Journal of Software Engineering Research and Development, 2025, 13:16, doi: 10.5753/jserd.2025.5097

Natural Language Processing in Software Engineering: A Systematic Literature Review

Gabriel Nogueira Pacheco [**Universidade Federal de São Paulo** | ***gpacheco@unifesp.br***]
Luiz Eduardo Galvão Martins [**Universidade Federal de São Paulo** | ***legmartins@unifesp.br***]
Ana Estela Antunes da Silva [**Universidade Estadual de Campinas** | ***aeasilva@unicamp.br***]
Niklas Lavesson [**Blekinge Institute of Technology** | ***niklas.lavesson@bth.se***]
Tony Gorschek [**Blekinge Institute of Technology** | ***tony.gorschek@bth.se***]

Abstract

Context: Software engineering (SE) artifacts and documents, such as requirements specifications, user stories, test cases, and concepts of operations (ConOps), are typically written in natural language, making their manipulation challenging. Natural Language Processing (NLP) is a viable solution for managing these tasks. *Objective*: To conduct a systematic literature review to explore the current use of NLP in SE artifacts and tasks, supplemented by a tertiary study focusing on the emerging role of Large Language Models (LLMs) in software engineering research. *Method*: We searched digital libraries for relevant papers and applied inclusion and exclusion criteria to filter the primary studies. We then analyzed NLP techniques applied to SE documents and examined their usage in this context. Our research methodology followed Kitchenham and Charters' guidelines. Additionally, we conducted a tertiary study to synthesize findings from existing systematic literature reviews and surveys specifically addressing LLMs in software engineering. *Results*: We selected 60 primary studies to identify the most common methods for NLP pipelines, feature extraction, language models, and machine learning algorithms used in SE. We also assessed the purposes of these methods, their benefits for SE, their difficulty, and their contribution to SE advancement. The tertiary study revealed a rapid proliferation of LLM-focused research, with comprehensive reviews documenting exponential growth in publications and widespread adoption across diverse SE tasks. *Conclusion*: Requirements are the most frequently addressed artifacts using NLP techniques, with preprocessing and part-of-speech (POS) tagging being widely used. There is a notable increase in the use of large language models for various SE tasks, such as requirements elicitation, source code generation, bug fixing, and software testing. The tertiary study confirms that LLMs represent a pivotal shift in the research landscape, warranting dedicated investigation to understand their transformative impact on NLP applications in software engineering.

Keywords: *Natural Language Processing, Software Engineering, Machine Learning, Literature Review*

1 Introduction

Natural language (NL) is widely used in Software Engineering (SE), not only for software and system requirement specifications, but also for documenting and reporting bugs Shehadeh et al. (2021); Gomes et al. (2023); Cho et al. (2022). This is because requirements are generally written in natural language owing to the ease of comprehension by software engineering experts and even stakeholders with limited knowledge and experience in the area. However, documents written in NL are often ambiguous Dalpiaz et al. (2018). The work and effort to perform software engineering tasks that use these documents can be time-consuming and error-prone. To address this issue, Natural Language Processing (NLP) is used to manage NL documents and artifacts because of the numerous advantages of automating SE processes. NLP is also utilized to help software developers write code and easily find bugs, as well as to help software designers analyze and build object-oriented diagrams. Another problem with some SE documents is the ambiguity and variability that comes with natural language. This can be minimized using NLP techniques and artificial intelligence (AI). Software testing is also receiving attention from the NLP community as NLP can be used with test cases to generate testing code or assist experts in this task.

In response to these challenges, NLP has emerged as a pivotal solution, leveraging automation to enhance the management and interpretation of NL documents and artifacts in SE Shreda and Hanani (2021); Halim and Siahaan (2019); Wein and Briggs (2021); Shakeri Hossein Abad et al. (2019); Arora et al. (2015); Li et al. (2015); Singh (2019). Beyond mere document management, NLP aids software developers in code composition and bug identification Mastropaolo et al. (2021); Alrashedy et al. (2020); Hu et al. (2018); Rani et al. (2021), while also assisting software designers in the analysis and construction of object-oriented diagrams Jaiwai and Sammapun (2017); Yang and Sahraoui (2022); Nasiri et al. (2020); Elallaoui et al. (2018); Cheema et al. (2023). By harnessing the capabilities of NLP techniques in conjunction with AI, the ambiguities and variabilities inherent in natural language within SE documents can be significantly mitigated, for example, the work of Ezzini et al. (2022), which aimed to address ambiguity detection.

Furthermore, the intersection of NLP and software testing is gaining momentum, with NLP demonstrating potential applications in generating testing code from test-cases or aiding experts in refining their testing strategies Blasi et al. (2023); Fischbach et al. (2023); Tahvili et al. (2020); Li et al. (2020); Viggiato et al. (2022).

In the past decade, researchers have made substantial ef-

forts to enhance the application of NLP techniques in SE. The predominant focus of these endeavors lies in effectively addressing the management of NL written requirements through the formidable capabilities of NLP, primarily centered on automation tasks, such as automatic extraction of requirements.

The proliferation of NLP techniques in SE over the last decade has been closely linked to the persistent challenge of ambiguity in requirements, as highlighted by numerous primary studies within the scope of this review. Ambiguity in NL-written requirements has been identified as a pervasive issue Ezzini et al. (2022); Asadabadi et al. (2020); Fantechi et al. (2023), introducing complexities and uncertainties that impede the seamless execution of SE tasks. The inherent vagueness Cruz et al. (2017) and interpretational nuances of NL pose formidable obstacles, often resulting in time-consuming and error-prone processes.

Recognizing the critical role that requirements play in the software development lifecycle, researchers have sought to leverage NLP to effectively navigate and mitigate the challenges posed by ambiguity Dalpiaz et al. (2019). By harnessing the power of NLP, the aim is to enhance the precision and clarity of NL requirements, ultimately facilitating more efficient automation tasks and reducing the cognitive load on software engineering professionals.

The primary objective of this study is to present a comprehensive systematic literature (SLR) review that delves into the predominant NLP methodologies and techniques shaping the contemporary landscape of NLP applications in software engineering. Guided by a set of research questions (RQs), our investigation sought to understand the use and impact of NLP techniques in software engineering processes.

The first set of inquiries encapsulated by RQ1 serves as a compass to guide our exploration. RQ1 is multifaceted, encompassing queries such as: ”What are the main NLP techniques used to analyze documents and artifacts produced in Software Engineering processes?” (RQ1); ”What are the purposes of the techniques found?” (RQ1.1); ”What are the benefits of the techniques found for Software Engineering processes?” (RQ1.2); and ”What are the difficulties in using the techniques encountered?” (RQ1.3). Complementing this, RQ2 takes a broader perspective, aiming to discern the overarching impact of NLP techniques on the advancement of Software Engineering. The research questions and objectives are presented in Table 1. The rationale for this multi-part structure is to systematically deconstruct the current state of practice. RQ1 serves as the foundation, identifying what techniques are being used. Following this, RQ1.1 explores the why, linking these techniques to the specific SE problems they aim to solve. To assess their value and practical impact, RQ1.2 investigates the benefits, while RQ1.3 provides a balanced perspective by examining the challenges and difficulties of their implementation. Finally, RQ2 synthesizes these findings to understand the cumulative impact of these technologies on the advancement of the SE field as a whole.

In preparing this review, we found that while several literature reviews on this topic exist, they tend to focus on specific sub-domains or artifacts. For instance, past studies have concentrated on NLP for user stories, requirements engineering , or requirements formalization. The primary novelty of our study lies in its deliberately broad scope. We did not find other reviews that addressed the general application of NLP across the entire SE landscape. As stated in our related work analysis, this comprehensive approach allows us to identify cross-cutting trends and map the utility of specific methods across different SE stages, an analysis not possible in more narrowly focused reviews.

It is crucial to distinguish between the traditional NLP techniques that form the basis of this review and the recent, transformative emergence of LLMs. The advent of frontier LLMs has caused a significant shift in the capacity and applicability of NLP in SE. To address this, we conducted a focused tertiary study, analyzing SLRs on the application of LLMs in SE to supplement our initial findings. This analysis revealed that the field is rapidly adopting decoder-only architectures, such as the GPT series, which excel at generative tasks like code generation and program repair. Our tertiary study also highlighted that while LLM research is heavily concentrated in the software development and maintenance phases, foundational areas like requirements engineering and software design remain significantly under-explored. Therefore, while this paper provides a deep analysis of the foundational NLP techniques identified in our original scope, this updated context on LLMs is essential for situating our findings within the current, rapidly evolving state-of-the-art.

This study’s findings, which combine a systematic literature review of foundational NLP techniques with a tertiary analysis of recent work on LLMs, provide a comprehensive overview of the field’s contributions. Our primary review reveals that traditional NLP applications have been heavily concentrated on software requirements artifacts, using techniques like POS-tagging and models such as BERT to automate analysis and handle ambiguity. In contrast, our analysis of the LLM landscape shows a significant shift in focus towards generative, code-related tasks. This recent wave of research is predominantly applied to the software development and maintenance phases, with code generation and program repair being the most prevalent applications Hou et al. (2024). A key insight from this combined analysis is the identification of a critical gap: while foundational NLP has centered on requirements, the powerful new generative LLMs have so far largely overlooked this area, presenting a significant opportunity for future research.

The remainder of this paper is organized as follows: Section 2 presents the theoretical background and reviews related work in natural language processing for software engineering. Section 3 outlines the SLR methodology. Section 4 presents the findings and analysis. Section 5 discusses the implications of the results, and Section 6 concludes with key insights and future research directions.

2 Background and Related Work

This section describes the necessary information about SE and NLP, their usual applications and the literature reviews that addressed similar topics related to SE.

2.1 Natural Language Processing for Software Engineering

Developing reliable software is difficult and laborious. SE techniques make development easier by presenting methods for organizing software products development, usually through teamwork, such as agile methods. SE processes are generally composed of developmental stages (e.g., planning, design, coding and testing). In the planning stage, the specifications of the software are created. These are called software requirements, which are the requirements that need to be included in the software final product, and are usually the guides for the entire development. Requirements are the most common software document, generally written in NL, mostly because of the interaction between stakeholders (clients, managers, developers, etc.). There are other types of requirements specifications, such as user stories and use cases. These requirements are more organized despite being written in NL, because they have specific formats that need to be followed.

In the design stage, software engineers create drafts to specify a new model to be developed. A common method for drafting models is the design of diagrams. Unified Modeling Language (UML) is a commonly used tool for creating object-oriented diagrams that help software engineers model new products. In the coding stage, the developers transform the model into code: they write the source code of the software being developed to include the functionality specified by the requirements. In the testing stage, the software is tested to verify bugs and errors before deploying the software in production.

Processing text written in NL is challenging. For this purpose, NLP studies techniques and methods that can manipulate human-language content. Natural Language Processing is divided into two sub-fields: Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU attempts to make machines comprehend natural language text, syntax, and semantics. Examples of NLU applications are word embeddings, which are semantic vectors that can relate similar words by proximity. On the other hand, NLG attempts to make machines generate text like a human being. An example of an application of NLG is chat bots, which have recently become popular. NLP pipelines almost always include some sort of preprocessing such as text cleaning, tokenization, POS tagging, and parsing. Vectorization is a common method for transforming text into numeric vectors to fit as an input to machine learning algorithms. A vast crescent area of NLP is the use of LLMs, which are machine learning models that are trained in a language with a very large corpus of text, and can be used to perform many NLP tasks such as sentiment analysis, text generation, machine translation, summarization, question answering and many others.

2.2 Related Work

We were unable to find literature reviews that focused on NLP in the general context of software engineering. Some reviews related to SE have focused on specific areas or software artifacts. Raharjana et al. (2021) conducted an SLR focused on NLP being applied to user stories, a specific-format type of requirements document. One of their findings indicates that POS tagging is the most commonly utilized NLP technique for processing user stories, although semantic approaches are also beginning to emerge. Wang et al. (2023b) conducted a literature review on applications of machine learning (ML) and deep learning (DL) in software engineering, not only focusing on NLP. The authors noted enhancements achieved by DL classification models over ML, as well as unique contributions by DL to SE tasks that traditional ML alone can not address.

Liu et al. (2022) conducted a literature review about the state-of-the-art artificial intelligence (AI) and its branches (machine learning and NLP) in the field of requirements engineering. Their study highlighted that certain NLP techniques enhance ML performance, whereas others may have a detrimental effect on performance. Omran and Treude (2017) published a systematic literature review to analyze the usage of NLP libraries by software engineering papers and how the choice of library was justified. Their analysis revealed that despite not being utilized in the papers under review, spaCy demonstrated the most promising accuracy. However, they also noted a significant potential for further enhancement. Casamayor et al. (2011) conducted a state-of-the-art review of the application of text mining to architectural software design. Gupta and Gupta (2019) published a literature review investigating the application of NLP in the field of Mining Software Repositories, which mainly focused on sentiment analysis, summarization, traceability, norm mining, and mobile analytics. Kolahdouz-Rahimi et al. (2023) conducted a systematic literature review to outline state-of-the-art NLP and ML in requirements formalization. They discovered that heuristic NLP approaches are the most common requirements formalization techniques, primarily operating on structured and semi-structured data. Sonbol et al. (2022) conducted a systematic literature mapping to investigate the use of NLP-based text representation techniques to support requirements engineering tasks. Their findings suggest that approximately two-thirds of the retrieved publications address tasks such as requirements classification, requirements traceability, ambiguity detection, and extracting requirements from reviews and documents. In addition, there is a growing trend in the utilization of advanced word-embedding techniques. While existing literature reviews provide valuable, in-depth analysis of specific domains, such as NLP for user stories, requirements engineering, or bug fixing, our study's broader scope yields several novel insights that are not apparent from a more targeted perspective. Unlike focused reviews, our work identifies which NLP techniques are versatile enough to be applied across multiple stages of the software lifecycle. For example, we found that techniques such as Named-Entity Recognition (NER) and TF-IDF are used in studies related to both the requirements and coding stages, whereas other methods like Latent Dirichlet Allocation (LDA) and constituency parsing were identified exclusively in studies concerning requirements. This allows us to map the utility of specific methods across the SE process, an analysis not possible in reviews confined to a single area.

Recent SLRs have begun to map the impact of LLMs on SE. A notable example is a comprehensive SLR by Hou et al. (2024) that synthesized 395 research papers published up to

January 2024. This review identified over 70 distinct LLMs being applied to SE tasks, which are categorized into three primary architectures: encoder-only, encoder-decoder, and decoder-only. There is a clear trend towards decoder-only models, like the GPT series, which appeared in 195 of the reviewed papers and excel at generative tasks. The application of these models has been cataloged across 85 different SE tasks, with the majority of research heavily concentrated in the software development (56.65% of studies) and software maintenance (22.71% of studies) phases. Specifically, code generation and program repair are the most prevalent tasks in these domains. In stark contrast, foundational areas remain significantly under-researched, including requirements engineering (3.90% of studies), software design (0.92% of studies), and software management (0.69% of studies). The SLR also highlighted persistent challenges such as the massive computational cost of LLMs, their dependency on high-quality data, and the risk of generating incorrect code. However, it also pointed to major opportunities, including the development of code-specialized LLMs, the expansion of applications into less-explored SE phases, and the emergence of a new reciprocal field known as ”Software Engineering for Large Language Models” (SE4LLM). This body of work on LLMs defines the current frontier of research and provides essential context for understanding the findings on more foundational NLP techniques. In their 2023 survey, Fan et al. (2023) outline the emerging landscape of Large Language Models (LLMs) for Software Engineering (SE), focusing on both their potential and key challenges. The authors' central finding is that while the emergent, creative properties of LLMs can be applied across the entire SE spectrum, they also introduce significant risks of incorrect ”hallucinated” outputs. Consequently, the survey highlights the pivotal role of hybrid techniques, where traditional SE methods like automated testing are used to filter and validate the creative but potentially flawed results from LLMs. The paper also identifies a notable research gap, finding that areas like requirements engineering and design are surprisingly underrepresented in the literature. Key open problems identified include the need for advanced prompt engineering and more robust scientific evaluation frameworks to manage the non-deterministic nature of these models. The work of Ozkaya (2023) frames the rise of LLMs as a pivotal but risky turning point for SE, advocating for a balanced approach of bold experimentation and caution. The paper outlines opportunities for LLMs to improve tasks like specification and test case generation , while highlighting significant risks such as amplifying biases from training data, concerns over content ownership, and a lack of model explainability that undermines trust. Ultimately, they argues the most profound implication is the urgent need to reform SE education to teach future engineers how to validate AI-assisted outputs, treat data as code, and prioritize ethics. To supplement our findings and address the recent, transformative impact of LLMs, we conducted a focused tertiary study analyzing those three review papers. Our work distinguishes itself from existing SLRs through its deliberately broad scope, aiming to provide a comprehensive overview of NLP applications across the entire field of Software Engineering. While other literature reviews tend to focus on specific aspects or subdomains within this intersection, our approach seeks to capture the full breadth of how NLP techniques are being applied to SE challenges, providing a broad perspective on this rapidly evolving landscape.

3 Methodology

In this section, we present the planning, conducting, and reporting of the literature review. The research methodology for this review was based on the guidelines of Kitchenham and Charters (2007). The rationale for this choice is grounded in the framework's suitability for the specific context of SE research. These guidelines provide a rigorous and transparent three-phase process, comprising Planning, Conducting, and Reporting, as illustrated in Figure 1, which is essential for minimizing bias and ensuring the replicability of the review. We investigated whether similar work was conducted by searching the *Compendex* and *Google Scholar* digital libraries. The following string was used to search for titles and abstracts:

(Natural Language Processing) AND (Software Engineering) AND (Systematic Literature Review).

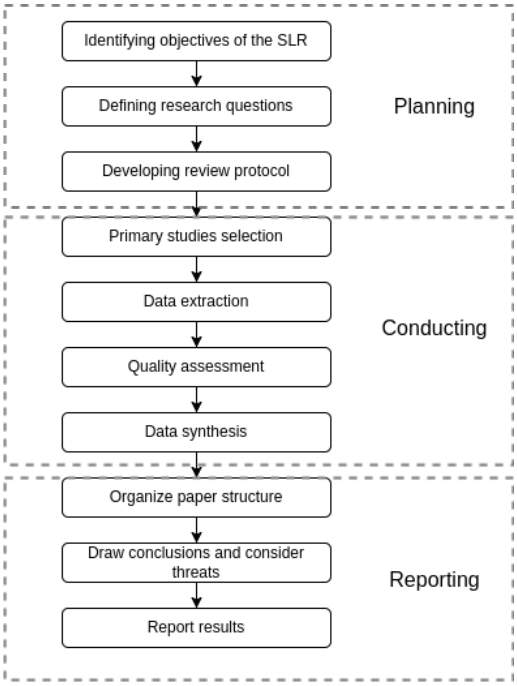

Figure 1. Steps in the SLR process

None of the retrieved literature reviews completely answered the research questions. The research questions and their objectives are described in Table 1. The main aim of this study is to survey and analyze state-of-the-art NLP techniques applied to Software Engineering documents.

3.1 Planning

Here, we present the search strategy and eligibility criteria defined in our protocol for conducting the review. The re-

Table 1. Research questions for the literature review

ID	Research question	Objective
RQ1	What are the main NLP techniques used to analyze documents and artifacts produced in Software Engineering processes?	To identify the main NLP approaches used in Software Engineering artifacts
RQ1.1	What are the purposes of the techniques found?	To understand for what purpose the techniques are used
RQ1.2	What are the benefits of the techniques found for Software Engineering processes?	To identify the advantages of using the techniques in Software Engineering
RQ1.3	What are the difficulties in using the techniques encountered?	To present the difficulties in using the approaches
RQ2	How has the use of NLP techniques contributed to the advancement of Software Engineering?	To verify the improvement of the Software Engineering processes due to NLP

view protocol was defined according to research questions and objectives.

3.1.1 Search strategy

The set of keywords used to formulate the search string is presented in Table 2. They were refined after a trial search of the selected digital libraries. The keyword ”Document” was used to filter out studies that were not related to SE documents and artifacts. Connecting the set of keywords with Boolean operators yields a defined search string:

(”Software Engineering”) AND (”Natural Language Processing” OR ”NLP”) AND (”Method” OR ”Process” OR ”Tool”) AND (”Document*”).

We limited the search to studies published between 2013 and 2023 to obtain the latest state-of-the-art approaches. The resources chosen for searching the studies were IEEE Xplore, ScienceDirect (Elsevier), and ACM Digital Library. We originally planned to use Springer Link, but the relevant papers retrieved were all book chapters that we did not have access to.

Table 2. Keywords for the search string

Keyword	Synonyms
Method	Process, Tool
Natural Language Processing	NLP
Software Engineering	
Document	

3.1.2 Eligibility criteria

We selected studies that were published as scientific articles in journals, conferences, magazines, transactions, and workshops.

Inclusion criteria: Studies that present or discuss approaches that apply Natural Language Processing to software engineering artifacts.

Exclusion criteria: Studies written in a language different from English, studies without an abstract, non-primary studies, studies published as doctoral proposals, studies that are older versions of other papers and studies not accessible to download.

We used the title and abstract of the articles to select studies based on the inclusion and exclusion criteria; however, when necessary, we opened the full article to evaluate the selection criteria. After that, we downloaded the papers and checked again for eligibility criteria and duplicates, removing studies that were not eligible.

3.2 Conducting

This section presents the conducting phase of the SLR. Herein, we present the primary study selection, data extraction, quality assessment, and data synthesis.

3.2.1 Primary studies selection

Studies were selected by reading the title and abstract and applying the inclusion and exclusion criteria presented in Section 3.1.2. For some studies, we needed to read parts of the text to be certain about their inclusion. We used the string defined in Section 3.1.1 to select studies from the chosen databases and applied the filter of publication date between 2013 and 2023. In the first search (Phase 1), we obtained 2510 studies from the IEEE Xplore, ScienceDirect, and ACM Digital Library. After applying the selection criteria to these studies (Phase 2), we selected 85 studies for data extraction and stored the study information on a spreadsheet. The final reduction in the number of studies from 85 to 60 occurred during Phase 3, the data extraction stage, after a full-text reading of the selected papers. Several studies were excluded at this point for specific reasons: we identified some papers that were older, preliminary versions of another study already included in our selection. Others were found to be duplicates published in a different venue. Furthermore, a small number of studies were removed because we were unable to obtain access to the full-text for a complete analysis. Finally, one study was excluded after a notice from the source library cast doubt on the integrity of the peer-review process for the conference where it was published. Table 3 lists the number of studies conducted in each phase.

Table 3. Number of selected studies from each library

Database	Phase 1	Phase 2	Phase 3
IEEE Xplore	124	33	24
ScienceDirect	1242	29	22
ACM Digital Library	1268	23	14
Total	2510	85	60

3.2.2 Data extraction

In the data extraction phase, we read the full text of the selected studies and extracted relevant information to answer

our research questions. However, we found that some studies were not related to NLP applied in SE, and some studies were versions of the same study but were published in another source, thus being classified as duplicates. Other studies were older versions of the same study; therefore, they were excluded. One particular study was also excluded due to a notice on the source library, pointing to evidence that casts doubt on the integrity of the peer review process for the conference in which the study was published.

We developed a form (spreadsheet) for extracting relevant information with metadata and properties related to the research questions described in Table 1. The extraction forms are listed in Table 4, and the DOI for the public data extraction form is provided in the Appendix A.

3.2.3 Quality assessment

For quality assessment, we created a quality checklist and assigned a score of 1 for each item in the study. If not, we assigned a zero, and if the study partially scored the item, then we assigned a 0.5. The maximum score was 7.0 and the minimum score was 0.0. Table 5 presents the quality scores of the selected studies.

3.3 Threats to Validity

3.3.1 Identification of primary studies

The initial pool of studies was obtained using the search string outlined in Section 3.1. It is not possible to retrieve all relevant existing studies; thus, some primary studies may have been missed, which could be a limitation of our study. In addition, our research was bounded from 2013 to 2023, so some relevant studies outside that range could also be missed and not included. Other databases that we initially planned to include for searching primary studies were not granted access to, which probably excluded some relevant papers.

3.3.2 Data extraction structure

Data extraction was performed using a predefined form tailored to our needs. The form was developed by considering the extraction attributes outlined in Section 3.2.2, which are related to the RQs addressed in the SLR. Owing to the structure of the data extraction, we believe that we limit the data that we can retrieve with the form properties; thus, important data could be missing or not properly addressed.

3.3.3 Quality assessment validity

We based the quality assessment of the primary studies on a checklist containing questions about aspects of the studies, as presented in Table 6. For each ”yes” answer, we added a score of 1.0. Otherwise, we added 0.0. When we could not answer clearly or when the question was partially answered, we added a 0.5. We present the quality scores in Section 3.2.3 (Table 5). Therefore, we believe that our choices for the questions could limit or bias the quality assessment to some extent.

3.4 Tertiary study: LLMs in SE Reviews

In the course of conducting this systematic literature review, a significant and rapidly emerging trend became apparent: the increasing application of Large Language Models (LLMs) to address a variety of software engineering challenges. While our initial review captured a broad spectrum of Natural Language Processing (NLP) techniques used in SE, the recent proliferation of studies focusing specifically on LLMs such as BERT Devlin et al. (2019), T5 Raffel et al. (2023), and their variants indicates a pivotal shift in the research landscape. The rapid growth in LLM-centric SE research warrants a more focused investigation. Recent comprehensive systematic literature reviews have documented this surge, with Hou et al. (2024) identifying 273 published papers in 2023 alone, and 46 relevant papers published within just one month in 2024, demonstrating the exponential growth of this field. The initial findings of this SLR show a notable increase in the use of LLMs for diverse SE tasks, including requirements elicitation, source code generation, bug fixing, and software testing. This surge suggests that a comprehensive understanding of the state-of-the-art now requires a dedicated analysis of how the research community is adopting and adapting these powerful models. Therefore, to supplement our primary SLR, we are conducting a tertiary study focused on existing systematic literature reviews and surveys of LLMs in software engineering. The objective of this tertiary study is to synthesize the findings from these reviews, providing a meta-level analysis of the trends, challenges, and future directions identified by other researchers in this specialized domain. This study aimed to synthesize the findings of recent reviews and surveys specifically focused on the intersection of LLMs and SE.

To achieve this, we performed a targeted search in the Arxiv and IEEE Xplore digital libraries, looking for papers with the keywords ”large language model”, ”software engineering”, and ”review” in their titles. To capture the most recent state-of-the-art, our search was limited to papers published between 2023 and 2024. The inclusion and exclusion criteria applied were the same used in our original SLR, ensuring the selection of relevant and accessible reviews. Our search yielded three review articles that were aligned with the scope of our study. These three selected reviews were then thoroughly analyzed, and their primary findings and results were summarized to provide a consolidated overview in the Results section of this study.

4 Results and Analysis

In this section we present the results and analysis of our study. We identified the artifacts that the primary studies used to apply their methods and tried to answer the research questions described in Table 1. In these studies, 40.9% of software artifacts were software requirements specifications. Software documentation was the second most used software artifact, which includes software manual documents and explicit comments in the source code. Software source code was used for various tasks and purposes, as explained in the following sections. Other types of software requirements, such as user stories and use cases, were widely used in these studies.

Table 4. Data extraction form

Data	Description	Related to
ID	Study identifier	Metadata
Database	Source of the study	Metadata
Title	Title of the paper	Metadata
Authors	Authors' names	Metadata
Source of publication	Journal, conference, etc. the study was published	Metadata
Year of publication	Year the study was published	Metadata
Country	Nation that the authors' affiliated university are located	Metadata
Approach class	Method, tool, experiment etc.	RQ1
Approach description	Description of the techniques used in the approach	RQ1
Paper motivation	Reason the authors had to develop the approach	RQ2
Purpose	For what it is used for	RQ1.1
SE artifact	What kind of artifact it is applied	RQ2
Benefits	What are the benefits of using it	RQ1.2
Difficulty	What are the difficulties in using the approach	RQ1.3
SE stage	At what stage of SE the technique is applied	RQ2
Validation context	Evaluation of the approach	RQ2

4.1 What are the main NLP techniques used to analyze documents and artifacts produced in Software Engineering processes? (RQ1)

To answer RQ1, we divided the techniques and methods into three groups: NLP pipeline techniques, feature extraction and language models. A brief definition of language models is provided in Section 4.1.3. This division was made for better organization of the analysis, as well as to group similar techniques. We also describe the machine learning algorithms that have been presented in some studies.

4.1.1 NLP pipeline techniques

The vast majority of studies used preprocessing. This includes data cleaning methods (stop-words, duplicates, and punctuation removal) and normalization methods (case folding). Part-of-speech (POS) tagging was also used to perform some NLP tasks in the SE documents, and was applied in approximately 41.6% of the total studies.

POS tagging is a technique that assigns a part-of-speech tag to each word in a sentence. Some POS-taggers are implemented using statistical algorithms such as the Hidden Markov Model Baum and Petrie (1966). Recent work in POS taggers utilizes modern techniques combining NLP and machine learning to predict the POS categories of words (e.g., Bidirectional Encoder Representations from Transformers (BERT) Devlin et al. (2019)).

Arora et al. (2015) developed a pipeline to check the conformance of templates for requirements. Their pipeline applied tokenization, POS-tagging, named entity recognition, and text-chunking to demarcate atomic noun phrases and verbal phrases, and then used a pattern-matching rule to check conformance. Jaiwai and Sammapun (2017) used POS tagging in their pipeline for processing requirements in Thai and extracted UML diagrams. Hamza and Walker (2015) developed a tool that receives a requirements document as input and processes it using POS tagging to remove adverbs, wh-adverbs, and determiners. Then, it sorts the sentences using the actor verb and removes duplicates to apply heuristics that organize and create feature models.

Tokenization is also a common method in NLP processes and is part of most NLP pipelines found in previous studies. Tokenization is a technique that divides a sentence into words and tokens. This process can be easily performed by splitting words by space or using more sophisticated methods, usually called tokenizers. Lemmatization and stemming are widely used in NLP pipelines. Lemmatization retrieves the lemma of the words, while stemming extracts the root (or base) form of the words.

Dependency and constituency parsing are the techniques used in syntactic analysis. Dependency parsing associates the grammatical relationship between words in a sentence (e.g., subject-object relationship). On the other hand, constituency parsing breaks down sentences into sub-phrases, also known as constituents (i.e., noun phrases, verb phrases, and prepositional phrases). Alrashedy et al. (2020) extracted questions from Stack Overflow and used them as inputs for a classification task. They cleaned the texts by removing non-alphanumeric characters, identified named entities using dependency parsing, and removed stop-words. Then, they performed lemmatization and stemming on the input texts to transform them into a feature matrix with term frequency-inverse document frequency (TF-IDF) and used it as input to supervised classifiers (Random Forest and XGBoost) for programming language classification. Figure 2 presents the NLP techniques found in this review.

Named entity recognition (NER) is a method used to identify named entities in text. For example, "Brazil" can be recognized as a country in pipelines containing NER modules. WordNet (Princeton) Fellbaum (1998) is a lexical database that stores synsets of words and their semantic relations, such as synonyms and antonyms. Coreference resolution is a technique that attempts to find all expressions that refers to the same entity in a text. Latent Dirichlet Allocation (LDA) is a widely used method for topic modeling.

The frequent application of these pipeline techniques goes beyond a simple count; it highlights their foundational importance for enabling more complex analyses. Syntactic processing steps like POS-tagging and dependency parsing are

Table 5. Quality scores assigned to the studies

Score	Studies
7.0	Arora et al. (2015); Sawant and Sengamedu (2022); Mastropaolo et al. (2021); Cho et al. (2022); Li et al. (2020)
6.5	Li and Nong (2022); AlDhafer et al. (2022)
6.0	Kaur and Kaur (2023); De Bortoli Fávero et al. (2022); Ezzini et al. (2022)
5.5	Gupta et al. (2013); Gomes et al. (2023); Casillo et al. (2022)
5.0	Hamza and Walker (2015); Alrashedy et al. (2020); Dalpiaz et al. (2019); Tahvili et al. (2020)
4.5	Shreda and Hanani (2021); Arunthavanathan et al. (2016); Zhai et al. (2020); Bhatia et al. (2020); Ezzini et al. (2021); Rani et al. (2021); Fischbach et al. (2023)
4.0	Jaiwai and Sammapun (2017); alsukhni (2021); Shakeri Hossein Abad et al. (2019); Stöckle et al. (2023); Yang and Sahraoui (2022); Viggiato et al. (2022)
3.5	Hu et al. (2018); Halim and Siahaan (2019); Blasi et al. (2023)
3.0	Sajid et al. (2017); Sawant et al. (2014); Nasiri et al. (2020); Asadabadi et al. (2020); Cheema et al. (2023); Li et al. (2015); Pérez et al. (2021)
2.5	Treude et al. (2015); Greghi et al. (2015); Fattahi and Mejri (2021); Wein and Briggs (2021); Ghaisas et al. (2013); Siahaan et al. (2023); Liu et al. (2014); Elallaoui et al. (2018); Cruz et al. (2017); Fantechi et al. (2023); Kadebu et al. (2023)
2.0	Shehadeh et al. (2021); Shu et al. (2016); Bidulya (2018); Arthur (2020)
1.5	Zamani (2021); Singh (2019)
1.0	Gilson and Weyns (2019); Lapeña et al. (2017)
0.5	Malhotra et al. (2016)

Table 6. Quality checklist evaluation questions

Question
Are the aims clearly stated?
Do the paper answer our research questions?
Are the approaches clearly defined?
Were the results compared to others?
If yes (previous question), were they obtained under similar circumstances?
Are the negative results presented/discussed?
Are the methods or tools used in the implementation described?

not end goals themselves but are critical for transforming unstructured natural language into a structured format that machines can interpret. This structured data is a prerequisite for subsequent tasks, as seen in the work of Arora et al. (2015), who used a full pipeline including tokenization, POS-tagging, and chunking to check the conformance of requirements templates, and Jaiwai and Sammapun (2017), who leveraged POS-tagging to extract elements for UML diagrams from requirements written in Thai. The implication is that the effectiveness of high-level SE tools, such as those for requirements validation or model generation, is deeply dependent on the quality and richness of these initial, foundational NLP processing steps.

4.1.2 Feature extraction and vectorization

Some methods in NLP are used for the extraction of features and vectorizing texts to serve as inputs to machine learning algorithms and their desired tasks. These vectors are also called *word embeddings* in the NLP and information retrieval context because they contain text information inside a vector. Term Frequency-Inverse Document Frequency (TF-IDF) is

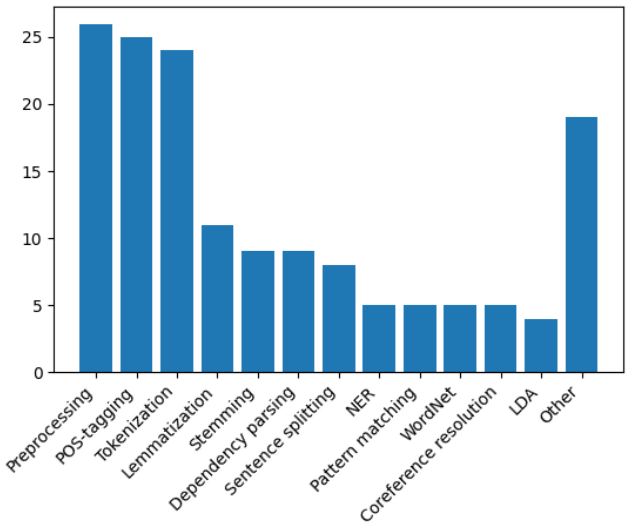

Figure 2. Frequency of techniques used in NLP

a well-known vectorization technique used to evaluate the importance of a word in a document relative to a collection of documents. It increases with the word's frequency in the document but is offset by its commonness across the corpus, highlighting significant words while reducing the weight of common ones Manning et al. (2008). Fattahi and Mejri (2021) created a tool for detecting spams from text messages and e-mails. They used bag-of-words and TF-IDF to transform the input text into vectors and used an ensemble learning algorithm for classification, combining Naive Bayes, Logistic Regression, Support Vector Machines (SVM), Nearest Centroid Classifier, XGBoost, k-Nearest Neighbours, and Perceptron. To improve the process of bug screening and fixing, Hu et al. (2018) developed a method to recommend

similar bug reports. Their approach leveraged a combination of TF-IDF and word embeddings to create rich vector representations of bug reports, calculating the similarity between them using the cosine function. This focus on semantic similarity as a performance metric proved effective, as their method improved recall-rate, mean average precision, and mean reciprocal rank by 7.89% to 8.96% compared to previous work.

Embeddings captures semantic and contextual information from word and sentences. A popular way to convert a text into a embedding vector is using word2vec Mikolov et al. (2013). Vectors created with word2vec can represent words and their semantic characteristics, allowing arithmetic operations to be performed, for example: vec(”king”) + vec(”woman”) = vec(”queen”). Shreda and Hanani (2021) used bag-of-words, TF-IDF, word2vec and BERT Devlin et al. (2019) for feature extraction. Then, they passed the features to the classifiers Naive Bayes, Support Vector Machines (SVMs), Logistic Regression and Convolutional Neural Network (CNN) to compare the results of each feature extraction approach. Finally, they combined the four feature models with CNN and Logistic Regression, which achieved the best results for non-functional requirements classification. Other alternatives to word2vec are Global Vectors for Word Representation (GloVe) Pennington et al. (2014) and FastText Bojanowski et al. (2016). BERT is widely used for word embedding; however, we describe it in more details in Section 4.1.3. Figure 3 shows the frequency of the feature extraction methods used in the studies.

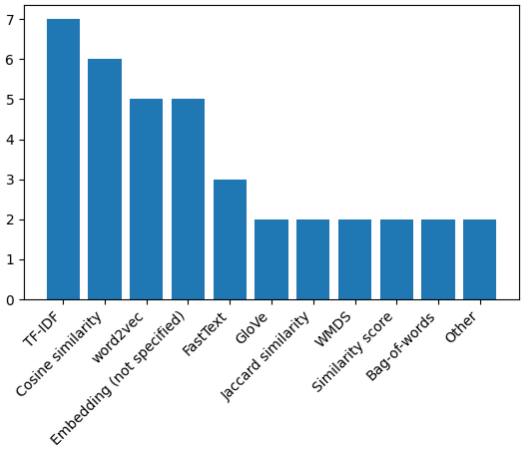

Figure 3. Frequency of feature extraction and vectorization methods

The variety of methods used for feature extraction illustrates a clear and impactful trend in the field: a shift from frequency-based representations to dense, semantic embeddings. While traditional methods like TF-IDF are still employed, often as a baseline, the move towards embeddings like word2vec signifies a crucial change in capability. These models capture the semantic context of words, allowing for a more nuanced understanding of meaning. The impact of this evolution is directly observable in studies that compare these methods. For instance, Shreda and Hanani (2021) experimented with both TF-IDF and word2vec, ultimately finding that a combination of feature models fed into a CNN achieved the best performance for classifying non-functional requirements. This shows that the choice of vectorization technique is not arbitrary but has a direct and measurable impact on the effectiveness of the final SE application.

4.1.3 Language models

Language models are neural networks trained to perform natural language tasks in one or more human languages. Usually they are trained to predict the next token or sentence, or to predict the surrounding tokens for the current token being read. Modern language models are neural networks based on the Transformer architecture Vaswani et al. (2017).

Bidirectional Encoder Representations from Transformers (BERT) Devlin et al. (2019) is a language model trained using a masked language modeling task. BERT can be interpreted as the encoder part of the Transformer model. BERT is widely used as an embedding method for machine learning algorithms because it generates contextual word embeddings. In this review, we found that BERT is the most used language model among first-generation Transformer-based language models in Software Engineering applications, as shown in Figure 4. Sawant and Sengamedu (2022) created a tool (named Doc2BP) to identify the best code practices in software documentation. They used BERT to encode embeddings to perform binary and few-shot labels classification, where there is only a limited amount of labeled data. The binary classifier part has a dropout and a linear layer with a sigmoid function, and the few-shot part has a cosine similarity score module (which is used to calculate the similarity between the support classes and the text using the cosine of the angle between the vectors) and a softmax function for multiclass classification. The practical benefits of language models are evident in their application results. To automate the classification of requirements, Kaur and Kaur (2023) developed a method using BERT as an embedding layer followed by a CNN. The performance of this approach was strong, as it outperformed baseline methods on most classes within the PROMISE dataset of functional and non-functional requirements.

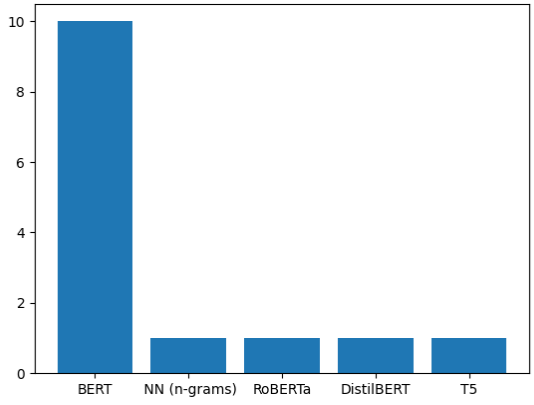

Figure 4. Frequency distribution of first-generation language models

Gomes et al. (2023) developed a method to classify long-lived bug reports. They extracted the bug reports from online resources and then labeled them as ”short-lived” and ”long-

lived" according to their fixing date-time, being "long-lived" above the 1 year threshold. They then cleaned the texts by removing non-alphabetical characters and stop-words and prepared the datasets for feature extraction, which were made using BERT and TF-IDF. Finally, they used these features to train machine learning algorithms for the classification task. Ezzini et al. (2022) developed a multi-solution method for the automatic handling of ambiguity detection and anaphora resolution in software requirements. The authors investigated six possible solutions for this purpose. For all of them, they first applied an NLP preprocessing pipeline to sentences. The pipeline consists of a tokenizer, sentence splitter, POS tagger, lemmatizer, constituency parser, dependency parser, coreference resolution, and semantic parser. Then, they tried three different solution categories: SpanBERT-based, ML-based and NLP-based. The SpanBERT-based solutions are SpanBERT-NLP, which is fine-tuned using the CoNLL2011 dataset, and SpanBERT-RE, which is fine-tuned using the DAMIR Ezzini et al. (2022) dataset from the authors. ML-based solutions: ML-LF, which is trained on language features collated from literature, and ML-FE, which is trained on feature embeddings extracted using BERT and Sentence-BERT (SBERT) Reimers and Gurevych (2019). The ML-ensemble solution is an ensemble classifier that combines the results of ML-LF and ML-FE. The NLP-based method uses two coreference resolvers on sentences.

In another application, Mastropaolo et al. (2021) aimed to support several complex, code-related tasks by fine-tuning a single encoder-decoder model, the Text-to-Text Transfer Transformer (T5). Their work successfully demonstrated the model's capacity to perform diverse generative tasks, including automatic bug fixing, the generation of asserts in test methods, and code summarization.

Frontier LLMs for SE LLMs have become a transformative force in numerous domains with a significant impact on the field of SE. The capacity of these models to process and generate human-like text has opened new frontiers for automating and augmenting various SE tasks. A recent SLR by Hou et al. (2024) provides a comprehensive analysis of this burgeoning field, synthesizing findings from 395 research articles published between January 2017 and January 2024. Our tertiary study shares the key findings from that SLR, providing an overview of how LLMs are currently being used in SE, including common data practices, optimization approaches, and the main application areas. Their review identified more than 70 different LLMs that are applied to SE tasks, which were classified into three primary architectures based on their structure and function.

Encoder-only Models: These models, such as BERT and its specialized variants like CodeBERT and GraphCode-BERT, function by processing an entire input sequence to create a hidden representation that captures the relationships between words and the overall context. Leveraging a distinctive bidirectional attention mechanism, they are highly effective in tasks that require a nuanced understanding of a complete code snippet or document. Their application is therefore primarily focused on analysis and comprehension tasks, including code review, bug report understanding, vulnerability detection, and bug localization.

Encoder-decoder Models: Architectures like T5, PLBART, and CodeT5 incorporate both an encoder module to ingest and understand the structure and semantics of an input, and a decoder module that uses this understanding to generate a target output. This design offers flexible training strategies, making them proficient in multifaceted tasks that require both comprehension and generation, such as converting code into human-readable summaries (code summarization), translating code from one programming language to another (code translation), and automated program repair.

Decoder-only Models: This category, which includes the widely adopted GPT series (GPT-3.5, GPT-4), Codex, and LLaMA, exclusively utilizes a decoder module to sequentially predict and generate text. These models have seen a surge in popularity, constituting 70.7% of the research in the first semester of 2024, largely due to their strength in generation-oriented tasks and their ability to perform well from simple instructions or a few examples without requiring extensive fine-tuning. Their primary applications in SE are consequently focused on tasks such as code generation, code completion, and test case generation.

The effectiveness of LLMs in SE applications heavily depends on the quality and nature of the underlying data used for training and fine-tuning Hou et al. (2024). Their SLR reveals distinct data practices within the SE domain, with datasets primarily sourced from four main categories: open-source, collected, constructed, and industrial repositories. Open-source datasets dominate the landscape, appearing in approximately 62.83% of studies that specified their data sources, largely due to their perceived authenticity and credibility. Notably, industrial datasets were utilized in only six studies, suggesting a significant gap between academic research and real-world industrial applications. Regarding data characteristics, text-based and code-based formats are the most frequently employed, aligning with LLMs' core strengths in natural language and code processing. The preprocessing workflows for these data types typically follow established pipelines that include data extraction, removal of unqualified entries, duplicate elimination, tokenization, and systematic partitioning into training, validation, and test sets. These preprocessing steps are crucial for ensuring data quality and model performance, though the specific approaches vary depending on the target SE application and the chosen model architecture. Beyond data preparation, optimizing LLMs for SE tasks requires careful consideration of tuning strategies, prompt engineering, and evaluation approaches Hou et al. (2024). Given the substantial computational costs associated with full model fine-tuning, researchers are increasingly adopting Parameter-Efficient Fine-Tuning (PEFT) methods, with the SLR identifying four prominent techniques: Low-Rank Adaptation (LoRA), prompt tuning, prefix tuning, and adapter tuning. Complementing these tuning approaches, prompt engineering has emerged as a powerful technique to guide model behavior without modifying core parameters, with eight distinct methods identified in the literature. Few-shot and zero-shot prompting represent the most popular approaches, utilized in 88 and 79 studies respectively, while more sophisticated methods like Chain-of-Thought (CoT) are being explored to enhance model reasoning capabilities. The assess-

ment of LLM performance in SE contexts employs metrics tailored to four primary problem types: regression, classification, recommendation, and generation tasks. Generation tasks commonly rely on metrics such as BLEU and Pass@k to evaluate code quality and correctness, while classification problems typically employ traditional measures like Precision, Recall, and F1-score to assess model accuracy and reliability.

Providing a complementary perspective, the survey by Fan et al. (2023) frames the LLM landscape around a central challenge: managing the ”emergent, creative properties” of the models, which, while powerful, also introduce a significant risk of incorrect ”hallucinated” outputs. Their central finding is the pivotal role of hybrid techniques, which combine traditional SE methods like automated testing with LLMs to filter and validate the creative but potentially flawed results. This survey also corroborates the existence of a notable research gap, finding that foundational areas like requirements engineering and design are surprisingly under-represented in the literature compared to code-centric tasks. Key open problems identified include the need for advanced prompt engineering strategies and the development of more robust scientific evaluation frameworks to manage the non-deterministic nature of these models.

4.1.4 Machine learning algorithms

Here we present some of the machine learning (ML) algorithms that were used along with NLP and vectorization techniques. Most algorithms are based on supervised learning, mainly for classification-based tasks. CNNs are widely used in many Artificial Intelligence fields, mainly within the domain of Computer Vision (image classification, image generation, segmentation, etc.). A Convolutional Neural Network (CNN) is a type of neural network particularly effective for image and video recognition tasks. It uses convolutional layers with filters to automatically and adaptively learn spatial hierarchies of features from input images, making it adept at identifying patterns such as edges, textures, and objects Krizhevsky et al. (2012). A Recurrent Neural Network (RNN) is a type of neural network designed to recognize patterns in sequences of data, such as time series or text. Unlike traditional neural networks, RNNs have connections that loop back on themselves, allowing them to maintain a ”memory” of previous inputs, which helps in processing sequential data Goodfellow et al. (2016). Despite this, many studies have employed other machine learning techniques. Naive Bayes was the most commonly used ML algorithm in the reviewed studies. Support Vector Machines are also widely used because of their excellent performance, having the same number of uses as CNNs and the LSTM recurrent networks. The performance of different machine learning algorithms is critical to their practical value in SE. In an effort to automatically classify issue reports, Cho et al. (2022) applied deep learning models (CNN and LSTM) to word embeddings generated from software user manuals. Their results demonstrated a significant improvement over the state-of-the-art, with their method yielding a 10% to 24% higher F1-score. Figure 5 shows the use of ML algorithms.

In a different context, Kadebu et al. (2023) sought to classify security-related requirements by applying a suite of traditional machine learning algorithms. Their findings provided clear performance outcomes, with Naive Bayes achieving the best results for binary classification, reporting 84% precision, 96% recall, and an 89% F1-score for the non-security requirements class. These results highlight how different algorithms can be evaluated and selected based on their measured effectiveness for a specific SE task.

Casillo et al. (2022) developed a method for detecting privacy disclosures from user stories. Their method takes three approaches: NLP-based features, privacy word’s dictionary features, and the use of transfer learning. The authors extracted NLP-based features using tokenization and then applied POS tagging, dependency parsing, and named entity recognition. These features are then converted to word embeddings and used as input to a CNN (1-dimensional convolution with dropout and max pooling). The dictionary features of privacy words (keywords, categories) are used in the same way as word embeddings and then input to a CNN. Finally, the authors fine-tuned a pre-trained CNN, concatenating the two flattened layers of the CNNs, merging them into one, and then forwarding them to the final fully connected layers and output layer (one neuron for classification).

The distribution of machine learning algorithms reveals an important pattern of co-evolution with feature extraction techniques. Simpler, traditional algorithms like Naive Bayes and SVM were frequently paired with classical feature representations like TF-IDF or bag-of-words. In contrast, deep learning models such as CNNs and LSTMs networks are almost exclusively coupled with dense semantic embeddings from models like word2vec or BERT. This strong correlation implies that the choice of a learning algorithm is highly dependent on the richness of the input features. The deep learning models are specifically chosen for their ability to learn complex patterns from the high-dimensional, contextual information that modern embeddings provide. This is demonstrated in studies like Cho et al. (2022), who used both CNN and LSTM models to effectively classify issue reports based on word embeddings derived from user manuals. This co-evolution suggests that future breakthroughs in NLP for SE could likely arise from the integrated design of both novel representation techniques and the learning architectures tailored to harness them.

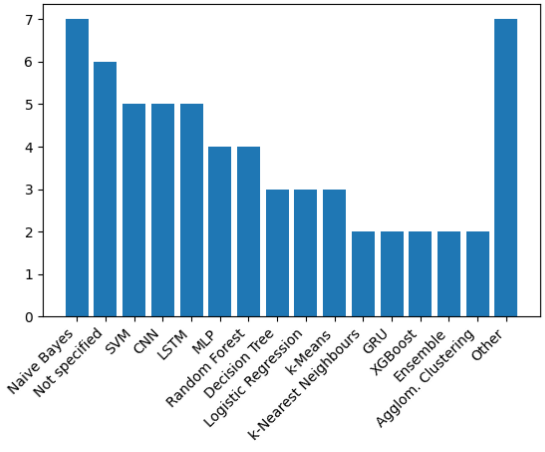

Figure 5. Frequency of machine learning algorithms

4.2 What are the purposes of the techniques found? (RQ1.1)

To answer RQ1.1, we identified some topics that grouped similar purposes: requirements classification and extraction, ambiguity handling, code and programming, software testing, bug fixing and reporting, and diagram generation. Many studies had domain-specific purposes, so it was difficult to classify them into one of the groups. Figure 6 shows the proportion of the topics and related studies.

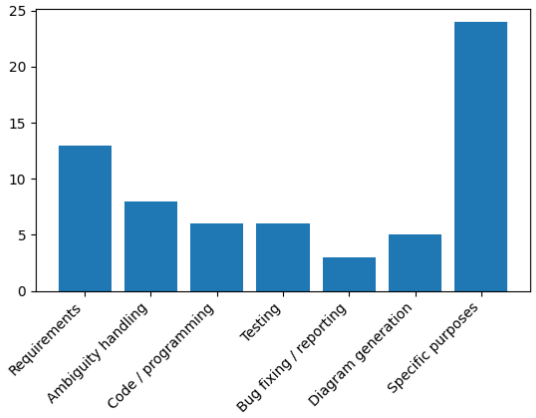

Figure 6. Categorization of purposes

4.2.1 Requirements classification and extraction

Software requirements are the predominant SE document that the studies used to apply NLP, mainly because requirements are usually written in natural language. The majority of studies have used NLP to perform some sort of requirement classification. For example, Li and Nong (2022) developed a method for the automatic classification of non-functional requirements using BERT word embeddings and Bi-LSTM recurrent neural networks.

Some studies aimed to extract or generate requirements, as seen in the work of Wein and Briggs (2021). The authors developed a method for extracting requirements from design documents using two approaches: extraction using base verbs and extraction using keywords. They applied an NLP pipeline with tokenization, POS tagging, and coreference resolution. They used a set of rules to include or exclude sentences and used TF-IDF to calculate the cosine similarity between the extracted requirements and a real database of requirements. As mentioned in Section 4.1.1, Arora et al. (2015) introduced a method for checking the conformance of requirement templates. Halim and Siahaan (2019) created a pipeline for classifying atomic and non-atomic requirements using tokenization, POS-tagging, stop-words removal, lemmatization, and ML algorithms.

4.2.2 Ambiguity handling

Ambiguity detection was the second most identified finality in the reviewed studies. Ambiguous words or sentences appear frequently in natural language texts and in the domain of software engineering, mostly in requirements. Because ambiguous requirements can be difficult to identify or resolve, some studies have attempted to address this issue. As described in Section 4.1.3, Ezzini et al. (2022) introduced a multi-solution method for automatic handling of ambiguity detection and anaphora resolution with the aim of reducing the potential for misinterpretation and the ensuing consequences of ambiguity in requirements. The same authors also performed another study Ezzini et al. (2021) with the same goal but with a different approach, published one year earlier.

Asadabadi et al. (2020) published a semi-automatic method for handling ambiguity in requirements by using NLP and fuzzy set theory. Fantechi et al. (2023) developed a tool-supported method to analyze the variability of ambiguous requirements using NLP tools and manual analysis. Another tool-supported method for detecting ambiguities in user stories using semantic similarity and visualization tools was proposed by Dalpiaz et al. (2019). Cruz et al. (2017) developed a method and tool for detecting and marking vague words in requirements using NLP and blacklists, with a set of rules for handling them.

4.2.3 Code and programming

Several studies have addressed code and programming-related tasks. The most common was code summarization, as shown by Mastropaolo et al. (2021), who presented a multi-task approach (code summarization, assertion generation in test cases, and bug fixing tasks) by fine-tuning the T5 model described in Section 4.1.3. Arthur (2020) presented a method for generating source code documentation summaries by applying POS tagging and context-free grammar rules.

Rani et al. (2021) developed a method for classifying class comment types using NLP and ML for Java, Python, and Smalltalk. To this end, the authors created a taxonomy called Class Comment Type Model (CCTM), in which they collected the data and manually classified the class comments, then validated and assigned the mapping categories for the programming languages. They then used CCTM to classify the comments automatically using NLP, textual analysis and ML. As described in Section 4.1.1, Alrashedy et al. (2020) developed a tool that provides an automatic way of classifying questions and code snippets of programming languages' source code from Stack Overflow.

On the other hand, Gupta et al. (2013) created a method and tool for precisely POS tagging program identifiers (i.e., class, function and variable names) in the source code. The approach relies on WordNet and grammatical constructions to assign POS tags to identifiers. Sawant and Sengamedu (2022) introduced a tool for identifying best practices from software documentation using deep learning. The focus on code and programming-related tasks identified in this review is strongly corroborated by the more recent, large-scale SLR from Hou et al. (2024). Their analysis found that tasks within the software development lifecycle phase are the most dominant application area for LLMs, accounting for 56.65% of all reviewed studies. Within this domain, code generation is the single most prevalent task, investigated in 118 separate studies, with models like the GPT series, Codex, and CodeGen being instrumental. Beyond generation, their re-

view also identifies a significant research focus on other key development tasks such as code completion, code summarization, code search, and program synthesis, further emphasizing that the direct application of NLP to programming activities is a primary objective of the research community.

4.2.4 Software testing

Testing is a very important phase in SE. Some studies have attempted to address the automatic generation of test cases and steps, as well as recommending improvement of test cases. Blasi et al. (2023) proposed a method for generating unit test cases with respect to the temporal constraints from Java methods using NLP. Fischbach et al. (2023) developed a tool-supported method for the automatic generation of test cases by extracting conditionals in natural language requirements using NLP. They experimented with BERT, DistilBERT Sanh et al. (2020), and RoBERTa Liu et al. (2019) for the embeddings but chose RoBERTa because the results showed the best performance. They also used Bi-LSTMs to classify causal sentences.

Tahvili et al. (2020) created a method for classifying software test cases as dependent or independent, using NLP and ML. They used embeddings to calculate latent semantic similarity and a supervised learning algorithm for classification. Li et al. (2020) proposed a method and tool for automating the test steps of software test descriptions using NLP and clustering. Their motivation was that due to numerous test steps in various natural language forms under manual inspection, testers may not realize semantically similar test steps and thus waste effort to implement duplicate test API methods for them. Viggiato et al. (2022) developed a method for recommending improvements to NL test cases using NLP techniques.

The application of NLP to software testing is a growing field, a trend confirmed by the findings of Hou et al. (2024), whose review categorized software quality assurance tasks (including testing) as representing 15.14% of the research landscape. Their work highlights several key purposes for LLMs in this domain. A primary application is automated test generation, where models are used to create diverse unit test cases to improve coverage and identify potential defects more efficiently. Another critical purpose is vulnerability detection, where LLMs assist developers by analyzing source code to find security weaknesses that traditional static analysis might miss. Furthermore, LLMs are increasingly being used for bug localization, a process that involves analyzing natural language bug reports to pinpoint the specific files or lines of code responsible for a defect.

4.2.5 Bug fixing or reporting

Fixing or reporting bugs is a daily task for almost all software engineers. As stated by Gomes et al. (2023), the manual handling of bug reports may be entirely subjective, tiresome and error-prone. We identified studies that attempted to mitigate this laborious task. As mentioned in Section 4.1.2, Hu et al. (2018) developed a method for assisting developers in bug screening and fixing by recommending similar bugs. The studies that have addressed this topic have already been mentioned in previous sections Mastropaolo et al. (2021); Gomes et al. (2023); Hu et al. (2018); Cho et al. (2022). The significant focus on bug fixing and reporting is reinforced by the review by Hou et al. (2024), which found that software maintenance tasks are the second most researched application area for LLM, comprising 22.71% of the analyzed studies. A paramount purpose within this area is automated program repair (APR), where models like the GPT series, Codex, and T5 are leveraged to analyze buggy code and generate correct patches automatically. Another key application identified is code review, where LLMs act as assistants to human reviewers by understanding code semantics, identifying potential quality issues, and suggesting improvements. The review also cataloged other related purposes, such as interactive debugging, automated bug reproduction, and duplicate bug report detection, demonstrating the versatile role of LLMs in streamlining the entire bug management lifecycle.

4.2.6 Diagram generation

Some studies had the final goal of generating UML or other diagrams. As cited in Section 4.1.1, Jaiwai and Sammapun (2017) created a method to facilitate the extraction of classes, attributes, relationships and operations of UML diagrams using requirements written in Thai. Yang and Sahraoui (2022) developed a method for creating UML diagrams from natural language specifications using NLP and ML classification. Nasiri et al. (2020) and Elallaoui et al. (2018) also created methods for generating UML class diagrams using NLP but from user stories. On the other hand, Cheema et al. (2023) developed a tool for automatic generation and visualization of data-flow diagrams from software documentation.

4.2.7 Specific purposes

Many studies had domain-specific purposes that did not fit into the broader categories previously discussed; these are listed in Table 7. A deep comparative analysis of the challenges and results for each of these purposes is not provided, as each niche domain was addressed by only one or two primary studies at most. Given this low frequency, a granular analysis for each would not be representative of wider trends. Therefore, Table 7 is presented primarily as a reference for readers who may be interested in exploring these specific emerging applications. Our analysis prioritizes the predominant purposes identified in the preceding sections, as they better reflect the overall state of NLP in the SE domain and align with the broad scope of this work.

4.3 What are the benefits of the techniques found for Software Engineering processes? (RQ1.2)

One of the benefits of the vast majority of studies was the automation of the task or goal that they were trying to achieve. Approximately 60.9% of the reviewed studies stated that their solution is automatic, which is a benefit compared to a manually performed task. In SE, manual tasks such as handling bug reports can be subjective and error-prone. Automation enforces consistency by applying the same rules and

Table 7. Domain-specific purposes

Specific purpose	Studies
Feature location	Pérez et al. (2021); Lapeña et al. (2017)
Security assistance	Malhotra et al. (2016); Stöckle et al. (2023)
Spam detection	Fattahi and Mejri (2021)
Defects detection	Liu et al. (2014)
Ontology generation	Shu et al. (2016)
Traceability management	Arunthavanathan et al. (2016)
User stories extraction from news	Siahaan et al. (2023)
Feature models generation	Hamza and Walker (2015)
EFSM generation	Greghi et al. (2015)
Recording design decisions	Gilson and Weyns (2019)
Search engine for scientific articles	Bidulya (2018)
Topic / title extraction	Sajid et al. (2017)
Explore multi-label classification	alsukhni (2021)
Detection of system use cases and validations	Ghaisas et al. (2013)
Modeling use cases	Sawant et al. (2014)
Software effort estimation	De Bortoli Fávero et al. (2022)
Translate comments to specifications	Zhai et al. (2020)
Clustering of glossary terms	Bhatia et al. (2020)
Privacy disclosure detection	Casillo et al. (2022)
Change impact analysis	Zamani (2021)
Discuss challenges	Treude et al. (2015)

models to every artifact, which reduces human error and subjective interpretation. This leads to more reliable and predictable outcomes, especially in large-scale projects where processing thousands of documents makes manual analysis infeasible. By automating these repetitive processes, NLP tools free up highly skilled engineers to focus on more complex and creative problem-solving.

Other benefits were identified during this review, such as the performance or efficiency increase that some studies achieved compared with older state-of-the-art methods Fattahi and Mejri (2021); Hu et al. (2018); Gupta et al. (2013); Mastropaolo et al. (2021); Kaur and Kaur (2023); Arora et al. (2015); Bidulya (2018); Zhai et al. (2020); Asadabadi et al. (2020). These are not merely abstract statistical gains; they have tangible impacts on the daily work of SE professionals. For example, Cho et al. (2022) reported that their method for classifying issue reports yielded a 10% to 24% higher F1-score than state-of-the-art methods. For an engineer, this translates to more accurate triaging of issues, ensuring that critical reports are addressed faster. Similarly, the work of Hu et al. (2018) improved the performance of recommending similar bugs by 7.89% to 8.96%, which directly accelerates the bug-fixing process by helping developers find existing solutions and reduce duplicate effort.

On the other hand, some studies Shehadeh et al. (2021); Sawant et al. (2014); Asadabadi et al. (2020) provides semi-automatic approaches to address their goals. A crucial set of benefits relates directly to improving software quality, including ambiguity minimization Greghi et al. (2015); Fantechi et al. (2023), reduction of defects, and the early discovery of errors Malhotra et al. (2016); Liu et al. (2014). These benefits are critical, as it is widely understood in software engineering that the cost of fixing a defect increases the later it is found in the development lifecycle. By using NLP to automatically detect ambiguous phrases in requirements documents, teams can prevent misinterpretations that lead to costly rework. Likewise, tools that support the early discovery of errors or potential defects enable teams to mitigate risks and improve the overall reliability and correctness of the final software product. The benefit of automation, identified as a primary goal in this review, is further detailed and quantified in the analysis by Hou et al. (2024). Their findings show that the application of modern LLMs offers a dual benefit of enhancing both development efficiency and the quality of software artifacts. A principal advantage is the acceleration of developer workflows; LLMs are shown to enhance code-writing efficiency, save developer time through automated code completion, and accelerate the bug-fixing process. By automating these routine processes, LLMs enable developers to focus on more complex tasks. Concurrently, these models contribute to a higher quality of output by reducing the risk of coding errors, improving test coverage , and enhancing software reliability and maintainability. Furthermore, LLMs improve software documentation quality and accelerate code comprehension among team members, fostering more robust and collaborative development procedures. Figure 7 shows the discrepancies in benefits found in this review.

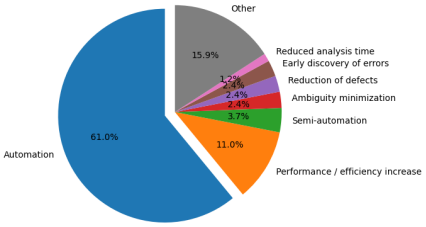

Figure 7. Proportion of identified benefits

4.4 What are the difficulties in using the techniques encountered? (RQ1.3)

To analyze the difficulty of using the techniques to answer RQ1.3, we defined four difficulty criteria: method complexity, required knowledge from the user, number of resources required for implementing the method (other methods, tools, etc.), and quantity of techniques for each study. Based on these criteria, we assigned a difficulty level for each study of easy, medium, and hard. The majority of the studies were classified as easy (51.7%) or medium (45%), whereas only two studies were classified as hard.

We identified a pattern for studies classified as easy: most of them uses only basic NLP pipeline techniques, such as tokenization and POS tagging. Thus, we assume that the utilization of the method is also easy, considering that the complexity is not high, neither the required knowledge from the user nor the number of resources required. The number of

techniques can be higher, but this does not affect the overall difficulty. Medium level approaches are studies that usually uses more complex methods, such as language models, or large resources, such as large datasets for training ML algorithms, but that do not require much knowledge from the user or a large number of methods. For the two studies classified as hard: Mastropaolo et al. (2021) and Shakeri Hossein Abad et al. (2019), we consider that, for Mastropaolo et al. (2021), the use of large language models and large datasets, as well as fine-tuning those models, requires a good knowledge of the techniques from the users and large amounts of data, in addition to the complexity of the Transformer-based models. For Shakeri Hossein Abad et al. (2019), the complexity of the methods is very high, as well as the knowledge required from the user to fully understand the approach. Figure 8 shows the proportion of each difficulty level in the studies.

Beyond that categorization, our analysis reveals that the difficulty of applying NLP techniques in Software Engineering is a multi-faceted concept influenced by several interconnected dimensions of complexity. Understanding these dimensions provides a more nuanced view of the challenges and resources involved in different approaches.

Algorithmic and Methodological Complexity: The inherent complexity of the chosen methods varies significantly. At the lower end of the spectrum are foundational NLP pipeline techniques such as tokenization, stemming, and POS-tagging. These are often available as well-documented, off-the-shelf functions in standard libraries, requiring minimal specific expertise to implement. At the higher end are sophisticated deep learning architectures like LLMs. The complexity here lies not only in understanding the underlying Transformer architecture but also in the process of applying it effectively. This includes complex tasks such as fine-tuning on domain-specific data, which requires significant expertise to perform correctly, as noted in the study by Mastropaolo et al. (2021) that involved fine-tuning the T5 model.

Data and Computational Resource Requirements: A major source of difficulty is the dependency on data and computational power. While simple rule-based systems may operate on small inputs, the majority of machine learning approaches require large, high-quality, and often manually-labeled datasets for training. The process of collecting, cleaning, and labeling this data represents a significant undertaking. State-of-the-art LLMs amplify this challenge, as training or even fine-tuning them demands massive computational resources (e.g., extensive GPU time) and enormous datasets, making their application a significant infrastructural and financial challenge.

Required User Expertise: The necessary knowledge for successfully applying these techniques varies widely. Implementing a basic NLP pipeline may only require general programming skills. In contrast, effectively leveraging an LLM or a custom deep learning model requires specialized knowledge in machine learning, including model architecture, hyperparameter tuning, prompt engineering, and an understanding of potential pitfalls like overfitting or data leakage. This steep learning curve can be a significant barrier to adoption for teams without dedicated AI/ML expertise.

Workflow Integration Complexity: Difficulty also arises from the integration of multiple techniques into a cohesive workflow. A simple, linear pipeline is far less complex than a multi-solution approach, such as the frontier LLMs or the one proposed by Ezzini et al. (2022), which combined and evaluated SpanBERT-based models, traditional machine learning classifiers, and NLP-based coreference resolvers. The complexity in such cases comes from orchestrating these disparate components and managing their interactions and dependencies.

In summary, the difficulty of using NLP in SE is not a monolithic concept but rather a spectrum defined by the interplay between algorithmic sophistication, data and resource needs, required user expertise, and the overall complexity of the engineered workflow.

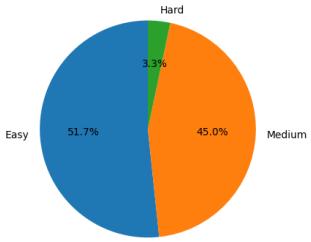

Figure 8. Proportion of difficulty levels

4.5 How has the use of NLP techniques contributed to the advancement of Software Engineering? (RQ2)

In our study, we identified some advancements in the SE processes, as analyzed in the previous sections. One of them is the automation of a variety of manual tasks, such as the process of writing natural language requirements, user stories, and use cases, as the automatic generation of textual specifications or the extraction of representations from the texts. This contributes to the process of requirements elicitation and extraction, as it makes it easier to extract assertive natural language requirements from the software context and helps analysts to identify irregular clauses. Another advancement is the generation and extraction of less ambiguous requirements using the ambiguity handling techniques presented in Section 4.2.2. Advancements in testing, such as test case generation and recommendation, have also been verified in some studies. Some studies have focused on source code automation, which is a significant contribution to the future of software engineering, eliminating the need for extensive code reviewing and bug fixing, as well as reducing the time spent by those processes.

4.5.1 NLP techniques and Software Engineering stages

We related the NLP techniques found in the four stages of software engineering: requirements, design, coding (development), and testing. Some of the identified NLP techniques and methods are present in almost all of them, as we can identify, being the techniques that most appeared in the primary studies. For example, preprocessing, tokenization, POS tagging, lemmatization, stemming, dependency parsing, sentence splitting, and pattern matching are text processing methods that we identified as being present in almost all SE

stages. Named-entity recognition is utilized in two stages: requirements and testing. TF-IDF appears in studies related to requirements and coding. WordNet has been identified in studies that addressed the design and coding stages. Other techniques were addressed only in one stage, as we can see with LDA, text chunking, and constituency parsing, which were identified in studies only related to requirements. Table 8 shows the relationship between the SE stages described and the NLP methods directly associated with them. In Figure 9, we see the number of NLP methods that only appeared in studies related to one of the SE stages.

Although this review has identified specific NLP techniques present in the requirements, design, coding, and testing stages, it is also valuable to consider the broader distribution of the research effort across the entire software development lifecycle. As noted in our tertiary study, the SLR by Hou et al. (2024) analyzed 395 primary studies and provides a clear quantitative insight into this landscape. Their findings reveal that the application of modern NLP, particularly LLMs, is heavily concentrated in the later stages of the cycle. Most of the research focuses on software development (56.65% of studies) and software maintenance (22.71%), where tasks like code generation and program repair are the most prevalent applications. In stark contrast, foundational areas such as requirements engineering (3.90%), software design (0.92%), and software management (0.69%) remain significantly under-researched. This distribution not only underscores the community's current focus on code-centric tasks but also highlights a substantial opportunity for future work to apply advanced NLP capabilities to the formative stages of software engineering, where they have the potential to make a profound impact. This research gap is reinforced by other perspectives, such as that of Ozkaya (2023), who argues for applying LLMs (AI assistants) across the entire development workflow. Identifying models like GPT-4 and BERT as drivers of a paradigm shift from search results to synthesized answers, this view also highlights the impact of code-specific assistants like GitHub's CoPilot. Crucially, this perspective points to significant opportunities in the very areas identified as under-researched, such as automated specification generation to improve requirements and the generation of more effective test cases. This vision of applying LLM capabilities across the full software lifecycle further underscores the importance of addressing the current research imbalance.

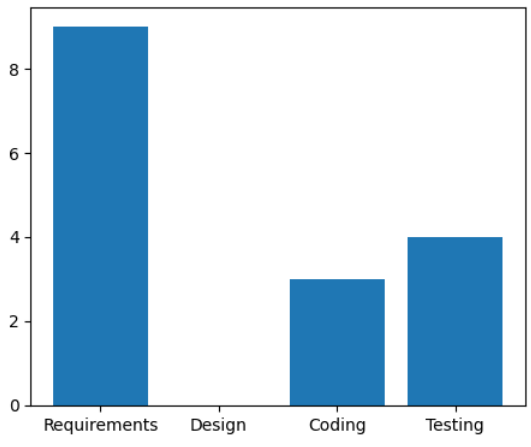

Figure 9. Number of methods appearing in only one SE stage

5 Discussion

Our combined analysis, drawing from both the primary SLR and the supplementary tertiary study, reveals several key insights into the state and trajectory of NLP in SE. A cohesive discussion of these findings highlights a field in a period of significant transition, defined by a technological shift from analytical to generative capabilities and a corresponding evolution in application focus.

The most significant finding is the clear technological progression from foundational, syntax-based techniques to sophisticated, semantic-based models. Our primary SLR on 60 studies confirmed that traditional NLP work heavily relies on foundational methods, with an increasing adoption of first-generation language models like BERT to better handle context and ambiguity. The implication of this initial shift is a move from simple pattern matching toward a deeper, more contextual understanding of language, which is crucial for the complexity of SE tasks.

This technological evolution has occurred alongside a distinct and revealing pattern of application. Our primary review found that traditional NLP efforts are overwhelmingly concentrated on requirements engineering, with tasks like requirements classification and ambiguity handling being paramount. In contrast, our tertiary study confirmed that the new frontier of LLMs is almost entirely focused on generative, code-related tasks within the development and maintenance phases, such as code generation and automated program repair. The most critical implication of this synthesis is the identification of a research gap: the most powerful generative models are not yet being widely applied to the long-standing challenges of requirements and design that were the primary focus of earlier NLP research.

Furthermore, our findings on the benefits and difficulties of these techniques indicate a maturing field. While automation remains the principal benefit sought across all studies, the capabilities of modern LLMs suggest a shift in goals from mere automation to active augmentation, where AI tools act as collaborative partners. This increasing sophistication, however, brings added complexity. The application of state-of-the-art LLMs requires significant expertise and computational resources, creating a higher barrier to entry compared to traditional methods. The key insight here is that for the SE field to fully harness the potential of modern NLP, future work must focus on bridging the identified application gap by applying generative models to the entire lifecycle, while simultaneously developing the rigorous engineering practices needed to manage their complexity and risks.

5.1 Limitations of the Study

Although this review was conducted using a rigorous methodology based on the Kitchenham and Charters guidelines, several limitations should be considered when interpreting its findings. These limitations pertain to the scope

Table 8. NLP techniques across SE stages

SE stage	Methods addressed in two stages	Exclusive
Requirements	NER, TF-IDF, Cosine similarity	LDA, Text chunking, Constituency parsing, Semantic parsing, LSA, Fuzzy logic, Jaccard's similarity, Bag-of-words, Paragraph similarity
Design	WordNet	
Coding	WordNet, TF-IDF, Cosine, Word embedding, Similarity score, T5	FastText, GloVe, Document embedding
Testing	NER, Word embedding, Similarity score, T5	Word Mover's Distance similarity, n-grams, RoBERTa, DistilBERT

of the literature search, the potential for bias in the analysis, and the constraints of the data synthesis process.

A primary limitation is the scope of the literature search, which cannot guarantee the retrieval of every relevant study. The findings are fundamentally shaped by the search string constructed from the keywords in Table 2, the selected digital libraries (IEEE Xplore, ScienceDirect, and ACM Digital Library), and the defined publication date range of 2013 to 2023. This temporal boundary was chosen to capture the latest state-of-the-art approaches, but consequently, some relevant foundational studies published outside this range may have been missed. Our inability to gain access to other planned databases may have also led to the exclusion of relevant papers. Furthermore, the review was limited to studies written in English, potentially missing research from other linguistic communities.

Potential for bias exists in the selection and analysis process. A publication bias may be present, as our focus on published scientific articles in journals, conferences, and workshops means we may have missed valuable contributions from industry white papers or open-source project documentation. Additionally, the quality assessment of primary studies was based on a predefined checklist of questions. While this process aimed for objectivity by assigning scores of 1.0 for a "yes," 0.0 for a "no," and 0.5 for a partial answer, the selection of the questions themselves and the subjective nature of what constitutes a "partial" answer could have introduced some bias into the quality scoring.

Finally, the review methodology has inherent constraints regarding data extraction and synthesis. The data extraction was performed using a predefined form with attributes tailored to the research questions. This structured approach ensures consistency but may have limited the depth of data captured from each study, as important nuances or data not fitting the form's properties could have been missed or not properly addressed. A further challenge was the categorization of a diverse set of NLP techniques and applications. This process of synthesis involves subjective interpretation, and for niche domains addressed by only one or two studies, a deep comparative analysis of challenges and results was not provided. Due to the wide variety of evaluation metrics, datasets, and validation contexts used across the primary studies, a direct quantitative meta-analysis of their results was not feasible.

5.2 Implications for Research and Practice

The findings of this study, which combine a systematic review of foundational NLP techniques with a tertiary analysis of LLMs, offer several actionable insights for the software engineering community. This section outlines the implications of these findings, first by exploring the theoretical shifts and future directions for researchers, and second by providing practical recommendations and guidelines for software development teams.

5.2.1 Theoretical Implications and Future Research Directions

This review provides a map of a field undergoing a significant transition, characterized by a technological progression from analytical, syntax-based techniques to sophisticated, generative models. The most critical theoretical implication drawn from this dual analysis is the identification of a significant research gap. While foundational NLP research has been heavily concentrated on the analysis of requirements artifacts, the community's most powerful new generative models have so far largely overlooked this area, remaining predominantly focused on code-related tasks within the development and maintenance phases.

This identified gap points to several clear and impactful future research directions. A primary avenue for future work is to apply the generative and analytical power of modern LLMs to the foundational areas of software engineering, such as requirements engineering, software design, and software management, which remain significantly underresearched in the LLM landscape. Research should also explore how these models can evolve beyond the simple automation of manual tasks toward more augmentative roles where they act as collaborative partners for software engineers. This evolution will likely require the development of code-specialized LLMs and will fuel the emergence of a critical new discipline known as "Software Engineering for Large Language Models" (SE4LLM), focused on creating the rigorous engineering practices needed to manage the complexity and risks of these advanced systems.

5.2.2 Practical Implications and Recommendations for Adoption

For practitioners, this review serves as a catalog of NLP techniques that can be applied to improve SE process efficiency and product quality. The documented benefits provide a clear case for adopting these technologies, including the automation of manual tasks that can be subjective and error-prone, increased performance and efficiency compared to older methods, and the potential for ambiguity minimization and the early discovery of errors.

To aid practitioners, this review offers guidelines for selecting appropriate NLP techniques based on specific SE tasks. For analytical tasks common in early lifecycle stages, such as requirements classification, ambiguity handling, or extracting diagrams from text, foundational NLP pipelines (e.g., POS-tagging, dependency parsing) and first-generation, encoder-only models like BERT are well-established and effective. These methods excel at structuring and analyzing textual artifacts to improve clarity and reduce errors early in development. In contrast, for generative tasks that dominate later lifecycle stages, including code generation, automated program repair, code summarization, and test case generation, practitioners should turn to frontier LLMs. Decoder-only models like the GPT series and encoder-decoder models like T5 represent the state-of-the-art for creating new content and accelerating developer workflows.

When planning for adoption, practitioners should make informed decisions based on the complexity and resources required. Implementing basic NLP pipelines is often straightforward and was classified as ”easy” in our analysis, requiring minimal specific expertise. However, leveraging state-of-the-art LLMs is a ”hard” undertaking that requires significant expertise in machine learning, massive high-quality datasets, and substantial computational resources. Given that even advanced models can produce incorrect or ”hallucinated” outputs, it is recommended to adopt a hybrid approach where traditional SE methods like automated testing are used to filter and validate the creative but potentially flawed results generated by LLMs.

6 Conclusion

This paper presented an overview of NLP in SE by synthesizing a SLR of 60 primary studies with a focused tertiary study on LLMs. Our findings reveal a field in a state of significant transition. The primary review showed that traditional NLP applications have been heavily concentrated on analyzing requirements artifacts, while the tertiary study confirmed that the new frontier of LLMs is predominantly focused on generative, code-related tasks in the development and maintenance phases. The most significant conclusion drawn from this dual analysis is the identification of a critical research gap: the community’s most powerful new models are not yet being widely applied to the foundational challenges of requirements and design.

Our literature review indicates that the utilization of traditional NLP methods in SE tasks is still in its early stages, with considerable untapped potential across several areas. Future research is poised to tackle challenges and explore new frontiers, with an emphasis on harnessing large language models based on Transformer networks to achieve superior performance compared to traditional NLP methods. This includes the potential expansion of NLP applications beyond the automation of manual tasks, which presents an exciting avenue for future exploration. The underlying technology continues to evolve rapidly; Transformer-based models are being trained for several areas, such as text and multimedia (audio, image, and video) generation. Concurrently, recent studies have attempted to address issues related to the financial cost and energy consumption for training such large models, for example, the works of Wang et al. (2023a) and Ma et al. (2024). New hardware may be developed in the future to leverage the efficiency of new techniques to scale large language models. Addressing these foundational challenges will be crucial for realizing the full potential of NLP and ushering in the next generation of intelligent software engineering tools.

Statements and Declarations

There are no conflicts of interest related to this research.

Acknowledgment

The authors wish to thank São Paulo Research Foundation (FAPESP). This work was partially supported by grants from 2023/03393-5 and 2025/16287-4.

References

AlDhafer, O., Ahmad, I., and Mahmood, S. (2022). An end-to-end deep learning system for requirements classification using recurrent neural networks. *Information and Software Technology*, 147:106877.

Alrashedy, K., Dharmaretnam, D., German, D. M., Srinivasan, V., and Aaron Gulliver, T. (2020). Scc++: Predicting the programming language of questions and snippets of stack overflow. *Journal of Systems and Software*, 162:110505.

alsukhni, B. (2021). Multi-label arabic text classification based on deep learning. In *2021 12th International Conference on Information and Communication Systems (ICICS)*, pages 475–477.

Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2015). Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41(10):944–968.

Arthur, M. P. (2020). Automatic source code documentation using code summarization technique of nlp. *Procedia Computer Science*, 171:2522–2531. Third International Conference on Computing and Network Communications (CoCoNet’19).

Arunthavanathan, A., Shanmugathasan, S., Ratnavel, S., Thiyagarajah, V., Perera, I., Meedeniya, D., and Balasubramaniam, D. (2016). Support for traceability management of software artefacts using natural language processing. In *2016 Moratuwa Engineering Research Conference (MERCon)*, pages 18–23.

Asadabadi, M. R., Saberi, M., Zwikael, O., and Chang, E. (2020). Ambiguous requirements: A semi-automated approach to identify and clarify ambiguity in large-scale projects. *Computers & Industrial Engineering*, 149:106828.

Baum, L. E. and Petrie, T. (1966). Statistical inference for

probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563.

Bhatia, K., Mishra, S., and Sharma, A. (2020). Clustering glossary terms extracted from large-sized software requirements using fasttext. In *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly Known as India Software Engineering Conference*, ISEC 2020, New York, NY, USA. Association for Computing Machinery.

Bidulya, Y. (2018). An approach to the development of software for effective search of scientific articles. In *2018 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC)*, pages 1–4.

Blasi, A., Gorla, A., Ernst, M. D., and Pezzè, M. (2023). Call me maybe: Using nlp to automatically generate unit test cases respecting temporal constraints. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Casamayor, A., Godoy, D., and Campo, M. (2011). Mining textual requirements to assist architectural software design: a state of the art review. *Springer Science+Business Media B.V.*

Casillo, F., Deufemia, V., and Gravino, C. (2022). Detecting privacy requirements from user stories with nlp transfer learning models. *Information and Software Technology*, 146:106853.

Cheema, S. M., Tariq, S., and Pires, I. M. (2023). A natural language interface for automatic generation of data flow diagram using web extraction techniques. *Journal of King Saud University - Computer and Information Sciences*, 35(2):626–640.

Cho, H., Lee, S., and Kang, S. (2022). Classifying issue reports according to feature descriptions in a user manual based on a deep learning model. *Information and Software Technology*, 142:106743.

Cruz, B. D., Jayaraman, B., Dwarakanath, A., and McMillan, C. (2017). Detecting vague words & phrases in requirements documents in a multilingual environment. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 233–242.

Dalpiaz, F., Ferrari, A., Franch, X., and Palomares, C. (2018). Natural language processing for requirements engineering: The best is yet to come.

Dalpiaz, F., van der Schalk, I., Brinkkemper, S., Aydemir, F. B., and Lucassen, G. (2019). Detecting terminological ambiguity in user stories: Tool and experimentation. *Information and Software Technology*, 110:3–16.

De Bortoli Fávero, E. M., Casanova, D., and Pimentel, A. R. (2022). Se3m: A model for software effort estimation using pre-trained embedding models. *Information and Software Technology*, 147:106886.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into uml use case diagrams using nlp techniques. *Procedia Computer Science*, 130:42–49. The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops.

Ezzini, S., Abualhaija, S., Arora, C., and Sabetzadeh, M. (2022). Automated handling of anaphoric ambiguity in requirements: A multi-solution study. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 187–199, New York, NY, USA. Association for Computing Machinery.

Ezzini, S., Abualhaija, S., Arora, C., Sabetzadeh, M., and Briand, L. C. (2021). Using domain-specific corpora for improved handling of ambiguity in requirements. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 1485–1497. IEEE Press.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., and Zhang, J. M. (2023). Large language models for software engineering: Survey and open problems.

Fantechi, A., Gnesi, S., and Semini, L. (2023). Vibe: Looking for variability in ambiguous requirements. *Journal of Systems and Software*, 195:111540.

Fattahi, J. and Mejri, M. (2021). Spaml: a bimodal ensemble learning spam detector based on nlp techniques. In *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, pages 107–112.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. MIT Press, Cambridge, MA.

Fischbach, J., Frattini, J., Vogelsang, A., Mendez, D., Unterkalmsteiner, M., Wehrle, A., Henao, P. R., Yousefi, P., Juricic, T., Radduenz, J., and Wiecher, C. (2023). Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study. *Journal of Systems and Software*, 197:111549.

Ghaisas, S., Motwani, M., and Anish, P. R. (2013). Detecting system use cases and validations from documents. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 568–573.

Gilson, F. and Weyns, D. (2019). When natural language processing jumps into collaborative software engineering. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 238–241.

Gomes, L., da Silva Torres, R., and Côrtes, M. L. (2023). Bert- and tf-idf-based feature extraction for long-lived bug prediction in floss: A comparative study. *Information and Software Technology*, 160:107217.

Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA.

Greghi, J. G., Martins, E., and Carvalho, A. M. B. R. (2015). Semi-automatic generation of extended finite state machines from natural language standard documents. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 45–50.

Gupta, S. and Gupta, S. K. (2019). Natural language processing in mining unstructured data from software repositories: a review. *Indian Academy of Sciences*.

Gupta, S., Malik, S., Pollock, L., and Vijay-Shanker, K. (2013). Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 3–12.

Halim, F. and Siahaan, D. (2019). Detecting non-atomic requirements in software requirements specifications using classification methods. In *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, volume 1, pages 269–273.

Hamza, M. and Walker, R. J. (2015). Recommending features and feature relationships from requirements documents for software product lines. In *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 25–31.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. (2024). Large language models for software engineering: A systematic literature review.

Hu, D., Chen, M., Wang, T., Chang, J., Yin, G., Yu, Y., and Zhang, Y. (2018). Recommending similar bug reports: A novel approach using document embedding model. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 725–726.

Jaiwai, M. and Sammapun, U. (2017). Extracting uml class diagrams from software requirements in thai using nlp. In *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5.

Kadebu, P., Sikka, S., Tyagi, R. K., and Chiurunge, P. (2023). A classification approach for software requirements towards maintainable security. *Scientific African*, 19:e01496.

Kaur, K. and Kaur, P. (2023). Bert-cnn: Improving bert for requirements classification using cnn. *Procedia Computer Science*, 218:2604–2611. International Conference on Machine Learning and Data Engineering.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Kolahdouz-Rahimi, S., Lano, K., and Lin, C. (2023). Requirement formalisation using natural language processing and machine learning: A systematic review.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

Lapeña, R., Font, J., Pastor, O., and Cetina, C. (2017). Analyzing the impact of natural language processing over feature location in models. *SIGPLAN Not.*, 52(12):63–76.

Li, B. and Nong, X. (2022). Automatically classifying non-functional requirements using deep neural network. *Pattern Recognition*, 132:108948.

Li, L., Li, Z., Zhang, W., Zhou, J., Wang, P., Wu, J., He, G., Zeng, X., Deng, Y., and Xie, T. (2020). Clustering test steps in natural language toward automating test automation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1285–1295, New York, NY, USA. Association for Computing Machinery.

Li, Y., Guzman, E., Tsiamoura, K., Schneider, F., and Bruegge, B. (2015). Automated requirements extraction for scientific software. *Procedia Computer Science*, 51:582–591. International Conference On Computational Science, ICCS 2015.

Liu, K., Reddivari, S., and Reddivari, K. (2022). Artificial intelligence in software requirements engineering: State-of-the-art. *IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*.

Liu, S., Sun, J., Liu, Y., Zhang, Y., Wadhwa, B., Dong, J. S., and Wang, X. (2014). Automatic early defects detection in use case documents. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, page 785–790, New York, NY, USA. Association for Computing Machinery.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Ma, S., Wang, H., Ma, L., Wang, L., Wang, W., Huang, S., Dong, L., Wang, R., Xue, J., and Wei, F. (2024). The era of 1-bit llms: All large language models are in 1.58 bits.

Malhotra, R., Chug, A., Hayrapetian, A., and Raje, R. (2016). Analyzing and evaluating security features in software requirements. In *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, pages 26–30.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mastropaolo, A., Scalabrino, S., Cooper, N., Palacio, D. N., Poshyvanyk, D., Oliveto, R., and Bavota, G. (2021). Studying the usage of text-to-text transfer transformer to support code-related tasks. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 336–347. IEEE Press.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Nasiri, S., Rhazali, Y., Lahmer, M., and Chenfour, N. (2020). Towards a generation of class diagram from user stories in agile methods. *Procedia Computer Science*, 170:831–837. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

Omran, F. N. A. A. and Treude, C. (2017). Choosing an nlp library for analyzing software documentation: A systematic literature review and a series of experiments. *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*.

Ozkaya, I. (2023). Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Software*, 40(3):4–8.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural*

Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pérez, F., Lapeña, R., Marcén, A. C., and Cetina, C. (2021). Topic modeling for feature location in software models: Studying both code generation and interpreted models. *Information and Software Technology*, 140:106676.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2023). Exploring the limits of transfer learning with a unified text-to-text transformer.

Raharjana, I. K., Siahaan, D., and Fatichah, C. (2021). User stories and natural language processing: A systematic literature review. *IEEE Access*.

Rani, P., Panichella, S., Leuenberger, M., Di Sorbo, A., and Nierstrasz, O. (2021). How to identify class comment types? a multi-language approach for class comment classification. *Journal of Systems and Software*, 181:111047.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.

Sajid, A., Jan, S., and Shah, I. A. (2017). Automatic topic modeling for single document short texts. In *2017 International Conference on Frontiers of Information Technology (FIT)*, pages 70–75.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sawant, K. P., Roy, S., Parachuri, D., Plesse, F., and Bhattacharya, P. (2014). Enforcing structure on textual use cases via annotation models. In *Proceedings of the 7th India Software Engineering Conference*, ISEC '14, New York, NY, USA. Association for Computing Machinery.

Sawant, N. and Sengamedu, S. H. (2022). Learning-based identification of coding best practices from software documentation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 533–542.

Shakeri Hossein Abad, Z., Gervasi, V., Zowghi, D., and H. Far, B. (2019). Supporting analysts by dynamic extraction and classification of requirements-related knowledge. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 442–453.

Shehadeh, K., Arman, N., and Khamayseh, F. (2021). Semi-automated classification of arabic user requirements into functional and non-functional requirements using nlp tools. In *2021 International Conference on Information Technology (ICIT)*, pages 527–532.

Shreda, Q. A. and Hanani, A. A. (2021). Identifying non-functional requirements from unconstrained documents using natural language processing and machine learning approaches. *IEEE Access*, pages 1–1.

Shu, Y., Lun, Y. H., Run, Y. X., and Ye, W. (2016). An automated method for constructing ontology. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 538–541.

Siahaan, D., Raharjana, I. K., and Fatichah, C. (2023). User story extraction from natural language for requirements elicitation: Identify software-related information from online news. *Information and Software Technology*, 158:107195.

Singh, M. (2019). Using natural language processing and graph mining to explore inter- related requirements in software artefacts. *SIGSOFT Softw. Eng. Notes*, 44(1):37–42.

Sonbol, R., Rebdawi, G., and Ghneim, N. (2022). The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review. *IEEE Access*, 10:62811–62830.

Stöckle, P., Wasserer, T., Grobauer, B., and Pretschner, A. (2023). Automated identification of security-relevant configuration settings using nlp. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

Tahvili, S., Hatvani, L., Ramentol, E., Pimentel, R., Afzal, W., and Herrera, F. (2020). A novel methodology to classify test cases using natural language processing and imbalanced learning. *Engineering Applications of Artificial Intelligence*, 95:103878.

Treude, C., Prolo, C. A., and Filho, F. F. (2015). Challenges in analyzing software documentation in portuguese. In *2015 29th Brazilian Symposium on Software Engineering*, pages 179–184.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Viggiato, M., Paas, D., Buzon, C., and Bezemer, C.-P. (2022). Using natural language processing techniques to improve manual test case descriptions. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, page 311–320, New York, NY, USA. Association for Computing Machinery.

Wang, H., Ma, S., Dong, L., Huang, S., Wang, H., Ma, L., Yang, F., Wang, R., Wu, Y., and Wei, F. (2023a). Bitnet: Scaling 1-bit transformers for large language models.

Wang, S., Huang, L., Gao, A., Ge, J., Zhang, T., Feng, H., Satyarth, I., Li, M., Zhang, H., and Ng, V. (2023b). Machine/deep learning for software engineering: A systematic literature review. *IEEE Transactions on Software Engineering, Vol. 49, No. 3*.

Wein, S. and Briggs, P. (2021). A fully automated approach to requirement extraction from design documents. In *2021 IEEE Aerospace Conference (50100)*, pages 1–7.

Yang, S. and Sahraoui, H. (2022). Towards automatically extracting uml class diagrams from natural language specifications. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, page 396–403, New York, NY, USA. Association for Computing Machinery.

Zamani, K. (2021). A prediction model for software requirements change impact. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1028–1032.

Zhai, J., Shi, Y., Pan, M., Zhou, G., Liu, Y., Fang, C., Ma, S., Tan, L., and Zhang, X. (2020). C2s: Translating natural language comments to formal program specifications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium*

on the Foundations of Software Engineering, ESEC/FSE 2020, page 25–37, New York, NY, USA. Association for Computing Machinery.

A Appendix: Data Extraction Form

To ensure transparency and support the reproducibility of this systematic literature review, the complete data extraction form containing the data synthesized from the 60 primary studies is publicly available. The form includes all fields described in Table 4 of our methodology. The full data extraction form can be accessed at the following permanent link: `https://doi.org/10.6084/m9.figshare.29367533.v1`.