Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 136 (2025) 701–706

www.elsevier.com/locate/procedia

35th CIRP Design 2025

Advancing Requirements Engineering with Large Language Models

Claudius Ellsel[a]*, Rainer Stark[a]

[a]*Chair of Industrial Information Technology, Technische Universität Berlin, Pascalstraße 8–9, 10587 Berlin, Germany*

* Corresponding author. Tel.: +49 30 314 22908. *E-mail address:* ellsel@tu-berlin.de

Abstract

This paper examines the application of Large Language Models (LLMs) in the requirements engineering process to improve the quality of textual product requirements. A custom-developed software tool demonstrates the utilization of LLMs in practically relevant workflows to reformulate requirements and assess their adherence to predefined quality criteria. In a comparative study, GPT-4o outperformed other models, closely aligning with human performance in quality assessments. These findings highlight the potential of LLMs to significantly enhance the process of working with textual product requirements, which often vary in quality and structure, with significant implications for the broader field of software and product development.

Keywords: Requirements Engineering; Product Requirements; Large Language Models; AI in Engineering; Quality Assessment

1. Introduction

Artificial Intelligence (AI) has long supported various engineering processes. With recent advances in Large Language Models (LLMs), new opportunities for optimizing engineering activities have emerged. Based on the transformer architecture, first introduced by Vaswani et al. [1], continuous improvements in LLMs have been seen in recent years [2]. This also includes open-source models, such as *LLaMA* (*Large Language Model Meta AI*) [3] and can be observed in respective benchmarks, such as [4] and [5].

1.1. Motivation

A promising application area is requirements engineering, which involves managing extensive volumes of textual product requirements. The text-heavy nature of this domain makes it well-suited for Natural Language Processing (NLP) with LLMs. Their application in the field of RE is actively discussed in the community [6]. However, despite their promise, LLMs face challenges such as limited long-term memory [7], problems with logic [8], and hallucinations [9], [10]. Addressing these limitations is crucial for their effective deployment in engineering contexts.

Initial research indicates that LLMs can automate and enhance requirements processes [11], [12], but practical integration remains in its early stages. This paper presents a novel software tool that leverages LLMs to refine product requirements by combining automated quality assessment and structured reformulation, demonstrating its integration into existing engineering workflows.

1.2. Existing research utilizing LLMs in requirements engineering

Recent studies explore using LLMs like *GPT-4* in requirements engineering, covering tasks from elicitation and user story generation to specification refinement.

Ronanki et al. evaluated *GPT-4*'s performance in requirements elicitation, finding that it outperformed human experts in abstraction, atomicity, consistency, correctness, and

2212-8271
10.1016/j.procir.2025.08.120

clarity. However, *GPT-4* showed weaknesses, sometimes generating ambiguous and unfeasible requirements [13].

Rahman and Zhu introduced *GeneUS* using *GPT-4* for automated user story generation through a Refinement and Thought (RaT) prompting technique, which significantly improved efficiency and developer satisfaction [10].

Krishna et al. benchmarked *GPT-4* and *CodeLlama* against a junior software engineer's evaluations using an IEEE-compliant software requirements specification (SRS) document. *CodeLlama* produced detailed, consistent documents but with redundancy, while *GPT-4* generated more concise and clear outputs [14].

Wang et al. developed *ChatCoder*, which refines code generation requirements through a two-phase dialogue system: paraphrasing for completeness and iterative deep questioning to enhance clarity [15].

The existing approaches show potential of the application of LLMs in the requirements engineering process. Thus, this paper implements a software tool that can integrate into the existing engineering process and support with refining product requirements.

2. Description of the approach

To effectively incorporate AI into engineering processes, it is important to ensure accessibility and usability. Thus, a software tool is proposed to seamlessly integrate into existing product development workflows, aiming to enhance the clarity and quality of requirement specifications using LLMs.

The tool's architectural concept follows a modular structure (see Fig. 1.), designed to seamlessly integrate into current engineering environments.

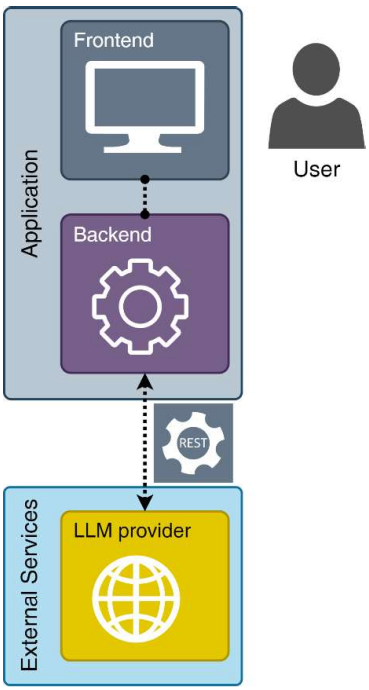

Fig. 1. Concept for the architecture of the software tool

The application consists of a user-friendly frontend and a robust backend that handles the processing logic, and communication with LLMs via external services. The frontend enables users to interactively review and refine requirements, while the backend manages the processing logic, handles data streams, and coordinates actions. The LLMs from external services are typically connected through a REST API.

The LLM-supported features include a numerical quality assessment, providing a percentage-based score to identify requirements needing improvement. Users can then modify requirements manually or with LLM assistance. The automated reformulation can follow two paths:

First, by following a predefined sentence template (e.g., the *FunctionalMASTER* template by *SOPHIST* [16] as shown in Fig. 2). Such templates divide requirements into predefined parts, thus ensuring a uniform structure with improved clarity.

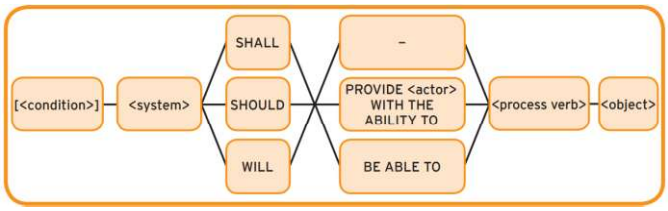

Fig. 2. *FunctionalMASTER* sentence template [16]

Second, the LLMs can also transform requirements into a machine-readable format, using special markup for entities.

In summary, the tool supports the following LLM-driven tasks:

- Quality assessment as quality score on a percentage scale
- Automatic requirement reformulation
  - … using a sentence template
  - … into a machine-readable format

These functionalities are achieved using system prompts—predefined instructions that guide LLMs to perform specific tasks [17], such as assessing requirement quality on a numerical scale.

3. Implementation of the approach

Following the conceptualization, the software prototype has been implemented in Python. Python's prominence in NLP [18] and AI [19] makes it well-suited for this project. As described in section 2, the software tool is split into a frontend and a backend; the implementation of each is described in the following sections.

3.1. Frontend

The frontend of the software tool is implemented using the *Custom Tkinter* toolkit [20]. This allows the creation of a modern, user friendly graphical user interface (GUI). Fig. 3 shows the main window.

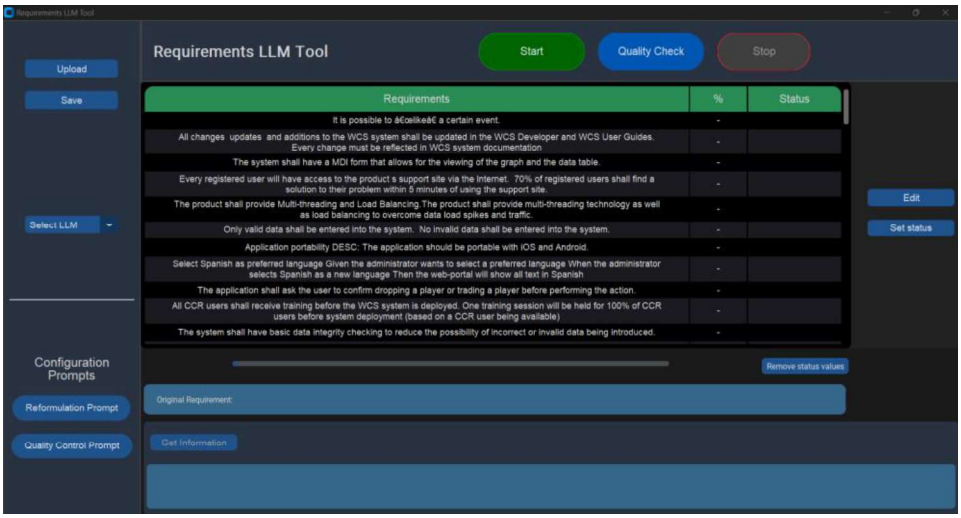

Fig. 3. User interface of the software tool

The tool provides users with an intuitive interface to efficiently handle requirement data. The sidebar on the left offers options for importing and saving requirement files, as well as LLM-specific settings. This also allows users to configure custom prompts (also see section 2) concerning the quality assessment and reformulation of the requirements.

At the core of the GUI is a table that displays all loaded requirements, allowing users to review, edit, or mark them for processing with LLMs. Each requirement can be further examined by clicking on a row, which reveals the original text below the table, ensuring transparency during modifications.

Additionally, the tool offers a feature for assessing the quality of requirements based on predefined criteria. The quality evaluation is displayed both as a percentage-based quality score, and optionally as a detailed text analysis, giving further insights.

Based on that, the requirements can be marked for automatic reformulation, which can be executed pressing the green “Start” button at the top.

3.2. Backend

The backend architecture is designed to support several core functionalities, such as reading requirement data, analyzing and reformulating requirements, enhancing their quality through LLMs.

The system uses the standardized Requirements Interchange Format (ReqIF) [21] for efficient data handling, simplifying exchange and processing. For the handling of the ReqIF files, the tool uses a dedicated Python library [22].

LLMs can be integrated via APIs from platforms like *OpenAI*, *OpenRouter*, and *Together AI* [17], [23], [24]. Also, local LLMs can be used in combination with tools like *LM Studio* or *Ollama* [25], [26]. ensuring a versatile and scalable architecture. These APIs rely on the REST architecture to facilitate communication between clients and servers, transmitting data in the JSON format.

3.3. Creation of the prompts

One important aspect of the tool concerns the system prompts that are used to instruct the LLMs for the specific actions (also see section 2). These are described in the following paragraphs.

3.3.1. Creation of the prompt for percentage quality evaluation

To assess the quality of requirements as a percentage scaled quality score, a system prompt with several criteria is used. These criteria include correctness, unambiguity, verifiability, modifiability, clarity, conciseness, and the avoidance of weak words, all of which can be evaluated without contextual information. This approach aligns with the research by Krishna et al. [14], who established similar evaluation criteria, supporting their efficacy for quality control. The requirement tool evaluates each requirement in isolation, providing a percentage score under any circumstances, ensuring a consistent and objective assessment of quality. See A.1. for the full system prompt.

3.3.2. Creation of prompts for requirement reformulation

The following two sections describe the prompts creation for the requirement reformulation.

3.3.2.1. Sentence template approach

The system prompt utilizes a sentence template approach to reformulate requirements in a precise and predefined manner. The *FunctionalMASTER* template by *SOPHIST* [16] (also see Fig. 2) is used. It is important to note that this only works for functional requirements. This approach guarantees clarity and precision, with an example provided to assist the LLM in

maintaining the original sentence elements without adding new information. A.2. contains the full system prompt.

3.3.2.2. Machine-readable format

For reformulating requirements into a machine-readable format, the system prompt contains custom guidelines. These guidelines specify formatting for entities, properties, and activities, using curly braces, square brackets, and angle brackets, respectively (see A.3. for the full system prompt). This structuring allows for accurate machine interpretation and processing, ensuring that requirements are reformatted in a way that machines can interpret and process effectively.

4. Evaluation and discussion

This evaluation examines LLMs' effectiveness in enhancing product requirements quality through a structured testing strategy. It involves screening ten LLMs with a template-based prompt, followed by a detailed evaluation of the top three models. The analysis identifies strengths and limitations, offering insights for future research.

4.1. Evaluation dataset

For the evaluation, a dataset curated by Shukla [27] has been utilized, consisting of 997 natural language requirements, both functional and non-functional, specifically focused on software projects. The methodology follows a structured selection and preprocessing approach to ensure repeatability. Given the limited availability of non-software-related requirement datasets, the decision was made to concentrate on software requirements for this study.

From the Shukla dataset, a random sample of 100 requirements was extracted to form the final evaluation dataset. This selection aimed to provide a representative yet manageable subset for analysis. Despite contained inconsistencies, the dataset provides a foundation to investigate LLMs' capabilities in reformulating and enhancing requirement clarity.

4.2. Selecting the LLMs

Ten LLMs were selected for preliminary evaluation based on *LMSYS* and *OpenRouter* rankings [5], [28], focusing on performance, popularity, and cost-efficiency. The selected models were: *GPT-4o*, *GPT-3.5 Turbo*, *Llama 3 8B Instruct*, *Claude 3.5 Sonnet*, *Gemini Pro 1.5*, *Gemma 2 9B*, *Yi Large*, *Mistral 7B Instruct*, *WizardLM-2 8x22B*, and *Qwen 2 72B Instruct*. These models were assessed using a sentence template prompt applied to three functional requirements of varying complexity. *GPT-4o*, *Gemma 2 9B*, and *Qwen 2 72B Instruct* excelled in precision, with *Gemma 2 9B* notable for its high score despite fewer parameters. These three were chosen for further evaluation due to their consistent performance.

4.3. Main evaluation

The main evaluation involves a comprehensive analysis of the three selected models: *Gemma 2 9B*, *Qwen 2 72B Instruct*, and *GPT-4o*. The focus is on the quality assessment of requirements on a percentage scale, as well as their reformulation using a sentence template approach, followed by transforming these requirements into a machine-readable format.

4.3.1. Quality assessment

To evaluate the quality assessment capabilities of the different LLMs, a straightforward approach is to compare their results directly. Table 1 presents the average quality ratings assigned by each model.

Table 1. Comparison of the different LLMs for assessing the quality of requirements

LLM	Average quality rating
Gemma 2 9B	74,33 %
Qwen 2 72B Instruct	78,25 %
GPT-4o	69,55 %

The results indicate that the LLMs produce relatively close average quality ratings.

To further evaluate the models, their assessments were compared to a manual evaluation of the first 20 requirements from the dataset. Table 2 shows the comparison between the LLM-generated ratings and human assessments.

Table 2. Comparison of the quality assessment of LLMs with human results

Assessment method	Average quality rating (first 20)	Average difference to manual assessment (first 20)
Gemma 2 9B	73,5 %	19,15 %
Qwen 2 72B Instruct	74,25 %	14,08 %
GPT-4o	69,5 %	13,75 %
Manual	66,65 %	-

As shown, the LLM assessments tend to be higher than the manual ratings on average. Notably, *GPT-4o* exhibits the closest alignment with human evaluation on average.

While the average quality rating provides a general overview, the differences between individual assessments are more telling. By summing and averaging these individual differences, it becomes evident that *GPT-4o* and *Qwen 2 72B Instruct* achieve more consistent alignment with manual evaluations, whereas *Gemma 2 9B* demonstrates greater variability.

4.3.2. Reformulating requirements

Following, the evaluation results for the requirement reformulation are presented.

4.3.2.1. Sentence template

The evaluation used a sentence template to reformulate 46 functional requirements from the dataset, as it supports only functional types. GPT-4o excelled, maintaining clarity and consistency by simplifying longer requirements while retaining essential details. Qwen 2 72B Instruct performed well on shorter requirements but struggled with complexity, often adding unnecessary phrases. Gemma 2 9B simplified lengthy requirements but sometimes added irrelevant conditions. Overall, GPT-4o was the most reliable for handling complex requirements.

4.3.2.2. Machine-readable format

In converting requirements to a machine-readable format, the three LLMs showed varied effectiveness. Gemma 2 9B compressed requirements into concise expressions but often lost critical details and misused parentheses, requiring manual corrections. Qwen 2 72B Instruct had inconsistent results, with multiple formulations and bracket errors. GPT-4o maintained structure with fewer errors, needing only minor adjustments. Despite these issues, GPT-4o was the most promising model, emphasizing the need for ongoing refinement and manual oversight to ensure accurate machine-readable outputs.

4.4. Discussion

The evaluation highlights *GPT-4o*'s effectiveness in aligning with human assessments for requirement quality, marking it as a valuable component of the proposed tool. This tool, which integrates LLMs into existing workflows, demonstrates significant potential to enhance requirements engineering by improving clarity and precision.

Despite its strengths, the tool's reliance on LLMs like *GPT-4o* reveals challenges, such as errors in machine-readable formats. These issues underscore the need for ongoing refinement in both LLM capabilities and tool integration to ensure reliable outputs. The comparative performance of models like *Qwen 2 72B Instruct* and *Gemma 2 9B* suggests that while LLMs can automate parts of the requirements process, human oversight remains crucial, particularly for complex and nuanced tasks.

Overall, the software tool offers a promising approach to augmenting requirements engineering processes, with implications for increased efficiency and accuracy. Continued enhancements in LLM integration and prompt design will be essential to fully leverage these capabilities within engineering workflows. The tool’s modular architecture, support for multiple LLMs, and cloud-based APIs provide inherent scalability, ensuring adaptability to future AI developments.

5. Conclusion and outlook

This paper proposed the application of Large Language Models (LLMs) for the requirements engineering process. The goal was to assess the capability of different LLMs for improving textual product requirements. For this task, a software tool has been developed that integrates in the existing workflows. It can be used to assess the quality of product requirements on a numerical basis and reformulate them based on two different structures. A comparative evaluation of different LLMs showed *GPT-4o* outperforming other models, closely approximating human performance in quality assessment. Overall, the potential of LLMs for the tasks could be demonstrated.

Future research may include the extension of the proposed tool with more functionality and templates to reformulate requirements. Also, the architecture and overall robustness of the tool can be further improved. Furthermore, as the field of LLMs progresses quickly, future models and improvements should be closely monitored and evaluated.

Further evaluation and validation of the approach and software prototype in real-world applications and workflows, particularly in industrial contexts such as mechanical and systems engineering, will be beneficial. This also should explore handling evolving requirement dependencies beyond isolated processing. Nonetheless, the software tool has reached a maturity level where it can provide tangible benefits and should be introduced to engineers in practice.

Acknowledgements

The authors would like to thank Mr. Fatih Kutlu for his support with the software development and evaluation.

Appendix A. Used system prompts for the LLMs

Following, the used system prompts of the LLMs for the different scenarios are appended.

A.1. System prompt to give a numerical quality assessment of requirements

```
Your task is to provide a percentage that indicates whether the requirement is well written or not, based on the following criteria:

• Correctness: The requirement must be factually correct and technically verifiable.
• Unambiguity: The requirement should allow only one interpretation.
• Verifiability: The requirement must contain concrete and measurable statements to verify if it has been met.
• Modifiability: The requirement should be easily modifiable without needing extensive changes.
• Clarity: The requirement should be clear and understandable.
• Conciseness: The requirement should be as short as possible but as long as necessary.
• Avoidance of "Weak Words": The requirement should not contain weak or vague words.

Provide a number as a result under any circumstances.
```

A.2. System prompt to reformulate requirements based on a sentence template

```
Reformulate the requirements into complete sentences in a prose-like manner while following this structure, using only the elements present in the original requirement. Do not add any new information that is not already included in the original requirement:

1. Conditional clause (only if the original requirement specifies a condition).
2. The "system word" (the subject in the sentence). Keep the system word unchanged.
3. Use a modal verb ("shall," "should," "will"). If one is already in the sentence, keep it unchanged.
4. Capability (e.g., "provide <actor> with the ability to," "be able to," or any other text as required).
5. Process verb (if mentioned in the original requirement).
6. Object (if mentioned in the original requirement).

Example: "If the user initiates a measurement within limits, the MRI system shall provide the user with the ability to start the measurement."
```

A.3. System prompt to reformulate requirements into a machine-readable format

```
Your task is to convert the requirement into a specific machine-readable format using the following rules:

• Use curly braces {} for entities (objects and system words).
• Use square brackets [] for properties, numerical values, and units, separating units with a vertical bar |.
• Use angle brackets <> for the process verb (activity).
• Do not put anything else in brackets.
Definitions:
• Entities: These are typically the main nouns in the sentence, such as physical items, components, and defined system elements.
• Properties and Numerical Values: These are adjectives or numerical descriptors that provide additional information about the entities, including numerical values and their units.
• Process Verbs (Activities): These are verbs that describe actions or processes involving the entities.

Examples:
1. "If the {traffic light} [state] = [green], the {car} shall <drive>"
2. "The {Stange_Schwungmassen_Pleuel_catpart} shall have a [weight] >= [500|kg]"
3. "The {Pleuelstange_catpart} shall have a [max_x_length] <= [45]"
```

References

[1] A. Vaswani *et al.*, “Attention Is All You Need,” Jun. 2017. [Online]. Available: http://arxiv.org/pdf/1706.03762v7

[2] S. Minaee *et al.*, “Large Language Models: A Survey,” Feb. 2024.

[3] Meta, “Introducing LLaMA: A foundational, 65-billion-parameter large language model.” [Online]. Available: https://ai.meta.com/blog/large-language-model-llama-meta-ai/

[4] E. Beeching *et al.*, “Open LLM Leaderboard - a Hugging Face Space by open-llm-leaderboard.” [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

[5] LMSYS, “LMSYS Chatbot Arena Leaderboard.” [Online]. Available: https://arena.lmsys.org/

[6] M. Borg, “Requirements Engineering and Large Language Models: Insights From a Panel,” *IEEE Softw.*, Jan. 2024.

[7] X. Amatriain, “Prompt Design and Engineering: Introduction and Advanced Methods,” Jan. 2024.

[8] S. Wang, Z. Wei, Y. Choi, and X. Ren, “Can LLMs Reason with Rules? Logic Scaffolding for Stress-Testing and Improving LLMs,” Jun. 21, 2024, *arXiv*: arXiv:2402.11442. Accessed: Nov. 15, 2024. [Online]. Available: http://arxiv.org/abs/2402.11442

[9] S. Peckham, J. Day, DianaHbr, and Microsoft Corporation, “Understanding strengths and weaknesses of Large Language Models.” [Online]. Available: https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/working-with-llms/use-case-recommend

[10] T. Rahman and Y. Zhu, “Automated User Story Generation with Test Case Specification Using Large Language Model,” Apr. 2024.

[11] A. Vogelsang and J. Fischbach, “Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline,” Feb. 2024. [Online]. Available: http://arxiv.org/pdf/2402.13823v3

[12] A. Tikayat Ray, B. F. Cole, O. J. Pinon Fischer, A. P. Bhat, R. T. White, and D. N. Mavris, “Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models,” *Systems*, vol. 11, no. 7, Jan. 2023.

[13] K. Ronanki, C. Berger, and J. Horkoff, “Investigating ChatGPT’s Potential to Assist in Requirements Elicitation Processes,” Jul. 2023.

[14] M. Krishna, B. Gaur, A. Verma, and P. Jalote, “Using LLMs in Software Requirements Specifications: An Empirical Evaluation,” Apr. 2024. [Online]. Available: http://arxiv.org/pdf/2404.17842v1

[15] Z. Wang, J. Li, G. Li, and Z. Jin, “ChatCoder: Chat-based Refine Requirement Improves LLMs’ Code Generation,” Nov. 2023. [Online]. Available: http://arxiv.org/pdf/2311.00272v1

[16] SOPHIST GmbH and SOPHIST GmbH, “MASTeR – Schablonen für alle Fälle.” [Online]. Available: https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf

[17] OpenAI, “OpenAI Platform.” [Online]. Available: https://platform.openai.com/docs/api-reference/making-requests

[18] D. A. Dahl, *Natural language understanding with Python: Combine natural language technology, deep learning, and large language models to create human-like language comprehension in computer systems*, 1st ed. Birmingham; Mumbai: Packt Publishing, 2023.

[19] S. Omari, K. Basnet, and M. Wardat, “Investigating large language models capabilities for automatic code repair in Python,” *Clust. Comput.*, Jan. 2024.

[20] T. Schimansky and GitHub, “CustomTkinter: A modern and customizable python UI-library based on Tkinter.” [Online]. Available: https://github.com/TomSchimansky/CustomTkinter

[21] C. Ebert and M. Jastram, “ReqIF: Seamless Requirements Interchange Format between Business Partners,” *IEEE Softw.*, vol. 29, no. 5, Jan. 2012.

[22] stanislaw and GitHub, “Python library for ReqIF format. ReqIF parsing and unparsing.” [Online]. Available: https://github.com/strictdoc-project/reqif

[23] OpenRouter, “Models.” [Online]. Available: https://openrouter.ai/docs/models

[24] Together AI, “Code/Language Models.” [Online]. Available: https://docs.together.ai/docs/language-and-code-models

[25] LM Studio, “LM Studio: Discover and run local LLMs.” [Online]. Available: https://lmstudio.ai/

[26] Ollama, “Ollama.” [Online]. Available: https://ollama.com/

[27] V. Shukla and Kaggle, “Software requirements dataset.” [Online]. Available: https://www.kaggle.com/datasets/iamvaibhav100/software-requirements-dataset

[28] OpenRouter, “LLM Rankings.” [Online]. Available: https://openrouter.ai/rankings