Large Language Models for Requirements Engineering: A Cross-Task Empirical Evaluation

Jacek Dąbrowski[1*], Manjeshwar Aniruddh Mallya[1], Alessio Ferrari[2,3], Mohammad Amin Zadenoori[4], Yijun Yu[5]

[1]Lero, the Research Ireland Centre for Software, University of Limerick, Ireland.
[2]Trinity College Dublin (TCD), Ireland.
[3]Consiglio Nazionale delle Ricerche (CNR), Italy.
[4]University of Padova, Italy.
[5]The Open University, UK.

*Corresponding author(s). E-mail(s): jacek.dabrowski@lero.ie;
Contributing authors: mallyaaniruddh@gmail.com;
alessio.ferrari@cnr.it; amin.zadenoori@unipd.it; y.yu@open.ac.uk;

Abstract

Requirements-related information is distributed across heterogeneous artefacts, including online user feedback, developer discussions, and software repositories. Extracting actionable requirements knowledge from these artefacts is labour-intensive and difficult to scale due to their volume, diversity, and unstructured nature. Recent advances in Large Language Models (LLMs) support a broad range of Requirements Engineering (RE) activities, ranging from requirements classification and traceability identification to requirements specification and traceability explanation generation. However, existing empirical evidence on the use of LLMs in RE remains fragmented across tasks, artefact types, and evaluation settings. Consequently, broader conclusions are difficult to draw about their capabilities and limitations. In particular, studies rarely provide cross-task empirical evaluations or replication packages, limiting reproducibility and the accumulation of empirical knowledge. To address this gap, this paper presents two complementary empirical studies evaluating LLMs across five RE-related activities. The first is a controlled experiment evaluating five lightweight open-source LLMs for feedback-driven requirements classification and specification generation. The second is an exploratory industrial case study evaluating two frontier LLMs for traceability link identification and traceability explanation

generation using software project artefacts. Classification and traceability identification were evaluated using quantitative metrics, while generation tasks were assessed through human evaluation. The results show that LLM performance is strongly task-dependent, ranging from moderate to high across the evaluated RE activities. No single model consistently outperformed others across all tasks, highlighting that effective LLM adoption in RE depends on selecting models and prompting strategies according to the target RE task. Our contributions are: (i) the first cross-task empirical evaluation of LLMs spanning five RE-related activities, (ii) replication materials supporting transparency and reproducibility, and (iii) a broader empirical understanding of the capabilities, limitations, and practical readiness of current LLMs for RE.

Keywords: Requirements Engineering, Large Language Models, User Feedback Analysis, Requirements Traceability, Requirements Specification, Mining Software Repositories.

1 Introduction

Modern Requirements Engineering (RE) depends on extracting actionable knowledge from heterogeneous software project artefacts, including online user feedback, issue tickets, developer discussions, and repository documentation [1–3]. These artefacts provide complementary evidence about stakeholder needs, implementation decisions, and software evolution [1, 4, 5]. This information supports core RE activities, from requirements elicitation [5] and classification [6] to specification and traceability [7, 8].

Extracting actionable requirements knowledge from these artefacts is however increasingly challenging due to their volume, diversity, and unstructured nature [5, 9, 10]. As software projects evolve, practitioners must continuously analyse information distributed across multiple artefacts to identify, classify, document, and trace requirements [1, 11, 12]. Consequently, automated support for analysing requirements-related artefacts has become an important research topic in RE and Software Engineering (SE) [10, 11].

Over the past decade, numerous Natural Language Processing (NLP) and Machine Learning (ML) techniques have been proposed to support analytical RE-related activities, including user feedback analysis, requirements classification, information extraction, and traceability recovery [8, 9, 13]. Although these approaches have demonstrated promising results, they are typically designed for individual tasks, require substantial labelled data, and often generalise poorly across projects and artefact types [3, 5]. Consequently, developing and maintaining task-specific solutions remains resource-intensive and difficult to generalise across projects and artefact types [3].

Recent advances in Large Language Models (LLMs) offer new opportunities to overcome these limitations [1, 11, 14]. Beyond improving analytical RE activities such as requiremetns classification [15] and traceability analysis [16], LLMs enable generative capabilities, including requirements specification [17] and explanation generation [1, 18]. Owing to their strong reasoning and text generation capabilities [19, 20], LLMs have rapidly gained attention across RE/SE [19]. Unlike

conventional NLP and ML approaches, LLMs can be adapted to diverse RE activities using natural-language prompts instead of task-specific models [1, 21].

Despite this growing interest, empirical evidence on the use of LLMs in RE remains fragmented across tasks, artefact types, and evaluation settings [1, 19, 22]. Consequently, it is difficult to establish a broader understanding of their capabilities and limitations or determine whether findings obtained in one RE context generalise to others [7, 8]. Moreover, few studies provide cross-task empirical evaluations or publicly available replication materials, limiting reproducibility and the accumulation of empirical knowledge [1, 23, 24].

This gap raises an important question: **how do current LLMs perform across diverse RE-related activities, and how do their capabilities and limitations vary across different RE contexts?** Addressing this question requires empirical evidence spanning multiple RE activities, artefact types, and evaluation settings.

To address this gap, we present two complementary empirical studies covering five RE-related activities.

Mallya et al. [17] previously reported the first study, which evaluated five lightweight open-source LLMs on three feedback-driven activities: non-functional requirements classification, user request classification, and requirements specification generation. This manuscript substantially extends that work by introducing a second empirical study. The second study is an exploratory industrial case study that evaluates two frontier LLMs for traceability link identification and traceability explanation generation using heterogeneous software project artefacts.

The two studies intentionally evaluate different categories of LLMs. The first study investigates whether lightweight open-source LLMs can support RE tasks with predefined outputs, including classification and requirements specification. The second study evaluates frontier LLMs on traceability analysis, which requires identifying semantic relationships across multiple software artefacts and generating evidence-grounded explanations. These studies move beyond task-specific evaluations by integrating evidence from complementary RE scenarios. This provides a broader empirical perspective on the capabilities and limitations of LLMs across requirements classification, specification, traceability, and explanation generation.

Rather than proposing a new LLM architecture or prompting strategy, this work aims to build cumulative empirical evidence on the capabilities and limitations of current LLMs for RE. By combining evidence from these complementary RE-related tasks, we identify strengths, limitations, and practical challenges that influence the effective adoption of LLMs in RE. The main contributions of this paper are as follows:

- the first cross-task empirical evaluation of LLMs spanning five RE-related activities;
- replication materials supporting transparency and reproducibility [25];
- a broader empirical understanding of the capabilities and limitations of current LLMs for RE.

The remainder of this paper is organised as follows. Section 2 introduces the background, and Section 3 reviews the related work. Section 4 presents the motivating scenarios for our studies. Sections 5 and 6 describe the two empirical studies

and present their results. Section 7 provides a cross-study discussion, and Section 8 concludes the paper.

2 Background

We now introduce the Large Language Models and prompt engineering strategies used in our study.

2.1 Large Language Models

Large Language Models (LLMs) are neural networks trained on large text corpora [11, 26]. They have demonstrated strong capabilities in reasoning, classification [27], summarization [17], and text generation [2]. These capabilities make them increasingly attractive for RE [1]. Unlike traditional Natural Language Processing (NLP) and Machine Learning (ML) approaches, LLMs can be adapted to diverse RE tasks using natural-language prompts rather than task-specific models [3].

LLMs can be broadly divided into two categories: *lightweight LLMs* and *frontier LLMs* [20, 28]. *Lightweight LLMs* require fewer computational resources and can be executed locally [1]. This makes them suitable for privacy-sensitive environments, controlled experimentation, and resource-constrained RE/SE projects [15]. However, their lower computational requirements come at the cost of reduced reasoning capabilities and smaller context windows compared with *frontier LLMs* [28]. In contrast, *frontier LLMs* provide substantially stronger reasoning capabilities and larger context windows, enabling the analysis of long and heterogeneous software artefacts [20]. Their reliance on provider-hosted infrastructure, however, may introduce higher computational costs and data privacy considerations [19]. Consequently, the choice of model depends on the characteristics of the RE/SE task [1]. *Lightweight LLMs* are well suited to analysing relatively short artefacts such as user feedback, whereas *frontier LLMs* are more appropriate for reasoning across multiple software artefacts, including project documentation, issue-tracking systems, and developer discussions.

To investigate LLM capabilities across different RE activities, this paper evaluates both lightweight and frontier models. **Study I** investigates five lightweight open-source LLMs (Llama 2[1], Llama 3[2], Mistral[3], Gemma 2[4], and Phi-3 Mini[5]) for feedback-driven RE tasks. **Study II** evaluates two frontier LLMs (GPT-5.5[6], and DeepSeek-R1[7]) for traceability analysis across software project artefacts.

The lightweight models were selected as a representative set of locally deployable LLMs from major AI developers. They differ in size and reasoning capability while remaining suitable for controlled local experimentation. Local execution provided full control over the experimental environment and ensured reproducibility. Rather than identifying a single best-performing model (e.g., [15]), **Study I** examines how representative lightweight LLMs perform across different RE tasks.

[1] https://huggingface.co/meta-llama/Llama-2-7b-hf
[2] https://huggingface.co/meta-llama/Meta-Llama-3-8B
[3] https://huggingface.co/mistralai/Mistral-7B-v0.3
[4] https://huggingface.co/google/gemma-2-9b
[5] https://huggingface.co/microsoft/Phi-3-mini-4k-instruct
[6] https://chatgpt.com/
[7] https://chat.deepseek.com/

Table 1 Large language models evaluated in this paper.

Model	Developer	Parameters	Context Window	Category	Study
Llama 2	Meta AI	7B	4K	Lightweight	I
Llama 3	Meta AI	8B	8K	Lightweight	I
Mistral	Mistral AI	7B	8K	Lightweight	I
Gemma 2	Google DeepMind	9B	8K	Lightweight	I
Phi-3 Mini	Microsoft	3.8B	4K	Lightweight	I
GPT-5.5	OpenAI	Not disclosed	128K	Frontier	II
DeepSeek-R1	DeepSeek AI	671B	128K	Frontier	II

In contrast, **Study II** evaluates frontier LLMs motivated by an industrial collaboration with Huawei Research Centre. Preliminary experimentation showed that lightweight models were impractical for traceability analysis over long and heterogeneous software artefacts because of their limited context windows and reasoning capabilities. Consequently, we evaluate GPT-5.5 and DeepSeek-R1, two representative frontier reasoning models with substantially larger context windows, accessed through their provider-hosted interfaces. These models represent complementary commercial and open-weight approaches to frontier LLM deployment in contemporary SE practice.

Table 1 summarises the LLMs evaluated in this paper. *Parameters* denote the approximate number of trainable model weights (where publicly available), while the *Context Window* indicates the maximum number of tokens the model can process in a single interaction. *Category* distinguishes lightweight and frontier models, and *Study* identifies the corresponding empirical study.

2.2 Prompting Strategies

Interaction with LLMs is achieved through *prompts*, i.e., natural-language instructions that define the task and expected output [1, 27]. The formulation of a prompt, commonly referred to as a *prompting strategy*, substantially influences model behaviour and output quality [20, 29]. Consequently, prompt engineering has become an important aspect of applying LLMs to RE/SE tasks [30]. Appropriate prompting can improve reasoning quality, increase output consistency, and reduce ambiguity in model responses [14]. These improvements are particularly important for RE tasks, where accurate interpretation and structured outputs are often required [1].

This paper considers five representative prompting strategies, summarised in Table 2. The strategies differ in how they guide model behaviour [1, 20, 27]. *Zero-shot* prompting relies solely on task descriptions. *Few-shot* prompting supplements the instruction with a small number of labelled examples. *Chain-of-thought* prompting encourages explicit reasoning before producing the final answer. *Constraint-based* prompting introduces explicit rules or templates to improve output consistency and structure. *Role-based* prompting assigns the model a specific professional perspective to guide its responses.

We selected these prompting strategies because they are widely adopted in the LLM for RE/SE literature and cover the principal mechanisms for guiding model behaviour [1, 14, 19]. **Study I** evaluates all five strategies to investigate their influence on classification and requirements generation tasks. In contrast, **Study II** focuses on zero-shot and chain-of-thought prompting. Rather than optimising prompt design [29],

the study evaluates the reasoning capabilities of frontier LLMs over heterogeneous software project artefacts in a realistic industrial setting.

Table 2 Overview of prompt strategies used in our study.

Prompt Strategy	Description and Example
Zero-shot	**Description:** A brief instruction describes the task, assuming the model can generalise from its pre-trained knowledge to perform it.
	Example: "Classify the following user feedback as a feature request, bug report, or usability issue."
Few-shot	**Description:** A few labelled examples show the desired pattern, assuming the model will apply it to new inputs.
	Example: "Example 1: 'Add dark mode' → Feature Request. Example 2: 'App crashes on login' → Bug Report. Now classify the next five items."
Chain-of-Thought	**Description:** The model is instructed to reason step by step and write intermediate steps before giving the final answer.
	Example: "Explain why this discussion fragment supports the given project goal, then identify the traceability link."
Constraint-based	**Description:** Prompts include explicit rules, templates, or conditions that the output must follow.
	Example: "Generate requirements that are testable, unambiguous, and measurable."
Role-based	**Description:** The model is assigned a specific role and responds using the perspective, tone, and knowledge expected from it.
	Example: "You are a requirements analyst. Rewrite the following user request as a formal functional requirement."

3 Related Works

This section reviews related work on modern Requirements Engineering and LLMs for RE, highlighting the research gaps addressed in this study.

3.1 Modern Requirements Engineering

Requirements Engineering (RE) has evolved substantially over the past two decades [9, 10, 12]. Traditional RE relied primarily on stakeholder interviews, workshops, and requirements specifications [31]. Modern software projects, however, continuously generate large volumes of requirements-related information throughout the software lifecycle. This shift has led to the emergence of Data-Driven Requirements Engineering (DDRE) [5, 13]. DDRE complements stakeholder knowledge with evidence extracted from heterogeneous software artefacts, such as user feedback [8, 32], issue reports [33], software repositories [34], and developer discussions [35].

The growing availability of software artefacts has stimulated research on Artificial Intelligence for Requirements Engineering (AI4RE) [36]. AI4RE applies Machine Learning (ML), information retrieval, knowledge-based techniques, and Natural Language Processing (NLP) to automate RE activities [5, 8]. These include, for example, requirements classification (e.g., functional versus non-functional requirements) [37], information extraction (e.g., feature descriptions from online user feedback) [38], traceability recovery (e.g., linking requirements to source code) [39], and defect detection (e.g., identifying ambiguous requirements) [40].

As most software artefacts are represented in natural language, Natural Language Processing for Requirements Engineering (NLP4RE) has become a major research direction [13]. NLP4RE extends traditional AI4RE by focusing on methods for understanding and processing textual software artefacts [3]. However, systematic literature reviews report that most AI- and NLP-based approaches address individual RE tasks, focus on a single artefact type (e.g., app reviews, issue reports, or requirements specifications) [41, 42], and require task-specific models or labelled datasets for supervised learning [1, 8, 43].

Recent advances in LLMs have significantly expanded these capabilities [43]. Unlike earlier AI and NLP approaches, LLMs support both analytical (e.g., requirements classification [15] and traceability recovery [16]) and generative RE activities (e.g., requirements specification [44] or model generation [45]) through natural-language prompting [1]. They can analyse diverse software artefacts and support multiple RE tasks using the same general-purpose model [21].

Overall, the literature demonstrates a clear evolution of modern RE. Research has progressed from stakeholder-centric practices towards data-driven analysis of heterogeneous software artefacts [5, 13, 46]. At the same time, AI and NLP approaches have evolved from task-specific techniques to general-purpose language models [3]. This evolution has laid the foundation for contemporary research on LLMs for RE [1, 21].

3.2 Large Language Models for Requirements Engineering

Research on LLMs for RE has grown rapidly in recent years [1, 47]. Existing studies investigate LLMs across a broad range of RE activities [30, 43]; and demonstrate their potential to support different stages of the requirements lifecycle. Recent systematic literature reviews also indicate a clear shift from traditional NLP pipelines towards prompt-driven and generative approaches (e.g., [1, 30, 43, 47]). In particular, recent studies increasingly focus on requirements elicitation and validation [48], while also exploring the integration of RE with downstream SE activities such as testing [49], modelling [45], and design support [50].

Current research covers both *analytical* and *generative* RE tasks [1, 18]. Papers investigating *analytical* tasks typically focus on requirements classification (e.g., functional versus non-functional requirements) [17] and traceability recovery (e.g., linking related artefacts) [16]. In contrast, papers investigating *generative* tasks commonly address requirements specification (e.g., generating user stories) [44], refinement (e.g., improving requirement quality [51], and explanation generation (e.g., explaining requirements) [51].

Table 3 Comparison between existing empirical studies on LLMs for Requirements Engineering and this work.

Dimension	**Existing Studies**	**This Work**
Evaluation scope	Single RE task	Multiple RE tasks
RE task type	Analytical *or* generative	Analytical *and* generative
Evaluation setting	Experiment *or* case study	Experiment *and* case study
Models	Single LLM family	Lightweight and frontier LLMs
Reproducibility	Limited replication support	Replication package

Although the range of RE tasks has broadened considerably, most empirical studies still evaluate LLMs on a single task or application scenario [1, 43]. Recent work also increasingly explores more knowledge-intensive activities, including requirements elicitation [18], validation [48], and specification generation [47]. Consequently, it remains unclear to what extent findings obtained for one RE task generalise to others [1, 30]. This limits a broader empirical understanding of LLM capabilities across RE tasks.

The range of evaluated artefacts has also expanded considerably. Early studies focused primarily on Software Requirements Specifications (SRS) [1, 44]. More recent work investigates a broader range of software artefacts, including user stories [17], issue reports [52], online user feedback [53], regulatory documents, technical documentation [30], and software repositories [1]. This trend reflects the increasing importance of heterogeneous software artefacts in modern RE.

Empirical studies investigate a growing range of LLMs and prompting strategies [16, 52]. However, most evaluations still rely on frontier models (e.g., GPT-based) and zero-shot or few-shot prompting [1]. Lightweight models (e.g., Mistral) and more advanced prompting strategies, including chain-of-thought reasoning [15], retrieval-augmented generation (RAG) [50], and agent-based workflows [4], have received comparatively less attention. Studies are also typically conducted either as controlled experiments [15] or as industrial case studies [44], with few combining both evaluation settings.

3.3 Research Gap

Recent AI4RE studies demonstrate the growing potential of LLMs for RE applications [1, 30, 43, 47]. However, the available empirical evidence remains fragmented [30, 47]. Table 3 summarises the principal differences between previous empirical studies and our work.

First, most studies investigate a single RE task (e.g., requirements classification [37] or traceability recovery [16]), limiting the generalisability of existing findings across different RE tasks.

Second, previous studies typically evaluate either analytical or generative RE tasks, and rely on either controlled experiments or industrial case studies [1, 43]. As a result, technical performance and practical applicability are rarely assessed together. In addition, most evaluations consider models from a single LLM family [1], providing limited insight into the trade-offs between lightweight and frontier LLMs [15].

Table 4 Motivating scenarios and supported RE activities.

Motivating Scenario	Supported RE Activities	Study
From User Feedback to Requirements	Requirements elicitation Requirements classification Requirements specification generation	I
Requirements Across Project Artefacts	Requirements traceability Design rationale understanding Requirements impact analysis	II

Finally, reproducibility remains an important challenge [1, 14]. While some studies release datasets, prompts, source code, or other research artefacts, replication support is often incomplete or unavailable [23]. This limits independent validation, cross-study comparisons, and the accumulation of reliable empirical evidence.

To address these gaps, our study provides a comprehensive cross-task evaluation of LLMs for RE. It evaluates multiple analytical and generative RE tasks using both lightweight and frontier LLMs, combines a controlled experiment with an industry-motivated case study, and releases a complete replication package to support reproducibility and future research.

4 Motivating Scenarios

We now present two complementary RE scenarios representing different stages of the requirements lifecycle. The first scenario is motivated by prior empirical RE research on the use of online user feedback [8]. Specifically, it builds on empirical studies, surveys, and practitioner interviews examining how software teams leverage user feedback to support RE and SE practice [54]. The second scenario is motivated by an industry-informed case study conducted in collaboration with Huawei Research Centre, focusing on the Rust open-source ecosystem[8]. It investigates requirements-related information distributed across heterogeneous software project artefacts to support requirements traceability, design rationale understanding, and impact analysis [3]. Table 4 provides an overview of the motivating scenarios, the supported RE activities, and the corresponding empirical studies.

4.1 Scenario 1: From User Feedback to Requirements

Modern mobile applications receive large volumes of online user reviews through app store platforms [8]. These reviews contain valuable requirements-related information, including feature requests, bug reports, and software quality concerns (e.g., usability and performance) [10]. Analysing user reviews can supports several RE activities [7]. However, their volume and noisy nature make manual analysis challenging [8]. Consequently, automated approaches for analysing user feedback and generating requirements specifications are increasingly important for supporting RE practice [13].

[8] https://rust-lang.org

Consider a development team maintaining a mobile application such as WhatsApp. Following a new release, the team receives thousands of user reviews. Some request new functionality (e.g., "add message scheduling"), others report defects (e.g., "messages fail to send"), while many describe software quality concerns such as poor performance ("the application is too slow to open") or usability issues ("the chat layout is confusing"). Although valuable, only a subset of reviews contains actionable requirements-related information.

To identify stakeholder needs, the team could automatically classify user reviews by request type (e.g., feature request, bug report, or other) and referenced non-functional requirements (NFRs). The resulting classifications would support *requirements elicitation* by extracting actionable stakeholder needs from large volumes of user feedback. They would support *requirements classification* by organising the elicited requirements according to request type and quality attributes. By quantifying the classified feedback, developers could identify the most frequently requested features, reported defects, and software quality concerns. This would support *requirements prioritisation* and guide software evolution [7, 8].

Having identified and classified the relevant feedback, the team could support the *requirements specification* activity. An automated approach could generate requirement statements, user stories, or an initial Software Requirements Specification (SRS). For example, the review "add dark mode" could be reformulated as the requirement statement "The system shall provide a dark mode option." Alternatively, it could be expressed as the user story "As a user, I want to enable dark mode so that I can comfortably use the application in low-light environments." Although these drafts would not replace a complete specification, they could reduce documentation effort, improve consistency, and maintain traceability between user feedback and documented requirements [54].

This scenario motivates **Study I**, which investigates lightweight open-source LLMs for supporting these RE activities.

4.2 Scenario 2: Requirements Across Project Artefacts

As software projects evolve, requirements-related information becomes distributed across software repository artefacts, including project roadmaps, GitHub issues, and developer discussions [10, 12]. Analysing the relationships between these artefacts supports RE activities such as requirements traceability, design rationale understanding, and requirements impact analysis [3]. However, their distributed and heterogeneous nature makes manual analysis challenging [5]. Consequently, automated traceability identification and rationale explanation are increasingly important for RE practice [16].

Consider the real-world open-source Rust programming language project, where contributors from both the community and industry (e.g., Huawei) collaborate on the language's development[9]. Requirements-related information is distributed across roadmap goals[10], GitHub issues[11], and Zulip developer discussions[12]. Understanding

[9] https://github.com/rust-lang/
[10] https://rust-lang.github.io/rust-project-goals/
[11] https://github.com/rust-lang/rust/issues/
[12] https://rust-lang.org/community/

the relationships between these artefacts would help contributors answer questions such as: *Which roadmap goal does this GitHub issue address? What functionality is being proposed? Why was this design decision made? How should the proposed feature be implemented?* Answering these questions supports requirements traceability, design rationale understanding, and anticipating the impact of proposed language changes [3]. However, manually reconstructing these relationships across numerous artefacts is time-consuming and error-prone.

The team could therfore use an automated approach to identify traceability links and generate evidence-grounded explanations across project artefacts. The identified links would support *requirements traceability* by linking roadmap goals with related GitHub issues and Zulip discussions. Contributors could then quickly determine which discussions concern a particular project goal, which issue operationalises it, and how the proposed feature evolves throughout development. The generated explanations would support *design rationale understanding* by explaining why these artefacts are linked and identifying the evidence supporting each relationship.

When planning language changes, contributors need to anticipate their implications before implementation. Traceability links and their evidence-grounded explanations would help identify related roadmap goals, implementation activities, and technical dependencies that may be affected by a proposed feature or modification. The same information would also help industrial adopters assess how language evolution could influence downstream software systems, such as operating systems, embedded software, and Rust-based applications. This would support *requirements impact analysis* and enable better-informed engineering decisions.

This scenario motivates **Study II**, which investigates state-of-the-art LLMs for supporting these RE activities across heterogeneous software project artefacts.

These two scenarios cover complementary stages of the RE lifecycle. Scenario I focuses on eliciting and documenting requirements from stakeholder feedback, whereas Scenario II focuses on tracing and explaining requirements across software project artefacts.

5 Study I: From User Feedback to Requirements

The first study investigates whether lightweight LLMs can support feedback-driven RE, based on the first motivating scenario (see Sect. 4). It evaluates five lightweight open-source LLMs across three complementary tasks: user request classification, NFR classification, and requirements specification generation. We first introduce the study context, including the key concepts, terminology, and RE problems considered in this study. We then present the empirical study design, experimental findings, and discuss their implications for RE practice.

5.1 Study Context

Our first study focuses on *app review analysis* [8]. Specifically, we consider *app reviews*, a form of online user feedback collected from mobile application platforms (e.g., Google Play Store). App reviews are among the most widely used sources of user feedback in RE. Their large volume and unstructured nature make manual analysis challenging,

motivating automated approaches for user feedback analysis [53, 54]. Each app review is treated as a *user feedback instance*.

User feedback conveys diverse types of information relevant to RE. We focus on two information types: *user requests* and *non-functional requirements (NFRs)*. User requests express users' needs to modify the software and are categorized as either *feature requests*, describing desired functionality, or *bug reports*, describing defects or incorrect system behavior [8]. NFRs are categorized according to the software quality attributes defined by the ISO/IEC 25010 quality model, including usability, reliability, performance, and security [55].

We investigate two complementary RE tasks. The first, *user feedback classification*, classifies each app review using two schemes: the *user request type* (e.g., *feature request* or *bug report*) and the *NFR type* (e.g., *performance*, *usability*, or *security*), each with an additional *other* category. The second, *requirements specification generation*, transforms one or more classified user feedback instances into structured requirements specifications, such as requirement statements or user stories, that capture the underlying stakeholder needs.

For example, the review *'The app crashes when saving a form"* is classified as a *bug report*, whereas *'I want the app to automatically save my progress"* is classified as a *feature request*. Similarly, *'The app responds very slowly when opening large files"* is classified into the *performance* NFR category. The feature request can then be transformed into the requirement statement *'The system shall automatically save user progress."* or expressed as the user story *"As a user, I want the app to automatically save my progress so that I do not lose data."*

5.2 Empirical Study Design

This section presents the empirical study conducted to evaluate the effectiveness of LLMs in analysing online user feedback to support RE.

5.2.1 Research Questions

The goal of this study is to evaluate LLMs in analysing online user feedback to support RE tasks. We specifically focus on three research questions:

- **RQ1**: How well do LLMs classify feedback by NFR type?
- **RQ2**: How well do LLMs classify feedback by user-request type?
- **RQ3**: How well do LLMs generate requirements specification?

In RQ1, we assess the models' ability to identify the NFR type mentioned in user reviews. RQ2 examines how accurately the models classify user feedback by request type. RQ3 evaluates their capability to generate requirements specification from the same feedback. All evaluations use human-annotated datasets (see Sect. 5.2.2). For RQ1–RQ2, model predictions are compared with annotations using precision, recall, and F1-score. For RQ3, SRSs generated from sampled reviews are assessed through human judgment based on predefined quality criteria.

5.2.2 Datasets

We use two annotated datasets of mobile app user feedback from prior studies [56, 57]. The first dataset was originally labeled with NFR types [57] and is used to answer RQ1. The second dataset contains app reviews annotated with user request types [56] and is used to answer RQ2. In addition, a sample of annotated reviews from both datasets is used to answer RQ3. We selected these datasets for their relevance and availability in a public repository [8]. Each dataset contains thousands of reviews from approximately a dozen apps across diverse domains and both major app stores. This diversity mitigates the app sampling problem and supports validity [8].

Non-Functional Requirements (NFR) Dataset. The first dataset, introduced by Lu and Liang [57], focuses on app reviews annotated with NFR types based on the software quality model [55]. The initial collected dataset comprised approximately 11,000 app reviews from two mobile applications in the books and communications categories. These reviews were drawn from both major app stores, the Apple App Store and Google Play Store. A subset of 4,000 review sentences was manually annotated according to five NFR categories.

User Request Dataset. The second dataset builds on previous studies [58, 59] and includes additional user feedback collected for this study [56]. The initial collected dataset covered about 10 mobile applications across more than 10 categories from both the Apple App Store and Google Play Store. From an initial pool of the collected reviews, a curated subset of 2,912 was manually annotated into three user request types: feature requests, bug reports, and other. This dataset provides a diverse and representative sample of user feedback for evaluating automated classification approaches.

5.2.3 Evaluation Metrics and Criteria

We applied both quantitative and qualitative methods. Standard ML metrics [4] were used for RQ1 and RQ2, as feedback classification is a classification task [60]. Specification generation (RQ3) was evaluated via criteria-based assessment [61].

Evaluation Metrics for Classification (RQ1–RQ2).

We compute precision, recall, and F1-score for two experiments: classifying user feedback by NFR type (RQ1) and by user request type (RQ2). Precision measures the proportion of correctly predicted labels among all predictions, while recall measures the proportion of correctly identified labels in the ground truth. Model performance is assessed by comparing each predicted label (user request or NFR type) with its annotated counterpart. Scores are calculated per class (e.g., feature request, bug report, other), along with macro averages. The macro average treats all classes equally.

Evaluation Criteria for Specification Generation (RQ3).

For RQ3, the quality of generated requirement specification is evaluated qualitatively across five criteria derived from established specification attributes such as completeness, consistency, and clarity [62]. Each criterion was adapted to suit the

Table 5 Evaluation criteria and scoring used to assess generated specification (RQ3).

Criterion	Description and Scoring
Structural Adherence	**Description:** Evaluates how well the generated specification follows the expected structure, ensuring coverage of all sections.
	Scoring: Originally 1–8 points (one per correctly included section), linearly rescaled to 1–5 for consistency; higher is better.
Completeness	**Description:** Evaluates the coverage of requirements identified from user feedback.
	Scoring: Rated on a 1–5 scale, with higher scores reflecting greater completeness of identified requirements from user feedback.
Fidelity	**Description:** Evaluates how faithfully the generated specification reflects user feedback, identifying fabricated requirements.
	Scoring: Rated on a 1–5 scale; higher scores indicate greater fidelity (rescaled from the proportion of fabricated to valid requirements).
Conciseness	**Description:** Evaluates redundancy and verbosity in the generated text, indicating how efficiently information is conveyed.
	Scoring: Rated on a 1–5 scale, with higher scores reflecting greater conciseness and lower values indicate increased verbosity.
Clarity	**Description:** Evaluates the clarity of generated requirements in terms of unambiguity, specificity, and interpretability.
	Scoring: Rated on a 1–5 scale, with higher scores reflecting greater clarity and linguistic precision.

characteristics of automatically generated specification. Table 5 outlines the evaluation rubric, which includes five criteria. These dimensions assess structural correctness, coverage of stakeholder input, factual grounding, semantic accuracy, and linguistic quality. Each criterion follows a defined scoring scheme combining quantitative (e.g., counts or coverage) and qualitative (e.g., 1–5 scale) measures, supporting systematic and replicable assessment of specification quality [61].

5.2.4 Experimental Setup and Procedure

We now describe the computational setup, prompting strategies, and evaluation procedures used in three experiments (RQ1–RQ3).

Computational Setup

All experiments were conducted under consistent hardware and parameter settings to ensure fairness and reproducibility. We evaluated five LLMs (see Sect. 2), running each model three times per task to reduce stochastic variance. Default parameters were used, with *temperature* set to 0 and a fixed *random seed* to minimise non-determinism. No hyperparameter tuning was applied to isolate the effects of prompting strategies. Experiments ran on a workstation with an NVIDIA RTX 4050 GPU (6 GB VRAM) and 16 GB RAM. Total runtime per model (3 runs) ranged from 20 min to 2 h for

user request classification (512 reviews) and 2–5 hours for NFR classification (1,278 reviews).

Prompting Strategies

We experiment with five prompting strategies (see Sect. 2). Prompting strategies are customised for each experiment (RQ1–RQ2).

Prompting Strategies for Classification (RQ1–RQ2). We apply three prompting strategies across two classification tasks (user request type and NFR type): *zero-shot*, *few-shot*, and *chain-of-thought (CoT)* prompting. These strategies capture increasing levels of reasoning and contextualization while remaining lightweight and reproducible. We omit more complex prompting (e.g., role-based or constraint-based), as classification primarily requires consistent label prediction rather than creative or constrained generation. Prompts are refined iteratively through a pilot study. Few-shot examples come from our dataset, and CoT prompts direct models to "*think step-by-step before giving the final category*".

Prompting Strategies for Specification Generation (RQ3). We use all five prompting strategies for the specification generation task. Beyond the three classification strategies (*zero-shot*, *few-shot*, *CoT*), we add *constraint-based* and *role-based* prompting to improve structural coherence and contextual relevance. These strategies better suit generative tasks that require creativity and controlled output. In the constraint-based setup, prompts specify that outputs follow a defined structure (e.g., functional and non-functional sections) and include constraints such as "*avoid implementation details*" and "*ensure each requirement is unique*".

Evaluation Procedures

We use quantitative evaluation for classification tasks (RQ1–RQ2) and qualitative evaluation for specification generation (RQ3).

Evaluation Procedure for Classification (RQ1–RQ2). For RQ1 and RQ2, we evaluate models on the corresponding annotated dataset for each classification experiment (Section 5.2.2). Each review is processed by the LLM under each prompting setup (zero-shot, few-shot, CoT), and predicted labels are compared with the ground truth. We calculate precision, recall, and F1-score per class, along with macro- and weighted averages. We report mean values over three runs.

Evaluation Procedure for Specification Generation (RQ3). For RQ3, we use 90 annotated reviews randomly sampled from our collected data. Half are taken from the user request dataset, and the other half from the NFR dataset. The same input is provided to each LLM under every prompting setup. Each model generates requirements specification following the template, including Introduction, Functional Requirements, NFRs, and Glossary sections. The first author evaluates the outputs using the five criteria (see Table 5). The results are verified through manual examination of user feedback. The scores are averaged across all samples.

5.3 Results

RQ1: How well do LLMs classify feedback by NFR type?
To answer RQ1, we evaluated how well LLMs classify user feedback by NFRs under three prompting strategies: zero-shot, few-shot, and chain-of-thought. Table 6 reports precision, recall, and F1 scores for each model, with the best results highlighted in bold. The effectiveness ranges from an F1 score of 0.40 to 0.55, with averages of 0.47, 0.49, and 0.51 for the zero-shot, few-shot, and chain-of-thought strategies, respectively. Across prompting strategies, performance improves steadily from zero-shot to few-shot to chain-of-thought, confirming that example-based and reasoning-enhanced prompts help LLMs better identify NFR types. Larger and newer models (Gemma, Llama 3, Mistral) generally outperform smaller or earlier ones (Llama 2, Phi-3 Mini). Gemma achieves the highest overall F1 score (0.55) and precision (0.53) under the chain-of-thought setting, while Llama 3 records the highest recall (0.59). Mistral performs most consistently across all prompting strategies, ranking near the top in zero-shot and few-shot modes. In contrast, Llama 2 yields the lowest scores across all metrics, while Phi-3 Mini remains stable but below the larger models.

Answer to RQ1: LLMs achieve moderate accuracy (F1 $\approx$ 0.47–0.51) for classifying feedback by NFR type. Chain-of-thought prompting performs best, with Gemma leading overall (P=0.53, R=0.57, F1=0.55).

Table 6 How well do LLMs classify feedback by NFR type? (RQ1)

Model	Zero-Shot			Few-Shot			Chain-of-Thought		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 2	0.44	0.36	0.40	0.49	0.39	0.43	0.52	0.46	0.49
Llama 3	0.42	**0.54**	0.47	0.46	**0.57**	0.51	0.48	**0.59**	0.53
Mistral	0.49	0.51	**0.50**	**0.54**	0.51	**0.52**	0.47	**0.59**	0.52
Gemma	**0.51**	0.48	0.49	0.51	0.49	0.50	**0.53**	0.57	**0.55**
Phi-3 Mini	0.44	0.54	0.48	0.46	0.54	0.50	0.44	0.48	0.46
Average	0.46	0.49	0.47	0.49	0.50	0.49	0.49	0.54	0.51

RQ2: How well do LLMs classify feedback by user request type?
To answer RQ2, we examined how LLMs classify user feedback by request type under three prompting strategies: zero-shot, few-shot, and chain-of-thought. Table 7 presents precision, recall, and F1 scores for each model, with the best results in bold. Model performance ranges from an F1 score of 0.32 to 0.74. The average F1 values are 0.59 for zero-shot, 0.68 for few-shot, and 0.64 for chain-of-thought prompting. Performance increases notably from zero-shot to few-shot prompting, while chain-of-thought yields moderate improvements. Larger and more recent models, such as Llama 3, Mistral, and Gemma, generally achieve higher accuracy than smaller or earlier ones, including Llama 2 and Phi-3 Mini. Llama 3 reaches the highest F1 score of 0.74 and recall of 0.75 under the few-shot setting. Mistral and Gemma perform consistently well across

all strategies. Llama 2 produces the lowest scores, while Phi-3 Mini remains stable but below the stronger models. Overall, few-shot prompting provides the best results (average F1 = 0.68); it suggests that including example-based context helps LLMs more accurately classify user feedback by request type.

Answer to RQ2: LLMs achieve moderate-to-high average accuracy (F1 ≈ 0.59–0.68) for classifying feedback by user request type. Few-shot prompting performs best, with Llama 3 leading overall (P=0.72, R=0.75, F1=0.74).

Table 7 How well do LLMs classify user feedback by request type? (RQ2)

Model	Zero-Shot			Few-Shot			Chain-of-Thought		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 2	0.28	0.36	0.32	0.57	0.56	0.57	0.60	0.42	0.49
Llama 3	**0.77**	0.67	**0.72**	0.72	0.75	**0.74**	0.71	0.70	0.71
Mistral	0.60	0.63	0.62	0.69	**0.74**	0.71	0.65	**0.72**	0.68
Gemma	0.66	**0.71**	0.68	0.68	0.67	0.68	0.65	0.60	0.63
Phi-3 Mini	0.69	0.51	0.59	0.68	0.67	0.68	0.67	0.70	**0.69**
Average	0.60	0.58	0.59	0.67	0.68	0.68	0.66	0.63	0.64

RQ3: How well do LLMs generate requirements specification?
To answer RQ3, each model was evaluated on five criteria: structure (SA), completeness (CO), fidelity (FI), conciseness (CN), and clarity (CL), as defined in Table 5. Table 8 reports results across models and prompting strategies. Overall, the models produced moderate-quality specification (mean=3.1; SD=0.8). Llama 3 with chain-of-thought prompting and Mistral with few-shot prompting achieved the highest mean score (3.6), showing strong structure and clarity (SA=4; CL=5). Llama 2 followed with stable mid-range scores (mean=3.2–3.4). Gemma produced the weakest but most consistent outputs (mean ≈ 2.9; SD=0.4). Phi-3 Mini showed high fidelity and conciseness (FI ≈ 4; CN=4–5) but low structure (SA=1–2) and high variability. Across prompting strategies, few-shot and chain-of-thought improved structure and clarity, whereas constraint-based prompting offered limited gains. Most models scored highest in clarity (CL=4–5) and completeness (CO=4) but lagged with fidelity and conciseness (FI, CN=2–3).

Answer to RQ3: LLMs produced moderate-quality specification, with Llama 3 and Mistral performing best; model and prompt choice strongly influenced output quality.

5.4 Discussion

Lightweight LLMs show promising potential for analysing user feedback in RE. Although more efficient and transparent than larger models, they are not yet suitable for reliable industrial use without further customization.

Table 8 LLM performance on requirement specification (RQ3). SA – Structure; CO – Compl.; FI – Fidelity; CN – Conciseness; CL – Clarity. Scores are on a 1–5 scale (higher = better). **Bold** indicates the best overall performance across all models and prompt strategies, based on the highest Mean score (averaged over the five criteria) and the lowest SD.

Model	Prompt Strategy	SA (1–5)	CO (1–5)	FI (1–5)	CN (1–5)	CL (1–5)	Mean	SD
Llama 2	Zero-Shot	3	4	3	3	4	3.4	0.49
	Few-Shot	3	4	3	2	4	3.2	0.75
	Chain-of-Thought	3	4	3	2	4	3.2	0.75
	Constraint-based	3	4	3	2	4	3.2	0.75
	Role-based	3	4	3	2	4	3.2	0.75
Llama 3	Zero-Shot	3	4	3	2	4	3.2	0.75
	Few-Shot	3	4	3	2	4	3.2	0.75
	Chain-of-Thought	4	4	3	2	5	**3.6**	1.02
	Constraint-based	3	4	3	2	4	3.2	0.75
	Role-based	3	4	3	2	4	3.2	0.75
Mistral	Zero-Shot	3	3	3	2	4	3.0	0.63
	Few-Shot	4	4	3	2	5	**3.6**	1.02
	Chain-of-Thought	3	4	3	2	4	3.2	0.75
	Constraint-based	3	4	3	2	4	3.2	0.75
	Role-based	3	4	3	2	4	3.2	0.75
Gemma	Zero-Shot	2	3	3	3	3	2.8	**0.40**
	Few-Shot	3	3	3	2	4	3.0	0.63
	Chain-of-Thought	3	3	3	2	4	3.0	0.63
	Constraint-based	2	3	3	2	4	2.8	0.75
	Role-based	3	3	3	2	4	3.0	0.63
Phi-3 Mini	Zero-Shot	1	4	4	4	2	3.0	1.22
	Few-Shot	2	4	4	4	3	3.4	0.89
	Chain-of-Thought	1	3	4	4	3	3.0	1.10
	Constraint-based	1	3	4	5	2	3.0	1.22
	Role-based	2	4	3	4	3	3.2	0.75
Average	–	2.6	4.0	3.0	2.3	3.8	3.1	0.8

A) Feedback Classification. In feedback classification, Llama 3 and Mistral achieved F1-scores around 0.74 for identifying user request types. This shows that lightweight models can reliably detect explicit requests such as feature suggestions or bug reports. Their performance in NFR classification was weaker, with the best F1-score reaching 0.55. This result aligns with prior studies where models also struggled to capture implicit qualities like usability or reliability [57]. The limitation likely stems from two factors. Lightweight LLMs find it difficult to interpret subtle, context-dependent expressions. They also lack sufficient exposure to RE-specific terminology and training data. Structured prompting with few-shot or reasoning examples led to only minor improvements. These findings suggest that contextual examples and reasoning cues can enhance understanding. However, overall accuracy remains moderate. In practice, this may produce noisy classifications that mislead analysts. As a result, important feedback can be missed, while irrelevant comments may be misclassified as critical requirements.

B) Requirements Specification Generation. Lightweight LLMs produced requirement specification that were generally clear, coherent, and complete. They

expressed user feedback readably but often failed to follow formal structures. Their outputs were sometimes verbose and required editing for conciseness and compliance with standards. Although the generated text was fluent, the models occasionally fabricated requirements, adding information absent from the original feedback. As a result, the content appeared plausible but not always accurate or grounded. In practice, these limitations mean lightweight LLMs can assist analysts by drafting initial requirement statements or summarising feedback. However, they still require human review to ensure accuracy and proper structure. With further refinement and adaptation, they could serve as drafting and documentation aids rather than autonomous specification generators.

C) Implications for Requirements Engineering. Lightweight LLMs can support RE tasks such as feedback filtering, classification, and initial specification drafting. However, they cannot yet replace human analysts. Their moderate precision and recall make them unreliable for use without supervision. These weaknesses may lead to missed insights or false positives that increase review effort. The generated outputs are clear and coherent but often lack factual grounding and formal consistency. This limits their value for downstream RE tasks, e.g., validation, and traceability. In several tasks, their performance is similar to earlier ML approaches [8, 57]. Larger models alone do not guarantee better outcomes in RE. Future work should adapt models to RE contexts using prompt design, fine-tuning, and retrieval-based approaches. These methods can improve accuracy and help lightweight LLMs better support early RE tasks.

5.5 Threats to Validity

Internal Validity. The main threat lies in the manual evaluation of generated requirement specification by a single evaluator, without validating its reliability. This introduces potential subjectivity. To mitigate this, we applied a systematic rubric with clear criteria and examples of both high- and low-quality specification. The rubric was used consistently across all outputs. In addition, prompts and model parameters were standardised to ensure uniform conditions.

External Validity. We used two publicly available datasets from different domains to reduce domain bias. Their diverse vocabulary helps generalisability, though an app review represents only one feedback type. Results may not generalise to industrial datasets or other contexts such as issue trackers. Moreover, the study focused on lightweight LLMs; larger models may yield different results.

Construct Validity. We employed standard precision, recall, and F1-score metrics for classification and a rubric assessing completeness, consistency, and correctness for generation. As qualitative evaluation relied on one evaluator, some interpretation bias may persist. Prompt phrasing may also influence results; this was mitigated by systematically applying established prompting strategies.

Conclusion Validity. All models were tested under identical settings and prompts. However, the limited sample size for specification generation lowers statistical power, and no inferential tests were applied, making results exploratory. While precision and recall were key metrics, tasks may favour one over the other, and F1 may not always be

optimal [63]. Our aim was to assess overall practical effectiveness rather than examine differences across RE-specific tasks.

6 Study II: Requirements Across Project Artefacts

In the second study, we investigate whether frontier LLMs can support requirements traceability across heterogeneous software project artefacts. Unlike the controlled experiment presented in Study I (Sect. 5), Study II is designed as an exploratory industrial case study based on observations from a research visit at Huawei Research Centre. We evaluate two frontier LLMs on two complementary RE tasks: traceability link identification and traceability explanation generation [1]. We first introduce the industrial study context, followed by the empirical study design, experimental findings, and their implications for RE practice.

6.1 Industrial Study Context

Our second study focuses on analysing requirements-related information distributed across software project artefacts. Specifically, we consider *project goals*[10], *GitHub tracking issues*[11], and *Zulip messages*[12] from the Rust open-source ecosystem[9]. These artefacts capture project objectives, implementation activities, technical decisions, and project evolution. As requirements-related information is distributed across multiple artefacts, reconstructing requirement evolution is labour-intensive, motivating automated support for traceability and rationale analysis.

A *project goal* defines a high-level development objective, its motivation, and expected outcomes. A *tracking issue* describes the implementation of a project goal, including implementation tasks, milestones, dependencies, and progress updates. A *Zulip message* captures developer discussions on project goals or tracking issues, including technical reasoning, design decisions, and implementation details. A *traceability link* links a project goal or tracking issue with a related Zulip message describing the same development objective or implementation activity.

The study evaluates two RE tasks. The first, *traceability link identification*, identifies Zulip messages related to a given project goal or tracking issue. The second, *traceability explanation generation*, produces an evidence-grounded explanation describing the identified traceability link and its supporting evidence.

For example, consider the project goal *"Improve compiler support for Rust-for-Linux by stabilizing the required language features."* A related Zulip message states: *"The remaining compiler changes have been merged, and Rust-for-Linux now builds successfully on nightly."* The first task identifies the message as related to the project goal. The second generates the explanation: *"The message provides evidence that the remaining compiler changes have been completed, demonstrating implementation progress towards the Rust-for-Linux compiler support goal."*

6.2 Empirical Study Design

This study evaluates the ability of frontier LLMs to identify traceability links across heterogeneous software project artefacts and generate evidence-grounded explanations. Unlike Study I (see Sect. 5), it adopts an industrial case study to evaluate readily available frontier LLMs in a realistic setting.

6.2.1 Research Questions

The goal of this study is to evaluate LLMs in supporting traceability analysis across distributed software project artefacts. We specifically focus on two research questions:

- **RQ4**: How well do LLMs identify traceability links across project artefacts?
- **RQ5**: How well do LLMs generate traceability explanations?

In RQ4, we assess the ability of LLMs to identify traceability links between Rust Project Goals or GitHub tracking issues and related Zulip discussions. RQ5 evaluates the quality of the explanations generated for the identified traceability links. All generated links and explanations are manually evaluated using predefined assessment criteria (see Sect. 6.2.4). For RQ4, retrieved links are validated by human assessors and analysed using Precision@N and NDCG@N. For RQ5 explanations are assessed through human judgment based on predefined quality criteria.

6.2.2 Dataset

We collected a new dataset from the Rust open-source ecosystem to evaluate traceability identification and explanation generation across software project artefacts[9]. The Rust ecosystem was selected because our industrial partner actively contributes to its development, making cross-artefact traceability a practically relevant engineering challenge. The dataset comprises nine traceability cases from the 2024–2026 Rust Project Goals initiative[10]. The cases were selected to ensure complete traceability scenarios, diversity of technical domains, and coverage of multiple roadmap years (2024–2026). Each case comprises a Rust Project Goal, its corresponding GitHub tracking issue[11], and the associated Zulip discussion[12]:

- **Project Goals** describe strategic objectives, motivations, and expected outcomes.
- **GitHub Tracking Issues** capture implementation planning, task management, dependencies, and progress updates.
- **Zulip Discussions** contain technical reasoning, design decisions, coordination activities, and implementation details.

We retained only project goals with linked GitHub tracking issues and related Zulip discussions. The associated Zulip discussions range from 384 to 39,343 messages, comprising 80,642 messages in total. Table 9 summarises the collected traceability cases, including their year, topic, description, and the number of associated Zulip messages. The cases span compiler engineering, tooling, language design, verification, infrastructure, and project governance. The dataset is available in the replication package [25].

Table 9 Overview of the Rust traceability dataset used in Study II. Each traceability case comprises a Rust Project Goal, its associated GitHub tracking issue, and the corresponding Zulip discussion. The table reports the project year, topic, description, and the number of associated Zulip messages.

Case	Year	Topic	Description	No. Msgs
1	2024	MIR Formality	Formal type system model	1,036
2	2024	Polonius	Scalable borrow checker	14,908
3	2024	Rust for Linux	Stable Linux kernel support	1,808
4	2025	Async Rust	Async language improvements	39,343
5	2025	Parallel Front End	Parallel compiler frontend	6,045
6	2025	StableMIR	Stable compiler API	2,982
7	2026	Build-std	Standard library builds	384
8	2026	Type Documentation	Type system documentation	13,047
9	2026	User Research	User research team	1,089
Total	–	–	–	**80,642**

6.2.3 Pilot Study

We first conducted a pilot study to assess the feasibility of LLM-based traceability identification and explanation generation across distributed software project artefacts. The pilot used two Rust project goals together with their associated GitHub issues and Zulip discussions. It refined the prompting strategy, artefact selection procedure, and manual validation protocol for the full study.

We initially evaluated the best-performing lightweight LLMs from Study I (see Sect. 2), namely Mistral and Llama. Both models performed well on feedback-driven RE tasks. However, their limited context windows restricted reasoning across heterogeneous software project artefacts. Local execution was also computationally demanding. We therefore selected frontier LLMs for the main study.

The pilot confirmed the feasibility of LLM-supported traceability analysis. Evidence-oriented and chain-of-thought prompting produced more technically grounded explanations than simple retrieval prompts. Finer-grained technical artefacts improved retrieval precision. In contrast, retrieving entire discussion threads introduced contextual noise through semantically related but unsupported discussions.

The pilot also showed that Goal–Zulip and Issue–Zulip retrieval are complementary tasks with different levels of ambiguity. We therefore evaluated them separately. Finally, the pilot confirmed the need for manual validation. Plausible-looking links often required verification against the surrounding discussions and Rust documentation. These findings informed the design of the full study.

6.2.4 Evaluation Metrics and Criteria

We applied both quantitative and qualitative methods. Retrieval metrics were used for RQ4, as traceability identification is formulated as a retrieval task. RQ5 was evaluated through criteria-based assessment using predefined quality attributes.

Evaluation Metrics for Traceability Identification (RQ4).

For RQ4, we evaluate the ability of LLMs to identify traceability links between Project Goals or GitHub tracking issues and related Zulip discussions. Retrieved

Table 10 Evaluation criteria and scoring used to assess generated traceability explanations (RQ5).

Criterion	Description and Scoring
Correctness	**Description:** Evaluates whether the explanation accurately justifies the identified traceability link and correctly describes the relationship between the linked artefacts.
	Scoring: Rated on a 1–5 Likert scale, where higher scores indicate greater correctness.
Evidence Grounding	**Description:** Evaluates whether the explanation is supported by information contained in the referenced artefacts without introducing unsupported claims.
	Scoring: Rated on a 1–5 Likert scale, where higher scores indicate stronger evidence grounding.

links are manually validated by human assessors. We report Precision@5 and Normalized Discounted Cumulative Gain (NDCG@5) [64]. We selected a cutoff of five because manual validation of retrieved links is time-intensive, while the highest-ranked candidates are the most relevant in practical traceability analysis. Precision@5 measures the proportion of validated links among the top five retrieved candidates, while NDCG@5 evaluates the quality of their ranking. Goal–Zulip and Issue–Zulip retrieval are evaluated separately. Recall is not reported because no complete ground truth of traceability links is available for the selected artefacts [64]. This is common in large-scale information retrieval tasks, where exhaustive identification of all relevant links is infeasible [24].

Evaluation Criteria for Explanation Generation (RQ5).

For RQ5, traceability explanations are evaluated qualitatively using two criteria: *Correctness* and *Evidence Grounding*. Table 10 summarises the evaluation rubric. Both criteria assess whether explanations correctly justify the identified traceability links and are supported by evidence from the referenced project artefacts. Each criterion is rated on a five-point Likert scale (Table 11) [61]. We report the mean scores across all validated traceability links.

6.2.5 Experimental Setup and Procedure

We now describe the computational setup, prompting strategies, and evaluation procedures used to address RQ4 and RQ5.

Computational Setup

We evaluated GPT-5.5 and DeepSeek-R1 using their latest publicly available versions through the official provider-hosted interfaces. GPT-5.5 was selected as a representative frontier LLM, whereas DeepSeek-R1 represents a leading open-weight reasoning model of interest to our industrial partner. Experiments were managed locally on a MacBook Pro (Apple M2 Pro, 32 GB RAM), while all model inference was performed on the providers' infrastructure.

Table 11 Interpretation of the five-point Likert scale used to evaluate explanation quality.

Score	Correctness	Evidence Grounding
1	Explanation is incorrect, misleading, or contradicts the referenced artefacts.	Evidence is missing, irrelevant, or does not support the identified traceability link.
2	Explanation contains major inaccuracies or unsupported reasoning.	Evidence is weak, vague, or only loosely related to the identified link.
3	Explanation is partially correct but incomplete or insufficiently justified.	Evidence is partially relevant but provides only limited support for the identified link.
4	Explanation is mostly accurate and clearly describes the relationship between the artefacts.	Evidence is relevant and provides adequate support for the identified link.
5	Explanation is accurate, precise, and fully consistent with the referenced artefacts.	Evidence is highly relevant, specific, verifiable, and fully supports the identified link.

Unlike Study I (Sect. 5), each experimental configuration was executed once. As this study aims to evaluate practical capability rather than performance variability, repeated executions were not required. The experimental design comprised 72 executions, covering all combinations of two frontier LLMs, two prompting strategies, two document types, and nine project topics. Response times ranged from approximately 30 seconds to 2 minutes per query, supporting interactive traceability analysis.

Prompting Strategies

We adopted two prompting strategies introduced in Section 2: zero-shot prompting and chain-of-thought (CoT) prompting. These strategies were selected because they provide two complementary reasoning settings while keeping the prompting design simple. Zero-shot prompting serves as a baseline, whereas CoT explicitly supports the multi-step reasoning required for traceability analysis. More complex strategies (e.g., few-shot or role-based prompting) were intentionally excluded to avoid introducing additional prompt-specific effects, as the objective of this study was to evaluate the reasoning capabilities of frontier LLMs rather than optimise prompt design.

Prompt development followed an iterative process during the pilot study. A small subset of project artefacts, excluded from the final evaluation, was used to refine task instructions, reduce ambiguities, and improve evidence grounding. The prompts were revised until they produced consistent outputs across both evaluated models. Once finalised, they remained unchanged throughout the experiment to ensure a fair comparison. Table 12 presents simplified examples of the zero-shot and CoT prompts. The complete prompt templates are available in the replication package [25].

Evaluation Procedures

We applied quantitative evaluation for traceability identification (RQ4) and qualitative evaluation for explanation generation (RQ5).

Table 12 Prompting strategies and simplified examples used in the study.

Prompting Strategy	Simplified Example
Zero-shot	Identify the most relevant Zulip discussion fragments related to the following Rust project goal. Return the top-5 candidate links together with a short explanation.
Chain-of-thought	Analyse the following Rust project goal and Zulip discussion fragments step by step. Explain why the artefacts may be related, identify supporting evidence, and return the top-5 most relevant traceability links with explanations.

Evaluation Procedure for Traceability Identification (RQ4). For each Project Goal or GitHub tracking issue, we provided the corresponding candidate Zulip discussions to the LLM. Under each prompting strategy, the model identified and ranked the five most relevant discussion fragments, extracted supporting evidence, and generated a short explanation for each identified link. Retrieved links were manually validated using predefined assessment guidelines. A traceability link was considered valid when the linked artefacts referred to the same project objective, implementation activity, technical concern, dependency, or design discussion. Precision@5 and NDCG@5 were then computed from the validated rankings.

Evaluation Procedure for Explanation Generation (RQ5). For RQ5, we manually evaluated the explanations associated with the validated traceability links using the criteria described in Sect. 5.2.3. Correctness and Evidence Grounding were assessed independently for each explanation using a five-point Likert scale. We report the mean and median scores across all validated traceability links.

To assess the reliability of the manual evaluation, a second author independently reviewed a randomly selected 10% subset of the evaluated cases. Cohen's Kappa reached 0.83 for traceability link validation [64], indicating almost perfect agreement, whereas Quadratic Weighted Cohen's Kappa reached 0.71 for explanation quality ratings, indicating substantial agreement [65].

6.3 Results

RQ4: How well do LLMs identify traceability links across project artefacts? To answer RQ4, we evaluated how well frontier LLMs identify traceability links between project goals, GitHub tracking issues, and Zulip discussions under two prompting strategies: zero-shot and chain-of-thought. Table 13 reports Precision@5 (P@5) and Normalized Discounted Cumulative Gain (NDCG@5) for Goal–Zulip and Issue–Zulip traceability link identification, with the best results highlighted in bold. For Goal–Zulip traceability link identification, chain-of-thought prompting consistently improved performance. The average P@5 increased from 0.59 to 0.73, while the average NDCG@5 increased from 0.87 to 0.90. ChatGPT achieved the highest NDCG@5 (0.94), whereas both models achieved the highest P@5 (0.73) under chain-of-thought prompting. For Issue–Zulip traceability link identification, ChatGPT

consistently outperformed DeepSeek. Under chain-of-thought prompting, it achieved the highest P@5 (0.91) and NDCG@5 (0.98). In contrast, DeepSeek performed best under zero-shot prompting, with performance decreasing under chain-of-thought prompting. Overall, the models performed better on Issue–Zulip than Goal–Zulip traceability link identification, achieving a higher average P@5 (0.77 versus 0.73).

Answer to RQ4: Frontier LLMs achieved moderate-to-high performance for traceability link identification. Chain-of-thought prompting improved Goal–Zulip links, while ChatGPT performed best overall (P@5=0.91, NDCG@5=0.98).

Table 13 Performance of frontier LLMs on traceability link identification (RQ4). P@5 and NDCG@5 are reported for Goal–Zulip and Issue–Zulip links under zero-shot and chain-of-thought prompting. Best results are shown in bold.

Model	Goal–Zulip Link				Issue–Zulip Link			
	Zero-Shot		Chain-of-Thought		Zero-Shot		Chain-of-Thought	
	P@5	NDCG@5	P@5	NDCG@5	P@5	NDCG@5	P@5	NDCG@5
ChatGPT	0.64	**0.94**	**0.73**	**0.91**	**0.78**	**0.94**	**0.91**	**0.98**
DeepSeek	0.53	0.80	**0.73**	0.89	0.67	**0.94**	0.62	0.89
Average	0.59	0.87	0.73	0.90	0.73	0.94	0.77	0.94

RQ5: How well do LLMs generate traceability explanations?
To answer RQ5, each model was evaluated on two criteria: correctness (CR) and evidence grounding (EG), as defined in Table 10. Table 14 reports results across models, prompting strategies, and traceability link types. Overall, the models produced high-quality traceability explanations. Mean scores ranged from 3.94 to 4.35, with average scores of 4.15 for Goal–Zulip explanations and 4.21 for Issue–Zulip explanations. DeepSeek with chain-of-thought prompting achieved the highest mean score for Goal–Zulip explanations (4.27), showing strong correctness and evidence grounding (CR=4.21; EG=4.33), while ChatGPT achieved the highest correctness score (CR=4.23). ChatGPT with zero-shot prompting achieved the highest mean score for Issue–Zulip explanations (4.35), demonstrating the best correctness and evidence grounding (CR=4.35; EG=4.34). Across prompting strategies, chain-of-thought prompting improved Goal–Zulip explanation quality for both models, whereas DeepSeek showed only modest improvements for Issue–Zulip explanations. Overall, Issue–Zulip explanations received slightly higher scores than Goal–Zulip explanations.

Answer to RQ5: Frontier LLMs produced high-quality traceability explanations. Chain-of-thought prompting improved Goal–Zulip explanations; ChatGPT achieved the best performance for Issue–Zulip explanations (Mean = 4.35).

Table 14 Performance of frontier LLMs on traceability explanation generation (RQ5). CR – Correctness; EG – Evidence Grounding. Scores are on a 1–5 scale (higher = better). Mean denotes the average of CR and EG. **Bold** indicates the best score for each evaluation criterion.

Model	Prompt Strategy	Goal–Zulip			Issue–Zulip		
		CR (1–5)	EG (1–5)	Mean	CR (1–5)	EG (1–5)	Mean
ChatGPT	Zero-Shot	3.89	3.99	3.94	**4.35**	**4.34**	**4.35**
	Chain-of-Thought	**4.23**	4.18	4.21	4.09	4.12	4.11
DeepSeek	Zero-Shot	4.14	4.19	4.17	4.17	4.15	4.16
	Chain-of-Thought	4.21	**4.33**	**4.27**	4.23	4.20	4.22
Average	–	4.12	4.17	4.15	4.21	4.20	4.21

6.4 Discussion

Frontier LLMs show promising potential for supporting requirements traceability across software project artefacts. Although they identify traceability links and generate explanations, human verification remains necessary for reliable industrial use.

A) Traceability Link Identification. Frontier LLMs achieved moderate-to-high performance in identifying traceability links. Performance was consistently higher for Issue–Zulip than Goal–Zulip links. This is expected because roadmap goals describe high-level strategic objectives, whereas GitHub issues refine these objectives into concrete implementation tasks. Zulip discussions typically focus on these implementation activities, making Issue–Zulip links semantically more direct. In contrast, Goal–Zulip links span different levels of abstraction and therefore require greater semantic reasoning. Chain-of-thought prompting particularly improved Goal–Zulip link identification, indicating that explicit reasoning helps bridge the gap between strategic objectives and implementation discussions. These findings suggest that frontier LLMs can reduce the effort of recovering traceability links across heterogeneous software project artefacts. Nevertheless, human verification remains necessary when traceability information supports downstream activities such as impact analysis or release planning.

B) Traceability Explanation Generation. Frontier LLMs generated explanations that were generally correct and evidence-grounded. They successfully explained why project artefacts were related and summarized the underlying design rationale. Goal–Zulip explanations benefited more from chain-of-thought prompting than Issue–Zulip explanations. This is expected as project goals capture high-level objectives, while the supporting evidence is distributed across multiple implementation discussions. Explaining these relationships therefore requires reasoning across different levels of abstraction. In contrast, GitHub issues and Zulip discussions are semantically closer and typically provide sufficient contextual evidence without additional reasoning. Automatically generated explanations can help developers understand why project artefacts are related and provide useful context during repository exploration. However, they should be treated as supporting evidence rather than authoritative rationale, particularly when decisions depend on interpretation of project history.

C) Implications for Requirements Engineering. Frontier LLMs achieved sufficient performance to support requirements traceability across heterogeneous software project artefacts. Both traceability identification and explanation generation can help practitioners reconstruct relationships between strategic project goals, GitHub

issues, and developer discussions with substantially less manual effort. This can improve design rationale understanding, requirements impact analysis, and developer onboarding. The results also highlight an important trade-off for industrial adoption. Frontier LLMs rely on commercial providers, raising concerns about vendor lock-in, privacy, and operational costs. However, our pilot study found that current off-the-shelf lightweight LLMs are not yet practical. Their limited context windows and slow local execution hinder analysis of large software repositories. Moreover, requirements traceability is only one of many RE activities. Organisations are therefore unlikely to customise and maintain lightweight LLMs solely for this task. General-purpose frontier LLMs that support multiple RE/SE activities may currently offer the more practical solution. Nevertheless, frontier LLMs should augment rather than replace engineers. Incorrect traceability links or explanations may otherwise propagate errors to engineering activities such as requirements impact analysis and release planning.

6.5 Threats to Validity

Internal Validity. The main threat lies in the manual validation of traceability links and explanation quality. Assessing whether a retrieved discussion genuinely supports a project goal or issue often requires interpretation of technical context and Rust-specific concepts. To mitigate this threat, we applied predefined validation criteria and evaluation guidelines throughout the study. A second evaluator independently assessed a randomly selected subset of the results. Substantial inter-rater agreement supported the reliability of the manual evaluation. When ambiguities arose, official Rust documentation and tutorials were consulted. Nevertheless, some interpretation bias may remain.

External Validity. A threat to external validity concerns the generalisability of the results. The study was conducted using a single case study based on the Rust ecosystem. The project was selected because it provides a rare combination of publicly available project goals, tracking issues, and developer discussions, enabling transparent and reproducible evaluation. Comparable industrial datasets are rarely publicly available due to confidentiality constraints. While Rust is a large and industrially relevant open-source project, it represents only one development context. Other open-source or industrial projects may follow different development practices and rely on different artefacts. We do not claim direct generalisability. Rather, the goal of this study is to provide an in-depth illustration of applying LLM-supported traceability analysis in a realistic open-source setting.

Construct Validity. Another threat concerns whether the applied measures adequately capture the intended requirements engineering constructs. Concepts such as traceability links, rationale, dependencies, and explanation quality originate from established requirements engineering and traceability research, whereas the analysed artefacts are informal and community-driven. As a result, some relationships may only partially reflect these constructs or oversimplify them. To mitigate this threat, we grounded the evaluation criteria in established traceability and information-retrieval measures, and assessed explanation quality using predefined dimensions of correctness and evidence grounding. Nevertheless, some constructs may remain underspecified due to the informal nature of the source artefacts.

Conclusion Validity. A threat to conclusion validity concerns the strength and reliability of the conclusions. The study does not aim to establish causal relationships or benchmark model performance. Instead, it reports observations from applying off-the-shelf LLMs in a realistic open-source software setting. The number of analysed artefacts is limited by the effort required for manual validation. Furthermore, no complete ground truth exists for the selected artefacts. These factors restrict the strength of the conclusions. The findings should therefore be interpreted as exploratory insights into the practical usefulness of LLM-supported traceability analysis rather than definitive evidence of effectiveness.

7 Cross-study Discussion

Although the two studies investigated different RE activities, artefacts, models, and evaluation settings, they provide complementary evidence on the capabilities and limitations of LLMs for RE. This section synthesises the findings and discusses their broader implications for RE research and practice.

7.1 Cross-Task Perspective on LLM Performance in RE

The two empirical studies provide a broader perspective on LLM capabilities across five complementary RE activities: requirements classification, requirements specification, traceability identification, and traceability explanation generation. Table 15 synthesises the empirical findings by summarising the best-performing model, prompting strategy, and overall performance achieved for each RE task. The synthesis shows that no single model or prompting strategy consistently achieved the best performance across all RE activities. Instead, LLM effectiveness was strongly task-dependent, with different models proving more suitable for different RE tasks. Performance ranged from moderate to high depending on the reasoning demands, artefact characteristics, and expected outputs of the target activity. Accordingly, the findings should be interpreted as evidence of the practical readiness of current LLMs for supporting specific RE activities rather than as a ranking of models or prompting strategies. From an RE perspective, these findings suggest that RE/SE practitioners should adopt LLMs selectively rather than uniformly across the requirements lifecycle. Model selection should be guided by the characteristics and reasoning demands of the target RE task rather than by overall model rankings. Human expertise nevertheless remains essential wherever engineering judgement and validation are required.

7.2 Prompting Matters, but Depends on the RE Task

Across both studies, prompting strategy consistently influenced LLM performance, although the most effective strategy varied according between RE activities. As summarised in Table 15, Chain-of-Thought prompting generally benefited reasoning-intensive tasks, including NFR classification, traceability identification, and traceability explanation generation. In contrast, few-shot prompting proved most effective for user request classification by providing representative examples that reduced ambiguity. These findings suggest that prompt engineering should be viewed as RE

Table 15 Cross-task empirical understanding of LLM performance across Requirements Engineering (RE) activities.

RE Task	Best Model	Best Prompt	Performance
NFR Classification	Gemma	Chain-of-Thought	Moderate
User Request Classification	Llama 3	Few-shot	High
Requirements Specification	Llama 3 / Mistral	CoT / Few-shot	Moderate
Traceability Identification	GPT-5.5	Chain-of-Thought	High
Traceability Explanation	GPT-5.5	Chain-of-Thought	Moderate–High

task-specific rather than universally applicable. Consequently, prompting strategies should be selected according to the reasoning demands and expected outputs of the target RE task rather than applied uniformly across the RE lifecycle.

7.3 Human Oversight Remains Essential

Although both studies demonstrate the potential of LLMs to support multiple RE activities, they also show that human oversight remains essential. The level of supervision required, however, depends on the complexity of the target RE task. Tasks involving relatively well-defined decisions, such as user request classification and traceability identification, generally produced reliable outputs that can reduce manual effort, although verification remains necessary. In contrast, tasks requiring more complex reasoning, synthesising information across multiple artefacts, or generating requirements-related artefacts were more susceptible to unsupported, incomplete, or insufficiently grounded outputs. These differences can largely be explained by the reasoning demands of individual RE activities. As tasks require synthesising information across multiple artefacts, preserving semantic consistency, or generating new textual artefacts, the likelihood of grounding and consistency issues increases. Consequently, LLMs should be viewed as decision-support tools that augment rather than replace RE practitioners.

7.4 Practical Implications for Requirements Engineering

The findings have several practical implications for RE practice. First, LLM adoption should be driven by the characteristics and reasoning demands of the target RE task rather than by model popularity or benchmark rankings. Lightweight open-source models performed well on short-context tasks, such as analysing user feedback. They also offer advantages in local deployment, reproducibility, and computational efficiency. However, our pilot experimentation showed that these models were impractical for traceability analysis across heterogeneous software project artefacts because of their limited context windows, weaker reasoning capabilities, and substantially longer execution times. Consequently, frontier models were adopted for Study II to support reasoning across long and heterogeneous artefacts. Second, model selection and prompt design should be considered jointly. Although prompting consistently influenced performance, the most effective strategy depended on the reasoning demands of the target RE task rather than on the model alone. Finally, LLM outputs should be

treated as decision-support artefacts rather than final engineering products, regardless of the model family. Requirements classifications, traceability links, generated specifications, and explanations should all be validated before being incorporated into SE practice. Overall, our findings indicate that the successful adoption of LLMs in RE depends less on identifying a universally superior model than on selecting models and prompting strategies that match the characteristics and reasoning demands of the target RE task.

7.5 Future Research Directions

The cross-task perspective presented in this work also highlights several opportunities for future research. First, improving factual grounding remains an important challenge, particularly for requirements specification generation and other generation-oriented RE activities. Second, future work should investigate hybrid approaches combining prompting with retrieval-augmented generation, repository-aware retrieval, or fine-tuning on RE-specific corpora to improve factual consistency and domain adaptation. More broadly, evaluating LLMs across multiple RE activities using common datasets, evaluation protocols, and replication packages would support more systematic cross-task comparisons and strengthen cumulative empirical evidence on the role of LLMs throughout the RE lifecycle. Rather than evaluating individual RE tasks in isolation, future studies should increasingly investigate how LLMs support complementary activities across the RE lifecycle. Future studies should therefore move beyond evaluating individual RE tasks in isolation and instead investigate how LLMs support complementary activities across the RE lifecycle.

8 Conclusion

Requirements-related information is increasingly distributed across heterogeneous software project artefacts, making many RE activities labour-intensive and difficult to scale. Large LLMs offer new opportunities to support these activities [43]. This paper addressed the fragmented empirical evidence on LLMs for RE through the first cross-task empirical evaluation spanning five representative RE activities. By integrating evidence from two complementary empirical studies, it provides a broader perspective on the capabilities and limitations of current LLMs across different RE contexts.

The findings demonstrate that LLM effectiveness is strongly RE task-dependent. Current LLMs can effectively support several RE activities, but their performance depends on the reasoning demands of the target task, the characteristics of the analysed artefacts, and the required outputs. Rather than identifying a universally superior model or prompting strategy, the results show that successful LLM adoption requires selecting models and prompts that are appropriate for the specific RE activity. Human expertise nevertheless remains essential wherever engineering judgement, factual grounding, and validation are required.

Beyond the individual empirical studies, this work contributes cumulative evidence on the practical readiness of current LLMs for RE. It demonstrates the value of evaluating LLMs across complementary RE activities rather than in isolation, providing a

broader understanding of where these models are already effective and where important limitations remain. This cross-task perspective also offers practical guidance for RE/SE practitioners seeking to integrate LLMs into their engineering workflows.

Finally, the released replication package, including datasets, prompts, scripts, and experimental materials, supports transparency and reproducibility [25]. We hope this work encourages future empirical studies spanning multiple RE activities, enabling a more cumulative understanding of the role of LLMs throughout the RE lifecycle.

Supplementary information. A replication package containing the datasets, prompts, scripts, and experimental materials is provided as supplementary material for peer review [25].

Acknowledgements. Study 1 was partially conducted within the MSc thesis of M. A. Mallya, supervised by J. Dąbrowski [66]. Study 2 was largely conducted during a research visit by J. Dąbrowski to the Huawei Ireland Research Center in Dublin, Ireland. The results contribute to the Prompt Me project [67]. This publication has emanated from research jointly funded by Taighde Éireann – Research Ireland under Grant Number 13/RC/2094_2 and co-funded by the European Union under the Systems, Methods, Context (SyMeCo) programme, Grant Agreement Number 101081459. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Declarations

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Zadenoori, M.A., Dąbrowski, J., Alhoshan, W., Zhao, L., Ferrari, A.: Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review (2025). https://arxiv.org/abs/2509.11446

[2] Arora, C., al.: In: Nguyen-Duc, A., Abrahamsson, P., Khomh, F. (eds.) Advancing Requirements Engineering Through Generative AI: Assessing the Role of LLMs, pp. 129–148. Springer, ??? (2024)

[3] Ferrari, A., Ginde, G.: Handbook on Natural Language Processing for Requirements Engineering: Overview, pp. 1–15 (2025). https://doi.org/10.1007/978-3-031-73143-3_1

[4] Dąbrowski, J., Cai, W., Bennaceur, A., Nuseibeh, B., Alrimawi, F.: Intelligent agents for requirements engineering: Use, feasibility and evaluation. In: 2025 IEEE 33rd International Requirements Engineering Conference (RE), pp. 535–543 (2025)

[5] Lim, S., Henriksson, A., Zdravkovic, J.: Data-driven requirements elicitation: A systematic literature review. SN Comput. Sci. **2**(1) (2021) https://doi.org/10.1007/s42979-020-00416-4

[6] Alhoshan, W., Ferrari, A., Zhao, L.: How Effective are Generative Large Language Models in Performing Requirements Classification? (2025). https://arxiv.org/abs/2504.16768

[7] Dąbrowski, J., Letier, E., Perini, A., Susi, A.: Mining user feedback for software engineering: Use cases and reference architecture. In: RE, pp. 114–126. IEEE, ??? (2022)

[8] Dąbrowski, J., Letier, E., Perini, A., Susi, A.: Analysing app reviews for software engineering: a systematic literature review. Empirical Softw. Engg. **27**(2) (2022)

[9] Dalpiaz, F., Ferrari, A., Franch, X., Palomares, C.: Natural language processing for requirements engineering: The best is yet to come. IEEE Software **35**(5), 115–119 (2018) https://doi.org/10.1109/MS.2018.3571242

[10] Maalej, W., Nayebi, M., Johann, T., Ruhe, G.: Toward data-driven requirements engineering. IEEE Softw. **33**(1), 48–54 (2016) https://doi.org/10.1109/MS.2015.153

[11] Hou, X., al.: Large language models for software engineering: A systematic literature review. ACM Trans. Softw. Eng. Methodol. (2024)

[12] Franch, X.: Data-driven requirements engineering: A guided tour. In: Ali, R., Kaindl, H., Maciaszek, L.A. (eds.) Evaluation of Novel Approaches to Software Engineering - 15th International Conference, ENASE 2020, Prague, Czech Republic, May 5-6, 2020, Revised Selected Papers. Communications in Computer and Information Science, vol. 1375, pp. 83–105. Springer, ??? (2020). https://doi.org/10.1007/978-3-030-70006-5_4 . https://doi.org/10.1007/978-3-030-70006-5_4

[13] Maalej, W., Nayebi, M., Ruhe, G.: Data-driven requirements engineering: an update. In: Sharp, H., Whalen, M. (eds.) Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 289–290. IEEE / ACM, ??? (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00041 . https://doi.org/10.1109/ICSE-SEIP.2019.00041

[14] Vogelsang, A., Fischbach, J.: Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline (2024). https://arxiv.org/abs/2402.13823

[15] Zadenoori, M.A., Martino, V.D., Dąbrowski, J., Franch, X., Ferrari, A.: Does

model size matter? A comparison of small and large language models for requirements classification. CoRR **abs/2510.21443** (2025) https://doi.org/10.48550/ARXIV.2510.21443 2510.21443

[16] Alturayeif, N., Ahmad, I., Hassine, J.: Tracellm: leveraging large language models with prompt engineering for enhanced requirements traceability. Requir. Eng. **31**(1) (2026) https://doi.org/10.1007/s00766-026-00460-1

[17] Mallya, M.A., Ferrari, A., Zadenoori, M.A., Dabrowski, J.: From online user feedback to requirements: Evaluating large language models for classification and specification tasks. In: Guizzardi, R.S.S., Araújo, J. (eds.) Requirements Engineering: Foundation for Software Quality - 32nd International Working Conference, REFSQ 2026, Poznań, Poland, March 23-26, 2026, Proceedings. Lecture Notes in Computer Science, vol. 16497, pp. 161–177. Springer, ??? (2026). https://doi.org/10.1007/978-3-032-21423-2_11 . https://doi.org/10.1007/978-3-032-21423-2_11

[18] Hess, A., Vogelsang, A., Franch, X., Herrmann, A., Kopczynska, S., Rachmann, A.: Opportunities and limitations of genai in RE: viewpoints from practice. In: Guizzardi, R.S.S., Araújo, J. (eds.) Requirements Engineering: Foundation for Software Quality - 32nd International Working Conference, REFSQ 2026, Poznań, Poland, March 23-26, 2026, Proceedings. Lecture Notes in Computer Science, vol. 16497, pp. 320–335. Springer, ??? (2026). https://doi.org/10.1007/978-3-032-21423-2_22 . https://doi.org/10.1007/978-3-032-21423-2_22

[19] Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., Zhang, J.M.: Large language models for software engineering: Survey and open problems. CoRR **abs/2310.03533** (2023) https://doi.org/10.48550/ARXIV.2310.03533 2310.03533

[20] Zhao, W.X., Zhou, K., Li, J., Tang, T., Dong, Z., Hou, Y., Zhang, B., Min, Y., Zhang, J., Liu, P., Wang, X., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Hu, Y., Nie, J.-Y., Wen, J.-R.: A survey of large language models. Frontiers of Computer Science **20**, 2012627 (2026) https://doi.org/10.1007/s11704-026-60308-3

[21] Cheng, H., Husen, J.H., Lu, Y., Racharak, T., Yoshioka, N., Ubayashi, N., Washizaki, H.: Generative ai for requirements engineering: A systematic literature review. Software: Practice and Experience **56**(2), 141–170 (2026) https://doi.org/10.1002/spe.70029 https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.70029

[22] Frattini, J., Motger, Q.: Towards a software reference architecture for natural language processing tools in requirements engineering. In: Guizzardi, R.S.S., Araújo, J., Brito, I.S.S., Sales, T.P., Horkoff, J., Insfrán, E., Vara, J.L., Paja, E., Mussbacher, G., Schneider, K., Karras, O., Adamo, G., Dabrowski, J., Dalpiaz, F., Fotrousi, F., Motger, Q., Deters, H., Droste, J., Hess, A., Schmid, E. (eds.) Joint Proceedings of REFSQ-2026 Workshops, Doctoral Symposium, Posters &

Tools Track, and Education and Training Track 30th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2026), Poznań, Poland, March 23-26, 2026. CEUR Workshop Proceedings, vol. 4208. CEUR-WS.org, ??? (2026). https://ceur-ws.org/Vol-4208/nlp4re-short1.pdf

[23] Abualhaija, S., Aydemir, F.B., Dalpiaz, F., Dell'Anna, D., Ferrari, A., Franch, X., Fucci, D.: Replication in requirements engineering: The NLP for RE case. ACM Trans. Softw. Eng. Methodol. **33**(6), 151 (2024) https://doi.org/10.1145/3658669

[24] Dąbrowski, J., Letier, E., Perini, A., Susi, A.: Mining and searching app reviews for requirements engineering: Evaluation and replication studies. Inf. Syst. **114** (2023)

[25] Dąbrowski et al.: Replication Package. The replication package will be made publicly available upon acceptance of the manuscript. (2026)

[26] Fan, A., al.: Large Language Models for Software Engineering: Survey and Open Problems . In: ICSE-FoSE, pp. 31–53. IEEE Computer Society, ??? (2023)

[27] Binkhonain, M., Alfayez, R.: Are prompts all you need? evaluating prompt-based large language models (llm) s for software requirements classification. Requirements Engineering, 1–21 (2025)

[28] Wang, F., Lin, M., Ma, Y., Liu, H., He, Q., Tang, X., Tang, J., Pei, J., Wang, S.: A survey on small language models in the era of large language models: Architecture, capabilities, and trustworthiness. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2. KDD '25, pp. 6173–6183. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3711896.3736563 . https://doi.org/10.1145/3711896.3736563

[29] Korn, A., Zaruchas, L., Arora, C., Metzger, A., Smolka, S., Wang, F., Vogelsang, A.: Reporting LLM prompting in automated software engineering: A guideline based on current practices and expectations. CoRR **abs/2601.01954** (2026) https://doi.org/10.48550/ARXIV.2601.01954 2601.01954

[30] Huang, K., Wang, F., Huang, Y., Arora, C.: Prompt engineering for requirements engineering: A literature review and roadmap. 2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW), 548–557 (2025)

[31] Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: ICSE '00, pp. 35–46. ACM, ??? (2000)

[32] Motger, Q., Oriol, M., Tiessler, M., Franch, X., Marco, J.: What about emotions? guiding fine-grained emotion extraction from mobile app reviews. In: 33rd IEEE International Requirements Engineering Conference, RE 2025, Valencia, Spain, September 1-5, 2025, pp. 6–18. IEEE, ??? (2025). https://doi.org/10.1109/RE63999.2025.00012 . https://doi.org/10.1109/RE63999.2025.00012

[33] Wang, X., Huang, J., Ye, C., Zhou, H.: Globaltagnet: A graph-based framework for multi-label classification in github issues. In: Liebel, G., Hadar, I., Spoletini, P. (eds.) 32nd IEEE International Requirements Engineering Conference, RE 2024, Reykjavik, Iceland, June 24-28, 2024, pp. 67–78. IEEE, ??? (2024). https://doi.org/10.1109/RE59067.2024.00017 . https://doi.org/10.1109/RE59067.2024.00017

[34] Morales-Ramirez, I., Kifetew, F.M., Perini, A.: Speech-acts based analysis for requirements discovery from online discussions. Inf. Syst. **86**, 94–112 (2019) https://doi.org/10.1016/J.IS.2018.08.003

[35] Kifetew, F.M., Perini, A., Susi, A., Siena, A., Muñante, D., Morales-Ramirez, I.: Automating user-feedback driven requirements prioritization. Inf. Softw. Technol. **138**, 106635 (2021) https://doi.org/10.1016/J.INFSOF.2021.106635

[36] Rani, L.M., Svensson, R.B., Feldt, R.: Ai for requirements engineering: Industry adoption and practitioner perspectives. In: 2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), pp. 244–251. IEEE Press, ??? (2025). https://doi.org/10.1109/ASEW67777.2025.00053 . https://doi.org/10.1109/ASEW67777.2025.00053

[37] Patel, K., Modi, H., Shah, U.: LACE-HC: A lightweight attention-based classifier for efficient hierarchical classification of software requirements. In: Hess, A., Susi, A. (eds.) Requirements Engineering: Foundation for Software Quality - 31st International Working Conference, REFSQ 2025, Barcelona, Spain, April 7-10, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15588, pp. 181–196. Springer, ??? (2025). https://doi.org/10.1007/978-3-031-88531-0_13 . https://doi.org/10.1007/978-3-031-88531-0_13

[38] Shah, F.A., Sabir, A., Sharma, R., Pfahl, D.: How effectively do llms extract feature-sentiment pairs from app reviews? In: Hess, A., Susi, A. (eds.) Requirements Engineering: Foundation for Software Quality - 31st International Working Conference, REFSQ 2025, Barcelona, Spain, April 7-10, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15588, pp. 123–138. Springer, ??? (2025). https://doi.org/10.1007/978-3-031-88531-0_9 . https://doi.org/10.1007/978-3-031-88531-0_9

[39] Paßlack, J., Specht, A., Herrmann, M., Elsofi, D.A.A., Ehl, M., Großer, K., Jürjens, J., Schneider, K.: Can we use llms to recover trace links between source code and security requirements? In: 33rd IEEE International Requirements Engineering Conference, RE 2025 Workshops, Valencia, Spain, September 1-5, 2025, pp. 223–232. IEEE, ??? (2025). https://doi.org/10.1109/REW66121.2025.00035 . https://doi.org/10.1109/REW66121.2025.00035

[40] Amna, A.R., Wautelet, Y., Poelmans, S., Heng, S., Poels, G.: The ambitrus framework for identifying potential ambiguity in user stories. J. Syst. Softw. **223**, 112357 (2025) https://doi.org/10.1016/J.JSS.2025.112357

[41] Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.-V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. ACM Comput. Surv. **54**(3) (2021) https://doi.org/10.1145/3444689

[42] Sonbol, R., Rebdawi, G., Ghneim, N.: The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review. IEEE Access **PP**, 1–1 (2022)

[43] Cheng, H., Husen, J.H., Lu, Y., Racharak, T., Yoshioka, N., Ubayashi, N., Washizaki, H.: Generative ai for requirements engineering: A systematic literature review. arXiv preprint arXiv:2409.06741 (2024)

[44] Pasquale, L., Ragone, A., Piemontese, E., Darban, A.A.: Exploring the use of llms for requirements specification in an IT consulting company. In: 33rd IEEE International Requirements Engineering Conference, RE 2025, Valencia, Spain, September 1-5, 2025, pp. 389–399. IEEE, ??? (2025). https://doi.org/10.1109/RE63999.2025.00045 . https://doi.org/10.1109/RE63999.2025.00045

[45] Chou, C.Y., Aydemir, F.B., Dalpiaz, F.: A comparative study of large and small language models for domain model extraction. In: Guizzardi, R.S.S., Araújo, J. (eds.) Requirements Engineering: Foundation for Software Quality - 32nd International Working Conference, REFSQ 2026, Poznań, Poland, March 23-26, 2026, Proceedings. Lecture Notes in Computer Science, vol. 16497, pp. 336–351. Springer, ??? (2026). https://doi.org/10.1007/978-3-032-21423-2_23 . https://doi.org/10.1007/978-3-032-21423-2_23

[46] Groen, E.C., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., Hosseini, M., Marco, J., Oriol, M., Perini, A., Stade, M.: The crowd in requirements engineering: The landscape and challenges. IEEE Software **34**(2), 44–52 (2017) https://doi.org/10.1109/MS.2017.33

[47] Hemmat, A., Sharbaf, M., Kolahdouz-Rahimi, S., Lano, K., Tehrani, S.Y.: Research directions for using llm in software requirement engineering: a systematic review. Frontiers in Computer Science **Volume 7 - 2025** (2025) https://doi.org/10.3389/fcomp.2025.1519437

[48] Franch, X., Gnesi, S., Paccosi, F., Quer, C., Semini, L.: Leveraging requirements elicitation through software requirement patterns and llms. In: Hess, A., Susi, A. (eds.) Requirements Engineering: Foundation for Software Quality, pp. 261–276. Springer, Cham (2025)

[49] Ullrich, J., Koch, M., Vogelsang, A.: From requirements to code: Understanding developer practices in llm-assisted software engineering. In: 2025 IEEE 33rd International Requirements Engineering Conference (RE), pp. 257–266 (2025)

[50] Ibtasham, M.S., Bashir, S., Abbas, M., Haider, Z., Saadatmand, M., Cicchetti,

A.: Reqrag: Enhancing software release management through retrieval-augmented llms: An industrial study. In: Hess, A., Susi, A. (eds.) Requirements Engineering: Foundation for Software Quality, pp. 277–292. Springer, Cham (2025)

[51] Lubos, S., Felfernig, A., Tran, T.N.T., Garber, D., El Mansi, M., Erdeniz, S.P., Le, V.-M.: Leveraging llms for the quality assurance of software requirements. In: 2024 IEEE 32nd International Requirements Engineering Conference (RE), pp. 389–397 (2024). https://doi.org/10.1109/RE59067.2024.00046

[52] Paiva, G., Canedo, E.D., Rocha Filho, G.P.: From issue titles to requirements: an empirical study of large language models and prompt engineering strategies. Requir. Eng. **31**(1), 1–23 (2026) https://doi.org/10.1007/s00766-026-00462-z

[53] Motger, Q., Oriol, M., Tiessler, M., Franch, X., Marco, J.: What about emotions? guiding fine-grained emotion extraction from mobile app reviews. In: 2025 IEEE 33rd International Requirements Engineering Conference (RE), pp. 6–18 (2025)

[54] Al-Subaihin, A.A., Sarro, F., Black, S., Capra, L., Harman, M.: App store effects on software engineering practices. IEEE Trans. Softw. Eng **47**(2), 300–319 (2021)

[55] ISO/IEC: Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models. International Organization for Standardization. https://www.iso.org/standard/35733.html

[56] Jha, N., Mahmoud, A.: Using frame semantics for classifying and summarizing application store reviews. Empirical Software Engineering **23**(6), 3734–3767 (2018) https://doi.org/10.1007/s10664-018-9605-x

[57] Lu, M., Liang, P.: Automatic classification of non-functional requirements from augmented app user reviews. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE 2017), pp. 344–353. Association for Computing Machinery (ACM), Karlskrona, Sweden (2017). https://doi.org/10.1145/3084226.3084241

[58] Chen, N., Lin, J., Hoi, S.C.H., Xiao, X., Zhang, B.: Ar-miner: Mining informative reviews for developers from mobile app marketplace. In: Proceedings of the 36th International Conference on Software Engineering (ICSE 2014), pp. 767–778. ACM, Hyderabad, India (2014)

[59] Maalej, W., Kurtanovic, Z., Nabil, H., Stanik, C.: On the automatic classification of app reviews. In: Proceedings of the 2016 IEEE 23rd International Conference on Requirements Engineering (RE), pp. 49–58. IEEE, Beijing, China (2016). https://doi.org/10.1109/RE.2016.23

[60] Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd edn. O'Reilly Media, ??? (2022)

[61] Kuckartz, U.: Qualitative Text Analysis: A Guide to Methods, Practice & Using Software, 1st edn. SAGE Publications, London / Los Angeles (2014). https://doi.org/10.4135/9781446288719

[62] Porter, D., DeFranco, J.F., Laplante, P.: Requirements specification automated quality analysis: Past, present, and future. Computer **58**(1), 101–104 (2025) https://doi.org/10.1109/MC.2024.3480629

[63] Berry, D.M.: Requirements engineering for artificial intelligence: What is a requirements specification for an artificial intelligence? In: REFSQ, pp. 19–25 (2022)

[64] Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, England (2008). https://stanford.edu

[65] Cohen, J.: Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. Psychological Bulletin **70**(4), 213–220 (1968) https://doi.org/10.1037/h0026256

[66] Mallya, M.A.: Developing software requirements using llms: A preliminary empirical study. Master of science in software engineering with data analytics (msc), University of Limerick, Limerick, Ireland (August 2025). Supervised by Dr. Jacek Dabrowski. Student ID: 24124133

[67] Dąbrowski, J.: Prompt Me: Intelligent Software Agent for Requirements Engineering (2025). https://prompt-me.github.io/