Evaluating the Capability of Prompted LLMs to Recommend NFR from User Stories: A Preliminary Study

José R. A. Pereira
Federal University of Campina Grande
Campina Grande, PB, Brazil
carlos.souza@virtus.ufcg.edu.br

Mirko Perkusich
Federal University of Campina Grande
Campina Grande, PB, Brazil
mirko@virtus.ufcg.edu.br

Felipe B. A. Ramos
Federal Institute of Paraíba
Santa Luzia, PB, Brazil
felipe.ramos@ifpb.edu.br

Danyllo W. Albuquerque
Federal University of Campina Grande
Campina Grande, PB, Brazil
danyllo.albuquerque@virtus.ufcg.edu.br

Kyller Costa Gorgônio
Federal University of Campina Grande
Campina Grande, PB, Brazil
kyller@virtus.ufcg.edu.br

Ângelo Perkusich
Federal University of Campina Grande
Campina Grande, PB, Brazil
perkusich@virtus.ufcg.edu.br

ABSTRACT

[Context] Non-functional requirements (NFRs) are critical to software quality but are often underspecified in agile projects. Previous work proposed NFRec, a k-nearest neighbors (kNN) recommender system, to support NFR elicitation based on structured User Story metadata. **[Objective]** This study investigates whether Large Language Models (LLMs), when prompted with structured representations of User Stories, can generate relevant NFRs comparable to those recommended by NFRec. **[Method]** We reused the original dataset of 246 User Stories and adopted the same evaluation protocol. The `gpt-4.1-mini` model was queried using zero-shot prompting, instruction tuning, and role-playing strategies. Predicted NFRs were evaluated using effectiveness measures against the original ground truth. **[Results]** The LLM achieved high recall and moderate F1 performance but lower precision due to frequent overgeneration. The quality of recommendations was highly sensitive to prompt design. In several cases, the model produced plausible NFRs not present in the baseline, suggesting that traditional metrics may understate its practical value. **[Conclusion]** Prompted LLMs offer a viable and flexible alternative for NFR elicitation, especially in cold-start scenarios where historical data is scarce. This study serves as an initial step toward LLM-assisted requirements engineering, opening up new directions for research in prompt engineering, hybrid models, and evaluation metrics that better reflect semantic relevance and practical utility.

KEYWORDS

Non-functional requirements; Agile projects; Large Language Models; Prompt engineering; Recommender systems.

1 Introduction

Agile Software Development (ASD) has become the dominant paradigm in software engineering due to its emphasis on adaptability, rapid feedback, and incremental delivery [3]. However, this flexibility often comes at the cost of structure, particularly in how software requirements are captured, shared, and reused across teams [5]. Among these requirements, Non-functional requirements (NFRs) are frequently overlooked or poorly specified in agile contexts, despite their critical impact on software quality and user satisfaction [7].

Previous work has explored the use of recommender systems based on similarity metrics to support NFR elicitation in ASD. One such approach is NFRec [22], which applies a k-nearest neighbors (kNN) algorithm to suggest relevant NFRs using a project-specific dataset derived from Scrum artifacts. While effective when sufficient historical data is available, this type of solution suffers from cold-start limitations and depends on the availability of structured, labeled examples [24, 26].

Meanwhile, the emergence of Large Language Models (LLMs), such as GPT-based models, has transformed many areas of software engineering [1, 4]. These models demonstrate strong reasoning capabilities and generalize well to new tasks through prompt-based interaction [2, 19], making them attractive candidates for replacing or complementing traditional recommenders. Recent systematic reviews have identified recommendation as one of the most common applications of LLMs in software engineering [11], and LLMs have already shown promising results in domains such as API suggestion [25], code completion [13], traceability link generation [18], and tag recommendation for developer forums [10].

In parallel, the recommender systems community has begun to explore LLMs as generative recommenders [16, 17], capable of producing suggestions directly from open-ended user inputs, without relying on structured historical data. These models are particularly effective in contexts where data is sparse, domain-specific, or expressed in natural language—all characteristics commonly observed in NFR elicitation tasks.

Despite this growing body of work, the potential of LLMs for recommending NFRs remains underexplored. This paper investigates whether prompt-based interaction with LLMs—specifically using the gpt-4.1-mini model—can be used to generate relevant NFR suggestions from functional requirement (FR) descriptions, even in zero-shot scenarios. We conducted a comparative evaluation using a previously established dataset and benchmark from the literature [22], enabling a head-to-head comparison between the LLM-based approach and the earlier kNN-based system.

The remainder of this paper is structured as follows. Section 2 introduces the original kNN approach and discusses

the growing use of LLMs in recommender systems. Section 3 presents our experimental design. Section 4 reports the results of our comparative study. Section 5 discusses this study's implications, while Section 6 describes the threats to validity. Finally, Section 7 concludes the paper and outlines future work.

2 Background and Related Work

This section provides the foundation for our study. We begin by introducing NFRec, a baseline kNN-based recommender for NFR elicitation. We then review how LLMs have been applied to general recommendation tasks, and conclude by discussing their use in SE contexts.

NFRec Background. NFRec is a recommender system designed to support the elicitation of NFRs in agile projects. It combines content-based and collaborative filtering to suggest NFRs based on similarities between structured FR profiles. These profiles encode project metadata derived from Scrum artifacts, including domain, platform, architecture, language, frameworks, APIs, persistence, and implemented tasks associated to FRs.

Each FR is annotated with one or more NFRs, grouped into three categories: (1) *Performance:* response time, capacity, transit delay, efficiency compliance; (2) *Reliability:* availability, integrity, fault tolerance, recoverability; and (3) *Security:* confidentiality, access control, authentication.

NFRs are structured as tuples of `NFR_Type`, `NFR_Attribute`, and `NFR_Sentence` (e.g., *"The system must ensure the response_time by responding within 2 seconds."*). This format supports both categorical and semantic evaluations (see Section 3).

Table 1 shows example FR profiles. The metadata fields (i.e., *Operation*, *Platform*) follow the taxonomy proposed by Dilorenzo et al. [9], which organizes user stories to promote reuse. This taxonomy enables structured comparison across FRs and underpins similarity-based retrieval in NFRec.

The dataset comprises 13 projects spanning domains such as education, business, and utilities. Table 2 summarizes the number of FRs, NFRs, and tasks per project. On average, each FR is linked to 1.8 NFRs.

NFRec uses kNN to recommend NFRs from similar FR profiles. Among several distance metrics tested, Manhattan distance with $k = 1$ yielded the best performance. In system-centric evaluation, NFRec achieved 78.5% precision, 81.6% recall, and 79.0% F-measure. User-centric evaluation showed consistent results (81.8% precision) and confirmed that development teams found the tool useful and easy to apply.

These findings establish a strong baseline but also expose limitations: NFRec depends entirely on structured input and prior data, making it susceptible to cold-start problems. In addition, the approach is limited to recommending NFRs that are explicitly associated with individual FRs in the dataset. System-wide or crosscutting quality attributes—such as high availability or global efficiency goals—are not represented, as the original study focused on localized, FR-linked NFRs. This motivates exploring prompt-based LLMs, which can operate in data-scarce environments and potentially reason about both local and global quality concerns.

LLMs in Recommender Systems. LLMs have emerged as flexible and powerful components within recommender system architectures. Recent surveys [16, 17] highlight the various roles that LLMs can play throughout the recommendation pipeline, including feature extraction, user modeling, and even direct generation of recommendations via natural language prompts.

Unlike conventional models that require structured user-item interaction data, LLMs leverage open-domain knowledge, semantic reasoning, and contextual understanding [14]. These characteristics make them particularly effective in cold-start scenarios and tasks where structured data is scarce, enabling zero-shot or few-shot performance [15]. Several studies propose frameworks for LLM-based recommendation, investigating how variations in prompt structure, task formulation, and context encoding impact performance [20, 29].

While these models demonstrate high recommendation accuracy, they also raise concerns about bias, hallucinations, and fairness [8]. Consequently, new evaluation strategies have been developed to assess LLM-based recommenders beyond traditional utility metrics, incorporating dimensions such as recency, diversity, and stability [14].

Recent efforts have further explored how ChatGPT can serve as a general-purpose recommender, analyzing the effects of prompt design, system roles, and user interaction strategies [21]. These findings emphasize the critical role of prompt engineering in shaping model outputs and offer practical guidance for adapting LLM-based recommenders to new domains, including tasks such as requirements elicitation.

LLMs for Recommendation in Software Engineering. In software engineering, LLMs have been applied to recommend code completions [13], APIs [12, 25], traceability links [18, 31], metadata [10], and method names [30]. These tasks often require aligning informal language with technical artifacts.

Models like BERT, CodeBERT, and GraphCodeBERT have outperformed classical IR approaches, especially when pre-trained on domain-specific corpora [10]. Prompt-tuning has also proven useful in adapting LLMs to software-specific tasks. Additionally, a systematic literature review [11] covering 395 studies found that recommendation is among the most frequent uses of LLMs in SE (6.77% of papers). However, nearly all focus on recommending code-level artifacts; the application of LLMs to NFR elicitation remains underexplored.

Given that NFRs are often implicit, informal, and context-dependent, LLMs—when guided by effective prompts—may offer a viable approach for generating relevant suggestions in settings where traditional, data-driven recommenders struggle. This work explores this hypothesis, focusing on the recommendation of NFRs from FR descriptions in agile project settings.

3 Methodology

This study adopts an explanatory sequential mixed-methods approach [6] to evaluate whether LLMs can effectively support the recommendation of NFRs from structured FR profiles. The approach comprises two phases: (i) an initial quantitative analysis using system-centric metrics, followed by (ii) a focused qualitative inspection of disagreement cases.

Table 1: Examples of structured FR profiles used in NFRec

ID	Mod.	Ope.	Plat.	Arch.	Domain	Obj.	Lang.	Fram.	API	DB	Tasks
f1	Authen.	Create Account	Web	MVC	Home Auto.	Prot.	Java	Node	Facebook	MongoDB	t1, t2
f2	Regist.	Insert Data	Web	Client-Serv	Edu	Prod.	JavaScript	Node	Mongoose	MongoDB	t3, t4
f3	Regist.	Insert Data	Web	Client-Serv	Edu	Prod.	Java	Springboot	JPA	MySQL	t3, t5
f4	Regist.	Retrieve Data	Web	Client-Serv	Edu	Prod.	TypeScript	Angular	Mongoose	MongoDB	t6, t7
f5	Regist.	Retrieve Data	Web	Layered	Info. Rec.	Prod.	Python	Django	Firebase	MySQL	t6, t7

Table 2: Summary of dataset used in NFRec

Project	Domain	FRs	NFRs	Tasks	NFRs/FR
P1	Entertainment	9	22	31	2.4
P2	Utilities	6	10	15	1.7
P3	Utilities	16	22	36	1.4
P4	Education	30	61	100	2.0
P5	Education	60	81	216	1.4
P6	Business	44	93	126	2.1
P7	Business	39	90	136	2.3
P8	Info. Assets	49	80	131	1.6
P9	Info. Assets	13	27	53	2.1
P10	Info. Assets	27	29	64	1.1
P11	Communications	36	85	91	2.4
P12	Info. Assets	23	37	53	1.6
P13	Office	31	56	75	1.8
Overall	—	383	693	1127	1.8

3.1 Dataset and Evaluation Metrics

This study reuses the same dataset and evaluation strategy adopted by NFRec (see Section 2). The dataset contains 383 FR profiles annotated with 693 NFRs, covering 13 projects from various domains. Each profile includes structured metadata derived from a taxonomy of user stories [9], as well as associated implementation tasks. The average NFRs-per-FR ratio is 1.8.

Following the original evaluation protocol, we adopted a system-centric approach to assess the performance of our model. Precision, Recall, and F-measure are used as metrics, allowing direct comparison with the NFRec baseline. We acknowledge that some FR profiles in the dataset lack NFR annotations, which may affect the observed performance. However, this limitation is consistent across both studies and does not compromise the comparative analysis.

3.2 LLM Configuration and Prompting Strategy

To generate NFR recommendations, we use the `gpt-4.1-mini` model via OpenAI's API. This model was selected for its favorable cost-to-performance ratio, with API costs of $0.40 per million tokens for input and $1.60 per million for output. All prompts were designed to remain within practical token limits while preserving clarity.

The prompt design followed established techniques such as Chain-of-Thought and Role-Playing [21, 27]. Specifically, the model was instructed to act as a Requirements Engineer capable of interpreting structured FR profiles and recommending suitable NFRs based on contextual factors such as domain, architecture, and implementation details.

It is important to note that the prompt did not constrain the number of NFRs to match the ground truth. The model was free to recommend as many requirements as it deemed relevant. This design choice reflects realistic usage scenarios and may result in a greater number of suggestions than those found in the reference dataset, potentially lowering precision but increasing coverage.

To improve transparency, the following box presents a summarized version of the prompt structure. In practice, the actual API interaction used a system message to define the model's role ("You are a Requirements Engineer") and a separate user prompt containing the remaining instructions. The full prompt configuration—including system and user roles—is available in the accompanying GitHub repository.

Prompt Template for Recommending NFRs

You are a Requirements Engineer. Based on the following structured input (including domain, architecture, platform, language, and associated tasks), recommend the most relevant NFRs. Use only the following types and attributes:
Performance: response_time, capacity, transit_delay, efficiency_complianc
Reliability: availability, integrity, fault_tolerance, recoverability
Security: confidentiality, access_control, authentication
For each recommended NFR, return a JSON object with the following structure:

- `NFR_Type`
- `NFR_Attribute`
- `NFR_Sentence` (e.g., "The system must ensure the response_time by responding within 2 seconds.")

Return a valid JSON array. Do not include explanations or additional text outside the array. Recommend multiple NFRs if contextually justified. The number of NFRs may exceed the number of tasks when appropriate.

Example:

Prompt (user input): "Given the following user story: 'As a user, I want to reset my password so I can regain access to my account.' Suggest relevant non-functional requirements."
LLM Output:

```
{
"NFR_Type": "Security",
"NFR_Attribute": "authentication",
"NFR_Sentence": "The␣system␣must␣ensure␣
     authentication␣by␣verifying␣the␣user's␣
     identity␣before␣allowing␣password␣reset."
},
{
"NFR_Type": "Reliability",
"NFR_Attribute": "recoverability",
"NFR_Sentence": "The␣system␣must␣ensure␣
     recoverability␣by␣allowing␣users␣to␣
     regain␣access␣even␣if␣their␣credentials␣
     are␣lost."
}
]
```

Although the prompt instructs the model to generate complete NFR sentences, our evaluation in this study considers only the `NFR_Type` and `NFR_Attribute` fields. While reference sentences exist in the dataset, we did not assess the similarity between generated and reference `NFR_Sentence`

values. This simplifies evaluation and focuses on the classification aspect of the task, but represents a limitation in terms of assessing the semantic quality of the output.

All datasets, prompt templates, and generated results are publicly available in the accompanying GitHub repository.

3.3 Quantitative Analysis

In the first phase of the evaluation, we conducted a quantitative analysis by comparing the NFRs recommended by the LLM against the reference dataset. Each predicted NFR was matched against the ground truth using an exact comparison of the `NFR_Type` and `NFR_Attribute` fields.

True Positives (TP) were counted when the LLM-predicted NFR matched one from the reference dataset. False Positives (FP) corresponded to predicted NFRs not present in the ground truth, while False Negatives (FN) referred to ground truth NFRs that the LLM did not recommend. Evaluation metrics—Precision, Recall, and F1-measure—were computed globally by aggregating TP, FP, and FN across all FR profiles. These results were then compared to those obtained by NFRec [22].

3.4 Qualitative Analysis of Disagreements

To complement the quantitative evaluation, we conducted a focused qualitative analysis of disagreement cases. Given the large number of mismatches between model predictions and reference annotations, we selected a representative project for manual inspection.

Specifically, we chose one project with a near-average agreement rate and a manageable number of disagreements. Each disagreement in this sample was reviewed by one of the authors to determine whether the additional NFRs suggested by the LLM—those not present in the ground truth—were valid yet missing recommendations, or simply incorrect predictions. This phase aims to uncover potential gaps in the reference dataset, as well as identify limitations in the model's reasoning or prompt configuration.

4 Results and Discussion

This section presents the results of our evaluation, organized into two parts. First, we report the quantitative performance of the LLM in terms of precision, recall, and F1-measure, comparing it to the NFRec baseline. Then, we provide a qualitative analysis of disagreement cases to better understand the nature of the mismatches and explore whether the LLM's broader recommendations may still offer practical value despite deviating from the ground truth.

4.1 Quantitative Analysis

Table 3 summarizes the number of TP, FP, and FN observed across the 13 projects in the dataset. These values were aggregated to compute macro-level effectiveness measures (i.e., Precision, Recall, and F1-measure), enabling a direct comparison with the NFRec baseline.

From these aggregate values, we derived the following performance metrics for the LLM:

- **Precision:** 24.6%
- **Recall:** 72.1%
- **F1-measure:** 36.7%

Table 3: TP, FP, and FN per project

Project	TP	FP	FN
P01	14	17	2
P02	6	7	1
P03	15	67	4
P04	46	129	9
P05	57	186	9
P06	47	108	20
P07	39	158	32
P08	37	242	24
P09	20	38	3
P10	12	108	14
P11	48	45	18
P12	16	60	11
P13	39	50	6
Total	**396**	**1215**	**153**

In comparison, the NFRec baseline reported 78.5% precision, 81.6% recall, and a 79.0% F1-measure [22]. The sharpest contrast lies in precision, which is substantially lower for the LLM-based approach.

However, these results should not be interpreted as evidence of poor performance. While the LLM produces more FPs, it also achieves high recall, indicating strong coverage of the relevant NFRs defined in the ground truth. The lower precision stems from the LLM's broader generalization capacity—an inherent feature of foundation models that rely on extensive world knowledge rather than local patterns.

Unlike NFRec, which is constrained to recommending NFRs previously observed in similar cases, the LLM is capable of suggesting requirements that may not have been documented in the reference dataset but are still contextually valid. This raises questions about the completeness and representativeness of the ground truth itself, particularly since many of the original projects offer limited information about their expected quality levels, target scale, or development maturity (e.g., MVPs vs. production-ready systems).

NFR recommenders are best viewed as exploratory assistants. In agile or early-stage projects, where requirements are often incomplete, it is better to surface plausible suggestions—even at the risk of false positives—than to miss critical quality concerns [23]. False positives can be reviewed and discarded, but false negatives may go unnoticed and harm system quality. Prior work emphasizes recall as a priority in such settings. Still, too many irrelevant suggestions can reduce trust in the tool, making it resemble a generic checklist. High recall is useful—but only if balanced with enough precision to keep recommendations actionable.

4.2 Qualitative Analysis of Disagreements

To complement the quantitative evaluation and contextualize the disagreement cases, we conducted a qualitative review. Given the large number of mismatches across the dataset, we selected project P09 as a representative case for in-depth analysis. This project exhibited a near-average agreement rate and a manageable number of disagreements (38 in total), enabling a thorough expert review.

The analysis shows that the LLM correctly identified the dominant NFR categories—Security, Performance, and Reliability—across most user stories. For example, in authentication-related scenarios (US01–US03), the model accurately recommended authentication, access control, and confidentiality, which were present in the ground truth. However, it also predicted additional performance-related attributes such as response time and capacity. These were not annotated but are arguably relevant in real systems, especially under production-like expectations.

Similar over-predictions occurred in US04 and US07, where the LLM suggested attributes like high availability and efficiency compliance. While these were absent from the manual annotations, they align with common quality expectations in user data management features. In account deletion (US05) and password recovery (US06) scenarios, the LLM correctly included security and reliability attributes and further proposed relevant qualities like recoverability and integrity, again extending beyond the ground truth.

In visualization features (US08), the LLM predicted performance attributes such as rendering time and support for concurrent users. While these recommendations were not included in the dataset annotations, they reflect real-world expectations that could enhance user experience and system robustness.

Notably, several of the LLM's predictions—such as high availability and support for concurrent users—refer to cross-cutting concerns typically addressed at the system level. In agile settings, such requirements are often captured in the Definition of Done or broader quality policies, rather than being explicitly tied to individual FRs (i.e., user stories). Since the reference dataset annotated only NFRs directly linked to specific FRs, this structural limitation may have caused valid, system-wide concerns to be labeled as false positives. This suggests that LLMs may complement conventional elicitation practices by surfacing global quality requirements that are often overlooked in FR-level analyses.

More broadly, the LLM appears to leverage domain knowledge and software engineering best practices to make generalizations that exceed the scope of the original annotations. In several cases, these generalizations resulted in plausible—though unsolicited—NFRs that were penalized in the quantitative evaluation despite their practical relevance.

In summary, the model tends to overemphasize performance-related attributes—such as response time, capacity, and efficiency compliance—even when the primary concern in a user story lies elsewhere. This behavior likely stems from the LLM's learned priors about common quality expectations in software systems. While some of these suggestions did not match the ground truth, they were defensible from a design and engineering perspective. These findings reinforce the potential of LLMs to support NFR elicitation by augmenting human judgment with contextual reasoning and industry-aligned recommendations.

5 Implications for Research and Practice

This study offers insights for both researchers and practitioners interested in enhancing NFR elicitation using LLMs. While our findings confirm that LLMs can produce a broader set of NFR recommendations than traditional approaches, they also expose challenges related to evaluation, dataset limitations, and prompt sensitivity. In what follows, we discuss implications for future research and practical adoption below.

Implications for Research. Our study highlights the potential and current limitations of LLMs in supporting NFR elicitation from structured textual input. Based on our results, we identify the following implications:

- *Limitations of Traditional Evaluation Metrics.* The LLM's low precision stemmed from suggesting plausible NFRs not in the ground truth. This highlights a limitation of conventional metrics (e.g., precision, F1) in capturing the value of generative models for early-stage elicitation. As noted by Jiang et al. [14], broader metrics like plausibility and diversity should be considered.
- *Sensitivity to Prompt Design.* We observed that output quality depends heavily on how the prompt is formulated. While our study used a consistent, carefully constructed zero-shot prompt, future work should investigate the impact of alternative prompting strategies and prompt variations on recommendation quality.
- *Potential for Dataset Enrichment*. Our qualitative analysis revealed several cases where LLM-generated NFRs, although absent from the reference dataset, were contextually valid and aligned with best practices. This suggests that LLMs could support the expansion of incomplete datasets, especially in cold-start or underspecified projects. Additionally, many relevant NFRs (e.g., availability, scalability) are usually crosscutting concerns typically defined in system-wide artifacts (e.g., Definition of Done) rather than attached to individual FRs. Future datasets should better reflect this layered nature of NFR specification.

Implications for Practice. This study reinforces the emerging role of LLMs as assistants in requirements elicitation. Despite their lower precision under conventional metrics, LLMs can serve as valuable tools to expand the scope of consideration, stimulate stakeholder discussion, and identify overlooked quality concerns.

- *Augmenting, Not Replacing, Human Expertise.* Our qualitative analysis revealed that many "false positives" generated by the LLM were, in fact, valid NFRs missing from the ground truth. This suggests that LLMs are best positioned as assistants that stimulate reflection and expand the analyst's perspective, rather than as automated replacements.
- *Reducing Cold-Start Limitations.* Unlike knowledge-based recommenders such as NFRec, LLMs do not require historical data to make suggestions. This can be particularly useful in early-phase projects or when facing new domains or architectures for which similar past cases are unavailable.
- *Enabling Lightweight NFR Support in Agile Teams.* Agile development processes often neglect systematic NFR elicitation due to time constraints or lack of formal structure. LLMs offer a low-effort way to generate initial

suggestions based on high-level context, which can be refined collaboratively with stakeholders.

Overall, our findings support the use of LLMs as practical support tools for NFR elicitation. However, they also underscore the need for careful prompt design, interpretive evaluation, and human oversight in their application.

6 Threats to Validity

We discuss potential threats to the validity of our findings using standard classifications: construct, internal, external, and reliability validity [28].

Construct Validity. Our evaluation assumes that the reference dataset and its manually annotated NFRs are complete and correct. However, as shown in our qualitative analysis, several LLM suggestions—labeled as false positives—may actually be valid but unannotated, revealing potential incompleteness or narrowness in the ground truth. Additionally, the dataset only includes NFRs explicitly linked to specific functional requirements, omitting crosscutting concerns (e.g., availability, scalability) that are often addressed at the system level in agile projects. This structural constraint may penalize otherwise reasonable recommendations. Our evaluation also focused solely on `NFR_Type` and `NFR_Attribute`, ignoring the `NFR_Sentence` field—potentially overlooking semantic differences or alignments in how requirements were phrased. Moreover, following the protocol used in NFRec, we relied on traditional system-centric metrics (precision, recall, and F1), which emphasize exact matches. This framing may underestimate the value of plausible, diverse, or contextually appropriate NFRs generated by the LLM, as emphasized by Jiang et al. [14].

Internal Validity. All experiments were run using GPT-4.1-mini with fixed parameters and temperature zero. We did not measure variation across multiple runs or models. Also, the quality expectations of the dataset projects are not clearly documented. These factors may influence how thoroughly NFRs were specified and annotated. In addition, the prompt was tailored to the dataset and task, raising the risk that results are specific to this configuration and may not generalize to other scenarios.

External Validity. This study is based on a single dataset composed of user stories from Brazilian software projects and focuses exclusively on three NFR categories. As such, the results may not capture the diversity, complexity, or rigor found in other domains, including safety-critical systems or large-scale industrial environments. Additionally, all user stories were written in Portuguese, which may introduce linguistic or cultural biases that limit generalizability to other languages or regions. Another limitation is the exclusive use of a single model—GPT-4.1-mini (OpenAI)—without comparison to other commercial or open-source LLMs. While this model offered competitive performance, different architectures or tuning strategies may yield varying results. Still, these constraints are consistent with our goal of exploring feasibility under controlled and reproducible conditions.

Reliability. The prompt template and all datasets are available online to support replication. However, the qualitative review of disagreements was limited to a single project (P09) and performed by a single researcher, which may introduce interpretation bias. Finally, our experiments used OpenAI's API with version-controlled prompts and temperature settings. However, commercial APIs may change behavior over time without notice, which can affect reproducibility.

7 Conclusions

This study demonstrates the feasibility of using LLMs to support the elicitation of NFRs in agile software projects. Unlike traditional data-driven recommenders like NFRec, LLMs operate without requiring historical datasets, making them especially useful in cold-start scenarios or settings with limited documentation. While the LLM in our study tended to overgenerate compared to the ground truth, many of its suggestions aligned with best practices and addressed relevant quality attributes not explicitly documented by human analysts.

At the same time, our results underscore key limitations. The model's output is sensitive to prompt phrasing, and conventional evaluation metrics—such as precision and recall—may not fully capture the practical utility or semantic correctness of generative outputs. This highlights the value of combining quantitative analysis with expert-led qualitative reviews to better interpret the nature of LLM-generated recommendations.

Future work should expand empirical validation across different domains, projects, and LLM variants, and evaluate the effectiveness of more advanced techniques such as few-shot prompting, instruction tuning, and retrieval-augmented generation (RAG). In parallel, hybrid architectures that combine the creativity of LLMs with the precision of structured recommenders (e.g., kNN-based systems) represent a promising research direction.

Finally, we encourage the development of richer datasets, standardized evaluation protocols, and shareable prompt templates to advance reproducibility in this area. We hope these initial findings will inspire future work on LLM-driven tools for NFR elicitation and broaden how we think about completeness and quality in early requirements engineering.

ARTIFACT AVAILABILITY

All artifacts related to this study—including prompts, scripts, detailed methodology, intermediate results, and evaluation resources—are publicly available to facilitate transparency, replication, and further research[1].

ACKNOWLEDGMENTS

This work has been partially funded by the project 'iSOP Base: Investigação e desenvolvimento de base arquitetural e tecnológica da Intelligent Sensing Operating Platform (iSOP)' supported by CENTRO DE COMPETÊNCIA EMBRAPII VIRTUS EM HARDWARE INTELIGENTE PARA INDÚSTRIA - VIRTUS-CC, with financial resources from the PPI HardwareBR of the MCTI grant number 055/2023, signed with EMBRAPII.

[1]Available at: https://anonymous.4open.science/r/NRFRecLLM-1ABE/

REFERENCES

[1] Danyllo Albuquerque, Everton Guimarães, Graziela Tonin, Pilar Rodríguezs, Mirko Perkusich, Hyggo Almeida, Angelo Perkusich, and Ferdinandy Chagas. 2023. Managing Technical Debt Using Intelligent Techniques - A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2202–2220. https://doi.org/10.1109/TSE.2022.3214764

[2] Yonatha Almeida, Danyllo Albuquerque, Emanuel Dantas Filho, Felipe Muniz, Katyusco de Farias Santos, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. 2024. AICodeReview: Advancing code quality with AI-enhanced reviews. *SoftwareX* 26 (2024), 101677.

[3] Corey Baham and Rudy Hirschheim. 2022. Issues, challenges, and a proposed theoretical core of agile software development research. *Information Systems Journal* 32, 1 (2022), 103–129.

[4] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the Opportunities and Risks of Foundation Models. *ArXiv* (2021). https://crfm.stanford.edu/assets/report.pdf

[5] Lan Cao and Balasubramaniam Ramesh. 2008. Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software* 25, 1 (2008), 60–67. https://doi.org/10.1109/MS.2008.1

[6] John W Creswell and J David Creswell. 2018. Mixed methods procedures. *Research Defign: Qualitative, Quantitative, and Mixed M ethods Approaches* 31, 3 (2018), 75–77.

[7] Karina Curcio, Tiago Navarro, Andreia Malucelli, and Sheila Reinehr. 2018. Requirements engineering: A systematic mapping study in agile software development. *J. Syst. Softw.* 139, C (May 2018), 32–50. https://doi.org/10.1016/j.jss.2018.01.036

[8] Yashar Deldjoo. 2024. Understanding biases in ChatGPT-based recommender systems: Provider fairness, temporal stability, and recency. *ACM Transactions on Recommender Systems* (2024).

[9] Ednaldo Dilorenzo, Emanuel Dantas, Mirko Perkusich, Felipe Ramos, Alexandre Costa, Danyllo Albuquerque, Hyggo Almeida, and Angelo Perkusich. 2020. Enabling the Reuse of Software Development Assets Through a Taxonomy for User Stories. *IEEE Access* 8 (2020), 107285–107300. https://doi.org/10.1109/ACCESS.2020.2996951

[10] Junda He, Xin Zhou, Bowen Xu, Ting Zhang, Kisub Kim, Zhou Yang, Ferdian Thung, Ivana Clairine Irsan, and David Lo. 2024. Representation learning for stack overflow posts: How far are we? *ACM Transactions on Software Engineering and Methodology* 33, 3 (2024), 1–24.

[11] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE] https://arxiv.org/abs/2308.10620

[12] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 293–304.

[13] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. 2022. Codefill: Multi-token code completion by jointly learning from structure and naming sequences. In *Proceedings of the 44th international conference on software engineering*. 401–412.

[14] Chumeng Jiang, Jiayin Wang, Weizhi Ma, Charles LA Clarke, Shuai Wang, Chuhan Wu, and Min Zhang. 2025. Beyond Utility: Evaluating LLM as Recommender. In *Proceedings of the ACM on Web Conference 2025*. 3850–3862.

[15] Genki Kusano, Kosuke Akimoto, and Kunihiro Takeoka. 2024. Are Longer Prompts Always Better? Prompt Selection in Large Language Models for Recommendation Systems. *arXiv preprint arXiv:2412.14454* (2024).

[16] Lei Li, Yongfeng Zhang, Dugang Liu, and Li Chen. 2023. Large language models for generative recommendation: A survey and visionary discussions. *arXiv preprint arXiv:2309.01157* (2023).

[17] Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Hao Zhang, Yong Liu, Chuhan Wu, Xiangyang Li, Chenxu Zhu, et al. 2025. How can recommender systems benefit from large language models: A survey. *ACM Transactions on Information Systems* 43, 2 (2025), 1–47.

[18] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability transformed: Generating more accurate links with pre-trained bert models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 324–335.

[19] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys* 55, 9 (2023), 1–35.

[20] Ahtsham Manzoor, Samuel C Ziegler, Klaus Maria Pirker Garcia, and Dietmar Jannach. 2024. ChatGPT as a conversational recommender system: A user-centric analysis. In *Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization*. 267–272.

[21] Dario Di Palma, Giovanni Maria Biancofiore, Vito Walter Anelli, Fedelucio Narducci, Tommaso Di Noia, and Eugenio Di Sciascio. 2024. Evaluating ChatGPT as a Recommender System: A Rigorous Approach. arXiv:2309.03613 [cs.IR] https://arxiv.org/abs/2309.03613

[22] Felipe Ramos, Alexandre Costa, Mirko Perkusich, Luiz Silva, Dalton Valadares, Ademar de Sousa Neto, Felipe Cunha, Hyggo Almeida, and Angelo Perkusich. 2025. A Data-Driven Recommendation System for Enhancing Non-Functional Requirements Elicitation in Scrum-Based Projects. *IEEE Access* 13 (2025), 44000–44023. https://doi.org/10.1109/ACCESS.2025.3548631

[23] John Slankas and Laurie Williams. 2013. Automated extraction of non-functional requirements in available documentation. In *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*. IEEE, 9–16.

[24] Jianling Wang, Haokai Lu, James Caverlee, Ed H Chi, and Minmin Chen. 2024. Large language models as data augmenters for cold-start item recommendation. In *Companion Proceedings of the ACM Web Conference 2024*. 726–729.

[25] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. 2022. Clear: contrastive learning for api recommendation. In *Proceedings of the 44th International Conference on Software Engineering*. 376–387.

[26] Yinwei Wei, Xiang Wang, Qi Li, Liqiang Nie, Yan Li, Xuanping Li, and Tat-Seng Chua. 2021. Contrastive learning for cold-start recommendation. In *Proceedings of the 29th ACM international conference on multimedia*. 5382–5390.

[27] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. arXiv:2303.07839 [cs.SE] https://arxiv.org/abs/2303.07839

[28] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.

[29] Lanling Xu, Junjie Zhang, Bingqian Li, Jinpeng Wang, Sheng Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2025. Tapping the Potential of Large Language Models as Recommender Systems: A Comprehensive Framework and Empirical Analysis. *ACM Transactions on Knowledge Discovery from Data* (2025).

[30] Jie Zhu, Lingwei Li, Li Yang, Xiaoxiao Ma, and Chun Zuo. 2023. Automating method naming with context-aware prompt-tuning. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 203–214.

[31] Jianfei Zhu, Guanping Xiao, Zheng Zheng, and Yulei Sui. 2022. Enhancing traceability link recovery with unlabeled data. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 446–457.