REQINONE: A Large Language Model-Based Agent for Software Requirements Specification Generation

Taohong Zhu*, Lucas C. Cordeiro*, Youcheng Sun†
*Department of Computer Science, The University of Manchester, Manchester, UK
{taohong.zhu, lucas.cordeiro}@manchester.ac.uk
†Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE
youcheng.sun@mbzuai.ac.ae

***Abstract*—Software Requirements Specification (SRS) is one of the most important documents in software projects, but writing it manually is time-consuming and often leads to ambiguity. Existing automated methods rely heavily on manual analysis, while recent Large Language Model (LLM)-based approaches suffer from hallucinations and limited controllability. In this paper, we propose REQINONE, an LLM-based agent that follows the common steps taken by human requirements engineers when writing an SRS to convert natural language into a structured SRS. REQINONE adopts a modular architecture by decomposing SRS generation into three tasks: summary, requirement extraction, and requirement classification, each supported by tailored prompt templates to improve the quality and consistency of LLM outputs.**

We evaluate REQINONE using GPT-4o, LLaMA 3, and DeepSeek-R1, and compare the generated SRSs against those produced by the holistic GPT-4-based method from prior work as well as by entry-level requirements engineers. Expert evaluations show that REQINONE produces more accurate and well-structured SRS documents. The performance advantage of REQINONE benefits from its modular design, and experimental results further demonstrate that its requirement classification component achieves comparable or even better results than the state-of-the-art requirement classification model.

***Index Terms*—Requirements Engineering, Software Requirements Specification, Large Language Models**

I. INTRODUCTION

Requirements engineering is a critical phase in software development, ensuring stakeholder needs are accurately captured and translated into implementable specifications. The Software Requirements Specification (SRS) is the primary outcome of requirements engineering, which defines the expected functionality, constraints, and operational environment of a software system [1]. A high-quality SRS must be unambiguous, complete, consistent, and traceable to guide design, implementation, and testing [2]. However, producing such specifications is challenging due to the reliance on natural language, which often leads to vagueness, contradictions, and increased communication overhead between stakeholders [3]. Tools like Visual Paradigm [4], ReqView [5], and Elementool [6] offer templates and diagrams but still require extensive manual input. NLSSRE [7] automates requirement extraction but does not support generating complete SRS content such as use cases and glossaries.

Recent advances in Large Language Models (LLMs) offer opportunities to automate and enhance requirements engineering tasks. Trained on vast real-world data, LLMs can generate human-like text with billions of parameters [8]–[10]. With the emergence of models such as LLaMA [11], GPT [12], and DeepSeek [13], techniques like zero-shot [14], few-shot [15], and chain-of-thought [16] prompting have significantly improved model performance across diverse tasks. In the context of requirements engineering, LLMs have been applied to classify requirements [17], [18], evaluate user story quality [19], assess completeness [20], and support information extraction and architectural design [21]–[24]. In [25], LLMs are employed to translate informal code comments into formal postcondition assertions. The framework in [26] leverages ChatGPT and counterexample-guided refinement to automatically extract and verify formal postconditions for Ethereum smart contract functions based on natural language descriptions. [27] designed a prompt to enable LLMs to directly generate a full SRS from natural language text. However, an SRS includes multiple sections, the task involves not only converting natural language into well-structured requirements but also correctly placing each one into the appropriate section. This makes SRS generation task more complex. Directly prompting an LLM to generate the full SRS can lead to hallucinations, underscoring the need for a suitable transformation strategy and well-crafted prompts [28], [29].

This paper introduces REQINONE, an LLM-based agent designed to automatically convert natural language texts, such as stakeholder requirements, meeting transcripts, and conversational records, into a structured SRS. REQINONE consists of three core components: Summary Task, Requirement Extraction Task, and Requirement Classification Task, each guided by a tailored prompt template to perform its specific role in the SRS generation process. By coordinating these components, REQINONE efficiently produces well-structured SRS documents from unstructured text.

Furthermore, we construct ReqFromSRS, a dataset consisting of 100 functional requirements and 100 non-functional requirements, manually extracted from 22 real-world SRS documents in the PURE dataset [30]. We release all prompt templates, code, datasets, generated SRSs, and experimental results in Github [31] to support future research.

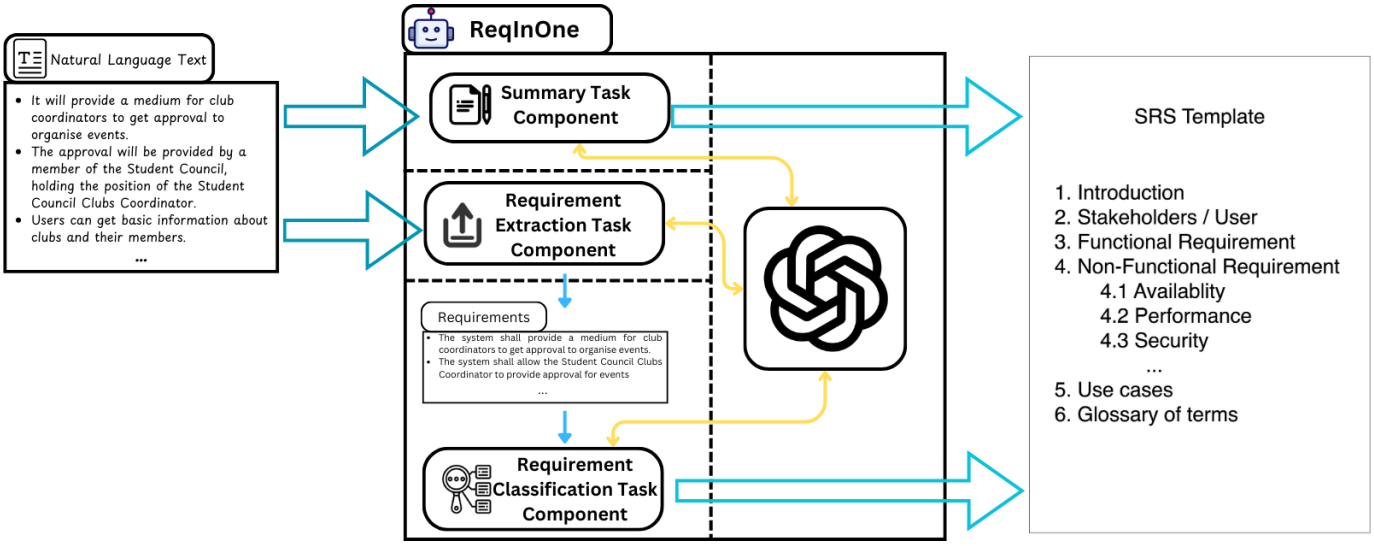

Fig. 1. Overview of the REQINONE

II. METHODOLOGY

A. *Overview of* REQINONE

Prior studies have shown that LLMs may struggle to effectively handle complex tasks when prompted to complete them in a single step. However, when guided by a chain-of-thought approach [16], such tasks can often be decomposed into a series of simpler sub-tasks, enabling more reliable and accurate performance. For instance, Tian et al. demonstrated that LLMs are less effective at directly repairing buggy code compared to scenarios where the error location is first identified by either human annotators or external tools, and the LLM is then responsible solely for repairing the localized snippet [32]. Inspired by this finding, we hypothesize that the task of converting natural language into a structured SRS, rather than being executed as a single-step transformation as in the work of Krishna et al. [27], can similarly benefit from decomposition into a set of simpler, more focused sub-tasks. By adopting this multi-step strategy, we aim to improve LLM performance and generate higher-quality SRS outputs.

Building on this hypothesis, we design REQINONE to follow the common steps taken by human requirements engineers when converting natural language text into a structured SRS. Rather than treating the conversion as a single-step process, REQINONE decomposes the task into three sub-tasks: Summary, Requirement Extraction, and Requirement Classification. To efficiently handle these sub-tasks, REQINONE consists of three specialized components, each responsible for executing a specific task. Through task scheduling and coordination among these components, REQINONE processes natural language input and generates a well-structured SRS.

The overall workflow of REQINONE is illustrated in Figure 1. It begins with an input of verbose natural language text, typically reflecting stakeholder needs and high-level system requirements. REQINONE first invokes the Summary Task component, which processes the input text for summarization. The Summary Task component prompts the LLM based on the natural language text and utilizes the output of the LLM to populate the SRS template. The generated content is used to populate specific sections of the SRS template. As an example of the SRS template shown in Figure 1, the Summary Task component is responsible for filling in summary-type sections of the SRS template, including the Introduction, Stakeholders/Users, Use Cases, and Glossary sections. Additionally, the SRS template illustrated in Figure 1 is merely an example, adapted from the template used in [27], which in turn is based on IEEE specifications [33]. Users are free to adopt any other SRS template that suits their specific needs. To do so, they simply need to identify the summary-type sections within their chosen template and accordingly modify the prompt template used in the Summary Task component (details in II-B).

Following this, REQINONE proceeds to the Requirement Extraction Task component, which extracts structured requirements from the natural language text. The Requirement Extraction Task component prompts the LLM based on the natural language text, and the output of the LLM constructs a list of structured requirements extracted from natural language text. This list of extracted requirements is then passed as input to the Requirement Classification Task component for further processing.

Once the list of extracted requirements is generated, it is passed to the Requirement Classification Task component for categorization. This component is responsible for classifying the requirements into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). Moreover, the Requirement Classification Task component further refines the classification of NFRs by assigning them to specific subtypes such as availability, performance, security, and other relevant NFR subtypes. The Requirement Classification Task component prompts the LLM to classify requirements and determines their category based on the output of the LLM.

The classified requirements are then populated into the corresponding sections of the SRS template. FRs are placed in the FRs section, while NFRs are categorized into subsections under NFRs section. Once all three components complete their respective tasks, the conversion of natural language input into a structured SRS is finalized. Through this coordinated execution of its three core components, REQINONE provides an efficient and automated solution for generating SRS documents.

B. Summary Task Component

To ensure that the Summary Task Component can generate the required summary-type sections accurately after prompting the LLM, we designed a prompt template specifically for the Summary Task. As shown in Figure 2, this prompt template consists of two main parts.

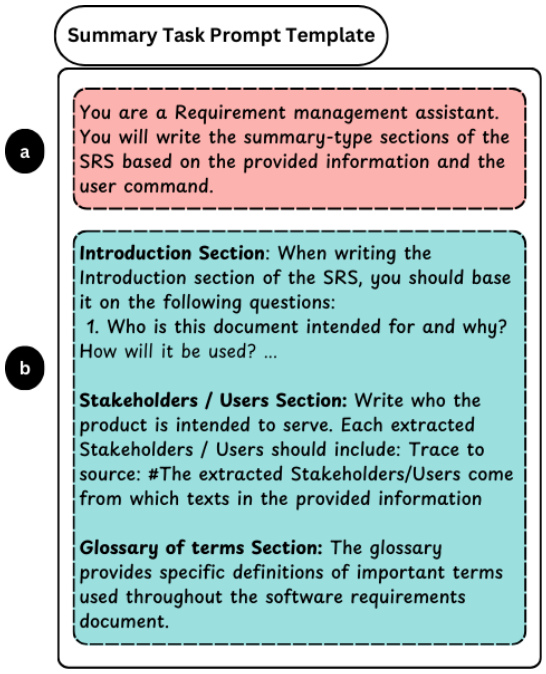

Fig. 2. Prompt Template for Summary Task

The first part involves Role Specification (Figure 2a), which instructs the LLM to assume the role of a requirement assistant responsible for generating content for various summary-type sections based on the provided natural language text.

The second part of the prompt template explicitly lists the sections for which content needs to be generated, along with detailed descriptions or definitions of the expected content for each section (Figure 2b). For example, we provide the description for the Stakeholders/Users Section: "Write who the product is intended to serve". Additionally, to mitigate hallucinations in the output of LLM, we specify that each stakeholder or user mentioned must be accompanied by an annotation indicating the text source from the provided natural language input. Similarly, the Glossary of Terms Section defines its content as: "The glossary provides specific definitions of important terms used throughout the software requirements document".

Furthermore, for the second part of the prompt template, in addition to providing explicit definitions for each section, researchers have found that including guiding questions can further help guide the LLM to produce more relevant and structured content—for example, as illustrated in Figure 2b for the Introduction section. If users find that providing only definitions does not yield satisfactory outputs, they may instead design guiding questions based on the expected content, enabling the LLM to generate results that better align with their intentions.

The Summary Task Component also includes a command list containing entries such as "Write Introduction Section" and "Write Stakeholders/Users Section." The Summary Task Component iterates through this command list, sequentially extracting commands and combining them with the prompt template before prompting the LLM. This process allows the LLM to focus on generating one section at a time, and the component populates each section into the SRS template accordingly.

Both the prompt template and the command list are designed to be extensible. If a required summary-type section is not initially included in the prompt template, users can extend the second part of the prompt template by adding relevant instructions in the same format. Likewise, by adding a corresponding command to the command list. This flexibility ensures that REQINONE remains adaptable to different SRS template structures and evolving requirements engineering needs.

C. Requirement Extraction Task Component

Similar to the Summary Task Component, we designed a specialized prompt template for the Requirement Extraction Task Component to facilitate prompting the LLM. The structure of this prompt template is illustrated in Figure 3 and consists of four main parts.

The first part is still Role Specification (Figure 3a), where the LLM is instructed to analyze the provided natural language text and extract relevant requirements.

The second part is Requirement Definition (Figure 3b), which provides the LLM with a clear definition of what a requirement is. By explicitly defining requirements within the prompt, the LLM gains a precise understanding of the target content it needs to extract from the natural language text, thereby improving extraction accuracy.

The third part is Requirements Pattern (Figure 3c). In this part, the LLM is directed to extract requirements that conform to the following standardized format: "The <subject clause> shall <action verb clause> <object clause> <optional qualifying clause>, when <condition clause>." This format is derived from INCOSE documentation [34] and serves as a guideline for the LLM, ensuring that all extracted requirements are expressed in a consistent and well-structured format. The purpose of enforcing this structured format is to promote clarity, consistency, and testability in requirement expressions.

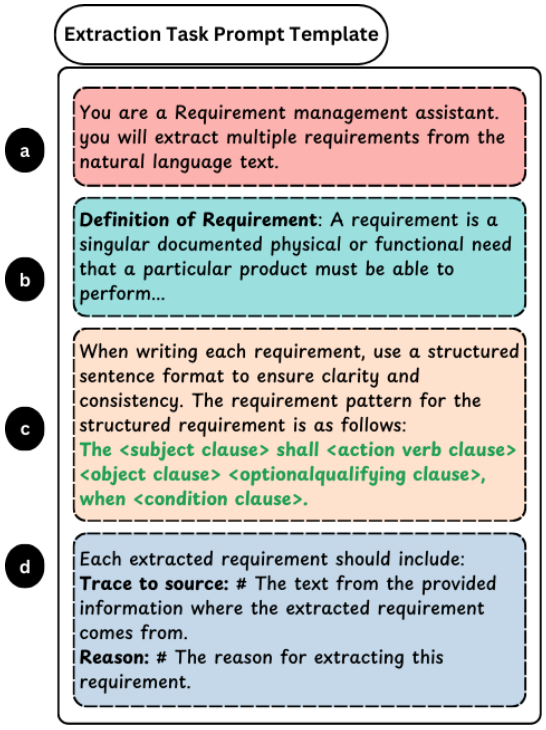

Fig. 3. Prompt Template for Requirement Extraction Task

Additionally, structured format phrasing helps reduce ambiguity, making the requirements easier to understand, validate, and trace throughout the software development lifecycle [35].

The fourth part includes Trace to Source (Figure 3ⓓ), where we instruct the LLM to append the source of each extracted requirement along with the reason for its extraction. This step is crucial in preventing hallucinations by compelling the LLM to justify the rationality of each requirement extraction based on trace to source before generating it. By linking each requirement to its original text source, we establish traceability within the SRS, ensuring that every requirement can be verified and traced back to its origin.

Once the Requirement Extraction Task Component provides the natural language text and the structured prompt template to the LLM, the LLM processes the input, analyzes the content, and returns a structured list of extracted requirements. The Requirement Extraction Task Component then organizes these extracted requirements into a requirements list, which is passed to the Requirement Classification Task Component for further classification and refinement.

D. Requirement Classification Task Component

In line with the design of the other components, we designed a specialized prompt template for the Requirement Classification Task Component. Upon receiving the requirement list from the Requirement Extraction Task Component, the Requirement Classification Task Component utilizes this prompt template to prompt the LLM. This prompt template transforms the LLM into a reasoning model specialized in classifying requirements into functional and non-functional categories.

As illustrated in Figure 4, the prompt template consists of four main parts. The first part is Task Specification (Figure 4ⓐ), which explicitly instructs the LLM to classify each requirement into either functional or non-functional categories. The second part is Definition of FRs and NFRs (Figure 4ⓑ), providing a clear classification standard by defining FRs and NFRs to help the LLM distinguish between these categories.

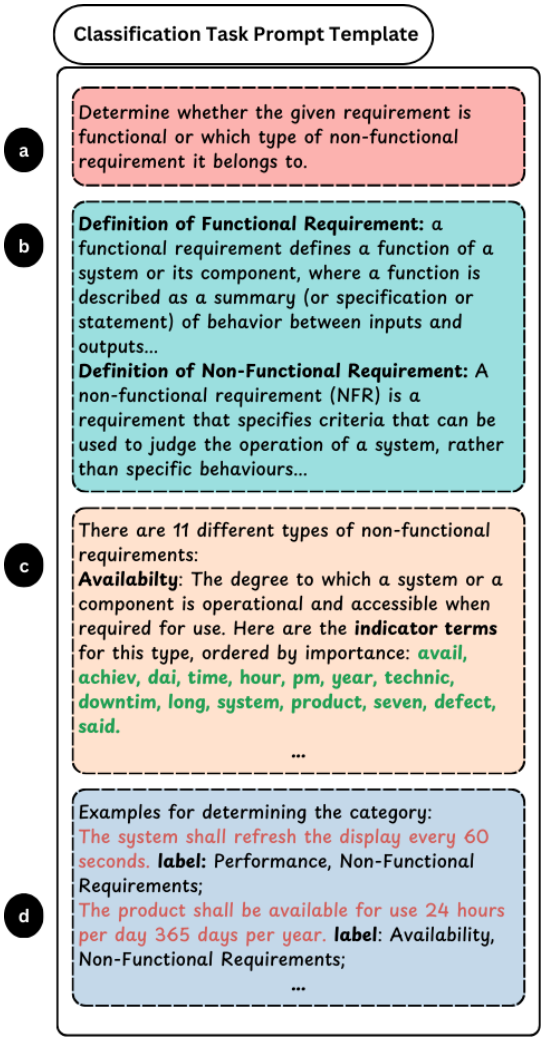

Fig. 4. Prompt Template for Requirement Classification Task

The third part, Detailed Classification of NFRs, explains various subtypes within NFRs (Figure 4ⓒ). This part defines 11 different subtypes of NFRs, such as Availability Requirement, Legal Requirement, and Maintainability Requirement, each accompanied by a corresponding definition. Additionally, Cleland-Huang et al. [36] proposed that different NFR types are often associated with specific keywords, which we refer to as indicator terms. These indicator terms are incorporated into the prompt template following the corresponding requirement definitions, helping guide the LLM toward more accurate classification.

In the fourth part (Figure 4ⓓ), we adopt the few-shot learning approach [15], where a set of labeled requirement

examples is provided to enhance classification accuracy. These examples cover all 11 NFR subtypes and also include representative samples of FRs, ensuring that the LLM learns from diverse cases to improve its classification ability.

Considering that there is still no consensus in the software engineering community on the concept of NFRs [37], [38], the categorization of NFRs may vary across different projects. NFRs in some cases may extend beyond the 11 NFR subtypes in the third part of the prompt template. To address this challenge, the prompt template is designed to be extensible. Users can customize the template by adding, modifying, or removing requirement category definitions based on their specific project needs or the adopted SRS template. Additionally, users can expand the few-shot learning examples by introducing new labeled requirements that align with the format used in the prompt template.

Once the LLM classifies all requirements in the list, an appropriate category label will be appended to each requirement. If a requirement is classified as NF, the label specifies which subtype of NFR it belongs to. The labeled requirements are then returned to the Requirement Classification Task Component, which organizes and populates them into the FRs Section or NFRs Section of the SRS template accordingly.

E. Summary and Insights

Unlike previous studies that typically instruct LLMs to generate the entire SRS in a single step, we introduces a novel strategy that decomposes the SRS generation task into three comparatively simpler sub-tasks: the Summary Task, the Requirement Extraction Task, and the Requirement Classification Task. Based on this strategy, we developed REQINONE, which composed of three corresponding components—each responsible for one sub-tasks. Every component is guided by a designed prompt template, tailored specifically for its corresponding task, and we adopt a zero-shot prompting approach to design these prompt templates [14].

A key advantage of our approach lies in its high degree of customizability. Users can freely adapt REQINONE to their preferred SRS format by modifying the contents of each prompt template. For instance, if a chosen SRS template includes a summary-type section not originally present in the Summary Task prompt template, the user can easily add the necessary description in the appropriate part of the prompt template. Conversely, if certain sections included in the original prompt template are irrelevant to the chosen SRS template, they can simply be removed.

Moreover, the use of prompt templates brings sustainability and extensibility to REQINONE. Users can continuously improve the output quality by adjusting the content of the prompt templates—for example, by modifying the definition of requirements in the requirement extraction prompt template, or by inserting more representative examples into the example part of the requirement classification prompt template. These kinds of adjustments allow REQINONE to be iteratively optimized over time, making its performance increasingly aligned with user expectations and domain-specific needs.

III. EVALUATION

Given that our approach decomposes the process of generating an SRS into multiple subtasks, we focus on several key aspects when evaluating REQINONE: the overall quality of the generated SRS, the quality of the requirements within the SRS, and whether the requirements are correctly categorized into their appropriate sections. To this end, we propose the following three research questions:

RQ1: How does the overall quality of SRSs generated by REQINONE using different LLMs compare to SRSs produced by existing automated SRS generation methods and those written by entry-level requirements engineers?

RQ2: How does the quality of requirements generated by REQINONE compare to those from existing automated methods and entry-level engineers?

RQ3: How well does REQINONE perform in the requirement classification?

A. Evaluation Design: User Study and Classification Task

RQ1 and RQ2 were addressed through a survey-based evaluation using a questionnaire, while RQ3 was addressed via a classification task using benchmark datasets.

The questionnaire consisted of two parts. **Part 1, addressing RQ1,** included five evaluation parameters drawn from prior literature [27], [39], [40]: Internal Consistency, Non-redundancy, Completeness, Conciseness, and Traceability. Participants read both the source text and the corresponding SRS before rating each parameter on a 1–5 Likert scale, where 1 indicates strong disagreement and 5 indicates strong agreement that the SRS meets the parameter. The aggregated scores were used to assess the overall quality of each SRS.

Part 2, for RQ2, followed a similar structure. Five requirements were randomly sampled from each SRS (ensuring one per category when possible). Each requirement was rated on five parameters: Unambiguous, Understandable, Correctness, Verifiable, and Conforming—also using a 1–5 Likert scale. As in RQ1, aggregated scores provided a quality assessment for each requirement.

Three software engineering experts participated in the study. Each evaluated five different SRSs: (1) SRS generated by REQINONE using ChatGPT-4o; (2) SRS generated by REQINONE using Llama3 (Version: Meta Llama3.1-8B); (3) SRS generated by REQINONE using DeepSeek-R1 (Version: DeepSeek-R1-0528-Qwen3-8B); (4) SRS generated by baseline, which directly uses GPT-4 to generate the SRS [27]; (5) SRS written by an entry-level requirements engineer [27]. To ensure unbiased evaluations, all participants were unaware of how each SRS was generated and had no prior involvement in this research.

To address RQ3, we evaluated REQINONE's classification component using the PROMISE dataset [41], where the task involved classifying requirements as functional or non-functional and further categorizing NFRs into specific subtypes. To test generalizability, we constructed a new dataset—ReqFromSRS—by manually extracting requirements from the PURE dataset [30]. We compared performance

against the NoRBERT baseline [17], using precision, recall, and F1 score as evaluation metrics.

ReqFromSRS Dataset: The PURE dataset is a collection of 79 SRS documents gathered from the web [30]. To evaluate the performance of REQINONE's requirement classification and its generalizability, we manually extracted 100 FRs and 100 NFRs from the SRSs in the PURE dataset.

- Among the 100 NFRs, there were 10 Usability Requirements (US), 21 Performance Requirements (PE), 24 Security Requirements (SE), 12 Availability Requirements (A), 12 Maintainability Requirements (MN), 7 Portability Requirements (PO), 4 Scalability Requirements (SC), 8 Look & Feel Requirements (LF), and 2 Legal Requirements (L).
- During the manual extraction process, we only selected FRs explicitly stated under the FRs section of the SRS and labeled them with **F**. Similarly, for NFRs, we only extracted those explicitly assigned a NFR subtype within the SRS and labeled them accordingly.

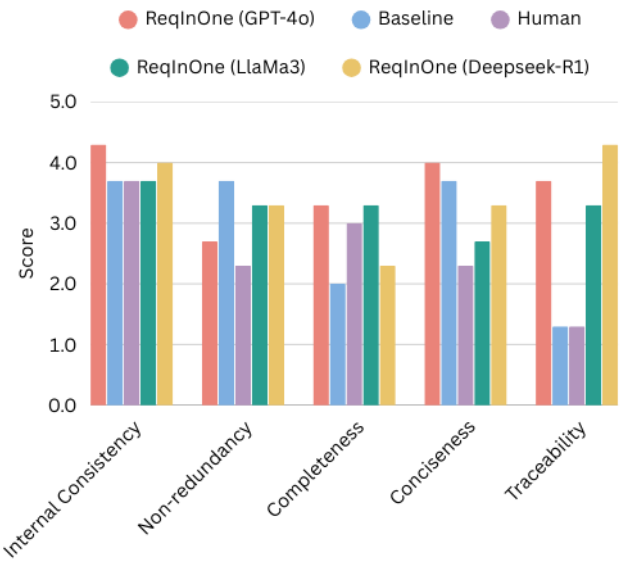

Fig. 5. Overall evaluation of the five SRSs across five quality parameters. Each score represents the average rating from experts.

B. RQ1: Overall Quality of SRSs

As illustrated in Figure 5, the SRS generated by REQINONE using GPT-4o, despite receiving a relatively low score in Non-redundancy, achieved the highest scores in Internal Consistency, Completeness, Conciseness, and Traceability. This indicates that REQINONE (GPT-4o) delivers the best overall SRS quality among all evaluated methods.

Compared with the human-written SRS, REQINONE (GPT-4o) consistently outperformed across all five evaluation parameters. In terms of Internal Consistency and Completeness, the human-written SRS scored 3.8 and 3.0, whereas REQINONE (GPT-4o) achieved 4.2 and 3.2 respectively indicating that REQINONE (GPT-4o) can now generate more logically coherent and coverage-complete specifications than entry-level requirements engineer in many cases.

When compared to the baseline, which used GPT-4, REQINONE (GPT-4o) also showed superior performance in four out of five parameters, especially Traceability, where REQINONE achieved a significantly higher score. This is particularly notable given that GPT-4—the model behind the baseline—is approximately 12 times more expensive to use than GPT-4o. This comparison not only confirms the strong performance of REQINONE, but also highlights its cost-effectiveness, offering better results at a low computational cost. This suggests that our proposed strategy of decomposing the SRS generation into subtasks can effectively enhance the performance of LLMs in SRS generation.

Regarding traceability, both the baseline and human-written SRS did not clearly show traceability, receiving the lowest scores in this parameter. In contrast, all three SRSs generated by REQINONE clearly maintained traceability, with Deepseek-R1 performing best.

Although the REQINONE powered by LLaMA3 and Deepseek-R1 did not achieve the highest overall scores, they did exhibit strengths in specific areas. Both outperformed the GPT-4o regarding Non-redundancy, suggesting that they may demonstrate stronger capability in extracting requirements.

> **Answer to RQ1:** REQINONE (GPT-4o), delivers the highest overall SRS quality among all evaluated parameters, outperforming both the human-written SRS and the baseline, while maintaining low computational cost. Additionally, LLLaMA3 and Deepseek-R1 also showed strengths in Non-redundancy, suggesting potential in requirement extraction.

C. RQ2: Quality of Generated Requirements

Figure 6 presents the evaluation of requirement quality in the generated SRSs. Overall, REQINONE using LLaMA3 produced the highest-quality requirements, while ReqInOne with Deepseek-R1 performed the worst.

Although REQINONE with LLaMA3 scored slightly lower than the baseline in the Unambiguous parameter, the difference was minimal. Both REQINONE (GPT-4o) and REQINONE (LLaMA3) achieved high scores in the Conforming parameter, outperforming the baseline. This improvement comes from using a requirement pattern in the requirement extraction prompt template, which guided LLMs to follow a consistent structure when generating requirements.

Although LLaMA3 performed better than GPT-4o in avoiding ambiguous phrasing and generating more easily understandable requirements, ambiguity remains a common challenge across all LLMs. These models often introduce unnecessary modifiers or redundant sentences, which can lead to vague or overly verbose requirements. Among the models evaluated, GPT-4o appeared to struggle with this issue the most. For instance, some requirements generated by GPT-4o included

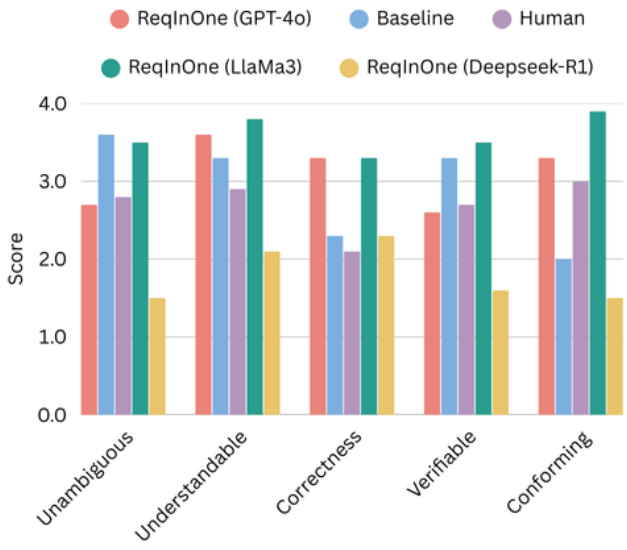

Fig. 6. Evaluation of requirement quality within the generated SRS documents. Each parameter score represents the average rating by experts on five selected requirements from each SRS.

subjective terms such as "appropriate" and "user-friendly", which are highly subjective and can introduce ambiguity. This observation is further supported by the evaluation results shown in Figure 5 (parameter: Non-redundancy) and Figure 6 (parameter: Unambiguous), both of which reflect lower scores for GPT-4o in these aspects. Therefore, future research could explore fine-tuning LLMs to reduce the use of highly subjective terms, which may lead to improved requirement quality.

Regarding Correctness, the baseline fell clearly behind REQINONE (LLaMA3) and REQINONE (GPT-4o), generating more requirements that were not grounded in the source text. This highlights the effectiveness of the Trace to Source part in the requirement extraction prompt template, which helps reduce hallucinations and ensures better alignment with the source content.

Answer to RQ2: REQINONE, particularly with LLaMA3, generated higher-quality and more consistent requirements than both the baseline and human-written SRSs, benefiting from well-designed structured prompt templates that improved clarity, correctness, and conformity.

D. RQ3: Requirement Classification Performance

a) Performance on PROMISE Dataset for FR/NFR Classification: To evaluate the performance of REQINONE 's Requirement Classification Component, we first assess its ability to classify requirements as either functional or non-functional on the PROMISE dataset. As illustrated in Table I, REQINONE using GPT-4o achieves competitive results when compared to the NoRBERT baseline. For FR, GPT-4o achieves an F1 score of 0.90, equal to NoRBERT, with a slightly lower precision (0.87 vs. 0.92) but notably higher recall (0.95 vs. 0.88). For NFR, GPT-4o outperforms the baseline in precision (0.96 vs. 0.92) while achieving the same F1 score of 0.93.

TABLE I
F/NFR CLASSIFICATION RESULTS ON PROMISE DATASET ACROSS DIFFERENT TOOLS

	FR			NFR		
Tool	P	R	F1	P	R	F1
NoRBERT (Baseline)	**.92**	.88	**.90**	.92	**.95**	**.93**
REQINONE (GPT-4o)	.87	**.95**	**.90**	**.96**	.90	**.93**
REQINONE (LlaMa3)	.75	.83	.78	.87	.81	.84
REQINONE (Deepseek-R1)	.76	.86	.80	.89	.81	.85

Meanwhile, REQINONE powered by Llama3 and DeepSeek-R1 also performs reasonably well. Llama3 achieves F1 scores of 0.78 (FR) and 0.84 (NFR), while DeepSeek-R1 yields 0.80 (FR) and 0.85 (NFR). Though they do not reach the level of GPT-4o or NoRBERT, their results suggest the potential of using local LLMs for future research in requirement classification tasks.

b) Performance on PROMISE Dataset for NFR Subtype Classification: To further evaluate the classification capabilities of REQINONE, we focus on the Classification of NFR Subtypes using the PROMISE dataset. This dataset includes 11 NFR subtypes: Availability (A), Fault Tolerance (FT), Legal (L), Look & Feel (LF), Maintainability (MN), Operational (O), Performance (PE), Portability (PO), Scalability (SC), Security (SE), and Usability (US). As shown in Table II, REQINONE using GPT-4o achieves a weighted F1 score of 0.81, which is nearly on par with the NoRBERT baseline score of 0.82. More importantly, GPT-4o surpasses NoRBERT in several individual subtypes, including Availability, Fault Tolerance, Look & Feel, and Maintainability in terms of precision, recall, and F1 score. This demonstrates that REQINONE understands certain non-functional subtypes more than NoRBERT.

The results of LLaMa3 and DeepSeek-R1 also indicate solid performance. Although their weighted F1 scores are lower than GPT-4o and NoRBERT, their results align with the earlier FR and NFR classification task and suggest local models remain viable options for requirement classification.

c) Performance on ReqFromSRS Dataset for FR/NFR Classification: Since our baseline NoRBERT is trained specifically on the PROMISE dataset, we constructed a new dataset—ReqFromSRS—to provide a more fair comparison and to evaluate the generalizability of REQINONE in classifying requirements. We performed the same FR/NFR classification task on this new dataset.

As shown in Table III, REQINONE using GPT-4o significantly outperforms NoRBERT across all evaluation metrics. It achieves the highest precision, recall, and F1 score for both FRs and NFRs, indicating its strong generalization capability to previously unseen data. Even when powered by local models such as LLaMa3 and DeepSeek-R1, REQINONE

TABLE II
CLASSIFICATION OF NFR SUBTYPES ON PROMISE DATASET.

Tool	**NoRBERT**	REQINONE **(GPT-4o)**	REQINONE **(LLaMa3)**	REQINONE **(Deepseek-R1)**
	P / R / F1	**P / R / F1**	**P / R / F1**	**P / R / F1**
A	.80 / .76 / .78	**.84 / 1 / .91**	.30 / .90 / .45	.77 / .95 / .85
FT	.60 / .60 / .60	**.67 / .80** / .73	.62 / .50 / .56	.78 / .70 / **.74**
L	**.91** / .77 / **.83**	.55 / **.85** / .67	.52 / .85 / .65	.60 / .69 /.64
LF	.81 / .79 / .80	**.91 / .84 / .88**	.78 / .76 / .77	.83 / .63 / .72
MN	.62 / .47 / .53	**.69 / .65 / .67**	.69 / .65 / .67	.58 / .65 / .61
O	.78 / **.84 / .81**	**.79** / .53 / .63	.79 / .44 / .56	.54 / .48 / .51
PE	**.92 / .87 / .90**	.87 / .83 / .85	.79 / .83 / .81	.81 / .81 / .81
SC	.76 / **.76 / .76**	.71 / .71 / .71	.73 / .52 / .61	**.79** / .52 / .63
SE	.90 / **.92** / .91	**.98** / .88 / .93	.98 / .74 / .84	.98 / .89 / **.94**
US	.83 / **.88** / .86	**.92** / .82 / **.87**	.90 / .66 / .76	.79 / .79 / .79
Weighted F1	**0.82**	**0.81**	**0.71**	**0.74**

TABLE III
F/NFR CLASSIFICATION RESULTS ON REQFROMSRS DATASET ACROSS DIFFERENT TOOLS

	FR			NFR		
Tool	P	R	F1	P	R	F1
NoRBERT (Baseline)	.84	.45	.59	.63	**.92**	.74
REQINONE (GPT-4o)	**.85**	**.87**	**.86**	**.87**	.85	**.86**
REQINONE (LlaMa3)	.80	.79	.79	.79	.80	.80
REQINONE (Deepseek-R1)	.82	.71	.76	.74	.84	.79

still outperforms NoRBERT. Both local models maintain a balanced performance with F1 scores of 0.76–0.80, surpassing NoRBERT, especially in FR recall, where NoRBERT performs poorly (0.45). Although NoRBERT achieves a relatively high recall (0.92) for NFRs, its NFR precision (0.63) and FR recall (0.45) are substantially lower, indicating that NoRBERT tends to classify most requirements as NFR.

Answer to RQ3: REQINONE demonstrates strong performance comparable to the NoRBERT baseline on the PROMISE dataset and exhibits significantly better generalization on the ReqFromSRS dataset. Even when using local models like LLaMA3 and DeepSeek-R1, REQINONE still offers promising classification performance.

IV. THREATS TO VALIDITY

a) Internal Validity: To minimize randomness in LLM outputs and obtain stable results, we set the temperature of all LLMs to 0. We also specify the exact versions of the LLMs used in the evaluation. These settings reduce the diversity of possible outputs and also enhance the reproducibility of our study. Additionally, when using local models such as LLaMa3 and DeepSeek-R1, we opted for their 8B parameter versions instead of larger alternatives. This decision was made to balance computational feasibility and evaluation time, but it may have limited the performance of these models.

b) Construct Validity: To answer RQ1 and RQ2, all three participants involved in the survey are experts in the field of software engineering. The meaning of each evaluation parameter in the questionnaire was clearly explained to ensure consistency. Nonetheless, human judgment is inherently subjective, and differences in individual understanding may have introduced scoring bias. We attempted to mitigate this by selecting the most representative parameters reported in existing literature to assess SRS and requirement quality.

c) Conclusion Validity: Instead of evaluating every requirement in the SRS, which would have imposed a heavy workload on the participants and possibly affected their judgment, we selected a sample of requirements that were as diverse as possible across different requirement types. This sampling strategy helps ensure coverage while maintaining evaluation quality, but it may still limit the comprehensiveness of our assessment.

V. CONCLUSION

In this paper, we proposed REQINONE, an LLM-based agent designed to automatically SRS by scheduling three tasks: summarization, requirement extraction, and requirement classification. Our evaluation shows that the SRS and individual requirements generated by REQINONE are of higher quality and more compliant with standard SRS guidelines than those produced by baseline methods or entry-level requirements engineers. Additionally, REQINONE achieves high accuracy and strong generalizability in the requirement classification task. These results demonstrate the potential of REQINONE to improve the efficiency of requirements engineering.

As part of future work, we aim to extend REQINONE by incorporating automated requirement validation mechanisms, enabling a more robust generation–validation–update workflow to further improve the quality and reliability of generated SRSs.

REFERENCES

[1] H. F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *IEEE software*, vol. 18, no. 4, p. 58, 2001.

[2] J. Doe, "Recommended practice for software requirements specifications (ieee)," *IEEE, New York*, 2011.

[3] F. Belfo, "People, organizational and technological dimensions of software requirements specification," *Procedia Technology*, vol. 5, pp. 310–318, 2012.

[4] "*VisualParadigm*." Available: https://www.visual-paradigm.com/, 2001.

[5] "*ReqView*." Available: https://www.reqview.com/, 2015.

[6] "*Elementool*." Available: https://www.elementool.com/, 2000.

[7] M. G. Georgiades and A. S. Andreou, "Automatic generation of a software requirements specification (srs) document," in *2010 10th International Conference on Intelligent Systems Design and Applications*, pp. 1095–1100, IEEE, 2010.

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[9] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[10] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[11] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[12] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[13] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[15] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.

[17] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "Norbert: Transfer learning for requirements classification," in *2020 IEEE 28th international requirements engineering conference (RE)*, pp. 169–179, IEEE, 2020.

[18] C. S. R. K. Surana, D. B. Gupta, S. P. Shankar, *et al.*, "Intelligent chatbot for requirements elicitation and classification," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pp. 866–870, IEEE, 2019.

[19] K. Ronanki, B. Cabrero-Daniel, and C. Berger, "Chatgpt as a tool for user story quality evaluation: Trustworthy out of the box?," in *International Conference on Agile Software Development*, pp. 173–181, Springer, 2022.

[20] D. Luitel, S. Hassani, and M. Sabetzadeh, "Improving requirements completeness: Automated assistance through large language models," *Requirements Engineering*, vol. 29, no. 1, pp. 73–95, 2024.

[21] K. Ronanki, C. Berger, and J. Horkoff, "Investigating chatgpt's potential to assist in requirements elicitation processes," in *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 354–361, IEEE, 2023.

[22] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "Ai-based question answering assistance for analyzing natural-language requirements," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1277–1289, IEEE, 2023.

[23] A. M. Abdelfattah, N. A. Ali, M. Abd Elaziz, and H. H. Ammar, "Roadmap for software engineering education using chatgpt," in *2023 International Conference on Artificial Intelligence Science and Applications in Industry and Society (CAISAIS)*, pp. 1–6, IEEE, 2023.

[24] A. El-Hajjami, N. Fafin, and C. Salinesi, "Which ai technique is better to classify requirements? an experiment with svm, lstm, and chatgpt," *arXiv preprint arXiv:2311.11547*, 2023.

[25] M. Endres, S. Fakhoury, S. Chakraborty, and S. K. Lahiri, "Can large language models transform natural language intent into formal method postconditions?," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1889–1912, 2024.

[26] G. Leite, F. Arruda, P. Antonino, A. Sampaio, and A. Roscoe, "Extracting formal smart-contract specifications from natural language with LLMs," in *International Conference on Formal Aspects of Component Software*, pp. 109–126, Springer, 2024.

[27] M. Krishna, B. Gaur, A. Verma, and P. Jalote, "Using llms in software requirements specifications: an empirical evaluation," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pp. 475–483, IEEE, 2024.

[28] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

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

[30] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th international requirements engineering conference (RE)*, pp. 502–505, IEEE, 2017.

[31] "*ReqInOne*." Available: https://github.com/TaohongZ/ReqInOne, 2025.

[32] X. Tian, "Evaluating the repair ability of llm under different prompt settings," in *2024 IEEE International Conference on Software Services Engineering (SSE)*, pp. 313–322, IEEE, 2024.

[33] IEEE, "IEEE recommended practice for software requirements specifications." IEEE Std 830-1998, 1998. pp. 1–40.

[34] "*Guide to Writing Requirements*." Available: https://www.incose.org/docs/default-source/working-groups/requirements-wg/gtwr/incose_rwg_gtwr_v4_040423_final_dra\-fts.pdf\?sfvrsn=5c877fc7_2, 2023.

[35] INCOSE, *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[36] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements engineering*, vol. 12, pp. 103–120, 2007.

[37] M. Glinz, "On non-functional requirements," in *15th IEEE international requirements engineering conference (RE 2007)*, pp. 21–26, IEEE, 2007.

[38] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are" non-functional" requirements really non-functional? an investigation of non-functional requirements in practice," in *Proceedings of the 38th international conference on software engineering*, pp. 832–842, 2016.

[39] K. E. Wiegers, "Writing quality requirements," *Software Development*, vol. 7, no. 5, pp. 44–48, 1999.

[40] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, *et al.*, "Identifying and measuring quality in a software requirements specification," in *[1993] Proceedings First International Software Metrics Symposium*, pp. 141–152, Ieee, 1993.

[41] C.-H. Jane, M. Sepideh, L. Huang, and P. Dan, "*PRMOISE NFR Dataset*." Available: https://zenodo.org/records/268542, 2007.