX-oriented software prototypes, allowing to conclude that the proposed model can evaluate UX instantly without interventions from end-users or UX designers [94].

In [99], a benchmark dataset is built for UX, based on textual software requirements crowdsourcing several UX experts. The paper develops a machine learning model to measure UX based on the dataset. The research describes the dataset characteristics and reports its statistical internal consistency and reliability. Results indicate a high Cronbach Alpha and a low root mean square error of the dataset, which leads the authors to conclude that the new benchmark dataset could be used to estimate UX instantly without the need for subjective UX evaluation. The dataset will serve as a foundation of UX features for machine learning models.

The work in [95] focuses on ensuring system security is addressed in the requirements management phase rather than leaving it for later phases in the software development process. The authors propose an approach to combine useful knowledge sources like customer conversation, industry best practices, and knowledge hidden within the software development processes. BERT is used in the proposed architecture to utilize its language understanding capabilities. The work also investigates the use of knowledge graphs to integrate information from various industry sources on security practices and vulnerabilities, ensuring that the requirements management team stays informed with critical data. Additionally, transfer learning is applied to reduce the expensive training demands of ML and DL models. The proposed architecture has been validated within the financial domain and agile development models. The authors propose that this approach could effectively integrate software requirements management with data science practices by leveraging the extensive information available in the software development ecosystem.

Software security is also a major concern in the work presented in [96]. Based on the principle that the root of a system security vulnerability can often be traced back to the requirements specification, the authors advocate a novel framework to provide an additional measure of predicting vulnerabilities at earlier stages of the SDLC. In the study in [96], the authors build upon their proposed framework and leverage state-of-the-art ML algorithms to predict vulnerabilities for new requirements, together with a case study on a large open-source-software (OSS) system, Firefox, evaluating the effectiveness of the extended prediction module. The results show that the framework could be a viable complement to the traditional vulnerability-fighting approaches.

Complying with the EU GDPR can be a challenge for small and medium-sized enterprises. The work reported in [97] considers GDPR-compliance as a high-level goal in software development that should be addressed at the beginning of software development, that is, during RE. The authors argue that NLP can be used to automate this process, and present initial work, preliminary results, and the current state of art on verifying requirements' GDPR-compliance.

In [98] the authors present a user feedback classifier based on ML for the classification of user reviews according to software quality characteristics compliant with the ISO 25010 standard. The proposed approach has been achieved by testing several ML algorithms, features, and class balancing techniques for classifying user feedback on a data set of 1500 reviews. The maximum F1 and F2 scores obtained were 60% and 73%, with recall as high as 94%. The authors conclude that the proposed approach does not replace human specialists, but helps in reducing the effort required for requirements elicitation.

The study in [100] reports on the development of a method of activity of ontology-based intelligent agent for evaluating initial stages of the software lifecycle. Based on the developed method, the intelligent agent evaluates whether the information in the SRS is sufficient or not. It provides a numerical assessment of the sufficiency level for each non-functional feature individually and for all features overall, along with a list of attributes (measures) or indicators that should be added to improve the SRS's completeness or level of sufficiency [100]. In experiments, the agent analyzed the

SRS for a transport logistics decision support system and determined that the information was not sufficient for assessing the quality by ISO 25010 and for assessing quality by metric analysis.

Software developers are gradually becoming aware that their systems have effects on sustainability [101]. Researchers are currently exploring approaches which strongly make use of expert knowledge to identify potential effects. In the work in progress research reported in [101], the authors looked at the problem from a different angle: they have worked on the exploration of a ML-based approach to identify potential effects. Such an approach allows to save time and costs but increases the risk that potential effects are overseen. First results of applying the ML-based approach in the domain of home automation systems are promising, but also indicate that further research is needed before the proposed approach can be applied in practice.

The growing complexity and ubiquity of software systems increase user reliance on their correct functioning, which in turn demands that these systems and their decision processes become more transparent. To achieve this, transparency requirements must be clearly understood, elicited, and refined into lower-level requirements [102]. However, there is still limited understanding of how the requirements engineering process should address transparency, including the roles and interactions among UX designers, data scientists, and other stakeholders. To address this gap, the work in [102] investigates the requirements engineering process for transparency through empirical studies with practitioners and other stakeholders. Preliminary findings indicate that further research is needed to develop effective solutions that support transparency in requirements engineering.

It is very important to deliver a defect free product that matches all the requirements specified by the client and must pass all the test cases [103]. To do this systematically, a requirement traceability matrix is used. A Requirement traceability matrix relates two items' lists in two dimensions. In this case, it shows the relations of all the requirements and the test cases, thus allowing to track the relation between the customer's requirements for the system and the test cases for validating the requirements. There are many approaches to create an efficient traceability matrix using Language processing and ML techniques but these approaches require a lot of data and do not take care of the real world knowledge and may lead to errors. In [103], the authors propose a knowledge engineering-based architecture to this problem with the use of ontology, machine learning and optimization algorithm which produces a dependability and steadiness of 97% and 95% respectively and the performance of the proposed model is compared with baseline approaches.

The authors in [104] propose an approach for test case generation using text classification, using the NB algorithm to identify preconditions and postconditions within software requirements. The approach categorizes software requirements into two categories: "none" and "both", which indicate the presence or absence of preconditions and postconditions in software requirements. The research employs the NB algorithm, a widely used probabilistic classification algorithm in text classification tasks [104]. It uses two libraries, namely Scikit-learn and Natural Language Toolkit (NLTK). The best accuracy score, which was obtained by the Scikit-learn model, was 0.86, which demonstrates the feasibility of reducing the effort and time required for classifying test case components based on software requirements. The proposed approach not only streamlines the identification of essential components in software requirements but also opens up possibilities for further automation and optimization of the testing process [104].

Accurate mapping of software requirements to tests is critical for ensuring high software reliability [105]. The dynamic nature of software requirements demands that these are traceable and measurable throughout the SDLC, in order to be able to plan software tests and integration tasks, during the development phase, and the evaluation and verification tasks, or the application of patches, during the operation phase. To address these challenges, a novel method is proposed in [105], combining NLP and ML to automate software requirement-to-test mapping. The proposed method formalizes the process of reviewing the recommendations generated by the automated system, enabling engineers to improve software reliability, and reduce cost and development time [105].

In Agile software project management methodologies, user requirements are frequently stated in the form of user stories, the smallest semi-structured specification units of user requirements. In [106], two metrics are applied in validating user stories: Testable and Valuable criteria from INVEST checklist[1]. The authors have applied supervised machine learning classifiers to automatically classify user stories according to those metrics. They have used industrial collected data for their dataset and applied the developed classifiers, having observed good values of accuracy and precision. After balancing the dataset, using data balancing techniques such as SMOTE, RUS, ROS, and Back translation (BT), the authors observed that, despite not seeing significant improvements in accuracy and precision, a significant improvement has been obtained in recall values across all the classifiers [106]. The research provides some promising insights into how the analysis of user stories can be used by the software industry to improve the quality of the software produced.

Providing automatic requirement analysis techniques for modeling and analyzing requirements is a must for saving manpower. In [107], a cloud service method for automated detection of quality requirements in SRS is proposed. The study also presents a novel approach for processing automatic classification of software quality requirements based on supervised machine learning techniques applied for the classification of training document and predict target document software quality requirements.

Approaches aiming to minimize the vulnerabilities in the software have been dominated by static and dynamic code analysis techniques, often using machine learning (ML). These techniques are designed to detect vulnerabilities in the post-implementation stage of the SDLC [96]. Accommodating changes after detecting a vulnerability in the system in later stages of the SDLC is very costly, sometimes even infeasible as it may involve changes in design or architecture. In [96], a framework to provide additional measures of predicting vulnerabilities at earlier stages of the SDLC is proposed.

In [108], a method for automatically generating test cases, for system testing and acceptance testing, from requirements is studied. The authors propose training data selection quality improvement technique in the cosine similarity with the test data, and have confirmed the effectiveness of the methods. A second method has also been proposed that adds the application judgment technique by the standard deviation value. The proposed methods have obtained the maximum value of accuracy with less training data.

Software used in communication systems is increasingly becoming more complex and larger in scale to accommodate various service requirements. Since telecom carrier networks serve as basic social infrastructures, it is important to maintain their reliability and safety as a critical lifeline [109]. The implementation of numerous quality improvement measures, however, has resulted in prolonged development periods and higher costs. To address these issues, the authors in [109] have been working on the automation of software testing, as this process has great influence in software quality. Typically, test cases are written by skilled engineers and are decided after multiple reviews, requiring a large amount of manpower in preparing them. The study in [109] has used the knowhow of skilled engineers in writing test cases as training data to automate the generation of homogeneous test cases through machine learning. The proposed method automatically extracts homogeneous test cases that are not dependent on skills and knowhow of the engineer writing the test cases from requirements specification documents. However, the required accuracy cannot be obtained by applying simple machine learning. To improve learning efficiency per unit of training data, without having to expand the training data, as the available quantity of requirements specifications is limited, and this would increase cost, the authors propose a method to increase accuracy through the preparation of training data inputted into the machine learning process, and conclude on the effectiveness of the method [109].

Model analytics for defect prediction allows quality assurance groups to build prediction models earlier and to predict the defect-prone components before the testing phase for in-depth testing [110]. In [110], it is shown that ML-based defect prediction models using design-level metrics in conjunction

[1] https://scrum-master.org/en/creating-the-perfect-user-story-with-invest-criteria/

with data sampling techniques are effective in finding software defects. The study shows that design-level attributes have a strong correlation with the probability of defects and the SMOTE data sampling approach improves the performance of prediction models. When design-level metrics are applied, the Adaboost ensemble method provides the best performance to detect the minority class samples [110].

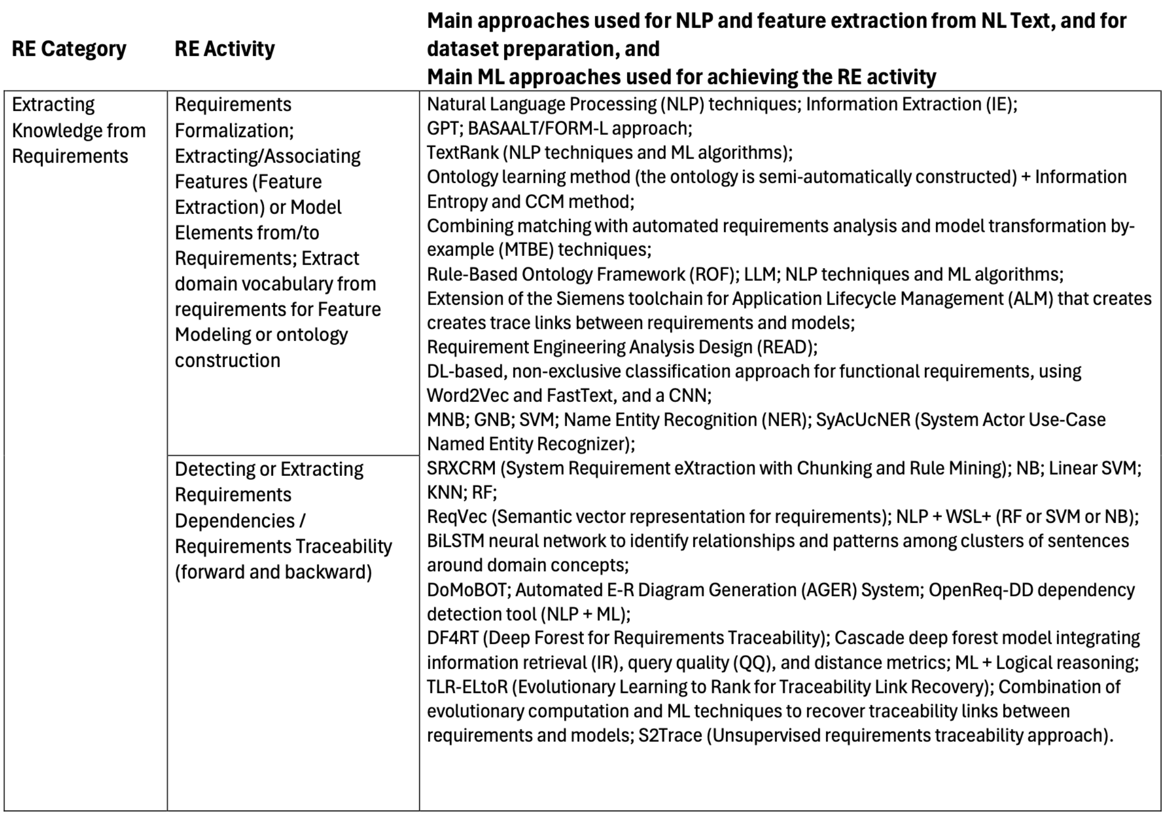
For quality assurance and maturity support of the final products, requirements must be verified and validated at different testing levels. To achieve this, requirements are manually labeled to indicate the corresponding testing level. The number of requirements can vary from few hundreds in smaller projects to several thousands in larger projects. Their manual labeling is time consuming and error-prone, thus sometimes incurring an unacceptable high cost. In [111], the initial results on an automated requirements classification approach proposal are reported. Requirements are automatically classified into two classes, using machine learning approaches: 'Integration Test' and 'Software Test'. The proposed solution may help the requirements engineers by speeding up the requirements classification and thus reducing the time to market of final products.

5.2.4. Extracting Knowledge from Requirements

The extraction of knowledge from requirements specifications enables getting semantically rich objects and concepts for building more formal models of such requirements or of the intended system. Several reviewed references target this goal, either for formalizing requirements, generating feature models, domain models or use case models, or to build domain vocabularies or ontologies that may help in further tasks towards designing and building the intended system.

In table 5, the main NLP and ML-based approaches for extracting knowledge from requirements are presented. As seen in the table, several techniques are reported in the studied literature, and these are addressed in this section, along with each studied reference.

Table 5. Main ML-based approaches to Extracting Knowledge from Requirements



RE Category	RE Activity	Main approaches used for NLP and feature extraction from NL Text, and for dataset preparation, and Main ML approaches used for achieving the RE activity

Extracting Knowledge from Requirements	Requirements Formalization; Extracting/Associating Features (Feature Extraction) or Model Elements from/to Requirements; Extract domain vocabulary from requirements for Feature Modeling or ontology construction	Natural Language Processing (NLP) techniques; Information Extraction (IE);
		GPT; BASAALT/FORM-L approach;
		TextRank (NLP techniques and ML algorithms);
		Ontology learning method (the ontology is semi-automatically constructed) + Information Entropy and CCM method;
		Combining matching with automated requirements analysis and model transformation by-example (MTBE) techniques;
		Rule-Based Ontology Framework (ROF); LLM; NLP techniques and ML algorithms;
		Extension of the Siemens toolchain for Application Lifecycle Management (ALM) that creates creates trace links between requirements and models;
		Requirement Engineering Analysis Design (READ);
		DL-based, non-exclusive classification approach for functional requirements, using Word2Vec and FastText, and a CNN;
		MNB; GNB; SVM; Name Entity Recognition (NER); SyAcUcNER (System Actor Use-Case Named Entity Recognizer);

	Detecting or Extracting Requirements Dependencies / Requirements Traceability (forward and backward)	SRXCRM (System Requirement eXtraction with Chunking and Rule Mining); NB; Linear SVM; KNN; RF; ReqVec (Semantic vector representation for requirements); NLP + WSL+ (RF or SVM or NB); BiLSTM neural network to identify relationships and patterns among clusters of sentences around domain concepts; DoMoBOT; Automated E-R Diagram Generation (AGER) System; OpenReq-DD dependency detection tool (NLP + ML); DF4RT (Deep Forest for Requirements Traceability); Cascade deep forest model integrating information retrieval (IR), query quality (QQ), and distance metrics; ML + Logical reasoning; TLR-ELtoR (Evolutionary Learning to Rank for Traceability Link Recovery); Combination of evolutionary computation and ML techniques to recover traceability links between requirements and models; S2Trace (Unsupervised requirements traceability approach).

Natural Language Processing (NLP) techniques have demonstrated their effectiveness in analyzing technical specification documents. One such technique, Information Extraction (IE), enables the automated processing of SRS by transforming unstructured or semi-structured text into structured data. This allows requirements to be converted into formal logic or model elements, enhancing their clarity and usability.

In [114], the authors introduce an IE method specifically designed for SRS data, analyze the proposed technique on a set of real requirements, and exemplify on how information obtained using their technique can be converted into a formal logic representation.

ChatGPT has also shown the potential in assisting software engineers in extracting model elements from requirements specification. In [58], ChatGPT is used, among other things, to requirements analysis and domain and design modeling. The study identifies lack of traceability and inconsistencies among produced artifacts as the main drawbacks. This demands human involvement in RE activities, but can serve as a valuable tool in the SDLC, enhancing productivity.

User demand is the key to software development. The domain ontology established by artificial intelligence can be used to describe the relationship between concepts in a specific domain, which can enable users to agree on conceptual understanding with developers [112]. The study in [112] uses the ontology learning method to extract the concept, and the ontology is constructed semi-automatically or automatically. Because the traditional weight calculation method ignores the distribution of feature items, the authors introduce the concept of information entropy, and the CCM method is further integrated [112]. This method allows improving the automation degree of ontology construction, and also make the user requirements of software more accurate and complete.

The BASAALT/FORM-L approach, seen before, can also be used to extract knowledge from textual requirements [83]. The approach creates semantically precise and simulable models while integrating missing contextual details. These models can then be reviewed by stakeholders to ensure alignment with intended requirements.

In [81], a machine learning-based approach is proposed to formalize NL requirements for critical embedded systems. Using spaCy's pre-trained NLP model, the method performs tokenization, PoS tagging, and dependency parsing. A rule-based strategy extracts text chunks, ensuring domain independence. These chunks are reordered into a standardized format, with deviations signaling quality issues. The structured output supports test oracle generation, software implementation, and requirement assessment through similarity and complexity metrics.

In [113], the authors address how the production of model transformations (MT) can be accelerated by automating transformation synthesis from requirements, examples and metamodels. A synthesis process is introduced, based on metamodel matching, correspondence patterns between metamodels, and completeness and consistency analysis of matches [113]. The authors also address how to deal with the limitations of metamodel matching by combining matching with automated requirements analysis and model transformation by-example (MTBE) techniques [113]. In practical examples, a large percentage of required transformation functionality can usually be constructed automatically, thus potentially reducing development effort. The efficiency of synthesized transformations is assessed [113].

The authors in [115], elaborate on previous work and propose a Rule-Based Ontology Framework (ROF) for Auto-Generating Requirements Specification. ROF covers the processes of requirements elicitation and of requirements documentation. The output of the elicitation process is a list of final requirements that are stored in an ontology structure, called Requirements Ontology (RO) [115]. From the RO, the documentation process automatically generates two outputs: process model, in the Business Process Model and Notation (BPMN) standard; and, SRS documents, in the IEEE standard [115]. The authors analyze the feasibility of implementing ROF in Information System (IS) projects through a case study on lecturer workload calculation at an Indonesian university. Using qualitative and quantitative methods, they assess each output and conclude that ROF effectively minimizes effort in generating requirements specifications [115].

Given the rise of LLMs, the authors in [116] investigate their potential for extracting domain models from agile product backlogs. They compare LLMs against (i) a state-of-practice tool and (ii) a specialized NLP approach, using a dataset of 22 products and 1679 user stories. This research marks an initial step toward leveraging LLMs or tailored NLP for automated model extraction from requirements text [116].

Model-Based Software Engineering (MBSE) offers various modeling formalisms, including domain models, which capture key concepts and relationships in class diagrams, in early design stages. Domain modeling transforms informal NL requirements into concise, analyzable models. However, existing automated approaches face three key challenges: insufficient accuracy for direct use, limited modeler interaction, and a lack of transparency in modeling decisions [117]. To address this, the authors in [117] propose an algorithm that enhances bot-modeler interaction by identifying and suggesting alternative configurations. Upon modeler approval, the bot updates the domain model. Evaluations show the bot achieves median F1 scores of 86% for Found Configurations, 91% for Offered Suggestions, and 90% for Updated Models, with a median processing time of 55.5 ms.

In [118], the authors employ NLP techniques and ML algorithms to automatically extract and rank the requirements terms to support high-level feature modeling. For that, they propose an automatic framework composed of noun phrase identification technique for requirements terms extraction and TextRank combined with semantic similarity for terms ranking. The final ranked terms are organized as a hierarchy, which can be used to help name elements when performing feature modeling [118]. In the quantitative evaluation, the proposed extraction method performs better than three baseline methods in recall with comparable precision. Their adapted TextRank algorithm can rank more relevant terms at the top positions in terms of average precision compared with most baselines [118].

Feature models (FM) provide a visual abstraction of the variation points in the analysis of Software Product Lines (SPL), which comprise a family of related software products. FMs can be manually created by domain experts or extracted (semi-)automatically from textual documents such as product descriptions or requirements specifications [119]. In [119] a method to quantify and visualize whether the elements in a FM (features and relationships) conform to the information available in a set of specification documents is proposed. Both the correctness (choice of representative elements) and completeness (no missing elements) of the FM are considered.

Requirements traceability helps to ensure that the developed system fulfills all requirements and prevents failures. For safety-critical systems, traceability is mandatory to ensure that the system is implemented correctly. Establishing and maintaining trace links are hard to achieve manually in today's complex systems [120]. In [120], the authors propose a tool for establishing bi-directional traceability links between requirements and model-based designs using Artificial Intelligence (AI). The tool is an extension of the Siemens toolchain for Application Lifecycle Management (ALM), systems engineering, and embedded software design. The proposed tool creates trace links between requirements written in natural language and CapitalTM software, AUTOSAR, SysML, UML, or Arcadia models for system/software design. The authors describe the implemented use-cases of tracing system/SW/HW requirements to system architecture models and provide an overview of the tool architecture.

Retrieving and extracting software information from SRS is crucial for SPL development. While NLP techniques like information retrieval and machine learning have been proposed for optimizing requirements specifications, they remain underutilized due to the complexity of organizational information and subsystem inter-dependencies [43]. A simple multi-class classification framework is insufficient to address these challenges. To overcome this, the work in [43] proposes a deep learning-based, non-exclusive classification approach for functional requirements. It utilizes Word2Vec and FastText word embeddings to represent documents and train a convolutional neural network (CNN). The study uses manually categorized enterprise data (AUTOSAR) for training and compares the impact of Word2Vec and FastText embeddings with pre-trained models available online.

The problems associated with the requirement analysis and class modeling can be overcome by the appropriate employment of ML. In [121], the authors propose a system, requirement engineering analysis design (READ) to generate UML class diagram using NLP and domain ontology techniques. They have implemented the READ system in Python and it successfully generates the UML class diagram i.e., class name, attributes methods, and relationships from the textual requirements written in English. To assess the performance of the proposed system, it was evaluated on publicly available standards, and the experimental results show that it outperforms the existing techniques for object-oriented based software designing.

The research reported in [122] applies NB classifiers – Multinomial and Gaussian over different SRS documents and classify the software requirement entities (Actors and Use Cases) using ML based methods. The study used SRS documents of 28 different systems, and labels for the entities 'Actor' and 'Use Case' have been defined. MNB is a popular classifier because of its computational efficiency and relatively good predictive performance [122]. Several classifiers have been tried out. The MNB recognizes Actors and Use Cases with an accuracy of 91%. Actors and Use Cases can be extracted with high accuracy from the SRS documents using MNB, which then can be used for plotting the Use Case diagram of the system [122]. Automated UML model generation approaches have a very prominent role in an agile development environment where requirements change frequently.

Existing NLP approaches for processing requirements documents are often limited to specific model types, such as domain or process models, and fail to reflect real-world requirements. To address this, in [123] a conceptual-level pre-processing pipeline is proposed, for automatically generating UML class, activity, and use case models. The pipeline consists of three steps [123]: (1) entity-based extractive summarization to highlight key requirement sections, (2) rule-based bucketing to categorize sentences for different UML models, and (3) sequence labeling to identify classes and attributes for class modeling. Since labeled datasets for this task are scarce, the authors have labeled the widely used PURE dataset, by tagging classes and attributes within the texts, on a word level, to train their supervised machine learning model [123].

In [124] a named entity recognition method called SyAcUcNER (System Actor Use-Case Named Entity Recognizer) is presented, with the goal of extracting the system, actor, and use case entities from unstructured English descriptions of user requirements for the software. SyAcUcNER uses SVM as an effective classifier, and uses a semantic role labeling process to tag the words in the text of user software requirements. SyAcUcNER is able to define the structure of a requirements engineering specialized Named Entity Recognizer (NER), and uses a specialized NER model as an approach for extracting actor and use case entities from the requirements description. Also, WEKA's SVM has been used to specify the semantic meanings of words in a certain domain of discourse, that is the SRS. The performance of SyAcUcNER is evaluated using a binomial technique, and the results from running it on text corpora from assorted sources gave weighted averages of 76.2% for precision, 76% for recall, and 72.1% for the F-measure [124].

The process of testing industrial systems, integrating highly configurable safety-critical components, is highly time-consuming and may require associations between product features and requirements demanded by customers [125]. ML has been used to help engineers in this task, by automating the extraction of associations between features and requirements. However, when requirements are written in NL, several additional difficulties arise. In [125], a NLP-based model, called SRXCRM (System Requirement eXtraction with Chunking and Rule Mining), is presented, which is able to extract and associate components from product design specifications and customer requirements, written in NL, of safety-critical systems. The model has a Weight Association Rule Mining framework that defines associations between components, generating visualizations that can help engineers in the prioritization of the most impactful features [125]. Preliminary results show that SRXCRM can extract such associations and visualizations [125].

ReqVec, a semantic representation for functional software requirements, is presented in [126]. ReqVec is a semantic vector representation for the requirements, that can be used different requirement-related applications, such as, requirement categorization or dependency, among others. ReqVec uses semantic dimensions, such as 'main actor', 'main action' and 'affected element', but the same approach can be applied using any improved set of dimensions [126]. ReqVec is calculated based on three main phases [126]: (1) a set of lexical and syntactic steps are performed to analyze textual requirements; (2)

semantic dimensions for requirements are calculated based on a words classifier and the well-known word embedding model Word2vec; (3) ReqVec is constructed based on the representations of these dimensions. Two experiments have been conducted to assess how ReqVec can capture meaningful semantic information to solve two well-known Requirements Engineering tasks: detecting semantic relation between requirements, and requirements categorization. The proposed representation was efficient enough to detect related requirements with 0.92 F-measure and to categorize requirements with 0.88 F-measure [126].

In [127], the authors develop an approach to extract domain models from problem descriptions written in NL by combining rules based on NLP with ML. First, they present an automated approach with an accuracy of extracted domain models higher than existing approaches. In addition, the approach generates trace links for each model element of a domain model. These trace links enable novice modelers to execute queries on the extracted domain models to gain insights into the modeling decisions taken for improving their modeling skills [127]. Preliminary results are positive.

Use Case Analysis is a graphical depiction used to explain the interaction between the user and the system for the given user's task, and it also denotes the extension/dependency of one use case to another. It is often used to identify, clarify, and categorize system requirements. Generating use cases from inherently ambiguous NL description of requirements may be a hard work, that can be automated using data-driven techniques [128]. The work in [128] presents an initial approach for the automated identification of use case names and actor names from the textual requirements specification using a NLP technique called Name Entity Recognition (NER), and extracts the named entity mentions in unstructured texts into predefined categories such as person names, organizations, locations, etc.

Model-Driven Software Engineering promotes using models and transformations to enhance system understanding. Domain modeling helps convert informal natural language requirements into concise, analyzable models but is time-consuming and requires expertise. Many approaches have been proposed to automatically extract domain concepts and relationships using extraction rules. However, relationships and patterns often remain hidden within sentences in a problem description [129], making their automatic recognition challenging due to the lack of contextual information and external knowledge about the domain concepts. To address these limitations, the authors in [129] propose a novel technique inspired by recent work on domain model extraction. Their approach customizes sentence boundaries to create clusters of sentences around domain concepts and employs a BiLSTM neural network to identify relationships and patterns among them. Additionally, they introduce a classification strategy for these relationships and patterns to implement the technique. Preliminary results indicate that this approach is promising and deserves further research [129].

In [130], a domain modeling bot called DoMoBOT is introduced, and implemented it in the form of a web-based prototype. According to the authors, DoMoBOT automatically extracts a domain model from a problem description written in NL with an accuracy higher than existing approaches. The bot also enables modelers to update a part of the extracted domain model and, in response, the bot re-configures the other parts of the domain model pro-actively. To improve the accuracy of extracted domain models, techniques of Natural Language Processing and Machine Learning are combined [130].

The work in [131] proposes an Automated E-R Diagram Generation (AGER) System that can generate an E-R Diagram from a given text in Natural Language. The AGER System parses an input text in natural language, semantically analyzes it and internally uses some domain specific databases and POS tagging to detect Entity and Relations from the given passage and builds a graph that represents the E-R Diagram [131]. During a project's requirements analysis phase, software engineers often engage in discussions with clients about the intended use cases. The conclusion of this process may yield a comprehensive E-R diagram, which serves as the blueprint for implementing and materializing the database relationships in later stages. The AGER system is aimed to assist in creating E-R Diagram directly from client's requirements in natural language [131].

With the widespread use of online forums and social media, gathering user feedback has become common, though such data is often fragmented and contains multiple viewpoints expressed over various messages. In [132], an argumentation-based CrowdRE approach is proposed to model these conversations as user argumentation models while preserving their original structure. Leveraging argumentation theory, the approach identifies new features and issues along with their supporting and opposing arguments. To achieve this, the study adopts abstract argumentation, bipolar argumentation, and coalition-based meta argumentation frameworks [132]. Moreover, automated support is provided through algorithms for bipolar argumentation, coalition-based meta argumentation, and an end-user voting mechanism. Various machine learning algorithms are employed to classify user comments into rationale elements and to identify conflict-free features or claims based on their supporting and attacking arguments. Initial results indicate that the approach effectively identifies features, issues, and their associated arguments with acceptable performance [132].

Some challenges have arisen in processing and analyzing user data for conversion into UML diagrams [133]. In response, the work in [133] introduces a novel approach that primarily aims to improve accuracy, shorten the time required to generate use cases from natural language descriptions, and address shortcomings in current technologies. The authors goal is to create a smart, precise system that not only saves time but also enhances user trust in the software.

In [134], the authors propose an approach in which users provide exemplary behavioral descriptions rather than explicit requirements. They reduce the problem of synthesizing a requirements specification from examples to one of grammatical inference, applying an active coevolutionary learning approach [134]. While such an approach typically requires numerous user feedback queries, the authors extend active learning by incorporating multiple oracles, known as proactive learning [134]. In this framework, the "user oracle" supplies input from the user, and the "knowledge oracle" provides formalized domain knowledge. Their two-oracle method, called the "first apply knowledge then query" (FAKT/Q) algorithm, is benchmarked against standard active learning, resulting in fewer required user queries and a faster inference process.

The work in [105], addressed in section 5.2.3, also proposes a method, combining NLP and ML, to automate extracting semantic elements for software requirement-to-test mapping. It formalizes recommendation reviews, enhancing traceability, reliability, and efficiency throughout the SDLC.

The paper in [135] summarizes the approach of the OpenReq-DD dependency detection tool developed at the OpenReq project, which allows an automatic requirement dependency detection approach. The core of this proposal is based on an ontology that defines dependency relations between specific terminologies related to the domain of the requirements. Using this information, it is possible to apply NLP techniques to extract meaning from these requirements and relations, and Machine Learning techniques to apply conceptual clustering, with the major purpose of classifying these requirements into the defined ontology [135].

Requirement dependencies affect many activities in the SDLC and are the basis for various software development decisions. Requirements dependencies extraction is, however, an error-prone and a cognitively and computationally complex problem, since most of the requirements are documented in natural language [136]. A two-stage approach is proposed in [136] to extract requirements dependencies using NLP and Weakly supervised learning (WSL). In the first stage, binary dependencies (basic dependent/independent relationships) are identified, and in the second stage, these are analyzed to determine the specific dependency type. An initial evaluation on the PURE dataset, using RF, SVM and NB was conducted. These three machine learners showed similar accuracy levels, although SVM required extra parameter tuning. The accuracy was further improved by applying weakly supervised learning to generate pseudo-annotations for unlabeled data [136]. The authors have defined a research agenda for assessing the use of their approach in different domains. To increase the semantic foundations, they intend to use evolving ontologies.

In [137], the authors present an approach to automatically identify requirement dependencies of type “requires” by using supervised classification techniques. The results indicate that the implemented approach can detect potential "requires" dependencies between requirements (formulated on a textual level). The approach has been evaluated on a test dataset and the conclusion is that it is possible to identify requirement dependencies with a high prediction quality. The proposed system has been trained and tested with different classifiers such as NB, Linear SVM, KNN, and RF. RF classifiers have correctly predicted dependencies with a F1 score of 82%.

Ignoring requirements inter-dependencies can adversely impact the design, development and testing of software products. In [138], a proposal addressing three main challenges is made: (1) NLP is studied to automatically extract dependencies from textual documents. Verb classifiers are used to automate elicitation and analysis of different types of dependencies (e.g: requires, coupling); (2) representation and maintenance of changing requirement dependencies from designing graph theoretic algorithms is explored; (3) the process of providing recommendations of dependencies is studied. The results, still preliminary, are aimed at assisting project managers to evaluate the impact of inter-dependencies and make effective decisions in software development life cycle [138].

Many supervised learning methods have been applied to requirements traceability recovery (RTR), yet their performance remains unsatisfactory, prompting the need for more effective models. In [139], a new cascade deep forest model for RTR, called DF4RT (Deep Forest for Requirements Traceability), is proposed with a novel composition aimed at enhancing performance. The model integrates three feature representation methods: information retrieval (IR), query quality (QQ), and distance metrics [139]. Additionally, it employs a layer-by-layer training approach to harness the benefits of DL while incorporating IR, QQ, and distance features to promote input diversity and enhance model robustness [139]. DF4RT was evaluated on four open-source projects and compared against nine state-of-the-art tracing approaches, showing average improvements of 94% in precision, 58% in recall, and 72% in F-measure [139]. The proposed approach is effective for RTR with good interpretability, few parameters, and good performance in small-scale data.

The traceability links between requirements and code are fundamental in supporting change management and software maintenance. Automatic trace retrieval can be performed via various tools such as Information retrieval or ML techniques [140]. These tools have, however, a low precision problem, which is primarily caused by the term mismatches across documents to be traced. The study in [140] proposes an approach that addresses the term mismatch problem to obtain the greatest improvements in the trace retrieval accuracy. The approach uses clustering in the automated trace retrieval process and, in an experimental evaluation against previous benchmarks, it showed results that allow one to conclude that the approach improves the trace retrieval precision [140].

Traceability Link Recovery (TLR) has been a topic of interest for many years within the software engineering community, and recently it has gained even more attention from both fundamental and applied research. However, there remains a significant gap between industry needs and the academic solutions proposed [141]. The work in [141] proposes an approach called Evolutionary Learning to Rank for Traceability Link Recovery (TLR-ELtoR), which combines evolutionary computation and machine learning techniques to recover traceability links between requirements and models by generating a ranked list of model fragments capable of fulfilling the requirement. TLR-ELtoR was evaluated in a real-world railway domain case study and compared against five TLR approaches (Information Retrieval, Linguistic Rule-based, Feedforward Neural Network, Recurrent Neural Network, and Learning to Rank). The results demonstrate that TLR-ELtoR achieved the best performance on most indicators, with a mean precision of 59.91%, recall of 78.95%, a combined F-measure of 62.50%, and a MCC value of 0.64 [141].

Many information retrieval-based approaches have been proposed to automatically recover software requirements traceability links. However, such approaches typically calculate textual similarities among software artifacts without considering specific features of different software artifacts, leading to less accurate results [142]. In [142], the authors present a hybrid method for recovering requirements traceability links by combining ML with logical reasoning to analyze both use case and code features. The approach first extracts semantic features from use cases and code, which are then used to train a classifier through supervised learning. Simultaneously, it examines the structural aspects of code to incrementally uncover traceability links using a set of defined reasoning rules. Experiments comparing this method to state-of-the-art techniques show that the proposed approach outperforms existing methods.

Supervised automated solutions to generate trace links use machine learning or deep learning techniques, but require large labeled datasets to train an effective model. Unsupervised solutions as word embedding approaches can generate links by capturing the semantic meaning of artifacts and are gaining more attention. Despite that, the authors in [143] argue that, besides the semantic information, the sequential information of terms in the artifacts would provide additional assistance for building the accurate links. In that sense, they propose an unsupervised requirements traceability approach (named S2Trace) which learns the Sequential Semantics of software artifacts to generate the trace links. Its core idea is to mine the sequential patterns and use them to learn the document embedding representation. Evaluation has been conducted on five public datasets, and results show that the proposed approach outperforms three typical baselines. The modeling of sequential information in [143] provides new insights into the unsupervised traceability solutions, and the improvement in the traceability accuracy further proves the usefulness of the sequential information.

5.2.5. Supporting Requirements Management and Validation, and Project Management

Requirements prioritization, assessing risk, namely the impact of requirements' change requests on the project outcomes, allocating requirement to software versions, among other activities, are important topics in supporting project management and requirements management and validation. Table 6, lists the main NLP and ML-based approaches for supporting requirements management and validation, and project management. These approaches, together with the reviewed literature references that use them, are addressed in this section.

Table 6. Main ML-based approaches to Supporting Requirements Management and Validation, and Project Management.

RE Category	RE Activity	Main approaches used for NLP and feature extraction from NL Text, and for dataset preparation, and Main ML approaches used for achieving the RE activity
Supporting Requirements Management and Validation, and Project Management	Requirements Prioritization	Adam algorithm; ARPT (Automated Requirement Prioritization Technique); Decision Tree, Random Forest, and K-Nearest Neighbors; NLP + ML; Combination of NLP techniques and Machine Learning algorithms; Adapted genetic K-means algorithm for software requirements engineering (GKA-RE), which automatically identifies the optimal number of clusters by dynamically readjusting initial seeds for improved quality; AI Task Allocation tool (ATA'); Tree-Family Machine Learning (TF-ML); Credal Decision Tree (CDT).
	Requirements allocation to software versions (considering as criteria: requirement development time, priority levels, and dependency relation-ships); Predicting if a software feature will be completed in its planned iteration	
	Project Management Risks Assessment / Req. Change Impact analysis on other requirements and on planned test cases	

The study in [144] proposes an optimization algorithm that aims to select features to give meaningful information about requirements. These features can be used to train a model for prioritizing requirements. The study examines how optimization algorithms select features and assign requirement priorities. It reveals that the Adam algorithm struggles with accurately prioritizing requirements due to the sparse matrix generated for the text dataset and high computational cost, and it fails to consider requirement dependencies. To address these issues, the paper introduces the Automated Requirement Prioritization Technique (ARPT) [144]. Compared to the Adam algorithm, ARPT achieves a much lower mean squared error of 1.29 versus 6.36, and its execution time is only 1.99 ms compared to 3380 ms [144].

Prioritizing software requirements for a release is complex, especially when requested features exceed development capacity. This challenge grows with multiple product and business lines, involving diverse stakeholders. The study in [145] applies ML to optimize release planning by analyzing key parameters influencing requirement inclusion. Five models were tested using accuracy, F1 score, and K-Fold Cross Validation. While most models achieved 80% accuracy, further investigation led to improved results. DT, Random Forest, and K-Nearest Neighbors performed best, with optimized Random Forest reaching 100% accuracy in some metrics but at high computational cost. Future work may refine other models through hyperparameter tuning.

The authors in [146] found thirteen Requirements Prioritization (RP) methods applying AI techniques such as ML, or genetic algorithms, 38% of which seek to improve the scalability problem, whereas 15% of them aim to improve the lack of automation issues along the RP process. In order to address the issues of scalability and lack of automation in RP, the study in [146] proposes a semi-automatic multiple-criteria prioritization method for functional and non-functional requirements of software projects developed within the Software Product-Lines paradigm. The proposed RP method is based on the combination of NLP techniques and ML algorithms, and empirical studies will be carried out with real web-based geographic information systems (GIS), for its validation [146].

Similar to standard systems, the identification and prioritization of the user needs are relevant to the software quality and challenging in SPL due to common requirements, increasing dependencies, and diversity of stakeholders involved [147]. As prioritization process might become impractical when the number of derived products grows, recently there has been an exponential growth in the use of AI techniques in different areas of RE. In [147], a semi-automatic multiple-criteria prioritization process is proposed, for functional and non-functional requirements (FR/NFR) of software projects developed within the SPL paradigm for reducing stakeholder participation.

The clustering stakeholder problem for system requirements selection and prioritization is considered inheritance in the area of requirements engineering. In [148], the authors apply clustering techniques from marketing segmentation to determine the optimal number of stakeholder groups. They introduce an adapted genetic K-means algorithm for software requirements engineering (GKA-RE) that automatically identifies the optimal number of clusters by dynamically readjusting initial seeds for improved quality. The method was tested on two RALIC system requirements datasets using various evaluation metrics, and its performance was compared with that of the standard K-means approach. The experimental results indicate the superiority of GKA-RE over K-means approach in obtaining higher values of evaluation metrics [148].

There are many fundamental prioritization techniques available to prioritize software requirements. In automating various tasks of software engineering, ML has shown useful positive impact. The work in [149] discusses the various algorithms used to classify and prioritize software requirements. The results in terms of performance, scalability and accuracy from different studies are contradictory in nature due to variations in research methodologies and the type of dataset used. Based on the literature survey conducted, the authors have proposed a new architecture that uses both types of datasets, i.e. SRS and user text reviews to create a generalized model. The proposed architecture attempts to extract features which can be used to train the model using ML algorithms. The ML algorithms for classifying and prioritizing software requirements will be developed and assessed based on performance, scalability and accuracy.

The study in [77], addressed above, explores a speech-act-based linguistic technique to analyze online discussions and extract requirements-relevant information, also for RP. By applying NLP and ML, user reviews from various sources are filtered and classified into categories like feature requests and bug reports. The authors refine their prior speech-act analysis approach and evaluate it on datasets from an open-source and an industrial project. The method achieves F-scores of 0.81 and 0.84 in classifying messages as Feature/Enhancement or Other. Results indicate a correlation between certain speech acts and issue priority, supporting automated RP.

The work in [150] introduces the AI Task Allocation tool (ATA') for distributing software requirements across different versions using an AI planning approach. The authors propose a model, expressed in an AI planning language, that represents a software project with a set of requirements and one or more development teams. The generated plan assigns requirements based on development time, priority levels, and dependency relationships [150]. A case study was conducted to evaluate the ATA' tool, and preliminary results show that the generated plans allocate requirements according to the specified criteria. These findings suggest that ATA' can effectively support the planning of incremental development projects by facilitating the allocation of requirements among teams.

In [151], the authors address the challenge of predicting whether a high-level software requirement will be completed within its planned iteration, a key factor in release planning. While prior research focused on predicting low-level tasks like bug fixes, this study analyzes iteration changes across three large IBM projects, revealing that up to 54% of high-level requirements miss their planned iteration. To tackle this, the authors develop and evaluate an ML model using 29 features derived from prior research, developer interviews, and domain knowledge. The model, tested at four requirement lifecycle stages, achieves up to 100% precision. Feature importance analysis shows that some factors are project-specific or stage-specific, while others, such as time remaining in the iteration and requirement creator, consistently influence predictions.

The success of NL interfaces in interpreting and responding to requests is, to a large extent, dependent on rich underlying ontologies and conceptual models that understand the technical or domain specific vocabulary of different users [152]. The effective use of NL interfaces in Software Engineering (SE) requires dedicated ontology models for software-related terms and concepts [152]. Although there are many SE glossaries, these are often incomplete and focus on specific sub-fields without capturing associations between terms, limiting their utility for NL tasks. To address this, the authors in [152] propose an approach that starts with existing glossaries and their defined associations, and uses ML to dynamically identify and document additional relationships between terms. The resulting semantic network is used to interpret NL queries in the SE domain and is enhanced with user feedback. Evaluated in the sub-domain of Agile Software Development, focusing on requirements-related queries, the approach shows that the semantic network significantly improves the NL interface's ability to interpret and execute user queries.

Risk prediction is critical in the SDLC, and can determine a project's success, so early risk prediction is essential. In [153], a new model based on a requirement risk dataset uses Tree-Family Machine Learning (TF-ML) approaches to predict requirements' risks. Ten different TF-ML techniques were evaluated to achieve minimal error and maximum accuracy, with Credal Decision Tree (CDT) outperforming the others. In 10-fold cross-validation, CDT achieved a Mean Absolute Error (MAE) of 0.0126, Root Mean Squared Error (RMSE) of 0.0888, Relative Absolute Error (RAE) of 4.498%, and Root Relative Squared Error (RRSE) of 23.741%, while accuracy, recall, and F-measure each reached 0.980 and Matthews Correlation Coefficient (MCC) was 0.975, corresponding to 98% overall accuracy. These results lead to recommend CDT for risk prediction in software requirements. Moreover, these findings can serve as a benchmark for future research, which should also address class imbalance and explore feature selection and ensemble learning strategies [153].

Software requirements Change Impact Analysis (CIA) is crucial in RE since changes are inevitable. When a requirement change is requested, its effects on all software artifacts must be evaluated to decide whether to accept or reject it. The research in [154] proposes a prediction model using a combination of ML and NLP techniques to forecast the impact of a requirement change on other requirements in the SRS document. This on-going work will evaluate the proposed model with relevant datasets for accuracy and performance. The resulting tool may be used by project managers to perform automated change impact analysis and make informed decisions about requirement change requests [154].

Incomplete requirements elicitation and elaboration often leads to functional requirements changes, which need to be controlled. Each function in a functional requirement includes its name, description, input/output specifications, and error messages. These specifications detail input and output names, data types, and constraints, and may or may not be linked to the database schema. Therefore, when changes occur in inputs or outputs that impact the database schema, both the schema and the corresponding test cases can be affected. The work in [155] presents an approach for analyzing the impact on test cases when inputs or outputs of functional requirements are modified. This method provides a structured change process to manage updates to functional requirements, test cases, and the database schema, and it also includes a rollback feature to reverse changes if necessary.

6. Discussion and Conclusions

In this section, the reviewed literature's results, presented in the previous section as belonging to five categories of RE tasks, is further discussed.

The main discussion topics aim to provide the answers to the previously stated research questions, namely:

RQ1 Which Requirements Engineering activities take most advantage of the use of Artificial Intelligence techniques?

RQ2 Which Artificial Intelligence techniques are most used in each Requirements Engineering activity?

RQ3 Which Artificial Intelligence techniques have the best results in each Requirements Engineering activity?

Figure 4 shows the percentage of references in the reviewed literature for each of the identified categories. One may see that 26% of the works reviewed have been categorized as "Classification of Requirements according to their Functional/Non-Functional nature" and another 26% as "Extracting Knowledge from Requirements". These are the RE activities with most AI-based proposals for automation or semi-automation of tasks. Then, there is "Improving the Quality of Requirements and Software", with 24% of the articles reviewed, "Supporting Requirements Elicitation" with 15%, and "Supporting Requirements Management and Validation, and Project Management" with 9%. Recall that some references may appear in more than one category, in the cases they address several RE tasks.

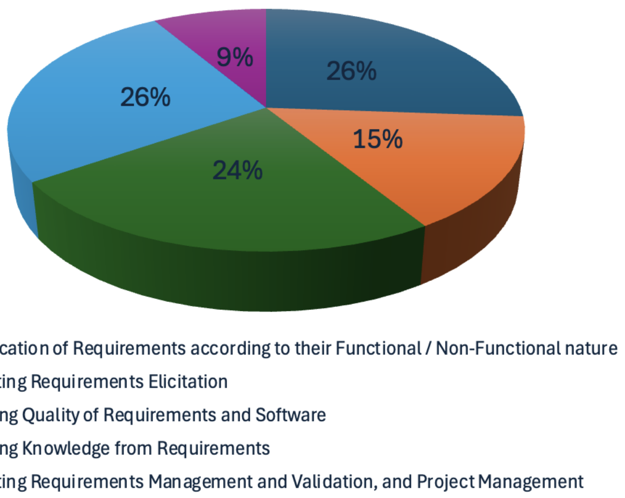


Figure 4. Categories of RE tasks found in the reviewed literature.

In the previous section, the main ML-based approaches for each of the five RE categories being considered in this article have been presented. In this section, an attempt is made to identify the most used and the ones that seem to yield better results.

Most of the studied approaches have used for text preprocessing and feature extraction, at the early phase of dealing with requirements written in NL, techniques for tokenization, normalization, lemmatization, and vectorization, in order to convert raw text into structured numerical feature vector representations. Examples of such techniques are Bag of words (BoW) [39], which quantifies the frequency of words in text documents, yielding a vector where each dimension corresponds to the frequency of a word, and TF-IDF [35,37,59], which weighs the word frequencies by how unique or important they are across documents. With standard BoW model, any word can have the highest term frequency, regardless of its semantic relevance. TF-IDF, which is a variation of the BoW, accounts for the word's prevalence throughout every document in a text set or corpus of documents. Once the requirements text is preprocessed and transformed into numerical feature vectors, a classification algorithm can be applied.

Besides BoW and TF-IDF, other used NLP techniques for data preprocessing and extracting syntactic and semantic features from natural language text, in preparation for any of the mentioned RE activities are Word2Vec, FastText, Doc2Vec and GloVe, which are able to capture semantic relationships between words. Tools like spaCy and NLTK also facilitate these steps. These techniques can be combined in a pipeline to convert raw NL text into structured numerical representations that an ML classifier can then use to learn patterns and make predictions.

To improve the accuracy and efficiency in classifying FR, NFR, and other specialized NFR requirements, after addressing the challenges posed by unstructured natural language data, many studies propose traditional ML algorithms, such as NB, DT, SVM, and Logistic Regression. The main ML approaches to classifying requirements as FR, NFR, or non-requirements, and for sub-categorizing NFR into their subcategories, are , SVM, NB, RF, KNN, among others (refer to table 2).

To identify preconditions and postconditions within requirements and categorize them into groups (e.g., "none" vs. "both", indicating the presence or absence of preconditions and postconditions in requirements), for test case generation, the used technique was NB with text preprocessing using libraries such as Scikit-learn and NLTK [104].

Supervised classifiers have also been used to automatically assess user stories against criteria (e.g., Testable and Valuable from the INVEST checklist) [106], and for automatically label requirements into testing levels (e.g., 'Integration Test' vs. 'Software Test') [111].

The results also show that the selection of the feature extraction techniques used after the NL text pre-processing phase, and before the ML classifiers entering into action, is important, as the chosen techniques influence the performance of the classifiers. For instance, selecting TF-IDF before different ML algorithms for classifying software requirements, namely DT, RF, LR, NN, KNN and SVM, yielded better performance than using Word2Vec before the same algorithms [37].

Some researchers further enhance these models using ensemble methods such as Bagging (Bagged KNN, Bagged DT, Bagged NB), RF, Gradient Boosting, AdaBoost, and majority voting ensembles. Some of the proposed ensemble techniques that have been proposed, and that seem to have yielded satisfactory results, are Bagged KNN, Bagged DT, and Bagged NB, which are ensemble methods where the respective classifiers (KNN, DT, and NB) are trained on different bootstrapped subsets of the training data [46,53]. This process, known as bagging (bootstrap aggregation), involves building several models and obtain several outputs, before combining and aggregating their predictions to enhance overall performance.

Other proposed ensemble methods, are Ensemble Grid Search classifier using 5 models (RF, MNB, GB, XGBoost, AdaBoost) [**?**] or Ensemble classifier combining 5 models (NB, SVM, DT, LR, SVC) [54]. These also work several solutions before aggregating results to obtain a better prediction.

Active Learning approaches, based on Strategies (e.g., uncertainty sampling) were used in the selection of the most informative data points for labeling, reducing manual effort in building high-quality training sets.

Ensemble Techniques such as boosting (Adaptive Boosting, Gradient Boosting, Extreme Gradient Boosting) have been applied to combine multiple models and enhance accuracy, particularly in identifying fault-prone requirements [82].

Supervised ML and Ensemble Methods, combining information retrieval, query quality, and distance metrics using cascade deep forest models (DF4RT) for traceability [139], have also been proposed to automate the generation of traceability links between requirements and other artifacts. Hybrid and Unsupervised Approaches, use sequential semantics (S2Trace) to mine patterns in requirements for more accurate link generation, have also been used for the same purpose [143].

Multiple ML classifiers (e.g., DT, RF, KNN) have been employed to predict requirement inclusion for releases [145,149]. Ensemble approaches and hyperparameter tuning further improve accuracy, addressing the complexity of prioritizing features when stakeholder demands exceed development capacity.

In [154], a combined NLP and ML model predicts how changes in requirements affect other parts of the SRS, supporting automated change impact analysis and aiding project managers in decision-making.

NLP techniques for extracting structured information (e.g., function names, I/O specifications) from functional requirements have also been used [155], followed by an ML-based approach to assess how changes affect test cases and database schemas, offering a structured change process and rollback features.

Some advanced Deep Learning approaches have also been proposed, including RNN-based models (Bi-LSTM, Bi-GRU) often combined with self-attention mechanisms and transformer-based architectures (BERT, BERT-CNN, PRCBERT) [40,41,45,60]. These models are designed to better capture contextual and sequential information in requirements texts. RNN and Variants, such as Bi-LSTM and Bi-GRU, capture sequential dependencies in requirements text, and have also been proposed for improving quality of requirements. Some of them incorporate self-attention mechanisms to better aggregate contextual information [40,60]. CNN has also been used, sometimes in hybrid models (e.g., BERT-CNN), to extract task-specific features from preprocessed text [45].

More advanced Transformer-based models, such as BERT, GPT-3.5, and PRCBERT are integrated for tasks like sentence classification, prompt-based learning, and even generating requirements, exploiting their contextual understanding. In fact, using Deep Learning transformer models such as BERT and GPT, and several variations (e.g., PRCBERT), is also becoming more common in classifying requirements as FR, NFR, or non-requirements, sub-categorizing NFR into their subcategories [45,47]. These type of models are, however, more demanding in terms of training data, and are more common for more complex RE activities, such as supporting requirements elicitation [66,74] or improving quality of requirements and software [85,89,95], as can be seen in tables 3 and 4, respectively.

For identifying requirements' ambiguities and generating use case specifications to zero-shot classification, the most used models have been BERT, GPT-3.5, and PRCBERT, due to their superior contextual understanding [64].

Other techniques, such as Zero-shot learning, have also been proposed to classify requirements without extensive labeled data.

Rasa-NLU and Rasa-Core open-source frameworks have been used for building conversational agents that can elicit requirements from stakeholders through natural language interactions [56]. The chatbot leverages an LSTM-based model for dialogue interpretation, followed by ML-based classification of the elicited requirements.

Large Language Models (LLMs), such as ChatGPT, have been used for generating model elements from natural language, in requirements analysis and domain and design modeling tasks [58,85,116]. These, however, have shown some issues with traceability and consistency, which indicate that human oversight remains necessary. To extract domain models from agile product backlogs, two approaches have been attempted, namely, the use of LLMs with specialized NLP tools [116], and bot-modeler interaction models for updating domain models [117].

In [144], an optimization algorithm is used to select meaningful features from requirements data. These features are used to train a model for prioritizing requirements. The Automated Requirement

Prioritization Technique (ARPT) improves on the Adam algorithm by reducing error (lower mean squared error) and execution time.

GKA-RE, a genetic K-means algorithm, is applied n [148] to group stakeholders. By dynamically readjusting initial seeds, the technique determines the optimal number of clusters, which aids in selecting and prioritizing requirements based on stakeholder segmentation.

Specialized Techniques for improving requirements quality have also been proposed.

NLP for linguistic feature extraction and , is used in [77]

Speech-Act Analysis, an NLP technique for extracting linguistic features, combined with ML-based classification, has been used to analyze online discussions by categorizing messages according to their speech acts. This approach enables filtering and categorizing user reviews (e.g., as feature requests or bug reports) and even issue priorities [77].

Approaches that combine textual and visual features (using autoencoders), called Multimodal Representations, are explored for processing user reviews with limited labeled data [69].

Clustering and hierarchical labeling techniques, for contextual word embeddings, have been proposed to group similar requirements, for identifying duplicates or handle multi-granularity issues. To creatively generate candidate requirements, by applying perturbations to original requirement descriptions, Adversarial Example Generation has been proposed [68].

Combining information from various sources in Knowledge Graphs (e.g., security practices) has also been used to enhance requirements management.

Ontology-based and Formal Methods, such as BASAALT and FORM-L were proposed for formalizing requirements, by generating semantically precise and simulable models, and support behavioral simulation, ensuring that ambiguities and inconsistencies are addressed [83].

Integrating ontologies with ML and optimization algorithms have been proposed for creating a traceability matrix, relating requirements and test cases while incorporating real-world knowledge, thus achieving high dependability and stability [103].

To extract domain concepts and relationships to build domain vocabularies or ontologies, methods that integrate information entropy and the CCM method have been proposed [112]. These methods help to improve the automation of ontology construction, enhancing the precision of user requirements.

The Rule-Based Ontology Framework (ROF) has been used to automatically generate requirements specifications and process models (BPMN, IEEE SRS) from elicitation outputs stored in an ontology [115].

In [152], ML is applied to extend existing SE glossaries, automatically identifying additional relationships between terms. This semantic network enhances natural language interfaces by improving domain understanding.

Other approaches used in improving requirements quality were Active Learning and AutoML, with techniques such as uncertainty sampling and tools such as TPOT [49]. These have been used to optimize model performance and reduce manual labeling efforts. And also, Transfer Learning and Text Augmentation techniques (e.g., ULMFiT and back translation) have been applied to improve classification performance, especially when data is limited [90].

Using cosine similarity and standard deviation-based methods for selecting and improving training data quality, in order to automatically generate test cases from requirements specifications with higher accuracy using fewer training data points, has also been used [108].

Converting SRS data into formal representations using Information Extraction (IE) techniques, to automatically transform unstructured or semi-structured SRS text into structured data that can be converted into formal logic or model elements, has also been proposed [114].

The combination of metamodel matching with transformation-by-example techniques, to accelerate model transformations from requirements by matching metamodels and analyzing consistency, has been used in [113].

To extract and hierarchically organize key requirement terms, a technique that combines noun phrase identification with TextRank and semantic similarity to rank terms has been used [118].

To extract domain-specific entities (e.g., system, actor, use case) from textual requirements, in [124] the SyAcUcNER approach has been proposed. It uses semantic role labeling and SVM to identify entities, aiding UML diagram generation.

To identify and map associations between features and requirements, especially for safety-critical systems, SRXCRM has been proposed. It applies weight association rule mining combined with NLP to visualize dependencies [125]. Additionaly, techniques like ReqVec have been used to generate semantic vector representations to capture relationships and dependencies among requirements [126].

In [150], an AI Task Allocation tool utilizes an AI planning language to distribute requirements across software versions based on development time, priority, and dependency relationships, supporting incremental project planning.

In [151], an ML model predicts whether high-level requirements will be met within planned iterations by using a set of 29 features, improving release planning through precise predictions of requirement completion.

In [153], a requirement risk prediction model uses various tree-family ML techniques, with the Credal Decision Tree (CDT) showing superior performance in terms of accuracy, error metrics, and overall precision. This helps forecast risks early in the SDLC.

7. Conclusions

In this article, the main AI techniques for Requirements Engineering, from the last five years, have been comprehensively reviewed, grouped in five RE tasks' categories: Classification of Requirements; Supporting requirements elicitation; Improving requirements and software quality; Extracting knowledge from requirements; and, Supporting requirements management and validation, and Project Management.

From the review, one can conclude that for supporting requirements classification, for identifying FR/NFR or for other purposes, the integration of traditional ML classifiers (NB, SVM, DT, RF, KNN) with effective NLP feature extraction techniques (BoW, TF-IDF, word embeddings) are the most used approaches.

Conversational agents and chatbots, built with frameworks like Rasa, combined with active learning strategies, have been proposed to facilitate the elicitation of requirements directly from stakeholders. This approach not only reduces manual effort but also minimizes errors by dynamically capturing and classifying requirements during stakeholder interactions.

For improving the quality of requirements, techniques such as the BASAALT/FORM-L approach and NLP4ReF leverage both rule-based and ML methods to formalize natural language requirements. By transforming unstructured text into precise, simulable models, these techniques support behavioral simulation and early detection of ambiguities, inconsistencies, and other quality issues, thereby ensuring that the requirements accurately reflect stakeholder needs.

The use of NLP techniques like Named Entity Recognition (NER), text chunking, and rule-based extraction, along with ML-based classification, has enabled the automated identification of domain-specific entities, such as classes, attributes, and relationships. This capability supports the generation of UML diagrams and domain models, thereby improving traceability and facilitating efficient system design.

And, the combination of NLP and ML techniques, by incorporating methods for traceability link recovery, risk prediction using tree-based ML approaches, and AI planning for requirement allocation, has been used to support requirements management. These methods support human decision-making across the SDLC by ensuring accurate change impact analysis, prioritization, and alignment with project goals.

Author Contributions: This research article is the result of research and findings of both authors, being the individual contributions as follows: Conceptualization, A.M.R.d.C.; methodology, A.M.R.d.C. and E.F.C.; investigation, A.M.R.d.C. and E.F.C.; resources, A.M.R.d.C.; formal analysis, A.M.R.d.C.; writing—original draft

preparation, A.M.R.d.C. and E.F.C.; writing—review and editing, A.M.R.d.C.. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: In this section you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
ASFR	Architecturally Significant Functional Requirement
BPMN	Business Process Model and Notation
CNN	Convolutional Neural Network
DL	Deep Learning
DT	Decision Tree
FR	Functional Requirement
GDPR	General Data Protection Regulation
GNB	Gaussian Naive Bayes
KNN	K-Nearest Neighbors
LLM	Large Language Model
LR	Linear Regression
LogR	Logistic Regression
ML	Machine Learning
MLP	Multilayer Perceptron
MNB	Multinomial Naive Bayes
NB	Naive Bayes
NFR	Non-Functional Requirement
NL	Natural Language
NLP	Natural Language Processing
NN	Neural Networks
PMBOK	Project Management Body of Knowledge
RE	Requirements Engineering
RF	Random Forest
RNN	Recurrent Neural Network
SDLC	Software development life-cycle
SRS	Software Requirements Specification
SVM	Support Vector Machines
UML	Unified Modeling Language

References

1. Liubchenko, V. The Machine Learning Techniques for Enhancing Software Requirement Specification: Literature Review. In Proceedings of the 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2023, Vol. 1, pp. 10–14. https://doi.org/10.1109/IDAACS58523.2023.10348647.
2. Jiang, Y.; Li, X.; Luo, H.; Yin, S.; Kaynak, O. Quo vadis artificial intelligence? *Discover Artificial Intelligence* **2022**, *2*.
3. Min, B.; Ross, H.; Sulem, E.; Veyseh, A.P.B.; Nguyen, T.H.; Sainz, O.; Agirre, E.; Heintz, I.; Roth, D. Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Comput. Surv.* **2023**, *56*. https://doi.org/10.1145/3605943.
4. Fernandes, J.M.; Machado, R.J. *Requirements in Engineering Projects*; Springer Cham, 2016.

5. Pressman, R. *Software Engineering: A Practitioner's Approach*; McGraw-Hill higher education, McGraw-Hill Education, 2010.
6. van Lamsweerde, A. *Requirements Engineering: From System Goals to UML Models to Software Specifications*; Wiley, 2009.
7. Page, M.J.; McKenzie, J.E.; Bossuyt, P.M.; Boutron, I.; Hoffmann, T.C.; Mulrow, C.D.; Shamseer, L.; Tetzlaff, J.M.; Akl, E.A.; Brennan, S.E.; et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* **2021**, *372*, [https://www.bmj.com/content/372/bmj.n71.full.pdf]. https://doi.org/10.1136/bmj.n71.
8. Nagarhalli, T.P.; Vaze, V.; Rana, N.K. Impact of Machine Learning in Natural Language Processing: A Review. In Proceedings of the 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 1529–1534. https://doi.org/10.1109/ICICV50876.2021.9388380.
9. Badillo, S.; Banfai, B.; Birzele, F.; Davydov, I.I.; Hutchinson, L.; Kam-Thong, T.; Siebourg-Polster, J.; Steiert, B.; Zhang, J.D. An Introduction to Machine Learning. *Clinical Pharmacology & Therapeutics* **2020**, *107*, 871–885. https://doi.org/https://doi.org/10.1002/cpt.1796.
10. Pouyanfar, S.; Sadiq, S.; Yan, Y.; Tian, H.; Tao, Y.; Reyes, M.P.; Shyu, M.L.; Chen, S.C.; Iyengar, S.S. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* **2018**, *51*. https://doi.org/10.1145/3234150.
11. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J. Classification of natural language processing techniques for requirements engineering. *arXiv preprint arXiv:2204.04282* **2022**.
12. Zhang, H.; Shafiq, M.O. Survey of transformers and towards ensemble learning using transformers for natural language processing. *Journal of Big Data* **2024**, *11*. https://doi.org/10.1186/s40537-023-00842-0.
13. Zamani, K.; Zowghi, D.; Arora, C. Machine Learning in Requirements Engineering: A Mapping Study. In Proceedings of the 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW), 2021, pp. 116–125. https://doi.org/10.1109/REW53955.2021.00023.
14. Liu, K.; Reddivari, S.; Reddivari, K. Artificial Intelligence in Software Requirements Engineering: State-of-the-Art. In Proceedings of the 2022 IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI), 2022, pp. 106–111. https://doi.org/10.1109/IRI54793.2022.00034.
15. Navaei, M.; Tabrizi, N. Machine Learning in Software Development Life Cycle: A Comprehensive Review. *ENASE* **2022**, *1*, 344–354.
16. Abdelnabi, E.A.; Maatuk, A.M.; Hagal, M. Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques. In Proceedings of the 2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA, 2021, pp. 288–293. https://doi.org/10.1109/MI-STA52233.2021.9464433.
17. Kulkarni, V.; Kolhe, A.; Kulkarni, J. Intelligent Software Engineering: The Significance of Artificial Intelligence Techniques in Enhancing Software Development Lifecycle Processes. In Proceedings of the Intelligent Systems Design and Applications; Abraham, A.; Gandhi, N.; Hanne, T.; Hong, T.P.; Nogueira Rios, T.; Ding, W., Eds. Springer International Publishing, 2022, pp. 67–82.
18. Sofian, H.; Yunus, N.A.M.; Ahmad, R. Systematic Mapping: Artificial Intelligence Techniques in Software Engineering. *IEEE Access* **2022**, *10*, 51021–51040. https://doi.org/10.1109/ACCESS.2022.3174115.
19. Sufian, M.; Khan, Z.; Rehman, S.; Haider Butt, W. A Systematic Literature Review: Software Requirements Prioritization Techniques. In Proceedings of the 2018 International Conference on Frontiers of Information Technology (FIT), 2018, pp. 35–40. https://doi.org/10.1109/FIT.2018.00014.
20. Talele, P.; Phalnikar, R. Classification and Prioritisation of Software Requirements using Machine Learning – A Systematic Review. In Proceedings of the 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2021, pp. 912–918. https://doi.org/10.1109/Confluence51648.2021.9377190.
21. Kaur, K.; Kaur, P. The application of AI techniques in requirements classification: a systematic mapping. *Artificial Intelligence Review* **2024**, *57*, 57.
22. López-Hernández, D.A.; Octavio Ocharán-Hernández, J.; Mezura-Montes, E.; Sánchez-García, A. Automatic Classification of Software Requirements using Artificial Neural Networks: A Systematic Literature Review. In Proceedings of the 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT), 2021, pp. 152–160. https://doi.org/10.1109/CONISOFT52520.2021.00030.
23. Pérez-Verdejo, J.M.; Sánchez-García, A.J.; Ocharán-Hernández, J.O. A Systematic Literature Review on Machine Learning for Automated Requirements Classification. In Proceedings of the 2020 8th International

Conference in Software Engineering Research and Innovation (CONISOFT), 2020, pp. 21–28. https://doi.org/10.1109/CONISOFT50191.2020.00014.

24. Li, X.; Wang, B.; Wan, H.; Deng, Y.; Wang, Z. Applications of Machine Learning in Requirements Traceability: A Systematic Mapping Study (S). In Proceedings of the SEKE, 2023, pp. 566–571.
25. Yadav, A.; Patel, A.; Shah, M. A comprehensive review on resolving ambiguities in natural language processing. *AI Open* **2021**, *2*, 85–92. https://doi.org/https://doi.org/10.1016/j.aiopen.2021.05.001.
26. Aberkane, A.J.; Poels, G.; Broucke, S.V. Exploring Automated GDPR-Compliance in Requirements Engineering: A Systematic Mapping Study. *IEEE Access* **2021**, *9*, 66542–66559. https://doi.org/10.1109/ACCESS.2021.3076921.
27. da Cruz Mello., O.; Fontoura, L.M. Challenges in Requirements Engineering and Its Solutions: A Systematic Review. In Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 2: ICEIS. INSTICC, SciTePress, 2022, pp. 70–77. https://doi.org/10.5220/0011079000003179.
28. Ahmed, S.; Ahmed, A.; Eisty, N.U. Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review. In Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), 2022, pp. 112–119. https://doi.org/10.1109/SERA54885.2022.9806783.
29. Sonbol, R.; Rebdawi, G.; Ghneim, N. The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review. *IEEE Access* **2022**, *10*, 62811–62830. https://doi.org/10.1109/ACCESS.2022.3182372.
30. Atoum, I.; Baklizi, M.K.; Alsmadi, I.; Otoom, A.A.; Alhersh, T.; Ababneh, J.; Almalki, J.; Alshahrani, S.M. Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review. *IEEE Access* **2021**, *9*, 137613–137634. https://doi.org/10.1109/ACCESS.2021.3117989.
31. Santos, R.; Groen, E.C.; Villela, K. An Overview of User Feedback Classification Approaches. In Proceedings of the REFSQ workshops, 2019, Vol. 3, pp. 357–369.
32. Ahmad, A.; Feng, C.; Tahir, A.; Khan, A.; Waqas, M.; Ahmad, S.; Ullah, A. An Empirical Evaluation of Machine Learning Algorithms for Identifying Software Requirements on Stack Overflow: Initial Results. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), 2019, pp. 689–693. https://doi.org/10.1109/ICSESS47205.2019.9040720.
33. Budake, R.; Bhoite, S.; Kharade, K. Identification and Classification of Functional and Nonfunctional Software Requirements Using Machine Learning. In Proceedings of the AIP Conference Proceedings, 2023, Vol. 2946. https://doi.org/10.1063/5.0178116.
34. Talele, P.; Phalnikar, R. Multiple correlation based decision tree model for classification of software requirements. *International Journal of Computational Science and Engineering* **2023**, *26*. https://doi.org/10.1504/ijcse.2023.131502.
35. Peer, J.; Mordecai, Y.; Reich, Y. NLP4ReF: Requirements Classification and Forecasting: From Model-Based Design to Large Language Models. In Proceedings of the IEEE Aerospace Conference Proceedings, 2024. https://doi.org/10.1109/AERO58975.2024.10521022.
36. Alhoshan, W.; Ferrari, A.; Zhao, L. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology* **2023**, *159*. https://doi.org/10.1016/j.infsof.2023.107202.
37. Talele, P.; Phalnikar, R.; Apte, S.; Talele, H. Semi-automated Software Requirements Categorisation using Machine Learning Algorithms. *International Journal of Electrical and Computer Engineering Systems* **2023**, *14*. https://doi.org/10.32985/ijeces.14.10.3.
38. Tasnim, A.; Akhter, N.; Ali, K. A Fine Tuned Ensemble Approach to Classify Requirement from User Story. In Proceedings of the 2023 26th International Conference on Computer and Information Technology, ICCIT 2023, 2023. https://doi.org/10.1109/ICCIT60459.2023.10441143.
39. Budake, R.; Bhoite, S.; Kharade, K. Machine Learning-Based Identification as Well as Classification of Functional and Non-functional Requirements. In Proceedings of the Lecture Notes in Electrical Engineering, 2024, Vol. 1087. https://doi.org/10.1007/978-981-99-6690-5_38.
40. Kaur, K.; Kaur, P. SABDM: A self-attention based bidirectional-RNN deep model for requirements classification. *Journal of Software: Evolution and Process* **2024**, *36*. https://doi.org/10.1002/smr.2430.
41. AlDhafer, O.; Ahmad, I.; Mahmood, S. An end-to-end deep learning system for requirements classification using recurrent neural networks. *Information and Software Technology* **2022**, *147*. https://doi.org/10.1016/j.infsof.2022.106877.
42. Patel, V.; Mehta, P.; Lavingia, K. Software Requirement Classification Using Machine Learning Algorithms. In Proceedings of the 2023 International Conference on Artificial Intelligence and Applications, ICAIA 2023

and Alliance Technology Conference, ATCON-1 2023 - Proceeding, 2023. https://doi.org/10.1109/ICAIA57370.2023.10169588.
43. Jp, S.; Menon, V.; Soman, K.; Ojha, A. A Non-Exclusive Multi-Class Convolutional Neural Network for the Classification of Functional Requirements in AUTOSAR Software Requirement Specification Text. *IEEE Access* **2022**, *10*. https://doi.org/10.1109/ACCESS.2022.3217752.
44. Nayak, U.A.; Swarnalatha, K.; Balachandra, A. Feasibility Study of Machine Learning & AI Algorithms for Classifying Software Requirements. In Proceedings of the MysuruCon 2022 - 2022 IEEE 2nd Mysore Sub Section International Conference, 2022. https://doi.org/10.1109/MysuruCon55714.2022.9972410.
45. Kaur, K.; Kaur, P. BERT-CNN: Improving BERT for Requirements Classification using CNN. In Proceedings of the Procedia Computer Science, 2022, Vol. 218. https://doi.org/10.1016/j.procs.2023.01.234.
46. Vijayvargiya, S.; Kumar, L.; Murthy, L.; Misra, S. Software Requirements Classification using Deep-learning Approach with Various Hidden Layers. In Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS 2022, 2022. https://doi.org/10.15439/2022F140.
47. Luo, X.; Xue, Y.; Xing, Z.; Sun, J. PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models. In Proceedings of the ACM International Conference Proceeding Series, 2022. https://doi.org/10.1145/3551349.3560417.
48. Magalhaes, C.; Araujo, J.; Sardinha, A. MARE: An Active Learning Approach for Requirements Classification. In Proceedings of the IEEE International Conference on Requirements Engineering, 2021, Vol. 2021-January. https://doi.org/10.1109/RE51729.2021.9714537.
49. J.M., P.V.; Á.J., S.G.; J.O., O.H.; E., M.M.; K., C.V. Requirements and GitHub Issues: An Automated Approach for Quality Requirements Classification. *Programming and Computer Software* **2021**, *47*. https://doi.org/10.1134/S0361768821080193.
50. Quba, G.; Qaisi, H.A.; Althunibat, A.; Alzu'Bi, S. Software Requirements Classification using Machine Learning algorithm's. In Proceedings of the 2021 International Conference on Information Technology, ICIT 2021 - Proceedings, 2021. https://doi.org/10.1109/ICIT52682.2021.9491688.
51. Dave, D.; Anu, V. Identifying Functional and Non-functional Software Requirements from User App Reviews. In Proceedings of the 2022 IEEE International IOT, Electronics and Mechatronics Conference, IEMTRONICS 2022, 2022. https://doi.org/10.1109/IEMTRONICS55184.2022.9795770.
52. Dave, D.; Anu, V.; Varde, A. Automating the Classification of Requirements Data. In Proceedings of the IEEE International Conference on Big Data, Big Data 2021, 2021. https://doi.org/10.1109/BigData52589.2021.9671548.
53. Vijayvargiya, S.; Kumar, L.; Malapati, A.; Murthy, L.; Misra, S. Software Functional Requirements Classification Using Ensemble Learning. In Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2022, Vol. 13381 LNCS. https://doi.org/10.1007/978-3-031-10548-7_49.
54. Rahimi, N.; Eassa, F.; Elrefaei, L. An ensemble machine learning technique for functional requirement classification. *Symmetry* **2020**, *12*. https://doi.org/10.3390/sym12101601.
55. Panichella, S.; Ruiz, M. Requirements-collector: Automating requirements specification from elicitation sessions and user feedback. In Proceedings of the IEEE International Conference on Requirements Engineering, 2020, Vol. 2020-August. https://doi.org/10.1109/RE48521.2020.00057.
56. Surana, C.R.K.; Shriya.; Gupta, D.; Shankar, S. Intelligent Chatbot for Requirements Elicitation and Classification. In Proceedings of the 2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings, 2019. https://doi.org/10.1109/RTEICT46194.2019.9016907.
57. Li, L.; Jin-An, N.; Kasirun, Z.; Piaw, C. An empirical comparison of machine learning algorithms for classification of software requirements. *International Journal of Advanced Computer Science and Applications* **2019**, *10*. https://doi.org/10.14569/IJACSA.2019.0101135.
58. Kim, D.K.; Chen, J.; Ming, H.; Lu, L. Assessment of ChatGPT's Proficiency in Software Development. In Proceedings of the Congress in Computer Science, Computer Engineering, and Applied Computing, 2023. https://doi.org/10.1109/CSCE60160.2023.00421.
59. Apte, S.; Honrao, Y.; Shinde, R.; Talele, P.; Phalnikar, R. Automatic Extraction of Software Requirements Using Machine Learning. *Lecture Notes in Networks and Systems* **2023**, *719 LNNS*. https://doi.org/10.1007/978-981-99-3758-5_33.

60. Chatterjee, R.; Ahmed, A.; Anish, P. Identification and Classification of Architecturally Significant Functional Requirements. In Proceedings of the 7th International Workshop on Artificial Intelligence and Requirements Engineering, AIRE 2020, 2020. https://doi.org/10.1109/AIRE51212.2020.00008.
61. Baker, C.; Deng, L.; Chakraborty, S.; Dehlinger, J. Automatic multi-class non-functional software requirements classification using neural networks. In Proceedings of the International Computer Software and Applications Conference, 2019, Vol. 2. https://doi.org/10.1109/COMPSAC.2019.10275.
62. Lafi, M.; Abdelqader, A. Automated Business Rules Classification Using Machine Learning to Enhance Software Requirements Elicitation. In Proceedings of the 2023 International Conference on Information Technology: Cybersecurity Challenges for Sustainable Cities, ICIT 2023 - Proceeding, 2023. https://doi.org/10.1109/ICIT58056.2023.10226076.
63. Gobov, D.; Huchenko, I. Software requirements elicitation techniques selection method for the project scope management. In Proceedings of the CEUR Workshop Proceedings, 2021, Vol. 2851.
64. Yeow, J.; Rana, M.; Majid, N.A. An Automated Model of Software Requirement Engineering Using GPT-3.5. In Proceedings of the 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems, ICETSIS 2024, 2024. https://doi.org/10.1109/ICETSIS61505.2024.10459458.
65. Tizard, J.; Devine, P.; Wang, H.; Blincoe, K. A Software Requirements Ecosystem: Linking Forum, Issue Tracker, and FAQs for Requirements Management. *IEEE Transactions on Software Engineering* **2023**, *49*. https://doi.org/10.1109/TSE.2022.3219458.
66. Shen, Y.; Breaux, T. Stakeholder Preference Extraction from Scenarios. *IEEE Transactions on Software Engineering* **2024**, *50*. https://doi.org/10.1109/TSE.2023.3333265.
67. Devine, P.; Koh, Y.; Blincoe, K. Evaluating software user feedback classifier performance on unseen apps, datasets, and metadata. *Empirical Software Engineering* **2023**, *28*. https://doi.org/10.1007/s10664-022-10254-y.
68. Gudaparthi, H.; Niu, N.; Wang, B.; Bhowmik, T.; Liu, H.; Zhang, J.; Savolainen, J.; Horton, G.; Crowe, S.; Scherz, T.; et al. Prompting Creative Requirements via Traceable and Adversarial Examples in Deep Learning. In Proceedings of the IEEE International Conference on Requirements Engineering, 2023, Vol. 2023-September. https://doi.org/10.1109/RE57278.2023.00022.
69. Gôlo, M.; Araújo, A.; Rossi, R.; Marcacini, R. Detecting relevant app reviews for software evolution and maintenance through multimodal one-class learning. *Information and Software Technology* **2022**, *151*. https://doi.org/10.1016/j.infsof.2022.106998.
70. Kauschinger, M.; Vieth, N.; Schreieck, M.; Krcmar, H. Detecting Feature Requests of Third-Party Developers through Machine Learning: A Case Study of the SAP Community. In Proceedings of the Annual Hawaii International Conference on System Sciences, 2023, Vol. 2023-January.
71. Mehder, S.; Aydemir, F.B. Classification of Issue Discussions in Open Source Projects Using Deep Language Models. In Proceedings of the IEEE International Conference on Requirements Engineering, 2022. https://doi.org/10.1109/REW56159.2022.00040.
72. Devine, P. Finding Appropriate User Feedback Analysis Techniques for Multiple Data Domains. In Proceedings of the International Conference on Software Engineering, 2022. https://doi.org/10.1109/ICSE-Companion55297.2022.9793732.
73. Araujo, A.; Marcacini, R. Hierarchical Cluster Labeling of Software Requirements using Contextual Word Embeddings. In Proceedings of the ACM International Conference Proceeding Series, 2021. https://doi.org/10.1145/3474624.3477067.
74. Bhatia, K.; Sharma, A. Sector classification for crowd-based software requirements. In Proceedings of the ACM Symposium on Applied Computing, 2021. https://doi.org/10.1145/3412841.3442005.
75. Do, Q.; Bhowmik, T.; Bradshaw, G. Capturing creative requirements via requirements reuse: A machine learning-based approach. *Journal of Systems and Software* **2020**, *170*. https://doi.org/10.1016/j.jss.2020.110730.
76. Iqbal, T.; Seyff, N.; Mendez, D. Generating requirements out of thin air: Towards automated feature identification for new apps. In Proceedings of the IEEE 27th International Requirements Engineering Conference Workshops, REW 2019, 2019. https://doi.org/10.1109/REW.2019.00040.
77. Morales-Ramirez, I.; Kifetew, F.; Perini, A. Speech-acts based analysis for requirements discovery from online discussions. *Information Systems* **2019**, *86*. https://doi.org/10.1016/j.is.2018.08.003.
78. Do, Q.; Chekuri, S.; Bhowmik, T. Automated Support to Capture Creative Requirements via Requirements Reuse. In Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2019, Vol. 11602 LNCS. https://doi.org/10.1007/978-3-030-22888-0_4.

79. Tizard, J.; Wang, H.; Yohannes, L.; Blincoe, K. Can a conversation paint a picture? Mining requirements in software forums. In Proceedings of the IEEE International Conference on Requirements Engineering, 2019, Vol. 2019-September. https://doi.org/10.1109/RE.2019.00014.
80. Peng, S.; Xu, L.; Jiang, W. Itemization Framework of Requirements using Machine Reading Comprehension. In Proceedings of the SPIE - The International Society for Optical Engineering, 2022, Vol. 12451. https://doi.org/10.1117/12.2656629.
81. Ehresmann, M.; Beyer, J.; Fasoulas, S.; Schorfmann, M.; Brudna, T.; Schoolmann, I.; Brüggmann, J.; Hönle, A.; Gerlich, R.; Gerlich, R. ExANT: Exploring NLP AI Systems for Requirements Development. In Proceedings of the International Astronautical Congress, IAC, 2023, Vol. 2023-October.
82. Muhamad, F.; Hamid, S.A.; Subramaniam, H.; Rashid, R.A.; Fahmi, F. Fault-Prone Software Requirements Specification Detection Using Ensemble Learning for Edge/Cloud Applications. *Applied Sciences (Switzerland)* **2023**, *13*. https://doi.org/10.3390/app13148368.
83. Nguyen, T.; Sayar, I.; Ebersold, S.; Bruel, J.M. Identifying and fixing ambiguities in, and semantically accurate formalisation of, behavioural requirements. *Software and Systems Modeling* **2024**. https://doi.org/10.1007/s10270-023-01142-0.
84. Berhanu, F.; Alemneh, E. Classification and Prioritization of Requirements Smells Using Machine Learning Techniques. In Proceedings of the 2023 International Conference on Information and Communication Technology for Development for Africa, ICT4DA 2023, 2023. https://doi.org/10.1109/ICT4DA59526.2023.10302263.
85. Luitel, D.; Hassani, S.; Sabetzadeh, M. Improving requirements completeness: automated assistance through large language models. *Requirements Engineering* **2024**, *29*. https://doi.org/10.1007/s00766-024-00416-3.
86. Habib, M.; Wagner, S.; Graziotin, D. Detecting Requirements Smells with Deep Learning: Experiences, Challenges and Future Work. In Proceedings of the IEEE International Conference on Requirements Engineering, 2021, Vol. 2021-September. https://doi.org/10.1109/REW53955.2021.00027.
87. Liu, C.; Zhao, Z.; Zhang, L.; Li, Z. Automated conditional statements checking for complete natural language requirements specification. *Applied Sciences (Switzerland)* **2021**, *11*. https://doi.org/10.3390/app11177892.
88. S., S.; L.P., S.; S., B. A study on Quality Assessment of Requirement Engineering Document using Text Classification Technique. In Proceedings of the 2nd International Conference on Electronics and Sustainable Communication Systems, ICESC 2021, 2021. https://doi.org/10.1109/ICESC51422.2021.9532736.
89. Moharil, A.; Sharma, A. Identification of Intra-Domain Ambiguity using Transformer-based Machine Learning. In Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering, NLBSE 2022, 2022. https://doi.org/10.1145/3528588.3528651.
90. Subedi, I.; Singh, M.; Ramasamy, V.; Walia, G. Application of back-translation: A transfer learning approach to identify ambiguous software requirements. In Proceedings of the ACMSE Conference - ACMSE 2021: The Annual ACM Southeast Conference, 2021. https://doi.org/10.1145/3409334.3452068.
91. Onyeka, E.; Varde, A.; Anu, V.; Tandon, N.; Daramola, O. Using Commonsense Knowledge and Text Mining for Implicit Requirements Localization. In Proceedings of the International Conference on Tools with Artificial Intelligence, ICTAI, 2020, Vol. 2020-November. https://doi.org/10.1109/ICTAI50040.2020.00146.
92. Femmer, H.; Müller, A.; Eder, S. Semantic Similarities in Natural Language Requirements. In Proceedings of the Lecture Notes in Business Information Processing, 2020, Vol. 371 LNBIP. https://doi.org/10.1007/978-3-030-35510-4_6.
93. Memon, K.; Xiaoling, X. Deciphering and analyzing software requirements employing the techniques of Natural Language processing. In Proceedings of the ACM International Conference Proceeding Series, 2019. https://doi.org/10.1145/3325730.3325757.
94. Atoum, I. Measurement of key performance indicators of user experience based on software requirements. *Science of Computer Programming* **2023**, *226*. https://doi.org/10.1016/j.scico.2023.102929.
95. Althar, R.; Samanta, D. BERT-Based Secure and Smart Management System for Processing Software Development Requirements from Security Perspective. In Proceedings of the Lecture Notes on Data Engineering and Communications Technologies, 2022, Vol. 132. https://doi.org/10.1007/978-981-19-2347-0_34.
96. Imtiaz, S.; Amin, M.; Do, A.; Iannucci, S.; Bhowmik, T. Predicting Vulnerability for Requirements. In Proceedings of the IEEE 22nd International Conference on Information Reuse and Integration for Data Science, IRI 2021, 2021. https://doi.org/10.1109/IRI51335.2021.00028.
97. A.-J.Aberkane. Automated GDPR-compliance in requirements engineering. In Proceedings of the CEUR Workshop Proceedings, 2021, Vol. 2906.

98. dos Santos, R.; Villela, K.; Avila, D.; Thom, L. A practical user feedback classifier for software quality characteristics. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2021, Vol. 2021-July. https://doi.org/10.18293/SEKE2021-055.
99. Atoum, I.; Almalki, J.; Alshahrani, S.; Shehri, W. Towards Measuring User Experience based on Software Requirements. *International Journal of Advanced Computer Science and Applications* **2021**, *12*. https://doi.org/10.14569/IJACSA.2021.0121137.
100. Hovorushchenko, T.; Pavlova, O. Method of activity of ontology-based intelligent agent for evaluating initial stages of the software lifecycle. In Proceedings of the Advances in Intelligent Systems and Computing, 2019, Vol. 836. https://doi.org/10.1007/978-3-319-97885-7_17.
101. Groher, I.; Seyff, N.; Iqbal, T. Towards automatically identifying potential sustainability effects of requirements. In Proceedings of the CEUR Workshop Proceedings, 2019, Vol. 2541.
102. Chazette, L. Mitigating challenges in the elicitation and analysis of transparency requirements. In Proceedings of the IEEE International Conference on Requirements Engineering, 2019, Vol. 2019-September. https://doi.org/10.1109/RE.2019.00064.
103. Adithya, V.; Deepak, G. OntoReq: An Ontology Focused Collective Knowledge Approach for Requirement Traceability Modelling. In Proceedings of the Lecture Notes in Networks and Systems, 2021, Vol. 239 LNNS. https://doi.org/10.1007/978-3-030-77246-8_34.
104. Fadhlurrohman, D.; Sabariah, M.; Alibasa, M.; Husen, J. Naive Bayes Classification Model for Precondition-Postcondition in Software Requirements. In Proceedings of the 2023 International Conference on Data Science and Its Applications, ICoDSA 2023, 2023. https://doi.org/10.1109/ICoDSA58501.2023.10277397.
105. Sawada, K.; Pomerantz, M.; Razo, G.; Clark, M. Intelligent requirement-to-test-case traceability system via Natural Language Processing and Machine Learning. In Proceedings of the IEEE 9th International Conference on Space Mission Challenges for Information Technology, SMC-IT 2023, 2023. https://doi.org/10.1109/SMC-IT56444.2023.00017.
106. Subedi, I.; Singh, M.; Ramasamy, V.; Walia, G. Classification of Testable and Valuable User Stories by using Supervised Machine Learning Classifiers. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2021, 2021. https://doi.org/10.1109/ISSREW53611.2021.00111.
107. Merugu, R.; Chinnam, S. Automated cloud service based quality requirement classification for software requirement specification. *Evolutionary Intelligence* **2021**, *14*. https://doi.org/10.1007/s12065-019-00241-6.
108. Ueda, K.; Tsukada, H. Accuracy Improvement by Training Data Selection in Automatic Test Cases Generation Method. In Proceedings of the 2021 9th International Conference on Information and Education Technology, ICIET 2021, 2021. https://doi.org/10.1109/ICIET51873.2021.9419636.
109. Kikuma, K.; Yamada, T.; Sato, K.; Ueda, K. Preparation method in automated test case generation using machine learning. In Proceedings of the ACM International Conference Proceeding Series, 2019. https://doi.org/10.1145/3368926.3369679.
110. Kaya, A.; Keceli, A.; Catal, C.; Tekinerdogan, B. Model analytics for defect prediction based on design-level metrics and sampling techniques. In Proceedings of the Model Management and Analytics for Large Scale Systems, 2019. https://doi.org/10.1016/B978-0-12-816649-9.00015-6.
111. Petcuşin, F.; Stănescu, L.; Bădică, C. An Experiment on Automated Requirements Mapping Using Deep Learning Methods. In Proceedings of the Studies in Computational Intelligence, 2020, Vol. 868. https://doi.org/10.1007/978-3-030-32258-8_10.
112. Zhang, J.; Yuan, M.; Huang, Z. Software Requirements Elicitation Based on Ontology Learning. In Proceedings of the Communications in Computer and Information Science, 2019, Vol. 861. https://doi.org/10.1007/978-981-15-0310-8_6.
113. Lano, K.; Kolahdouz-Rahimi, S.; Fang, S. Model Transformation Development Using Automated Requirements Analysis, Metamodel Matching, and Transformation by Example. *ACM Transactions on Software Engineering and Methodology* **2022**, *31*. https://doi.org/10.1145/3471907.
114. Koscinski, V.; Gambardella, C.; Gerstner, E.; Zappavigna, M.; Cassetti, J.; Mirakhorli, M. A Natural Language Processing Technique for Formalization of Systems Requirement Specifications. In Proceedings of the IEEE International Conference on Requirements Engineering, 2021, Vol. 2021-September. https://doi.org/10.1109/REW53955.2021.00062.
115. Yanuarifiani, A.; Chua, F.F.; Chan, G.Y. Feasibility Analysis of a Rule-Based Ontology Framework (ROF) for Auto-Generation of Requirements Specification. In Proceedings of the IEEE International Conference on Artificial Intelligence in Engineering and Technology, IICAIET 2020, 2020. https://doi.org/10.1109/IICAIET49801.2020.9257838.

116. Arulmohan, S.; Meurs, M.J.; Mosser, S. Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2023, 2023. https://doi.org/10.1109/MODELS-C59198.2023.00096.
117. Saini, R.; Mussbacher, G.; Guo, J.; Kienzle, J. Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling* **2022**, *21*. https://doi.org/10.1007/s10270-021-00942-6.
118. Zhang, J.; Chen, S.; Hua, J.; Niu, N.; Liu, C. Automatic Terminology Extraction and Ranking for Feature Modeling. In Proceedings of the IEEE Int'l Conf. on Requirements Engineering, 2022, Vol. 2022-August. https://doi.org/10.1109/RE54965.2022.00012.
119. Sree-Kumar, A.; Planas, E.; Clarisó, R. Validating Feature Models with Respect to Textual Product Line Specifications. In Proceedings of the ACM International Conference Proceeding Series, 2021. https://doi.org/10.1145/3442391.3442407.
120. Bonner, M.; Zeller, M.; Schulz, G.; Beyer, D.; Olteanu, M. Automated Traceability between Requirements and Model-Based Design. In Proceedings of the CEUR Workshop Proceedings, 2023, Vol. 3378.
121. Bashir, N.; Bilal, M.; Liaqat, M.; Marjani, M.; Malik, N.; Ali, M. Modeling Class Diagram using NLP in Object-Oriented Designing. In Proceedings of the IEEE 4th National Computing Colleges Conference, NCCC 2021, 2021. https://doi.org/10.1109/NCCC49330.2021.9428817.
122. Vineetha, V.; Samuel, P. A Multinomial Naïve Bayes Classifier for identifying Actors and Use Cases from Software Requirement Specification documents. In Proceedings of the 2nd International Conference on Intelligent Technologies, CONIT 2022, 2022. https://doi.org/10.1109/CONIT55038.2022.9848290.
123. Schouten, M.; Ramackers, G.; Verberne, S. Preprocessing Requirements Documents for Automatic UML Modelling. In Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2022, Vol. 13286 LNCS. https://doi.org/10.1007/978-3-031-08473-7_17.
124. Imam, A.; Alhroob, A.; Alzyadat, W. SVM Machine Learning Classifier to Automate the Extraction of SRS Elements. *International Journal of Advanced Computer Science and Applications* **2021**, *12*. https://doi.org/10.14569/IJACSA.2021.0120322.
125. Leitão, V.; Medeiros, I. SRXCRM: Discovering Association Rules between System Requirements and Product Specifications. In Proceedings of the CEUR Workshop Proceedings, 2021, Vol. 2857.
126. Sonbol, R.; Rebdawi, G.; Ghneim, N. Towards a Semantic Representation for Functional Software Requirements. In Proceedings of the 7th International Workshop on Artificial Intelligence and Requirements Engineering, AIRE 2020, 2020. https://doi.org/10.1109/AIRE51212.2020.00007.
127. Saini, R.; Mussbacher, G.; Guo, J.; Kienzle, J. Towards Queryable and Traceable Domain Models. In Proceedings of the IEEE International Conference on Requirements Engineering, 2020, Vol. 2020-August. https://doi.org/10.1109/RE48521.2020.00044.
128. Tiwari, S.; Rathore, S.; Sagar, S.; Mirani, Y. Identifying Use Case Elements from Textual Specification: A Preliminary Study. In Proceedings of the IEEE International Conference on Requirements Engineering, 2020, Vol. 2020-August. https://doi.org/10.1109/RE48521.2020.00059.
129. Saini, R.; Mussbacher, G.; Guo, J.; Kienzle, J. A Neural Network Based Approach to Domain Modelling Relationships and Patterns Recognition. In Proceedings of the 10th International Model-Driven Requirements Engineering Workshop, MoDRE 2020, 2020. https://doi.org/10.1109/MoDRE51215.2020.00016.
130. Saini, R.; Mussbacher, G.; Guo, J.; Kienzle, J. DoMoBOT: A bot for automated and interactive domain modelling. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings, 2020. https://doi.org/10.1145/3417990.3421385.
131. Ghosh, S.; Bashar, R.; Mukherjee, P.; Chakraborty, B. Automated generation of e-r diagram from a given text in natural language. In Proceedings of the International Conference on Machine Learning and Data Engineering, iCMLDE 2018, 2019. https://doi.org/10.1109/iCMLDE.2018.00026.
132. Khan, J. Mining requirements arguments from user forums. In Proceedings of the IEEE International Conference on Requirements Engineering, 2019, Vol. 2019-September. https://doi.org/10.1109/RE.2019.00059.
133. Osman, M.; Alabwaini, N.; Jaber, T.; Alrawashdeh, T. Generate use case from the requirements written in a natural language using machine learning. In Proceedings of the 2019 IEEE Jordan International

Joint Conference on Electrical Engineering and Information Technology, JEEIT 2019 - Proceedings, 2019. https://doi.org/10.1109/JEEIT.2019.8717428.

134. Wever, M.; Rooijen, L.V.; Hamann, H. Multioracle coevolutionary learning of requirements specifications from examples in on-the-fly markets. *Evolutionary Computation* **2019**, *28*. https://doi.org/10.1162/evco_a_00266.
135. Motger, Q.; Borrull, R.; Palomares, C.; Marco, J. OpenReq-DD: A requirements dependency detection tool. In Proceedings of the CEUR Workshop Proceedings, 2019, Vol. 2376.
136. Deshpande, G.; Arora, C.; Ruhe, G. Data-driven elicitation and optimization of dependencies between requirements. In Proceedings of the IEEE International Conference on Requirements Engineering, 2019, Vol. 2019-September. https://doi.org/10.1109/RE.2019.00055.
137. Atas, M.; Samer, R.; Felfernig, A. Automated Identification of Type-Specific Dependencies between Requirements. In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI 2018, 2019. https://doi.org/10.1109/WI.2018.00-10.
138. Deshpande, G. SReYantra: Automated software requirement inter-dependencies elicitation, analysis and learning. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019, 2019. https://doi.org/10.1109/ICSE-Companion.2019.00076.
139. Wang, B.; Deng, Y.; Wan, H.; Li, X. DF4RT: Deep Forest for Requirements Traceability Recovery Between Use Cases and Source Code. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2023. https://doi.org/10.1109/SMC53992.2023.10394259.
140. Al-walidi, N.; Azab, S.; Khamis, A.; Darwish, N. Clustering-based Automated Requirement Trace Retrieval. *International Journal of Advanced Computer Science and Applications* **2022**, *13*. https://doi.org/10.14569/IJACSA.2022.0131292.
141. A.C., M.; R., L.; Ó., P.; C., C. Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train control and management case. *Journal of Systems and Software* **2020**, *163*. https://doi.org/10.1016/j.jss.2020.110519.
142. Wang, S.; Li, T.; Yang, Z. Exploring Semantics of Software Artifacts to Improve Requirements Traceability Recovery: A Hybrid Approach. In Proceedings of the Asia-Pacific Software Engineering Conference, APSEC, 2019, Vol. 2019-December. https://doi.org/10.1109/APSEC48747.2019.00015.
143. Chen, L.; Wang, D.; Wang, J.; Wang, Q. Enhancing Unsupervised Requirements Traceability with Sequential Semantics. In Proceedings of the Asia-Pacific Software Engineering Conference, APSEC, 2019, Vol. 2019-December. https://doi.org/10.1109/APSEC48747.2019.00013.
144. Talele, P.; Phalnikar, R. Automated Requirement Prioritisation Technique Using an Updated Adam Optimisation Algorithm. *International Journal of Intelligent Systems and Applications in Engineering* **2023**, *11*.
145. Fatima, A.; Fernandes, A.; Egan, D.; Luca, C. Software Requirements Prioritisation Using Machine Learning. In Proceedings of the International Conference on Agents and Artificial Intelligence, 2023, Vol. 3. https://doi.org/10.5220/0011796900003393.
146. Lunarejo, M. Requirements prioritization based on multiple criteria using Artificial Intelligence techniques. In Proceedings of the IEEE International Conference on Requirements Engineering, 2021. https://doi.org/10.1109/RE51729.2021.00072.
147. Limaylla, M.; Condori-Fernandez, N.; Luaces, M. Towards a Semi-Automated Data-Driven Requirements Prioritization Approach for Reducing Stakeholder Participation in SPL Development †. *Engineering Proceedings* **2021**, *7*. https://doi.org/10.3390/engproc2021007027.
148. Reyad, O.; Dukhan, W.; Marghny, M.; Zanaty, E. Genetic K-Means Adaption Algorithm for Clustering Stakeholders in System Requirements. In Proceedings of the Advances in Intelligent Systems and Computing, 2021, Vol. 1339. https://doi.org/10.1007/978-3-030-69717-4_21.
149. Talele, P.; Phalnikar, R. Software Requirements Classification and Prioritisation Using Machine Learning. In Proceedings of the Lecture Notes in Networks and Systems, 2021, Vol. 141. https://doi.org/10.1007/978-981-15-7106-0_26.
150. Pereira, F.; Neto, G.; Lima, L.D.; Silva, F.; Peres, L. A Tool For Software Requirement Allocation Using Artificial Intelligence Planning. In Proceedings of the IEEE International Conference on Requirements Engineering, 2022, Vol. 2022-August. https://doi.org/10.1109/RE54965.2022.00032.
151. Blincoe, K.; Dehghan, A.; Salaou, A.D.; Neal, A.; Linaker, J.; Damian, D. High-level software requirements and iteration changes: a predictive model. *Empirical Software Engineering* **2019**, *24*. https://doi.org/10.1007/s10664-018-9656-z.

152. Liu, Y.; Lin, J.; Cleland-Huang, J.; Vierhauser, M.; Guo, J.; Lohar, S. SENET: A Semantic Web for Supporting Automation of Software Engineering Tasks. In Proceedings of the 7th International Workshop on Artificial Intelligence and Requirements Engineering, AIRE 2020, 2020. https://doi.org/10.1109/AIRE51212.2020.00011.
153. Khan, B.; Naseem, R.; Alam, I.; Khan, I.; Alasmary, H.; Rahman, T. Analysis of Tree-Family Machine Learning Techniques for Risk Prediction in Software Requirements. *IEEE Access* **2022**, *10*. https://doi.org/10.1109/ACCESS.2022.3206382.
154. Zamani, K. A Prediction Model for Software Requirements Change Impact. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, 2021. https://doi.org/10.1109/ASE51524.2021.9678582.
155. Cherdsakulwong, N.; Suwannasart, T. Impact Analysis of Test Cases for Changing Inputs or Outputs of Functional Requirements. In Proceedings of the 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2019, 2019. https://doi.org/10.1109/SNPD.2019.8935754.
156. Olson, R.; Bartley, N.; Urbanowicz, R.; Moore, J. Evaluation of a tree-based pipeline optimization tool for automating data science. In Proceedings of the Proc. Conf. on Genetic and Evolutionary Computation GECCO'16, Denver, 2016, p. 485–492.