TYPE Original Research
PUBLISHED 21 March 2025
DOI 10.3389/fcomp.2025.1519437

EDITED BY
Hongji Yang,
University of Leicester, United Kingdom

REVIEWED BY
Bruno Gadelha,
Federal University of Amazonas, Brazil
Mohd Sameen Chishti,
Norwegian University of Science and Technology, Norway

*CORRESPONDENCE
Sobhan Y. Tehrani
✉ sobhan.tehrani@ucl.ac.uk

RECEIVED 29 October 2024
ACCEPTED 28 February 2025
PUBLISHED 21 March 2025

CITATION
Hemmat A, Sharbaf M, Kolahdouz-Rahimi S, Lano K and Tehrani SY (2025) Research directions for using LLM in software requirement engineering: a systematic review. *Front. Comput. Sci.* 7:1519437. doi: 10.3389/fcomp.2025.1519437

Research directions for using LLM in software requirement engineering: a systematic review

Arshia Hemmat[1], Mohammadreza Sharbaf[1], Shekoufeh Kolahdouz-Rahimi[2], Kevin Lano[3] and Sobhan Y. Tehrani[4]*

[1]Department of Software Engineering, University of Isfahan, Isfahan, Iran, [2]School of Arts, University of Roehampton, London, United Kingdom, [3]Department of Informatics, King's College London, London, United Kingdom, [4]Department of Computer Science, University College London, London, United Kingdom

Introduction: Natural Language Processing (NLP) and Large Language Models (LLMs) are transforming the landscape of software engineering, especially in the domain of requirement engineering. Despite significant advancements, there is a notable lack of comprehensive survey papers that provide a holistic view of the impact of these technologies on requirement engineering. This paper addresses this gap by reviewing the current state of NLP and LLMs in requirement engineering.

Methods: We analyze trends in software requirement engineering papers, focusing on the application of NLP and LLMs. The review highlights their effects on improving requirement extraction, analysis, and specification, and identifies key patterns in the adoption of these technologies.

Results: The findings reveal an upward trajectory in the use of LLMs for software engineering tasks, particularly in requirement engineering. The review underscores the critical role of requirement engineering in the software development lifecycle and emphasizes the transformative potential of LLMs in enhancing precision and reducing ambiguities in requirement specifications.

Discussion: This paper identifies a growing interest and significant progress in leveraging LLMs for various software engineering tasks, particularly in requirement engineering. It provides a foundation for future research and highlights key challenges and opportunities in this evolving field.

KEYWORDS

software development, requirement engineering, Large Language Models (LLMs), systematic literature review, requirement specification

1 Introduction

Large Language Models (LLMs) are powerful AI systems trained on extensive text datasets to produce human-like language. They are commonly applied in areas such as machine translation, content creation, chatbots, and summarizing information across various sectors. More recently, they have opened up transformational possibilities in many spheres of Software Requirement Engineering (SRE). As a discipline, Requirements Engineering (RE) plays a *critical role* in the software development lifecycle by bridging stakeholder needs and technical implementations, ensuring that final software products meet specified objectives and quality standards. LLMs, with their *transformative potential*, can fundamentally improve the RE process by automating tasks such as requirement elicitation and specification, which have historically been time-consuming and error-prone.

With natural language processing capabilities, LLMs have now become an active partner in Requirements Engineering, powering automation in elicitation, specifications generation, quality assurance, and many other aspects of the requirements. These are state-of-the-art architectures, including GPT, BERT, and their domain-specific variants, each of which has different advantages with respect to structured text analysis and generation and, therefore, are very helpful in Requirement Engineering (RE) activities where the requirements are written in a natural language, such as Wang et al. (2024), Krishna et al. (2024), and Ma et al. (2024). Therefore, LLMs have begun to attract attention as a tool that can help RE processes become more efficient and accurate and cover the gap between high-level requirements and technical implementation.

They bring generative and analytical strengths to RE tasks, from creating UML diagrams to supporting formal specifications and verification. In this respect, models such as Codex and ChatGPT have been used in generating code snippets, verifying requirements, and enabling automated test case generation that alleviates human engineers' work and rushes project timelines accordingly (Wang et al., 2024; Luitel et al., 2024; Li et al., 2024). Besides mere generation, their contribution in generating formalized specifications and integrating requirements with domain-specific constraints has given more depth and relevance to the RE documentation, such as those found in Kogler et al. (2024), Gomez et al. (2024), and Frydenlund et al. (2024). Not all challenges have been resolved, however. Some examples include LLMs' difficulty with domain-specific subtleties of languages, hallucination, and inability to comprehend complex dependencies in various RE contexts (Alhanahnah et al., 2024; Rahman and Zhu, 2024; Sarsa et al., 2022). Various tuning and prompt engineering methods have been carried out to optimize the performance of LLMs for RE.

Fine-tuning techniques include LoRA and prompt tuning, which have proved to be very effective in making LLM outputs align with the requirements of RE and thus have enabled even complex specifications to be processed with higher relevance and accuracy (Lee et al., 2024; Bhattacharya et al., 2023). Few-shot and iterative prompting are some of the most important techniques in prompt engineering for structured input to LLMs, which could yield responses according to the RE standards. Refer to Mandal et al. (2023), Lubos et al. (2024), and White et al. (2024) for examples. Evaluation metrics such as precision and recall, along with expert-based assessments, also play an important role in validating generated outputs by LLMs to ensure reliability and usability in the SRE environment. Despite this, other factors remain potential barriers to the wide adoption of SRE for LLMs.

Common issues include problems with processing domain requirements, the effectiveness of prompts, and technical inability to handle structured RE inputs that mainly cause inconsistencies and errors in output (Zhang et al., 2023b; Xie et al., 2023; Abukhalaf et al., 2024). This implies the need for more robust frameworks and evaluation techniques that handle SRE-specific requirements. These gaps have been addressed through the emergence of research on hybrid approaches, such as combinations of LLMs with traditional RE tools or human-in-the-loop systems that further enhance the reliability and accuracy of the LLM-generated outputs (Wu et al., 2023; Hasan et al., 2023). In particular, these advances point to a range of *key challenges and opportunities* for natural language processing and LLM technologies: from improving the precision of requirement extraction to orchestrating collaborative workflows where both machine and human expertise can contribute to higher-quality software designs.

In this article we present a systematic mapping study which reviews current applications, challenges, and optimization techniques for LLMs in SRE. The different types of LLMs used in RE are classified in detail, input-output mechanisms are considered, and their relevance for specific RE tasks. Strategies related to tuning and prompt engineering, which help optimize the performance of LLMs in SRE, will be discussed. The rest of the paper details these elements and sets up a base for future work on leveraging LLMs effectively within SRE. It also informs both researchers and practitioners about methodologies for maximizing the potential of LLMs while working around their existing limitations.

The rest of this paper is organized as follows. Section 3 briefly explains the research goals and methodology for performing this study. Section 4 reports the extracted results based on the research questions. Section 5 provides an overall discussion of the results to clarify the open directions and research challenges. Finally, Section 6 concludes the paper and highlights areas for future work.

2 Survey of related work on LLMs and software requirements

The integration of Large Language Models (LLMs) into Software Engineering, particularly in the context of Requirements Engineering (RE), has been the focus of several studies. To provide a comprehensive overview, this section categorizes and analyzes surveys under three themes: Surveys on NLP in Software Engineering, Surveys on LLMs in Software Engineering, and Surveys on LLMs in Software Engineering Requirements. Each theme highlights the contributions, strengths, limitations, and emerging trends of relevant studies (Al-Hossami and Shaikh, 2022; Necula et al., 2024; Fan et al., 2023).

2.1 Surveys on NLP in Software engineering

Natural Language Processing (NLP) has played a pivotal role in bridging human-readable requirements and technical implementations in Software Engineering. A significant contribution in this area includes an extensive taxonomy for Code Intelligence (CI), emphasizing the role of conversational agents in software development and education (Al-Hossami and Shaikh, 2022). Studies in this domain chronicle three decades of deep learning techniques applied to source code, capturing the evolution of NLP applications in Software Engineering. However, these studies often lack a focus on specialized domains like Requirements Engineering, limiting their utility for such specific tasks. Another comprehensive review highlights trends and applications of NLP in Requirements Engineering over a period spanning from 1991 to 2023 (Necula et al., 2024). While offering actionable insights and identifying emerging directions, these studies primarily address classical NLP methods with minimal focus on advancements in LLMs.

2.2 Surveys on LLMs in software engineering

Large Language Models represent a transformative advancement in Software Engineering, offering capabilities that span across automation, code generation, and design. Surveys in this domain explore the utility of LLMs in tasks such as coding, design, and requirements, while identifying challenges and opportunities for future research (Fan et al., 2023; Zhang et al., 2023a). These works provide a detailed review of LLM technologies, including pre-training objectives, downstream evaluations, and their applications in software testing and deployment. Despite their technical depth, these surveys often lack practical guidance for applying LLMs specifically in Requirements Engineering. Some studies also unify the perspectives of NLP and Software Engineering, discussing the role of LLMs in testing, deployment, and operations, thus offering a broader outlook on their impact (Zhang Z. et al., 2023). However, the absence of detailed analysis tailored to requirements engineering use cases limits the applicability of these findings for RE-specific tasks.

2.3 Surveys on LLMs in software engineering requirements

Requirements Engineering is a critical phase in software development, and recent studies have begun exploring the potential of LLMs in this domain. One notable study conducts a SWOT analysis of LLMs in requirements elicitation, analysis, and validation, providing a structured assessment of their strengths and weaknesses (Arora et al., 2023). Another study offers practical guidelines for leveraging LLMs in RE tasks, focusing on task-specific recommendations (Vogelsang and Fischbach, 2024). While these works provide valuable insights, they often lack empirical evidence to support their proposed methodologies. Additionally, theoretical discussions on integrating formal methods with LLMs underscore the importance of validating LLM-generated outputs but offer limited practical implementation guidance (Spoletini and Ferrari, 2024). A systematic literature review further extends the understanding of generative AI in RE processes, although it highlights challenges related to implementation and practical use cases (Cheng et al., 2024).

2.4 Emerging trends and research opportunities

The reviewed surveys collectively highlight broader trends and significant research opportunities. An increasing adoption of LLMs is evident across various stages of software engineering, particularly in coding and design (Fan et al., 2023; Zhang et al., 2023a). However, practical integration of LLMs in Requirements Engineering remains underexplored, with empirical studies addressing real-world applications being limited (Vogelsang and Fischbach, 2024; Arora et al., 2023). Key challenges include the validation of LLM-generated requirements, addressing issues like hallucination and consistency, and enhancing fine-tuning techniques for domain-specific RE tasks (Spoletini and Ferrari, 2024; Cheng et al., 2024). Future research should focus on developing robust frameworks for validating and optimizing LLM outputs tailored to the nuanced demands of Requirements Engineering.

While these surveys provide valuable insights into the integration of NLP and LLMs in Software Engineering and Requirements Engineering, gaps remain in focused research on LLMs tailored specifically for RE tasks. Nonetheless, the field demonstrates significant upward momentum, with promising opportunities for advancing methodologies and addressing existing challenges (Fan et al., 2023; Zhang et al., 2023a; Arora et al., 2023).

3 Methods

To conduct this systematic mapping study, we followed the guidelines presented by Brereton et al. (2007) and Petersen et al. (2015). The goal of this study is to answer the following research questions.

Q1. (Role of LLM in RE process): what roles do Large Language Models (LLMs) serve in the requirement engineering (RE) process?

Q2. (Evaluation techniques for LLM in RE): what Techniques are used to optimize and evaluate the LLMs in software requirement engineering?

Q3. (Trend and opportunities): what are the challenges and directions for utilizing LLMs in requirement engineering?

We carried out a literature review of published studies covering the use of LLMs for requirement engineering in the last five years (2020–2024). This provides a broad range of information on the prevalence and application of LLMs in the RE process. We provided a complete replication package to enable the easy reproduction of our systematic review. The prepared package that has been made publicly available[1] includes a spreadsheet containing a list of filtered articles, the classification of primary studies, and extracted data.

We started the search process by selecting the electronic repositories we used for the search, such as IEEE Xplore,[2] ACM Digital Library,[3] Science Direct,[4] Google Scholar,[5] Wiley Online Library,[6] and Springer Link.[7] We identified keywords and formulate appropriate search strings from research questions and a broad investigation of known primary studies. After a series of test executions and refinements, we grouped keywords into two sets include keywords that defined "Requirement Engineering Process" and related to "LLMs" as are described in Table 1. Every search string must contain at least one keyword from each term set. Moreover, to reduce the likelihood of bias, we provide inclusion (I) and exclusion (E) criteria as follows:

1 https://github.com/MSharbaf/LLMs_in_RE
2 http://ieeexplore.ieee.org
3 http://dl.acm.org
4 http://www.sciencedirect.com
5 https://scholar.google.com/
6 https://onlinelibrary.wiley.com
7 http://www.springer.com

TABLE 1 Search string keywords.

Term set	Keywords
Set1	Requirement Engineering Process, Requirement Specification, Requirement Formalization RE, Software Requirement Engineering, Requirements Elicitation, Software Specification
Set2	Large Language Model, LLM, Pre-trained Language Model, Transformer Model Deep Learning Model, Generative Language Model, AI Language Model

- I1: Papers published between January 2020 and September 2024.
- I2: Publications that describe the use of LLMs for extracting/specifying software requirements
- I3: Publications in peer-reviewed journals, conferences, and workshops
- I4: Publication in English
- E1: Publications not written in English
- E2: Summary, survey, or review publications
- E3: Publications not focusing on RE process

Figure 1 outlines our search and selection process. This process involved an initial automatic database search and a subsequent iterative snowballing-based search. We performed the defined search queries on each repository to acquire a set of relevant articles based on the search string, which was configured to result only in research papers, such as journal articles, conference papers, and workshop papers. This activity yielded 287 articles, where the number of articles obtained from each repository is shown in Figure 1. The initial pool of studies, comprising 73 articles, was generated after collecting all the relevant papers and eliminating duplicates. In the next step, an initial filtering was applied by inspecting the title, abstract, introduction, and conclusion. To further enhance the precision of article selections, we applied inclusion and exclusion criteria, resulting in 28 articles. Then, we employed backward and forward snowballing-based search to guarantee thorough coverage of the study's extensive scope. The snowballing process was repeated in three iterations, with the newly included articles leading to the identification of 6 new articles that met the selection criteria In the next step, we reviewed the full texts of the 34 obtained articles and selected 29 of them through a manual screening process as primary studies to identify and collect from them the appropriate and relevant information to answer our research questions.

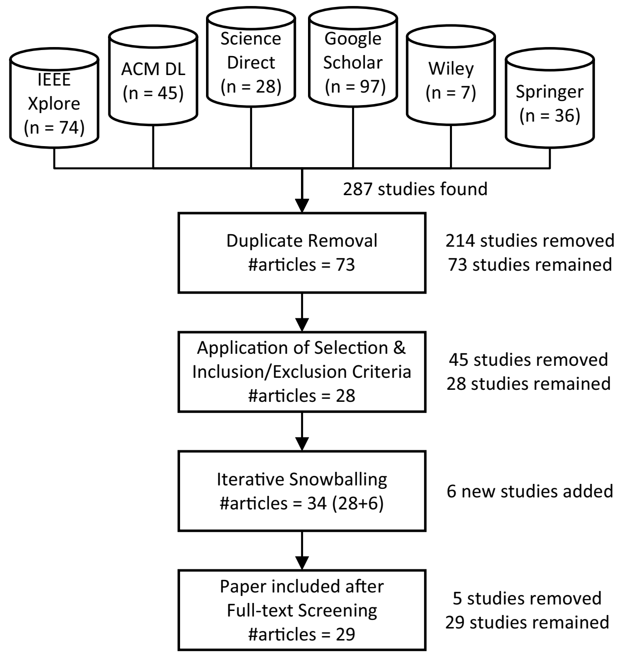
IEEE Xplore (n = 74)
ACM DL (n = 45)
Science Direct (n = 28)
Google Scholar (n = 97)
Wiley (n = 7)
Springer (n = 36)
287 studies found
Duplicate Removal #articles = 73
214 studies removed 73 studies remained
Application of Selection & Inclusion/Exclusion Criteria #articles = 28
45 studies removed 28 studies remained
Iterative Snowballing #articles = 34 (28+6)
6 new studies added
Paper included after Full-text Screening #articles = 29
5 studies removed 29 studies remained

FIGURE 1
Overview of search and selection process.

4 Results

In this section, we report the classification results of the investigated approaches for each research question.

4.1 Q1: What roles do LLMs serve in RE process?

This research investigates how LLMs are integrated into RE process, focusing on their contributions to Software Requirements Engineering (SRE). The main question is divided into three sub-questions:

- **Q1.1:** What types of LLM architectures are commonly employed in SRE?
- **Q1.2:** What are the typical inputs provided to LLMs in the RE process, and what outputs do they generate?
- **Q1.3:** Which stages of the RE process integrate LLMs most effectively, and what roles do they fulfill?

Together, these questions provide a comprehensive overview of the types of LLMs used, their input-output mechanisms, and their functional roles across different stages of RE.

4.1.1 Q1.1: What types of LLM are commonly employed in SRE?

The following subsections categorize the types of LLMs employed in RE processes, focusing on their specific applications and architecture, as presented in Table 2.

4.1.1.1 Generative pre-trained transformers (GPT)

Generative Pre-trained Transformers (GPT), including models like ChatGPT and GPT-3, have seen wide usage in RE. These models aid in generating requirement documents, system code, and test cases, thereby facilitating automation in repetitive tasks and accelerating documentation workflows. For instance, ChatGPT has been employed to produce initial drafts of requirement specifications and even code snippets, which were later refined by human engineers. Additionally, GPT models support progressive prompting techniques to enhance the relevance and specificity of generated outputs iteratively. The output is a specification of the maximum probability output from the model given the input text (prompt + NL requirements) and the trained model parameters.

$$y_i = \text{GPT}(x_i) = \arg\max P(y|x, \theta_{GPT})$$

Where x_i is the input text (such as a user prompt), y_i represents the generated output, and θ_{GPT} represents the trained model parameters.

4.1.1.2 Code-focused language models

Code-centric models, such as Codex, are designed explicitly for programming-related tasks. In RE, Codex is often used to directly translate natural language requirements into executable code, thus bridging the gap between high-level requirements and specific technical implementations. Codex also plays a crucial role in test case generation, helping to validate requirements by automatically creating test scripts based on initial requirement inputs, which helps streamline the verification process.

$$\text{Code} = \text{Codex}(\text{Requirement}) = \arg\max P(\text{Code}|\text{Requirement}, \theta_{Codex})$$

where Requirement is the input requirement text, and Code is the generated executable code.

4.1.1.3 Bidirectional encoder representations (BERT) and variants

BERT and its derivatives provide significant advantages in analyzing textual requirements due to their context-aware nature. These models support RE by performing tasks such as requirement classification, contradiction detection, and keyword extraction. For example, BERT-based models can scan requirement documents to identify conflicting requirements or to extract entities relevant to the project, thereby ensuring consistency and quality.

4.1.1.4 Transformer-based models

Models like T5 and Llama 2 extend the use of transformer architectures to tasks like paraphrasing and summarizing requirements. This capability is essential in RE, where technical language must often be translated into more accessible terms for various stakeholders. Transformer-based models also enable the summarization of extensive requirement documents, making it easier for stakeholders to comprehend complex information.

$$\text{Paraphrase} = \text{T5}(x) = \arg\max P(\text{Paraphrase}|x, \theta_{\text{T5}})$$

where x is the input text, such as a requirement statement, and Paraphrase is the output.

4.1.1.5 Multimodal models

Multimodal models, which can process both text and visual data, are particularly beneficial for RE tasks that involve diagrams or visual aids. These models can integrate user interface sketches, workflow diagrams, and other visual assets alongside text-based requirements, enabling comprehensive understanding and documentation. This is particularly useful in software projects that require detailed graphical interfaces or workflow visualization.

TABLE 2 Detailed categorization of LLMs in requirement engineering.

Category	#Studies	Papers
Generative pre-trained transformers (GPT)	23	P1, P2, P4, P5, P6, P7, P9, P10, P11, P13, P14, P15, P17, P18, P20, P21, P22, P24, P25, P26, P27, P28, P29
Code-focused language models	14	P2, P3, P6, P8, P12, P15, P16, P19, P21, P23, P26, P27, P28, P29
Bidirectional encoder representations (BERT) and variants	5	P5, P11, P17, P23, P24
Transformer-based models	10	P6, P7, P8, P14, P16, P19, P21, P23, P25, P28
Multimodal models	6	P10, P13, P15, P16, P18, P22
Specialized models	9	P1, P5, P11, P12, P15, P18, P22, P24, P27
Vision-language models	7	P3, P8, P10, P13, P16, P18, P25

$$\text{Output} = \text{MultimodalModel}(\text{Text}, \text{Image})$$

Where Text and Image are inputs to the multimodal model, producing a combined output.

4.1.1.6 Specialized models

Specialized models are tailored to meet the specific needs of a given domain, such as finance, healthcare, or education. They are trained on domain-specific data to better understand unique terminology and nuances, providing more accurate interpretations of requirements in specialized sectors. In RE, such models improve requirement extraction and ensure precision in niche industries by reducing ambiguity and misinterpretation.

$$P(y|x, D) = \text{SpecializedModel}(x, D)$$

where x is the input text, y is the output, and D denotes the domain-specific dataset.

4.1.1.7 Vision-language models

Vision-language models bridge the gap between textual requirements and visual design elements. These models are useful in RE when the project involves visual requirements, such as user interface design or physical system blueprints. These models enhance clarity in design-related requirements by aligning visual and text-based requirements and validate the alignment between visuals and textual descriptions.

$$\text{Output} = \text{VisionLangModel}(\text{Text}, \text{Visual})$$

where Text is the textual requirement and Visual is the corresponding image or design input.

TABLE 3 Overview of input categories for LLM4SE studies.

Input category	#Studies	Papers
Natural language requirements	21	P1, P2, P3, P4, P5, P6, P8, P10, P11, P12, P13, P14, P15, P18, P19, P20, P21, P25, P26, P27, P28
Technical documentation	6	P7, P10, P11, P14, P22, P28
Programming codes	7	P7, P16, P17, P18, P23, P24, P25
Examples and sketches	1	P23
Model and formal specifications	2	P9, P29

TABLE 4 Overview of output categories for LLM4SE studies.

Output category	#Studies	Papers
Modeling and diagrams	5	P1, P4, P19, P20, P26
Software requirements and specifications	9	P2, P11, P12, P13, P15, P21, P22, P27, P28
Formal specifications and DSL	4	P3, P7, P9, P29
Simulation and predictions	3	P5, P6, P18
Code and pseudocode	6	P13, P16, P17, P23, P24, P25

4.1.2 Q1.2: What are the typical inputs provided to LLMs in the RE process, and what outputs do they generate?

In the Requirement Engineering (RE) process, Large Language Models (LLMs) are provided with a range of input types, each tailored to produce specific outputs that meet the needs of software engineering tasks. This section outlines the primary categories of inputs and outputs, along with examples from studies that utilize each type.

4.1.2.1 Input categories

As shown in Table 3, we define the following attributes to categorize the input of LLMs identified in the investigated articles.

- **Natural language requirements** (e.g., **P1, P2, P5**): natural language inputs include descriptions of software requirements, user stories, and project goals. These inputs help the model understand high-level objectives and generate models, diagrams, or structured requirements. Studies like **P8, P13, P19** use natural language prompts to create UML diagrams, while others generate detailed Software Requirements Specifications (SRS).
- **Technical documentation** (e.g., **P7, P10, P11**): technical documentation, such as project glossaries, specifications, or structured documents, provides a foundation for generating formal specifications or structured software requirements. For example, **P22** uses technical specifications to generate formalized requirements that align with engineering standards.
- **Programming codes** (e.g., **P7, P16, P17**): code inputs, such as snippets or buggy programs, enable LLMs to generate formal specifications, bug fixes, or refactored code. In **P24** and **P25**, models analyze code snippets to identify faults or generate summaries of code functionalities.
- **Examples and sketches** (e.g., **P23**): example-based inputs include sketches or examples provided as a basis for LLMs to generate code or design structures. These inputs help guide the model in generating structured outputs, as in **P23**, where sketches are used to illustrate pseudo-code or code sketches.
- **Model and formal specifications** (e.g., **P9, P29**): these inputs include formal language specifications like Alloy models or other domain-specific languages (DSLs), allowing LLMs to create structured code or models aligned with RE standards. Studies like **P29** use faulty Alloy specifications as input, prompting the model to generate corrected formal specifications.

4.1.2.2 Output categories

As shown in Table 4, we define the following attributes to categorize the output of LLMs identified in the investigated articles.

- **Modeling and diagrams** (e.g., **P1, P4, P19**): LLMs generate models and diagrams, including UML diagrams (use case, class, sequence) based on natural language inputs or technical documentation. These outputs, seen in studies like **P20** and **P26**, help visualize system structures and interactions.
- **Software requirements and specifications** (e.g., **P2, P11, P12**): generated specifications include Software Requirements Specifications (SRS), user stories, or functional requirements. In **P22** and **P27**, LLMs produce comprehensive specifications that capture both functional and non-functional aspects of a system.
- **Formal specifications and DSL** (e.g., **P3, P7, P9**): formal outputs involve DSL or structured code that represents precise specifications, such as JML (Java Modeling Language) or Alloy code. Studies like **P29** focus on generating or repairing formal specifications in Alloy, ensuring syntactic and semantic correctness.
- **Simulation and predictions** (e.g., **P5, P6, P18**): simulation models, such as system dynamics or event-driven models, are generated for tasks like testing or predictive analysis. In **P18**, the model produces programming exercises with sample solutions and code explanations, simulating educational or training tasks.
- **Code and pseudocode** (e.g., **P13, P16, P17**): LLMs generate code outputs, including corrected or refactored code snippets, pseudocode, and bug fixes. For instance, **P23** uses code sketches as input to generate working code snippets, while **P24** focuses on identifying faults and generating solutions in code.

Each of these input and output categories supports specific tasks in RE, enabling LLMs to generate relevant outputs that align with RE standards and requirements.

TABLE 5 Overview of RE process stages utilizing LLMs in SE tasks.

RE process stage	#Studies	Papers
Requirements elicitation and analysis	6	P5, P14, P19, P21, P27, P28
Requirements modeling and specification	10	P1, P6, P7, P8, P12, P13, P18, P19, P20, P22
Formal specification and verification	5	P3, P7, P9, P12, P29
Validation and quality assurance	10	P2, P10, P11, P13, P14, P15, P16, P17, P23, P24

4.1.3 Q1.3: Which stages of the RE process integrate LLMs most effectively, and what roles do they fulfill?

In the Requirement Engineering (RE) process, Large Language Models (LLMs) are applied across different stages to support diverse tasks. Each stage leverages LLMs in unique ways, enabling more efficient, accurate, and automated RE tasks. Below, we detail the primary stages where LLMs are integrated, along with examples from studies that use LLMs for each purpose (see Table 5).

4.1.3.1 Requirements elicitation and analysis

Requirements elicitation and analysis involve identifying and refining a project's initial requirements. LLMs play a significant role in this stage by processing natural language inputs and generating initial requirement drafts or analyses.

- LLMs assist in identifying missing terminology or critical information in initial requirements, enhancing completeness and accuracy (**P5**).
- During the elicitation process, models provide feedback on requirement feasibility and clarity, as seen in studies like **P14**, where they assess requirement appropriateness and correctness.
- Models are used to convert user stories into structured requirements, allowing for a smoother transition from high-level descriptions to specific RE inputs (**P19, P21**).
- LLMs facilitate interactive requirement elicitation by interpreting and analyzing inputs from stakeholders, as in studies **P27** and **P28**.

4.1.3.2 Requirements modeling and specification

This stage involves creating structured representations of requirements, such as UML models or formalized specifications. LLMs streamline this process by generating, validating, or converting requirements into structured formats.

- LLMs generate UML diagrams like use case, class, and sequence diagrams based on natural language requirements, as demonstrated in studies like **P1, P8**.
- In the modeling phase, models can also translate high-level requirements into more formal or structured specifications, enabling effective visualization and model verification (**P6, P12**).
- Models like those in **P7** support specification generation and verification tasks, automating repetitive modeling processes in the design phase.
- LLMs assist in creating architecture specifications from RTL code, a critical part of the specification generation stage (**P8**).
- LLMs provide visualization of requirements through model specifications, enabling clearer representation of system functionalities, as explored in studies like **P13, P18, P19, P22**.

4.1.3.3 Formal specification and verification

In this stage, LLMs generate formalized requirements that adhere to specific standards or programming languages. They are often used to verify requirement correctness and consistency.

- LLMs translate natural language specifications into formal code or Domain-Specific Language (DSL), enhancing precision and enabling formal verification (**P3**).
- During verification, models assist in checking for syntactic and semantic correctness, ensuring that the generated formal specifications meet predefined standards (**P7, P9**).
- Studies like **P12** explore how LLMs can be applied to generate and verify formal specifications, enhancing system reliability through rigorous validation.
- Models are used in debugging and repairing specifications, such as Alloy models, where they provide automated repairs to faulty code or models (**P29**).

4.1.3.4 Validation and quality assurance

Validation and quality assurance (QA) involve evaluating the completeness, accuracy, and feasibility of requirements to ensure they meet project standards and objectives. LLMs play a substantial role in automating and enhancing these evaluations.

- In QA tasks, LLMs review and validate Software Requirements Specifications (SRS), providing insights on completeness, clarity, and correctness (**P2, P10**).
- Models can automate the review of technical documents, identifying potential issues and generating suggestions for improvement (**P11, P15**).
- Quality assessment tasks include evaluating requirements based on dimensions such as appropriateness, feasibility, and consistency, as seen in studies **P14** and **P17**.
- LLMs are also applied to validate generated code, test case specifications, and fault localization, ensuring outputs align with RE standards and objectives (**P13, P16, P24**).
- In **P23**, models contribute to identifying code faults and generating corrective patches, supporting quality improvement in RE outputs.

Each of these RE process stages showcases the versatility of LLMs, highlighting their ability to enhance the RE process through automation, formal verification, quality assurance, and interactive requirement elicitation.

4.2 Q2: What techniques are used to optimize and evaluate the LLMs in SRE?

Optimizing and evaluating Large Language Models for Requirement Engineering (LLM4RE) involves a range of specialized techniques to enhance model performance and ensure the accuracy of outputs tailored to software engineering (SE) tasks. This question explores these methods through three subquestions:

- **Q2.1** examines various *tuning techniques* that adjust model parameters to better align with SE-specific tasks, helping LLMs generate responses that are relevant, contextually accurate, and aligned with requirements engineering goals.
- **Q2.2** investigates *prompt engineering techniques*, which optimize how inputs are structured to elicit high-quality, targeted responses from LLMs in SE scenarios.
- **Q2.3** focuses on the *evaluation metrics* used to assess the performance of LLM4SE, detailing how these metrics help measure the effectiveness, reliability, and precision of model outputs in requirements engineering contexts.

Together, these subquestions provide a comprehensive overview of the strategies that optimize LLM4RE, covering both enhancement and assessment techniques to improve the application of LLMs in SE tasks.

4.2.1 Q2.1: What tuning techniques are used to enhance the performance of LLMs in SE tasks?

Optimizing large language models (LLMs) for Software Engineering (SE) tasks, specifically in Requirement Engineering (RE), employs various tuning techniques to enhance model performance and align outputs with domain-specific needs. These tuning strategies are designed to refine model parameters and response behaviors to improve accuracy, relevance, and efficiency in SE tasks. Below, we discuss the primary tuning strategies used in LLM4RE, accompanied by references to relevant studies that utilize each method (see Table 6).

- **Full fine-tuning** (e.g., **P25**): this comprehensive tuning approach involves updating all model parameters using a task-specific dataset. Full fine-tuning is resource-intensive, but allows deep integration of SE knowledge into the model. By adjusting all weights, full fine-tuning helps the model respond with high specificity and adaptability to SE tasks, making it highly effective for complex requirement generation and analysis tasks.
- **Prompt tuning** (e.g., **P2**): prompt tuning is a parameter-efficient method where only prompt embeddings are trained while keeping other parameters unchanged. This strategy allows for targeted modifications to the model's responses to SE-specific queries, enhancing its ability to interpret and process SE prompts with minimal computational load. Prompt tuning is ideal in scenarios where quick adjustments are needed without altering the entire model structure.

TABLE 6 Overview of tuning strategies for LLM4SE.

Tuning strategy	#Studies	Papers
Full fine-tuning	1	P25
Prompt tuning	1	P2
Context tuning	1	P2
Low-rank adaptation	2	P10, P16
Custom decoder fine-tuning	1	P11
Instruction fine-tuning	1	P16
Fine-tuning with domain specific knowledge	1	P14
No tuning techniques	23	P1, P3, P4, P5, P6, P7, P8, P9, P12, P13, P15, P17, P18, P19, P20, P21, P22, P23, P24, P26, P27, P28, P29

- **Context tuning** (e.g., **P2**): context tuning involves supplementing the input prompt with additional SE-specific information, enabling the model to access a broader range of relevant knowledge without changing its internal parameters. By incorporating project context or SE-related terms, context tuning enhances response relevance and improves the model's performance in specific requirements engineering tasks.
- **Low-rank adaptation (LoRA)** (e.g., **P10, P16**): LoRA introduces low-rank matrices to fine-tune only a subset of model parameters, which reduces memory usage and computational cost. In Quantized LoRA (QLoRA), quantization is applied to enhance efficiency further. This technique has been applied to models like LLaMA2-13B in SE contexts, providing adaptable performance in requirements engineering without a complete overhaul of the model's architecture. LoRA is well-suited for large-scale SE tasks requiring computational efficiency.
- **Custom decoder fine-tuning** (e.g., **P11**): in custom decoder fine-tuning, the model's decoder layer is adjusted to better interpret and produce SE-specific language. This technique tailors the output generation mechanism to the syntactic and semantic demands of SE tasks, particularly helpful in tasks such as specification synthesis. The model generates more accurate, SE-focused outputs by customizing the decoding process.
- **Instruction fine-tuning** (e.g., **P16**): instruction fine-tuning trains the model on SE task-specific instructions, enabling it to follow structured guidelines closely. This technique has been applied to models like CodeUp-13B-Chat, making them proficient in tasks where structured, step-by-step responses are needed, such as documenting requirements or generating test cases. By aligning the model's responses to SE-specific instructions, instruction fine-tuning helps achieve greater accuracy in SE tasks.
- **Fine-tuning with domain-specific knowledge** (e.g., **P14**): this approach involves training the model on datasets that include SE-specific terminology and methodologies, such as requirements engineering and test-driven development. By

TABLE 7 Overview of prompt engineering techniques for LLM4SE.

Prompt engineering technique	#Studies	Papers
Few-shot prompting	11	P3, P4, P5, P7, P8, P9, P16, P19, P21, P25, P28
Zero-shot prompting	3	P9, P16, P21
Iterative prompts	10	P2, P4, P6, P7, P8, P13, P15, P18, P20, P26
Contextual prompts	14	P1, P2, P3, P10, P11, P12, P14, P17, P19, P22, P23, P24, P27, P29

fine-tuning on domain-relevant data, the model incorporates specialized knowledge that enhances its performance in SE contexts, allowing it to generate accurate, domain-specific responses for tasks like requirements analysis and system design.

- **No tuning techniques** (e.g., **P1, P3, P4, P5, P6, P7, P8, P9, P12, P13, P15, P17, P18, P19, P20, P21, P22, P23, P24, P26, P27, P28, P29**): in some cases, LLMs are applied directly to SE tasks without any tuning modifications, relying on pre-trained capabilities. This approach provides a baseline for evaluating the model's natural adaptability to SE tasks. While it may lack the specificity of tuned models, the baseline performance offers insights into the inherent versatility of LLMs and establishes a point of comparison for assessing the effectiveness of tuning strategies.

Each of these tuning strategies contributes uniquely to optimizing LLM4RE, allowing models to deliver more accurate, context-sensitive, and resource-efficient responses tailored to software engineering tasks. These techniques enable LLMs to handle the complexities of requirements engineering with increased precision and contextual alignment.

4.2.2 Q2.2: What prompt engineering techniques are applied to improve the performance of LLMs in SE tasks?

Prompt engineering is critical for guiding large language models (LLMs) in Requirement Engineering (RE) tasks. It allows precise control over model outputs by structuring inputs effectively. This section details four key prompt engineering techniques utilized to enhance LLM4RE and references relevant studies (see Table 7).

4.2.2.1 Zero-shot prompting

Zero-shot prompting allows the model to generate responses without any prior example in the prompt, relying solely on the model's pre-trained knowledge. In the context of Software Engineering (SE), this approach is often used for more straightforward tasks where minimal domain-specific context is necessary, such as providing concise definitions, brief code snippets, or summarizing short requirements. Because zero-shot prompting does not supply the model with explicit examples, the instructions must be sufficiently clear and well-structured to guide the LLM effectively. Despite its simplicity, zero-shot prompting can be surprisingly effective for simpler or standard tasks, but it may require more careful prompt engineering or additional context for more complex SE scenarios.

- Zero-shot prompting is often used in simpler SE tasks where model responses are more predictable and can be generated without specific examples (e.g., **P9**).
- This technique requires clear task instructions within the prompt to guide the LLM effectively. For instance, **P21** provides structured prompts for SE tasks to ensure clarity and accuracy.
- In complex scenarios, zero-shot prompting is often combined with context-rich instructions to improve response relevance (e.g., **P16**).

4.2.2.2 Few-shot prompting

Few-shot prompting involves providing the model with a small set of examples in the prompt to illustrate the task at hand. These examples help the LLM understand the desired format, style, or structure of the output, reducing ambiguity. In SE contexts, few-shot prompts are particularly beneficial for tasks that require a specific style of writing (e.g., user stories, requirement statements, or test cases) or adherence to certain design patterns. By seeing well-chosen examples, the model can more accurately infer how to generate responses that align with SE standards or domain conventions. Careful selection of representative examples, covering typical use cases and edge cases, can significantly improve the consistency and accuracy of the model's outputs.

- Few-shot prompting can guide the model with relevant examples, enabling it to mimic the patterns and syntax found in the examples to generate accurate SE outputs (e.g., **P3, P4, P5**).
- In **P7**, predefined examples were used in initial prompts for specification generation, followed by feedback-driven adjustments to improve results iteratively.
- Few-shot prompting combined with example-based instructions helps the LLM interpret complex SE tasks, such as generating Data Flow Diagrams (DFDs) and modeling requirements (e.g., **P19, P25**).
- Studies like **P16, P21** use few-shot prompting to show examples of DSL (Domain-Specific Language) syntax, helping LLMs produce outputs compatible with SE frameworks.

4.2.2.3 Iterative prompts

Iterative prompting is a technique where the LLM is prompted multiple times with progressively refined instructions, allowing the model to improve its outputs iteratively. This approach is especially useful for more complex or exploratory SE tasks, where the first response may only be a draft or partial solution. Through iterative prompting, users can analyze the LLM's output, identify gaps or inconsistencies, and provide additional guidance or corrections in subsequent prompts. This feedback loop helps the model converge toward higher-quality results. Common applications in SE include refining requirement statements, incrementally generating design models, or progressively clarifying ambiguities within a specification.

- This approach is beneficial for SE tasks requiring gradual refinement, such as extracting requirements or refining system models (e.g., **P2, P4, P6**).
- Iterative prompting can involve asking follow-up questions or providing additional instructions based on previous responses, improving output quality for tasks like generating code or requirements summaries (e.g., **P8, P13**).
- Studies like **P15, P18, P20** use iterative prompts to address complex requirements engineering tasks, adjusting model responses based on feedback or verification tools.
- In **P26**, path-based prompt augmentation was applied, refining model outputs over multiple prompts to improve specification accuracy.

4.2.2.4 Contextual prompts

Contextual prompts are designed to provide the model with extensive information about the task or domain, enhancing the model's ability to produce relevant and accurate responses. Such prompts typically include detailed background, domain-specific terminologies, relevant standards (e.g., ISO 29148 for requirements), or even partial SE artifacts like user stories or architecture diagrams. By embedding this information directly into the prompt, the model can leverage a richer context to generate solutions that align more closely with project-specific needs or conventions. Contextual prompting is especially powerful for tasks where general knowledge is insufficient, and precise domain context is critical–such as validating regulatory requirements, generating UML diagrams with correct syntax, or adhering to specific modeling notations.

- Contextual prompts typically include additional information, such as user stories, design specifications, or SE project descriptions, to clarify the model's task (e.g., **P1, P2, P3**).
- In **P10** and **P14**, prompts incorporated project-specific terminology and quality characteristics based on ISO 29148, guiding the LLM to respond accurately within a defined SE framework.
- Providing context can help the model understand SE-specific syntax or expected formats, such as DSL code or PlantUML, and improve the relevance of generated outputs (e.g., **P11, P12**).
- Contextual prompts are also used to create structured tasks in SE, where prompts outline task descriptions, input formats, and examples to guide the model's understanding of expected outputs (e.g., **P19, P22, P23**).
- Techniques like co-prompting and context-rich instructions can enhance model adaptability, as seen in **P27** and **P29**, by incorporating multiple prompts that provide progressive layers of context.

4.2.3 Q2.3: How are evaluation metrics utilized to assess the performance of LLM4SE tasks?

Evaluation metrics are essential for assessing the effectiveness and quality of LLMs in SE tasks. These metrics measure various aspects of model performance, such as correctness, precision, and overall reliability, ensuring that the models meet the specific requirements of SE tasks. Below, we detail the primary evaluation metric categories and references to studies that apply each approach (see Table 8).

TABLE 8 Overview of evaluation metrics for LLM4SE.

Evaluation metric category	#Studies	Papers
Correctness and completeness metrics	10	P1, P2, P3, P6, P7, P8, P9, P13, P19, P20
Performance metrics	6	P7, P10, P17, P20, P24, P25
Agreement and consistency metrics	5	P11, P14, P16, P26, P29
Precision and recall metrics	6	P5, P12, P14, P21, P22, P28
Qualitative and expert-based evaluation	5	P4, P8, P10, P15, P27
Code quality and maintainability metrics	5	P6, P18, P23, P24, P29

4.2.3.1 Correctness and completeness metrics

Correctness and completeness metrics assess how accurately and comprehensively LLMs perform SE tasks, ensuring that generated outputs fully represent the specified requirements.

- These metrics evaluate if the generated outputs, like UML models or requirements documents, are correct in structure and completeness (e.g., **P1, P2, P3**).
- Studies like **P6** and **P8** assess correctness by comparing model outputs to predefined requirements, examining factors like adherence to system specifications.
- Completeness metrics are applied to ensure that all essential elements of a model or specification are present, such as actors, relationships, or sequence steps in UML diagrams (e.g., **P19, P20**).
- In some cases, correctness is evaluated based on successful verification of generated specifications, where completeness is measured by matching outputs with user stories or predefined functional requirements (e.g., **P7, P13**).

4.2.3.2 Performance metrics

Performance metrics gauge the efficiency and reliability of LLM outputs, particularly in tasks involving repetitive or large-scale generation, like specification synthesis.

- These metrics assess the overall reliability of outputs, including metrics like the number of specifications passing verification (e.g., **P7**).
- In **P10** and **P17**, performance metrics include the accuracy of model responses across multiple iterations, particularly focusing on the model's ability to fulfill SE tasks consistently.
- Metrics like Correct@6, which measures the number of correct outputs within six iterations, are used to benchmark performance against other SE tools (e.g., **P24, P25**).

- Studies also use comparison-based metrics, evaluating the LLM's performance against other state-of-the-art SE tools, such as ARepair or ICEBAR, providing insight into relative efficiency (e.g., **P20**).

4.2.3.3 Agreement and consistency metrics

Agreement and consistency metrics evaluate how consistently LLMs generate outputs in line with human assessments or predefined standards, ensuring reliability across multiple runs.

- Agreement metrics, such as Cohen's Kappa, measure the alignment between model assessments and those of human evaluators, quantifying consistency (e.g., **P14**).
- BLEU, ROUGE-L, METEOR, and BERTSim scores are employed to evaluate output consistency in tasks that require high linguistic similarity to human-generated texts (e.g., **P29**).
- In **P16**, agreement metrics help determine the LLM's accuracy in interpreting SE requirements, while consistency metrics validate that outputs are reproducible across multiple prompts.
- McNemar's Test is sometimes used to validate model outputs statistically, comparing consistency in fulfilling requirements across different prompt variations (e.g., **P26**).

4.2.3.4 Precision and recall metrics

Precision and recall metrics are vital for evaluating the accuracy and relevancy of LLM outputs in SE, especially in tasks requiring high retrieval quality and minimal error rates.

- **Precision** measures the accuracy of model outputs by calculating the ratio of correctly identified SE elements to all identified elements, which is essential for tasks like identifying terminology in requirements (e.g., **P5, P12**).
- **Recall** evaluates the model's ability to capture all relevant SE information by assessing the comprehensiveness of the generated outputs. Recall is calculated as the ratio of correctly identified relevant items to the total number of relevant items, ensuring the model's outputs cover all necessary aspects in RE tasks (e.g., **P21, P22**).
- F1 scores, a harmonic mean of precision and recall, are frequently used to balance accuracy and completeness in SE tasks, ensuring both aspects are considered (e.g., **P14, P28**).

4.2.3.5 Qualitative and expert-based evaluation

Qualitative metrics, often derived from expert feedback, assess the quality and usability of model outputs beyond quantitative metrics, capturing human perceptions of usefulness and accuracy.

- Expert-based evaluations assess outputs based on relevance, clarity, and adherence to requirements, allowing domain experts to qualitatively gauge LLM performance (e.g., **P4, P8**).
- RUST (Readability, Understandability, Specifiability, Technical aspects) scores are employed in studies like **P10** to evaluate user stories generated by the model.
- Feedback-based evaluations capture developer responses to model outputs, assessing the LLM's capability to meet real-world SE needs (e.g., **P15, P27**).

4.2.3.6 Code quality and maintainability metrics

These metrics measure the technical quality of code generated by LLMs, focusing on aspects like readability, maintainability, and conformance to coding standards.

- Metrics such as the Jaccard index, Cosine Similarity, and Validity@K assess generated code's structural and syntactic quality, comparing it to SE standards (e.g., **P6, P18**).
- Studies like **P23** evaluate generated code based on quality indicators such as maintainability and readability, ensuring that outputs are correct and usable in long-term projects.
- Consistency metrics like Pass@k and Correctness@k assess the model's ability to generate correct and usable code within specific tolerances, allowing for effective benchmarking against SE requirements (e.g., **P24, P29**).

Each evaluation metric significantly assesses LLM4SE's performance and reliability, ensuring that models produce outputs that meet the quality standards required for software engineering tasks.

4.3 Q3: What are the challenges and directions for utilizing LLMs in RE?

The utilization of Large Language Models (LLMs) in Requirement Engineering (RE) brings both significant opportunities and notable challenges. These challenges highlight current limitations and complexities in adopting LLMs, while future directions offer promising paths for improvement and innovation. Below, we detail the primary challenges and directions, with relevant studies for each category (see Tables 9, 10).

4.3.1 Challenges in utilizing LLMs in requirement engineering

In the following, we discuss challenges in the use of LLMs in the RE process, building upon our research results presented in Section 4.

- **Understanding and knowledge challenges** (e.g., **P3, P4, P15**): LLMs often struggle with domain-specific knowledge, leading to errors or oversights in requirements interpretation. Challenges include a lack of understanding of company-specific rules and limited context in domain-specific tasks (**P22, P28**).
- **Output quality and completeness issues** (e.g., **P2, P6, P8**): quality challenges involve incomplete or vague outputs, often resulting in requirements that lack detail or specificity. Studies such as **P10** and **P11** report hallucination issues, where LLMs generate misleading or incorrect information, necessitating extensive manual corrections.
- **Technical and procedural limitations** (e.g., **P4, P7, P12**): technical limitations include difficulties in handling structured inputs and maintaining consistency across iterative tasks. For example, **P24** notes that LLMs may fall short in tasks requiring

TABLE 9 Challenges in utilizing LLMs in requirement engineering.

Challenge category	#Studies	Papers
Understanding and knowledge challenges	7	P3, P4, P15, P19, P21, P22, P28
Output quality and completeness issues	10	P2, P6, P8, P10, P11, P13, P14, P18, P27, P29
Technical and procedural limitations	5	P4, P7, P12, P24, P26
Challenges in code and testing	6	P7, P15, P17, P18, P23, P25
Prompting and input challenges	6	P16, P20, P21, P23, P27, P28
Empirical and experimental limitations	3	P23, P24, P25
Formatting and structural issues	2	P26, P29

complex programming language syntaxes or in generating precise engineering models (**P26**).

- **Challenges in code and testing** (e.g., **P7, P15, P17**): LLMs often struggle with generating complete test cases, patching errors, or understanding code semantics. Studies like **P18** indicate that only a fraction of generated test cases met testing standards, while **P25** notes challenges in program synthesis and debugging.
- **Prompting and input challenges** (e.g., **P16, P20**): effective prompt construction is critical, yet challenging due to the limited token capacity of LLMs. Studies like **P21** emphasize issues with prompt effectiveness, while **P28** highlights that prompts lacking domain context may result in ill-formed or incomplete requirements.
- **Empirical and experimental limitations** (e.g., **P23, P24, P25**): empirical limitations include difficulties in setting optimal hyperparameters and experimental setups, which may affect the reliability of LLMs in RE. **P23** points out that experimental limitations can reduce LLM adaptability, with some studies relying on empirically chosen settings that might benefit from further tuning.
- **Formatting and structural issues** (e.g., **P26, P29**): formatting issues arise due to LLMs generating outputs with structural inconsistencies or syntax errors. For instance, **P29** notes struggles with structural issues in formal specifications, including type mismatches and improper operator usage in generated code.

4.3.2 Future directions for utilizing LLMs in RE

In the following, we discuss directions for future work on utilizing LLMs for RE process, building upon our research results presented in Section 4.

- **Evaluation and metrics** (e.g., **P1, P3, P14**): there is a need for refining evaluation metrics to assess LLM outputs in RE tasks accurately. For instance, **P15** suggests increasing sample sizes and utilizing diverse UML models to improve output validation across various projects.
- **Model improvement and fine-tuning** (e.g., **P2, P6, P10**): fine-tuning LLMs to handle RE tasks with domain-specific data is a priority. Studies like **P12** recommend using techniques that allow models to handle complex specifications and longer text sequences, while **P11** focuses on improving fine-tuning to reduce hallucinations.
- **Prompt engineering and techniques** (e.g., **P4, P7, P17**): enhanced prompt engineering is critical for achieving high-quality outputs. Future work emphasizes refining prompt patterns and exploring advanced prompt structures to reduce errors and improve output specificity (**P18, P19**).
- **Integration and adaptation** (e.g., **P4, P8, P20**): integrating LLMs more effectively with existing RE tools and adapting them for broader RE workflows is suggested. **P24** discusses the need for LLMs to adapt within complex software design workflows, enhancing tool compatibility and task accuracy.
- **Performance and reliability** (e.g., **P7, P9, P13**): improving LLM reliability and consistency in RE processes remains a priority. Studies like **P25** emphasize the importance of enhancing generation efficiency, while **P13** suggests methods to improve reliability in code synthesis.
- **Hybrid approaches** (e.g., **P22, P26, P28**): hybrid methods that combine LLMs with traditional RE tools or human-in-the-loop validation are promising directions. For instance, **P29** proposes integrating ChatGPT with conventional tools for improved performance and accuracy, minimizing issues like hallucinations and inconsistency.

TABLE 10 Future directions for utilizing LLMs in requirement engineering.

Direction category	#Studies	Papers
Evaluation and metrics	4	P1, P3, P14, P15
Model improvement and fine-tuning	5	P2, P6, P10, P11, P12
Prompt engineering and techniques	8	P4, P7, P17, P18, P19, P21, P24, P27
Integration and adaptation	4	P4, P8, P20, P24
Performance and reliability	4	P7, P9, P13, P25
Hybrid approaches	4	P22, P26, P28, P29

Tables 9, 10 outline the challenges and future directions for improving LLM performance and reliability in RE, emphasizing the need for refined evaluation, enhanced model integration, and hybrid approaches to optimize LLM application across RE tasks.

5 Discussion

In this section, we try to draw some conclusions based on the results of each research question.

5.1 The role of LLM in RE process

Our investigation into the roles of LLMs in RE highlights several key patterns in their application and effectiveness across various RE tasks. Below, we summarize the primary findings for each subquestion.

5.1.1 Types of LLMs used in RE (Q1.1)

Across the studies, GGPT models are most frequently used in RE, with notable applications in automating documentation, code generation, and iterative refinement of requirements. This preference is attributed to GPT models' versatility and effectiveness in handling natural language, which is central to the RE process. Code-centric models like Codex and task-specific transformers (e.g., BERT for classification) further complement GPT models, addressing technical and analytical needs in RE. This combination reflects a strategic use of LLMs where both generative and specialized models are utilized to balance creativity and precision in RE tasks.

5.1.2 Input and output mechanisms (Q1.2)

The studies reveal a reliance on natural language requirements, technical documents, and code snippets as primary inputs, with outputs ranging from UML diagrams and test cases to structured requirement specifications. This diverse input-output handling reflects LLMs' flexibility in adapting to the varied nature of RE tasks, supporting requirements elicitation, modeling, and validation. The range of artifacts generated by LLMs, such as software specifications and code snippets, highlights their broad applicability, indicating an emerging pattern where LLMs serve as adaptable tools capable of supporting multiple facets of RE workflows. However, challenges persist regarding the quality and accuracy of the generated software specifications, as LLMs can produce hallucinations–plausible yet incorrect outputs–containing irrelevant or inappropriate elements. Therefore, a thorough investigation is needed to address this issue and develop solutions for the effective use of LLMs in requirements engineering.

5.1.3 Stage-wise integration of LLMs (Q1.3)

LLMs are effectively integrated at different stages of the RE process, particularly in requirements elicitation, modeling, and validation. For example, LLMs assist in refining initial requirements, generating formal models, validating specifications, and highlighting their multi-stage utility. The pattern of stage-wise integration underscores LLMs' role in enhancing accuracy and efficiency across the RE process, from initial requirements gathering to quality assurance, demonstrating their potential as comprehensive aids in end-to-end RE workflows. Although these findings highlight the strategic potential of LLMs in RE, leveraging their adaptability to navigate the complex, multi-stage nature of the process, several challenges persist. These include the risk of generating inaccurate or inconsistent results, reliance on high-quality training data, and difficulties in interpreting or validating complex outputs, all of which can impact their reliability and necessitate careful oversight in critical RE tasks.

5.2 Techniques to optimize and evaluate LLM4RE

Optimizing and evaluating LLM4RE employs various tuning, prompt engineering, and evaluation techniques tailored to software engineering needs. We summarize the key findings for each subquestion.

5.2.1 Tuning techniques for SRE tasks (Q2.1)

Our findings indicate a preference for parameter-efficient tuning techniques, such as Low-Rank Adaptation (LoRA) and prompt tuning, which offer computationally economical methods to adapt LLMs for SE-specific tasks. These techniques help models generate relevant responses to SE contexts while preserving computational efficiency. Full fine-tuning, though less common, is selectively used in complex SE tasks, where complete parameter adjustment can optimize model alignment with RE goals. This balanced use of tuning approaches underscores a strategic focus on efficiency without sacrificing model relevance in SE tasks.

Our analysis revealed that a substantial portion of the surveyed models do not employ fine-tuning techniques in the context of SRE. Several factors could explain this trend. First, fine-tuning large language models can be computationally expensive, requiring significant GPU resources and extended training times, which might not be feasible for many research teams or industry practitioners. Second, certain LLMs already possess robust baseline capabilities that can produce adequate performance on simpler requirement tasks without the added complexity of fine-tuning. Furthermore, some researchers may perceive prompt engineering as more flexible and cost-effective than adjusting internal model weights, since prompts can be iteratively refined with far less computational overhead. Lastly, licensing or data availability concerns can also discourage fine-tuning, particularly when data is confidential, limited, or requires a high level of preprocessing.

5.2.2 Prompt engineering techniques for SRE tasks (Q2.2)

Prompt engineering emerges as essential for guiding LLM responses in SE, with techniques such as few-shot prompting and iterative prompting providing practical ways to structure inputs for higher output quality. Few-shot prompting, which includes examples within the prompt, is particularly beneficial in generating SE-specific outputs, while iterative prompts allow continuous refinement of responses. Contextual prompts further enhance output accuracy by embedding domain-specific information. This layered prompting strategy maximizes response relevance and specificity in SE outputs. Notably, zero-shot prompting had the least reported usage in SRE contexts, possibly because it lacks the direct guidance from datasets or domain-specific examples that the other three strategies incorporate. Requirements tasks often demand precise understanding of domain constraints and stakeholder needs–demands that are better met by few-shot, iterative, or contextual prompts, each of which embeds domain knowledge or partial requirement statements into the prompt. By contrast, zero-shot prompts offer no tailored examples or situational cues, making them less effective for SRE's complexity, terminology, and accuracy requirements. Consequently, it appears more rational for researchers to adopt prompting methods that leverage some form of guided input, suggesting that zero-shot prompting alone is insufficient to handle the nuanced demands of software requirement engineering tasks.

The following are real-world prompt engineering examples applied in SRE, based on the work of Ronanki et al. (2024) and White et al. (2023). These prompts focus on tasks such as **requirements classification**, **requirements tracing**, **specification disambiguation**, **API generation**, and **change request simulation**.

Cognitive verifier (Ronanki et al., 2024): Classify the given list of requirements into functional (F) and non-functional requirements (NF). Ask me questions if needed to break the given task into smaller subtasks. All outputs from smaller subtasks must be combined before generating the final output.

Tracing cognitive verifier (Ronanki et al., 2024): List the IDs of requirements related to the [deprecated] feature in the requirements specification document below. Ask me questions if needed to break down the task into smaller subtasks. Combine outputs before finalizing the response.

Context manager (Ronanki et al., 2024): Classify the given list of requirements into functional (F) and non-functional requirements (NF). When you provide an answer, explain the reasoning and assumptions behind your response. Address potential ambiguities or limitations.

Tracing context manager (Ronanki et al., 2024): List the IDs of requirements related to the [deprecated] feature in the requirements document. Explain reasoning and assumptions. Address ambiguities for a more complete and accurate response.

Persona (Ronanki et al., 2024): Act as a requirements engineering domain expert and classify the given list of requirements into functional (F) and non-functional requirements (NF).

Tracing persona (Ronanki et al., 2024): Act as an expert and list the IDs of requirements dependent on the [deprecated] feature in the requirements document.

Question refinement (Ronanki et al., 2024): Classify the given list of requirements into functional (F) and non-functional requirements (NF). If needed, suggest a better version of the question that incorporates information specific to this task and ask me if I would like to use your question instead.

Tracing question refinement (Ronanki et al., 2024): List the IDs of requirements related to the [deprecated] feature from the requirements document below. If needed, suggest a better version of the question to incorporate specific information.

Template (Ronanki et al., 2024): Read the following list of requirements and return the IDs of non-functional requirements only. Write the result as a list like: (ID=X) (ID=Y) (ID=Z) where X, Y, and Z are IDs of non-functional requirements.

Tracing template (Ronanki et al., 2024): List the IDs of requirements related to the [deprecated] feature in the requirements document below. Follow the provided template when generating the output: ID list: X.X.X.X; X.X.X.X; X.X.X.X etc.

Specification disambiguation pattern (White et al., 2023): Identify ambiguous areas in requirement specifications provided by stakeholders and suggest refinements to clarify intent.

Example prompt: The following represents system requirements. Point out any areas that could be ambiguous or lead to unintended outcomes. Provide suggestions to make the language more precise.

API generator pattern (White et al., 2023): Generate API specifications from natural language descriptions or requirement lists.

Example prompt: Generate an OpenAPI specification for a web application that would implement the listed requirements.

Change request simulation pattern (White et al., 2023): Analyze the impact of changes in requirements and architecture.

Example prompt: My software system uses the OpenAPI specification that you generated earlier. Simulate a change where a new mandatory field needs to be added. List which functions and files will need modification.

These prompts showcase how LLMs are effectively used in white2023prompt, **classification**, and **validation**. By leveraging structured prompt engineering techniques, researchers and practitioners can optimize LLM responses to enhance SRE workflows.

5.2.3 Evaluation metrics in LLM4SE (Q2.3)

Evaluation metrics applied in LLM4SE span correctness, completeness, and qualitative assessments, emphasizing quantitative precision and expert-driven feedback. Correctness and performance metrics help validate model outputs against SE standards, while agreement and qualitative metrics provide insights into reliability and practical usability. This multidimensional evaluation approach ensures a comprehensive assessment of LLM performance, balancing technical accuracy with human-centric evaluations that address practical requirements in SE contexts. These findings underscore an optimization strategy that prioritizes efficiency, relevance, and accuracy, enabling LLMs to effectively meet RE's rigorous demands. However, achieving consistent and systematic improvements across various input cases requires the establishment of standard benchmarks and evaluation criteria. With many evaluation datasets from prior studies no longer accessible, creating a centralized repository of standard cases, methodologies, and evaluation procedures becomes essential–a critical step for advancing the field and ensuring the reliability of LLM-driven solutions in RE.

5.3 Challenges and future directions for LLMs in requirement engineering

Our investigation into the challenges and future directions for LLMs in RE reveals essential areas for improvement and promising opportunities for advancing the field. Below, we discuss key findings for each subquestion.

5.3.1 Challenges in utilizing LLMs in RE (Q3.1)

A major pattern identified is the difficulty LLMs face in handling domain-specific knowledge and context, as seen in issues like hallucination, incomplete outputs, and limitations in addressing complex syntaxes or structured requirements. Output quality and relevance remain significant challenges, especially in tasks requiring detailed technical understanding. Additionally, prompting limitations, empirical constraints, and formatting issues further highlight the need for specialized tuning and prompt engineering to improve LLM adaptability and response quality in RE tasks.

5.3.2 Future directions for LLMs in RE (Q3.2)

Future directions indicate a focus on refining evaluation metrics, enhancing model tuning with domain-specific data, and improving prompt engineering techniques. There is a push toward hybrid methods, integrating LLMs with traditional tools or human-in-the-loop systems, which hold promise for reducing hallucinations and boosting accuracy. Additionally, advancements in model fine-tuning and structured prompt patterns suggest that better alignment with RE-specific contexts can significantly enhance LLM reliability and performance.

Importantly, the findings indicate the need for continuous model adaptation and robust evaluation frameworks to maximize the utility of LLMs in RE, ensuring precise and contextually relevant outputs in software engineering applications. Therefore, a systematic approach is needed to enhance the precision and reliability of transforming natural language requirements into formal specifications. Validating outputs across multiple LLMs and ensuring consistency with the original requirements to enhance accuracy by identifying and mitigating errors or inconsistencies, leading to more reliable formalizations. Additionally, developing standardized evaluation metrics and benchmarks will allow for a systematic assessment of LLM effectiveness in software requirements formalization, facilitating comparison and driving advancements in future requirements engineering applications. These findings underscore the need for continuous model adaptation and robust evaluation frameworks to maximize the utility of LLMs in RE, ensuring precise and contextually relevant outputs in software engineering applications.

5.4 Threats to validity

5.4.1 Study search and selection bias

One key limitation is the potential for study search and selection bias. To mitigate this concern, we adopted a comprehensive approach following the guidelines established by Petersen et al. (2015). We aimed to select relevant studies to address these potential issues. We created a list of search strings derived from various terms in the field and tailored them for each digital repository and search engine. Additionally, we conducted both backward and forward snowballing on the selected articles to identify further relevant studies for our research. We defined inclusion and exclusion criteria to guide the initial selection of papers, which were then manually verified.

5.4.2 Empirical knowledge bias

This systematic review incorporates findings from 29 pertinent studies in the LLM4RE field and addresses three research questions. This necessitates a thorough manual analysis and comprehension of each study. During this process, biases may arise due to subjective assessments and personal experience. To mitigate potential errors, we conducted an extensive literature review to establish predefined categories and details for each research question.

6 Conclusion

In this paper, we surveyed the emerging field of LLM applications in RE. We identified key publications, explored current research areas, highlighted the challenges of using LLMs in RE, and proposed future research directions to advance the field. As LLMs continue to grow in power and user familiarity with them increases, they are poised to significantly impact RE performance, particularly in automating RE activities. One of the significant findings is the demonstrated effectiveness of LLMs, such as GPT-4 and CodeLlama, in generating and validating software requirements specifications with a high degree of accuracy and completeness. However, several challenges persist. The risk of hallucinations–plausible but incorrect outputs–remains a critical concern, as it can result in irrelevant or inappropriate elements in generated specifications. Additionally, the reliance on high-quality training data, difficulty in interpreting or validating complex outputs, and the computational expense of fine-tuning pose significant hurdles to the broader adoption of LLMs in RE. These challenges highlight the need for a systematic approach to improving the reliability, precision, and contextual relevance of LLM outputs. Future directions emphasize refining evaluation metrics, developing robust benchmarks, and enhancing model tuning with domain-specific data to address these limitations. Hybrid approaches, such as integrating LLMs with traditional tools or human-in-the-loop systems, show promise for reducing hallucinations and improving accuracy. Furthermore, structured prompt engineering and advancements in fine-tuning techniques are critical for aligning LLMs with the nuanced requirements of RE tasks. Establishing centralized repositories of standard cases, methodologies, and evaluation procedures will also be essential to enable systematic comparisons and foster consistent improvements across various use cases. Ultimately, the findings of this study underscore the need for continuous model adaptation, robust evaluation frameworks, and innovative hybrid solutions to maximize the utility of LLMs in RE. By addressing the outlined challenges and leveraging these advancements, LLMs can evolve into reliable and transformative tools for software requirements engineering.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

AH: Data curation, Formal Analysis, Investigation, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. MS: Conceptualization, Data curation, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing. SK-R: Conceptualization, Investigation, Methodology, Project administration, Supervision, Writing – review & editing. KL: Conceptualization, Project administration, Supervision, Writing – review & editing. ST: Conceptualization, Funding acquisition, Project administration, Supervision, Writing – review & editing.

Funding

The author(s) declare that no financial support was received for the research and/or publication of this article.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that Gen AI was used in the creation of this manuscript. It was only used to check grammar and improve the writing.

Publisher's note

References

Abukhalaf, S., Hamdaqa, M., and Khomh, F. (2024). "PathOCL: path-based prompt augmentation for OCL generation with GPT-4," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering (FORGE '24)* (New York, NY: Association for Computing Machinery), 108–118. doi: 10.1145/3650105.3652290

Alhanahnah, M., Hasan, M. R., and Bagheri, H. (2024). An empirical evaluation of pre-trained large language models for repairing declarative formal specifications. *arXiv [preprint]* arXiv:2404.11050. doi: 10.48550/arXiv.2404.11050

Al-Hossami, E., and Shaikh, S. (2022). A survey on artificial intelligence for source code: a dialogue systems perspective. *arXiv [Preprint].* arXiv:2202.04847. doi: 10.48550/arXiv.2202.04847

Arora, C., Grundy, J., and Abdelrazek, M. (2023). Advancing requirements engineering through generative ai: assessing the role of LLMS. *arXiv* [preprint] arXiv:2310.13976. doi: 10.1007/978-3-031-55642-5_6

Bhattacharya, P., Chakraborty, M., Palepu, K. N., Pandey, V., Dindorkar, I., Rajpurohit, R., et al. (2023). Exploring large language models for code explanation. *arXiv* [preprint] arXiv:2310.16673. doi: 10.48550/arXiv.2310.16673

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.* 80, 571–583. doi: 10.1016/j.jss.2006.07.009

Cheng, H., Husen, J. H., Peralta, S. R., Jiang, B., Yoshioka, N., Ubayashi, N., et al. (2024). Generative AI for requirements engineering: a systematic literature review. *arXiv* [preprint] arXiv:2409.06741. doi: 10.48550/arXiv.2409.06741

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., et al. (2023). "Large language models for software engineering: survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)* (Melbourne: IEEE), 31–53.

Frydenlund, E., Martínez, J., Padilla, J. J., Palacio, K., and Shuttleworth, D. (2024). Modeler in a box: how can large language models aid in the simulation modeling process? *Simulation*. 100, 727–749. doi: 10.1177/00375497241239360

Gomez, A. P., Krus, P., Panarotto, M., and Isaksson, O. (2024). Large language models in complex system design. *Proc. Design Soc.* 4, 2197–2206. doi: 10.1017/pds.2024.222

Hasan, M. R., Li, J., Ahmed, I., and Bagheri, H. (2023). Automated repair of declarative software specifications in the era of large language models. *arXiv* [preprint] arXiv:2310.12425. doi: 10.48550/arXiv.2310.12425

Kogler, P., Falkner, A., and Sperl, S. (2024). "Reliable generation of formal specifications using large language models," in *SE 2024-Companion* (Bonn: Gesellschaft für Informatik eV), 141–153.

Krishna, M., Gaur, B., Verma, A., and Jalote, P. (2024). Using llms in software requirements specifications: an empirical evaluation. *arXiv* [preprint] arXiv:2404.17842. doi: 10.1109/RE59067.2024.00056

Lee, J., Jung, W., and Baek, S. (2024). In-house knowledge management using a large language model: focusing on technical specification documents review. *Appl. Sc.* 14:2096. doi: 10.3390/app14052096

Li, M., Fang, W., Zhang, Q., and Xie, Z. (2024). Specllm: Exploring generation and review of vlsi design specification with large language model. *arXiv* [preprint] arXiv:2401.13266.

Lubos, S., Felfernig, A., Tran, T. N. T., Garber, D., El Mansi, M., Erdeniz, S. P., et al. (2024). "Leveraging llms for the quality assurance of software requirements," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)* (IEEE), 389–397.

Luitel, D., Hassani, S., and Sabetzadeh, M. (2024). Improving requirements completeness: Automated assistance through large language models. *Requirem Eng.* 29, 73–95. doi: 10.1007/s00766-024-00416-3

Ma, L., Liu, S., Li, Y., Xie, X., and Bu, L. (2024). Specgen: automated generation of formal program specifications via large language models. *arXiv* [preprint] arXiv:2401.08807.

Mandal, S., Chethan, A., Janfaza, V., Mahmud, S., Anderson, T. A., Turek, J., et al. (2023). Large language models based automatic synthesis of software specifications. *arXiv* [preprint] arXiv:2304.09181.

Necula, S.-C., Dumitriu, F., and Greavu-Şrban, V. (2024). A systematic literature review on using natural language processing in software requirements engineering. *Electronics* 13:2055. doi: 10.3390/electronics13112055

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: an update. *Inform. Softw. Technol.* 64, 1–18. doi: 10.1016/j.infsof.2015.03.007

Rahman, T., and Zhu, Y. (2024). Automated user story generation with test case specification using large language model. *arXiv* [preprint] arXiv:2404.01558. doi: 10.1109/RE59067.2024.00046

Ronanki, K., Cabrero-Daniel, B., Horkoff, J., and Berger, C. (2024). "Requirements engineering using generative AI: prompts and prompting patterns," in *Generative AI for Effective Software Development*, eds. A. Nguyen-Duc, P. Abrahamsson, and F. Khomh (Cham: Springer Nature Switzerland), 109–127.

Sarsa, S., Denny, P., Hellas, A., and Leinonen, J. (2022). "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research* (New York: ACM), 27–43.

Spoletini, P., and Ferrari, A. (2024). "The return of formal requirements engineering in the era of large language models," in *International Working Conference on Requirements Engineering: Foundation for Software Quality* (Cham: Springer), 344–353.

Vogelsang, A., and Fischbach, J. (2024). Using large language models for natural language processing tasks in requirements engineering: a systematic guideline. *arXiv* [preprint] arXiv:2402.13823. doi: 10.48550/arXiv.2402.13823

Wang, B., Wang, C., Liang, P., Li, B., and Zeng, C. (2024). How llms aid in uml modeling: an exploratory study with novice analysts. *arXiv* [preprint] arXiv:2404.17739. doi: 10.1109/SSE62657.2024.00046

White, J., Hays, S., Fu, Q., Spencer-Smith, J., and Schmidt, D. C. (2023). *ChatGPT prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design.*

White, J., Hays, S., Fu, Q., Spencer-Smith, J., and Schmidt, D. C. (2024). "ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," in *Generative AI for Effective Software Development* (Cham: Springer), 71–108.

Wu, Y., Li, Z., Zhang, J. M., Papadakis, M., Harman, M., and Liu, Y. (2023). Large language models in fault localisation. *arXiv* [preprint] arXiv:2308.15276. doi: 10.48550/arXiv.2308.15276

Xie, D., Yoo, B., Jiang, N., Kim, M., Tan, L., Zhang, X., et al. (2023). Impact of large language models on generating software specifications. *arXiv* [preprint] arXiv:2306.03324. doi: 10.48550/arXiv.2306.03324

Zhang, Q., Fang, C., Xie, Y., Zhang, Y., Yang, Y., Sun, W., et al. (2023a). A survey on large language models for software engineering. *arXiv* [preprint] arXiv:2312.15223. doi: 10.48550/arXiv.2312.15223

Zhang, Q., Zhang, T., Zhai, J., Fang, C., Yu, B., Sun, W., et al. (2023b). A critical review of large language model on software engineering: an example from chatGPT and automated program repair. *arXiv* [preprint] arXiv:2310.08879. doi: 10.48550/arXiv.2310.08879

Zhang, Z., Chen, C., Liu, B., Liao, C., Gong, Z., Yu, H., et al. (2023). Unifying the perspectives of nlp and software engineering: a survey on language models for code. *arXiv* [preprint] arXiv:2311.07989. doi: 10.48550/arXiv.2311.07989

SURVEYED PAPERS

[P1] Abukhalaf, S., Hamdaqa, M., and Khomh, F. (2024). Pathocl: Path-based prompt augmentation for ocl generation with gpt-4. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*. 108–118.

[P2] Alhanahnah, M., Hasan, M. R., and Bagheri, H. (2024). An empirical evaluation of pre-trained large language models for repairing declarative formal specifications. *arXiv [Preprint]* arXiv:2404.11050.

[P3] Arora, C., Grundy, J., and Abdelrazek, M. (2024). Advancing requirements engineering through generative ai: Assessing the role of llms. In *Generative AI for Effective Software Development* (Springer). 129–148.

[P4] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., et al. (2021). Program synthesis with large language models. *arXiv [Preprint]* arXiv:2108.07732.

[P5] Belzner, L., Gabor, T., and Wirsing, M. (2023). Large language model assisted software engineering: prospects, challenges, and a case study. In *International Conference on Bridging the Gap between AI and Reality* (Springer), 355–374.

[P6] Bhattacharya, P., Chakraborty, M., Palepu, K. N., Pandey, V., Dindorkar, I., Rajpurohit, R., et al. (2023). Exploring large language models for code explanation. *arXiv [Preprint]* arXiv:2310.16673.

[P7] De Vito, G., Palomba, F., Gravino, C., Di Martino, S., and Ferrucci, F. (2023). Echo: An approach to enhance use case quality exploiting large language models. In *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (IEEE), 53–60.

[P8] Frydenlund, E., Martínez, J., Padilla, J. J., Palacio, K., and Shuttleworth, D. (2024). Modeler in a box: how can large language models aid in the simulation modeling process? Simulation, 00375497241239360.

[P9] Gomez, A. P., Krus, P., Panarotto, M., and Isaksson, O. (2024). Large language models in complex system design. *Proceedings of the Design Society* 4, 2197–2206.

[P10] Hasan, M. R., Li, J., Ahmed, I., and Bagheri, H. (2023). Automated repair of declarative software specifications in the era of large language models. *arXiv [Preprint]* arXiv:2310.12425.

[P11] Herwanto, G. B. (2024). Automating data flow diagram generation from user stories using large language models. In *7th Workshop on Natural Language Processing for Requirements Engineering*.

[P12] Kogler, P., Falkner, A., and Sperl, S. (2024). Reliable generation of formal specifications using large language models. In SE 2024-Companion (Gesellschaft für Informatik eV), 141–153.

[P13] Krishna, M., Gaur, B., Verma, A., and Jalote, P. (2024). Using llms in software requirements specifications: An empirical evaluation. *arXiv [Preprint]* arXiv:2404. 17842.

[P14] Lee, J., Jung, W., and Baek, S. (2024). In-house knowledge management using a large language model: Focusing on technical specification documents review. *Applied Sciences* 14, 2096.

[P15] Li, J., Li, G., Li, Y., and Jin, Z. (2023). Enabling programming thinking in large language models toward code generation. *arXiv [Preprint]* arXiv:2305.06599.

[P16] Li, M., Fang, W., Zhang, Q., and Xie, Z. (2024). Specllm: Exploring generation and review of vlsi design specification with large language model. *arXiv [Preprint]* arXiv:2401.13266.

[P17] Lubos, S., Felfernig, A., Tran, T. N. T., Garber, D., El Mansi, M., Erdeniz, S. P., et al. (2024). Leveraging llms for the quality assurance of software requirements. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)* (IEEE), 389–397.

[P18] Luitel, D., Hassani, S., and Sabetzadeh, M. (2024). Improving requirements completeness: Automated assistance through large language models. *Requirements Engineering* 29, 73–95.

[P19] Ma, L., Liu, S., Li, Y., Xie, X., and Bu, L. (2024). Specgen: Automated generation of formal program specifications via large language models. *arXiv [Preprint]* arXiv:2401.08807.

[P20] Mandal, S., 859 Chethan, A., Janfaza, V., Mahmud, S., Anderson, T. A., Turek, J., et al. (2023). Large language models based automatic synthesis of software specifications. *arXiv [Preprint]* arXiv:2304.09181.

[P21] Rahman, T. and Zhu, Y. (2024). Automated user story generation with test case specification using large language model. *arXiv [Preprint]* arXiv:2404.01558.

[P22] Sarsa, S., Denny, P., Hellas, A., and Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[P23] Wang, B., Wang, C., Liang, P., Li, B., and Zeng, C. (2024). How llms aid in uml modeling: An exploratory study with novice analysts. *arXiv [Preprint]* arXiv:2404.17739.

[P24] Wei, B. (2024). Requirements are all you need: From requirements to code with llms. *arXiv [Preprint]* arXiv:2406.10101.

[P25] White, J., Hays, S., Fu, Q., Spencer-Smith, J., and Schmidt, D. C. (2024). Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative AI for Effective Software Development* (Springer). 71–108.

[P26] Wu, Y., Li, Z., Zhang, J. M., Papadakis, M., Harman, M., and Liu, Y. (2023). Large language models in fault localisation. *arXiv [Preprint]* arXiv:2308.15276.

[P27] Xie, D., Yoo, B., Jiang, N., Kim, M., Tan, L., Zhang, X., et al. (2023a). Impact of large language models on generating software specifications. *arXiv [Preprint]* arXiv:2306.03324.

[P28] Xie, D., Yoo, B., Jiang, N., Kim, M., Tan, L., Zhang, X., et al. (2023b). Impact of large language models on generating software specifications. *arXiv [Preprint]* arXiv:2306.03324.

[P29] Zhang, Q., Zhang, T., Zhai, J., Fang, C., Yu, B., Sun, W., et al. (2023). A critical review of large language model on software engineering: An example from chatgpt and automated program repair. *arXiv [Preprint]* arXiv:2310.08879.