Probing with Precision: Probing Question Generation for Architectural Information Elicitation

Gokul Rejithkumar
gokul.rejithkumar@tcs.com
TCS Research
Pune, India

Preethu Rose Anish
preethu.rose@tcs.com
TCS Research
Pune, India

Jyoti Shukla
jyotis.shukla@tcs.com
TCS Research
Pune, India

Smita Ghaisas
smita.ghaisas@tcs.com
TCS Research
Pune, India

ABSTRACT

Software Requirements Specifications (SRS) often lack the necessary level of specificity required by software architects to make well-informed architectural decisions. This deficiency compels software architects to probe business analysts to collect more details pertinent to architectural requirements from the clients. In our previous work, we introduced Probing Question-flows (PQ-flows) that can assist business analysts to probe stakeholders and gather architecturally significant information for the creation of a more comprehensive SRS. Key limitations of our previous work were the manually created templatized PQ-flows and the mapping of PQ-flows to the software requirements based on standard Vector Space Model. In this study, we propose a Retrieval Augmented Generation (RAG) prompting framework to address these limitations. We conducted experiments using ChatGPT and Mistral-7B models. We present our findings utilizing human and automated evaluation metrics on a subset of the publicly available PUblic REquirements (PURE) dataset.

CCS CONCEPTS

• **Software and its engineering** → **Requirements analysis**; • **Computing methodologies** → **Natural language generation**; • **Information systems** → *Information retrieval query processing*; *Question answering*.

KEYWORDS

Requirements Engineering, Probing Questions, Large Language Models, Prompting, Retrieval Augmented Generation, ChatGPT, Mistral

ACM Reference Format:
Gokul Rejithkumar, Preethu Rose Anish, Jyoti Shukla, and Smita Ghaisas. 2024. Probing with Precision: Probing Question Generation for Architectural Information Elicitation. In *Workshop on Multi-disciplinary, Open, and RElevant Requirements Engineering (MO2RE 2024), April 16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3643666.3648577

MO2RE 2024, April 16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0569-4/24/04
https://doi.org/10.1145/3643666.3648577

1 INTRODUCTION

Software Requirements Specifications (SRS) often specify requirements at a high level and lack the necessary level of specificity required to make well-informed architectural decisions [1–3]. For example, consider a requirement from the PUblic REquirements (PURE) dataset [6]—*The Event Recognition Processor CSC, identified XCP-ERP, receives raw CCD data from XCP-CCD, generates output Reports, and outputs them to XCP-DCX for compression and transmission to the Spacecraft. It also contains the Event Recognition Algorithm, the Centroid Algorithm, the bad pixel/row/column routines, bias algorithms, baseline correction, and mean row correction.* However, neither this requirement nor the SRS from which it was extracted specify critical architectural details such as: *the frequency and volume of raw CCD data being received from XCP-CCD, the format of the raw CCD data, the output format of Event Recognition Processor CSC, the latency requirement for processing raw CCD data and generating output Reports.* Such details are crucial for the software architects (SAs) to make well-informed architectural decisions pertinent to a given software requirement.

Clients often expect developers and SAs to possess an innate understanding of their requirements and therefore usually do not express such details clearly. As a result, the SAs may need to conduct additional formal or informal interviews with relevant stakeholders to clarify and gather missing information. Experienced SAs can often detect when critical details are absent and may resort to assuming the client's requirements. However, conducting additional interviews is time-consuming, and relying on assumptions for missing information can result in costly refactoring efforts during later stages of the project [2].

In our previous works [1–3], we discovered that skilled and experienced SAs ask Probing Questions (PQs) to address missing information in Software Requirements Specifications (SRS). We explored the perspectives of more than 40 experienced SAs regarding the process of identifying, analyzing, structuring, and assessing PQs with respect to five frequently occurring functional areas: *Audit Trail, Batch Processing, Business Process State Alerts, Report*, and *Workflow*. Based on this, we introduced five structured PQ-flows for each functional area, aiming to equip business analysts (BAs) to elicit a more comprehensive set of requirements that can contribute

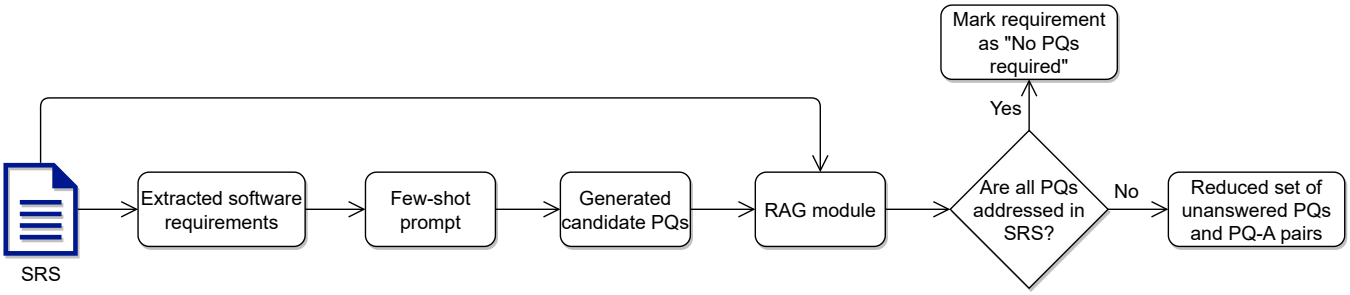

Figure 1: Retrieval augmented prompting framework

sufficient information to the architecture design process. Additionally, we presented an automated approach for analyzing an SRS and mapping requirements to their respective PQ-flows. The automated approach first determined whether a requirement qualifies as an Architecturally Significant Functional Requirement (ASFR) using a Naïve Bayes binary classifier. Subsequently, it mapped the ASFR to the relevant PQ-flows employing the Random K-label sets Ensemble multilabel classifier (RAkEL). The approach used the standard Vector Space Model (VSM) [15] to find instances where questions in a PQ-flow might have already been answered in different sections of the SRS. This involved transforming PQs in the PQ-flow and requirements in the SRS to a VSM and comparing them using a cosine similarity score. A high similarity score indicated that a particular PQ had already been addressed somewhere within the SRS. In such cases, these PQs were removed from the list of unanswered questions of the PQ-flow. The approach then involved presenting the unanswered PQs and the PQ-A pairs (answered PQs and requirement containing the answers to the addressed PQs) to the BA. The key shortcomings of this work were: (1) PQ-flows: Manually created templatized PQ-flows that may not elicit any new information and the limited adaptability of PQ-flow templates to evolving requirements in software due to their dynamic nature. For example, consider the alterations and incorporation of new software requirements prompted by the recent surge in generative AI models. Additionally, the sustainability and environmental requirements arising from the emerging field of Green AI [16]. (2) Restrictions: The confinement of PQ-flows to only five functional areas. (3) VSM and similarity score: A high cosine similarity score in a VSM does not necessarily imply that the requirement effectively addresses the PQ, as VSM with cosine similarity lacks consideration of semantic information [8].

We encountered only one work [5] in Requirements Engineering (RE), that aligned with our work, albeit for a different purpose. In [5], Ezzini et al. proposed a QA system based on machine reading comprehension to support the analysis of requirements. When presented with a question, the system retrieves relevant passages from SRSs and pertinent external documents that likely contain the answers to the questions posed. It further demarcates the answers to the questions, aiding various stakeholders, including requirements engineers. In this work, we introduce a Retrieval Augmented Generation (RAG) prompting framework for generating PQs [9]. In the literature on RE and Software Architecture, both Functional Requirements (FRs) and Non-Functional Requirements (NFRs) play crucial roles in architectural design [2, 13]. Consequently, our focus in this study is on Architecturally Significant Requirements (ASRs) – which include both FRs and NFRs rather than restricting them only to ASFRs.

Given a requirement, our framework: (1) Determines if it is as an ASR and generates semantically diverse candidate PQs in case the ASR requires more specificity. This is achieved using a few-shot prompt (2) Verifies whether the candidate PQs are already addressed in the SRS through retrieval augmented prompts, thus making a distinction between local and global contexts of the SRS. (Local context refers to the specific requirement at hand, while global context refers to the whole SRS.) This is achieved by combining cosine similarity with an LLM, thus taking into account contextual information too. (3) Filters out the addressed PQs and presents the remaining unanswered PQs and PQ-A pairs to the end user.

Figure 1 presents the retrieval augmented prompting framework. We conducted experiments with two Large Language Models (LLMs), ChatGPT (*gpt-3.5-turbo*) [12] and Mistral-7B model [7]. We annotated and evaluated our framework on 10 SRSs from the publicly available PUblic REquirements (PURE) dataset. We present our results using automated evaluation metrics for ASR identification and human evaluation metrics for the quality of generated PQs and PQ-A pairs.

The remainder of the paper is organized as follows: Section 2 provides details about the dataset and the retrieval augmented prompting framework, Section 3 delves into the results of our experiments, Section 4 discusses the threats to validity, and Section 5 concludes the paper.

2 METHODOLOGY

In this Section, we discuss the dataset details and the retrieval augmented prompting framework.

2.1 Dataset creation

We evaluated our framework using a subset of the PUblic REquirements (PURE) dataset [6], which consists of 79 Software Requirements Specifications (SRSs) publicly accessible on the web. Among these 79 documents in PURE, some are already manually ported to a common XML format by Ferrari et al. [6]. We utilized 10 XML-formatted documents for dataset creation. We extracted requirements from subsubsection (*text_body*) of these documents. To maintain coherence, we treated all sentences within a subsubsection as

a single requirement, as splitting them would render the requirement unintelligible. We extracted 1195 requirements from these 10 SRSs. The size of each SRS ranged from 10 to 148 pages, with each requirement having a word count between 10 and 1156 words.

The participants in the dataset annotation task comprised two authors of this paper and two software architects (SAs). The SAs have more than 15 years of experience in executing projects across diverse domains and technologies. The two authors assigned binary labels to the requirements in the dataset, indicating whether the requirement is an Architecturally Significant Requirement (ASR) or not. It is to be noted that every requirement underwent independent annotations by the two authors. In instances of disagreements, the requirement was escalated to the SAs. After annotation, the dataset contained a total of 522 ASRs. Each participant devoted a total of 5 working days to this task, averaging 4 hours per day.

2.2 Retrieval Augmented Prompting Framework

The retrieval-augmented prompting framework consists of two stages: (1) In the first stage, we identify a requirement as ASR and generate candidate PQs if necessary, focusing solely on the local context. However, due to the cross-referencing nature of the information in SRS, a PQ might be addressed in another section of the SRS. Consequently, some or all the candidate PQs generated in the first stage may be extraneous, as they could already be addressed in some other section of the SRS. We address this issue in the second stage by considering the global context. (2) In the second stage, we employ Retrieval Augmented Generation (RAG) to filter out PQs from the candidates that have already been answered in the SRS, resulting in a more refined set of relevant PQs and PQ-A pairs.

We utilized the Hugging Face PyTorch implementation of the Mistral-7B model, specifically we used the *zephyr-7b-beta*[1] version. We utilized the OpenAI API for interacting with *gpt-3.5-turbo* model (ChatGPT), specifically we used *gpt-3.5-turbo-0613*[2] version (training data up to September 2021). Figure 2 presents the schematic of RAG module. We ran all our experiments on an Nvidia Tesla V100 GPU with 32 GiB GPU memory and 60 GiB CPU RAM.

2.2.1 ASR Identification and Generation of Candidate PQs. Various prompting techniques, including zero-shot, few-shot, Chain-of-Thoughts (CoT) [18], Tree-of-Thoughts (ToT) [19], self-consistency (few-shot-CoT) [17], EmotionPrompt [10], and others, can be found in the literature of Natural Language Processing (NLP). According to [18], CoT negatively impacts the performance of models with fewer than 100B parameters. This also applies to ToT and self-consistency, as they incorporate CoT as a fundamental component. In [10], the authors proposed EmotionPrompt and claimed that LLMs possess emotional intelligence. They showed that the performance of LLMs can be enhanced by providing emotional stimuli by appending particular phrases to prompts, such as: "*This is very important to my career.*", "*Are you sure?*", "*You'd better be sure.*", among others. EmotionPrompt also yielded positive performance enhancements on models with very few parameters. Additionally, it also gave better performance than CoT. Consequently, we utilized few-shot prompts inspired by EmotionPrompt for ASR identification and candidate PQ generation.

Few-shot learning involves providing the model with a collection of well-crafted examples, each comprising input and the corresponding desired output related to the targeted task. By exposing the models to the expected inputs and outputs, the model gains a deeper comprehension of human intentions and the criteria for desired responses. For improved prompt clarity, we formulated a few-shot prompt, with six examples pertaining to ASR and another six belonging to the non-ASR class. We equally divided the examples in the prompt to address majority bias [21]. To mitigate recency bias [21], we alternated between ASR and non-ASR examples in the prompt. Building on [10], we enhanced the prompt by appending an emotional stimuli phrase "*This is very important to my career.*". To enable easier processing of the model's output, we instructed the model to generate the output in JSON format. Given a requirement, the prompt instructs the model to identify whether it is an ASR. Subsequently, if it is determined to be an ASR, the model checks whether PQs are required. If they are necessary, it generates PQs, taking into account only the given requirement (local context).

A system prompt offers context and guidance by outlining a particular goal or role for the LLM before presenting a question or task. Additionally, the system prompt plays a crucial role in establishing the overall tone of the LLM. While we instructed the models to generate its output in the format of a JSON object through the user prompt, we observed that instructing the LLMs to generate JSON in the user prompt alone occasionally resulted in a deviation. Therefore, we instructed the LLMs to generate its output in JSON format in the system prompt as well. Figure 3 illustrates the structure of the prompt template for ASR identification and candidate PQ generation including the system and user prompt.

We experimented with temperature values ranging from 0.1 to 1.5. The Mistral-7B model yielded the best classification results at temperature values of 0.8 and 0.9, while for the gpt-3.5-turbo model, the optimal temperature was found to be 0.5. We set the context window to 3200 since it addressed all the requirements and prompts in our dataset. During inference, we used a batch size of 4.

2.2.2 Filtering Extraneous PQs from Candidate PQs. As mentioned earlier, PQs within the generated candidate PQ set may have already been answered in a different section of the SRS. To filter out these extraneous PQs, we need to consider the global context. We employed RAG to factor in the global context since the limited context window of LLMs restricts passing in the entire SRS to the LLM.

Embeddings and Vector database. We extracted requirements from our dataset and segmented it into chunks of length of 350 tokens (*chunk size*), based on the average length of requirements in the dataset. To prevent the loss of information during this segmentation, we set an intersection window of 40 tokens (*chunk overlap*) between two consecutive chunks. We transformed the requirements in the SRS into mathematical representations (embeddings) that allows for efficient storage and retrieval from a database. This was achieved using a pretrained embedding model. We used the Hugging Face implementation of the *bge-large-en*[3] embedding model to

[1] https://huggingface.co/HuggingFaceH4/zephyr-7b-beta
[2] https://platform.openai.com/docs/models/gpt-3-5
[3] https://huggingface.co/BAAI/bge-large-en-v1.5

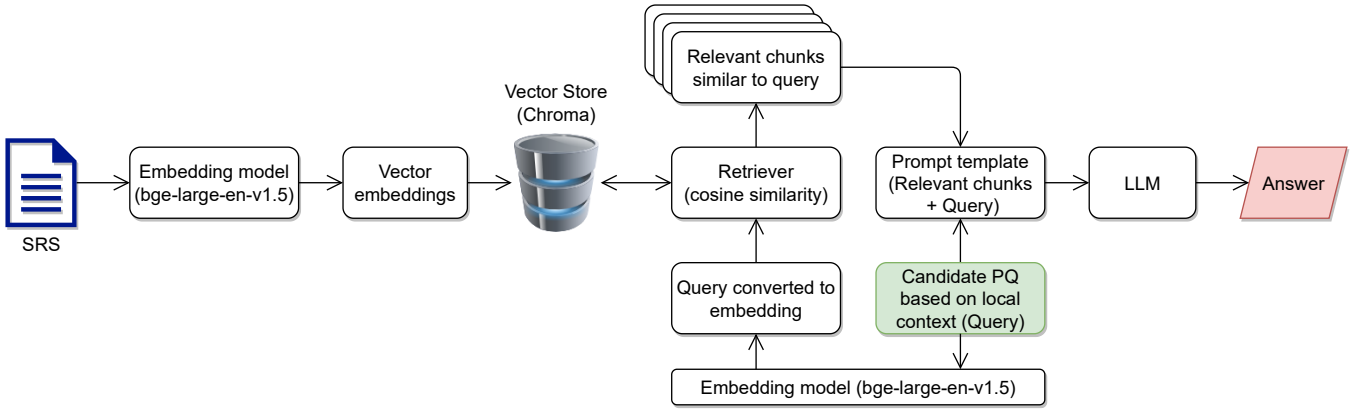

Figure 2: Retrieval Augmented Generation (RAG) module

convert the chunks to embeddings. We then used the open-source Chroma[4] vector database to store these embeddings for later retrieval.

Retrieval Augmented Prompting. In this step, we utilized the vector database and the LLM to retrieve relevant chunks and verify whether they address the candidate PQ using a prompt. Figure 4 illustrates the prompt template including the system and user prompt. The candidate PQ was converted into a query vector (embedding) using the *bge-large-en* embedding model. To retrieve chunks relevant to the query vector, we employed cosine similarity as the retrieval metric and retrieved the top 4 chunks showing the highest similarity to the query vector. Subsequently, we prompted the LLM with the retrieved chunks and the candidate PQ to assess whether the chunks answer the candidate PQ (Context window of 1800 and temperature of 0.6 for both models.). If the LLM failed to provide an answer to the candidate PQ based on the retrieved chunks, we retained the PQ as unanswered. However, if the LLM answered the candidate PQ, we returned the PQ-A pairs.

3 RESULTS

We evaluate the following: (1) Identification of ASR, (2) Performance of RAG module, and (3) Quality of the generated PQs.

3.1 ASR Identification

For the evaluation of ASR identification, we utilized the 10 SRSs that we had annotated with binary labels indicating ASR or Not ASR. We designated ASR as the positive class. We placed a greater emphasis on recall, as failure to identify an ASR instance would imply overlooking architecturally significant information. Therefore, we present our results using the F2-score, aiming to minimize false negatives. Table 1 presents the results of ASR identification for both models. As evident from Table 1, both models achieved an F2-score of 0.81. However, the Mistral-7B model exhibited a very low precision of 0.50. In contrast, ChatGPT exhibited a more balanced and acceptable precision and recall.

[4]https://www.trychroma.com/

System prompt

The assistant strictly adheres to the user's instructions and tasks. The tasks given by the user will be challenging, so the assistant should pay close attention while solving the provided complex tasks. The assistant's response is always a JSON object and does not include any additional details.

User prompt

Your task is to first check whether the given requirement statement is an Architecturally Significant Requirement (ASR). Second, if it is an ASR and the requirement seems to be incomplete or lacks necessary details, recommend up to six Probing Questions (PQs) that will aid me in producing a more complete ASR.

<Definition of ASR for LLM understanding>

Start of examples.
<Example 1: ASR> <Example 2: Not ASR>...<Example 12: Not ASR>
End of examples.

Now provide your analysis in JSON with keys 'Classification rationale', 'ASR', 'PQs required', 'PQs' for the following given requirement statement in triple backticks.

Given requirement statement: ```{}```

This is important to my career.

Figure 3: Prompt template for candidate PQ generation

System prompt

The assistant strictly adheres to the user's instructions and tasks. The tasks given by the user will be challenging, so the assistant should pay close attention while solving the provided complex tasks.

User prompt

Use the following pieces of context to answer the question. If you don't know the answer, just say "I don't know". Use three sentences maximum and keep the answer concise. No additional details.

Question: {candidate PQ}

Context: {context 1, context 2, context 3}

Give accurate answers. This is important to my career.

Figure 4: Prompt template for RAG

3.2 RAG Module

We assigned the evaluation of RAG module to the two SAs referenced earlier. The assessment of RAG module necessitates a

Table 1: Results of ASR identification

Model	Precision	Recall	F2-score
ChatGPT	**0.87**	**0.79**	**0.81**
Mistral-7B	0.50	0.96	0.81

document-level evaluation. We assessed 267 PQs, including PQ-A pairs, randomly sampled across 50 requirements from five SRSs. For each PQ, the SAs first verified whether the corresponding SRS answered the PQ. To find answers to the PQs, the SAs examined whether the keywords used in the PQ were explicated elsewhere in the corresponding SRS. Then, the SAs compared their answer with the answer generated by the retrieval augmented prompt. A score of 1 was assigned for correct answers, while a score of 0 was assigned for incorrect, hallucinated answers, or answer present in the SRS but not retrieved. The SAs devoted a total of 5 working days to this task, averaging 5 hours a day. We present our results using accuracy in Table 2 for both models. Out of the 50 requirements evaluated, Mistral-7B marked 11 as '*no PQ required*' and ChatGPT marked 14 as '*no PQ required*'.

Table 2: Results of RAG module

Model	Accuracy
ChatGPT	**0.88**
Mistral-7B	0.82

3.3 Quality of Generated PQs

The participants in this evaluation included the two SAs. To assess the quality of the generated PQs, we evaluated the same set of 50 requirements (267 PQs) randomly sampled from five SRSs as outlined in Section 3.2. In the absence of a ground truth for PQ evaluation, we employed qualitative human evaluation metrics. Several human evaluation metrics have been proposed in the literature [11] for evaluating generated questions. We specifically chose metrics that are most relevant to our task based on feedback from the SAs. We adopted individual human evaluation metrics—***Relevance***, ***Seeking New Information***, and ***Usefulness*** from [14]. Additionally, we adopted a group level metric—***Redundant*** from [20]. Although [14] and [20] are not directly related to PQ generation, the SAs deemed that these metrics are relevant to PQ quality evaluation. The SAs evaluated the PQs along the following axes, and chose from YES (1) or NO (0):

- ***Relevance:*** *We ask, "Is the generated PQ relevant to the given requirement?"*
- ***Seeking New Information:*** *We ask, "Does the PQ seek any new information not specified in the given requirement?"*
- ***Usefulness:*** *We ask, "Is the PQ useful with respect to the given requirement in eliciting architecturally significant information?"*
- ***Redundant:*** *We ask, "Are there multiple instances of paraphrased duplicate PQs within the set of PQs?" This metric is a group-level metric and applies to a set of PQs. A lower value of this metric is preferred since redundancy is not desirable.*

Table 3 presents the results of PQ quality evaluation. We present the results averaged across the 267 PQs. The task of PQ quality evaluation took a total of 6 working days, averaging 4 hours per day. We observed a very low value for the ***Redundant*** quality metric across both the models, indicating that questions generated based on the local context were semantically diverse. Figure 5 presents examples of the generated PQs based on local context and a refined set of filtered PQs based on the global context across both the models for a requirement from the PURE dataset.

4 THREATS TO VALIDITY

The first threat to validity is the absence of standard automated evaluation metrics for evaluating the quality of the generated PQs. To mitigate this threat, we asked two software architects (SAs) to verify the quality of generated rationales by leveraging human evaluation metrics such as ***Relevance, Seeking New Information, Usefulness,*** and ***Redundant***.

The second threat to validity is the reproducibility of the experiments reported in this paper. To support reproducibility, we have made our dataset available on demand. Additionally, we have also provided the hyperparameter values for all models. However, it should be noted that LLMs may not always generate the same answers, even if hyperparameters are the same. Behavior drift and generation quality deterioration have also been observed with ChatGPT [4].

5 CONCLUSION

In this study, we presented a retrieval augmented prompting framework designed to aid BAs to effectively probe project stakeholders and extract more precise architecturally significant information during the drafting of an SRS. Given an SRS, the framework generates PQs for each requirement within the SRS, focusing on the local context. Since PQs based on the local context might find answers in other parts of the SRS, the framework leverages RAG to consider the entire SRS comprehensively and expunge the answered PQs. Subsequently, the framework provides the generated PQs and their corresponding PQ-A pairs for requirements that necessitate posing a PQ. We used ChatGPT and Mistral-7B models for implementing the retrieval augmented prompting framework. ChatGPT's performance was found to be slightly better than the Mistral-7B model across both ASR identification and PQ generation. To assess the effectiveness of our approach, we conducted evaluations using a dataset comprising 10 SRSs sourced from the PURE dataset. Further, we plan to conduct empirical studies involving SAs and BAs to gauge the usefulness of the approach. The embedding model we utilized has not been pre-trained and fine-tuned on SRSs. We intend to conduct experiments by fine-tuning an embedding model and assess its retrieval efficacy, specifically for SRSs. Additionally, we plan to conduct an empirical study to assess the efficacy of our framework, incorporating objective measures such as the number of follow-up questions required after using our framework, time saved in software development, and the impact on efficiency in software development.

Table 3: Results of PQ quality evaluation using human evaluation metrics

Model	Relevance	Seeking New Information	Usefulness	Redundant
ChatGPT	**0.78**	**0.83**	**0.76**	**0.05**
Mistral-7B	0.74	0.76	0.71	0.02

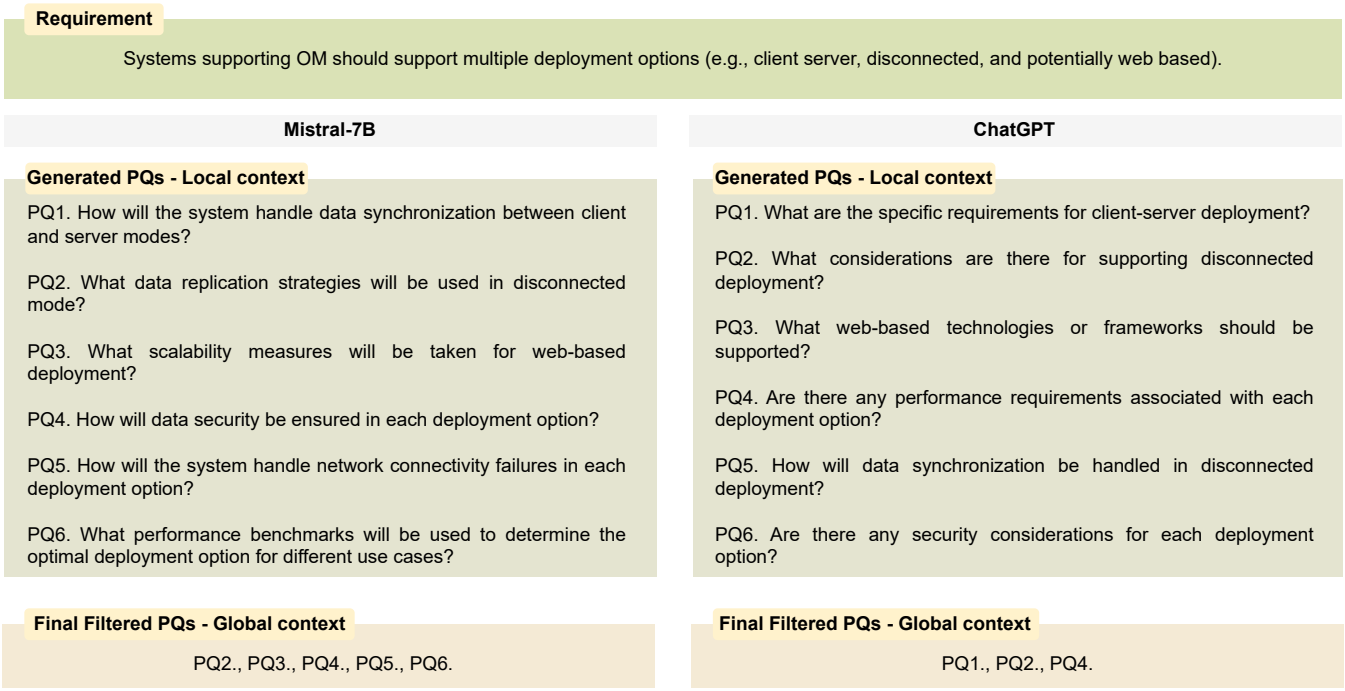

Figure 5: Examples of generated PQs

REFERENCES

[1] Preethu Rose Anish, Balaji Balasubramaniam, Jane Cleland-Huang, Roel Wieringa, Maya Daneva, and Smita Ghaisas. 2015. Identifying Architecturally Significant Functional Requirements. In *2015 IEEE/ACM 5th International Workshop on the Twin Peaks of Requirements and Architecture*. 3–8. https://doi.org/10.1109/TwinPeaks.2015.9

[2] Preethu Rose Anish, Balaji Balasubramaniam, Abhishek Sainani, Jane Cleland-Huang, Maya Daneva, Roel J. Wieringa, and Smita Ghaisas. 2016. Probing for Requirements Knowledge to Stimulate Architectural Thinking. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 843–854. https://doi.org/10.1145/2884781.2884801

[3] Preethu Rose Anish, Maya Daneva, Jane Cleland-Huang, Roel J. Wieringa, and Smita Ghaisas. 2015. What you ask is what you get: Understanding architecturally significant functional requirements. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 86–95. https://doi.org/10.1109/RE.2015.7320411

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? arXiv:2307.09009 [cs.CL]

[5] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh. 2023. AI-based Question Answering Assistance for Analyzing Natural-language Requirements. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1277–1289. https://doi.org/10.1109/ICSE48619.2023.00113

[6] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. PURE: A Dataset of Public Requirements Documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 502–505. https://doi.org/10.1109/RE.2017.29

[7] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B.

[8] Qilu Jiao and Shunyao Zhang. 2021. A Brief Survey of Word Embedding and Its Recent Development. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Vol. 5. 1697–1701. https://doi.org/10.1109/IAEAC50856.2021.9390956

[9] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[10] Cheng Li, Jindong Wang, Yixuan Zhang, Kaijie Zhu, Wenxin Hou, Jianxun Lian, Fang Luo, Qiang Yang, and Xing Xie. 2023. Large Language Models Understand and Can be Enhanced by Emotional Stimuli. arXiv:2307.11760 [cs.CL]

[11] Nikahat Mulla and Prachi Gharpure. 2023. Automatic Question Generation: A Review of Methodologies, Datasets, Evaluation Metrics, and Applications. *Prog. in Artif. Intell.* 12, 1 (jan 2023), 1–32. https://doi.org/10.1007/s13748-023-00295-9

[12] OpenAI. (n.d.). [n. d.]. *OpenAI platform*. Retrieved December, 2023 from https://platform.openai.com/docs/models

[13] Barbara Paech, Allen Dutoit, Daniel Kerkow, and Antje Knethen. 2002. Functional requirements, non-functional requirements, and architecture should not be separated -A position paper. (01 2002).

[14] Sudha Rao and Hal Daumé III. 2019. Answer-based Adversarial Training for Generating Clarification Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 143–155. https://doi.org/10.18653/v1/N19-1013

[15] G. Salton, A. Wong, and C. S. Yang. 1975. A Vector Space Model for Automatic Indexing. *Commun. ACM* 18, 11 (nov 1975), 613–620. https://doi.org/10.1145/

361219.361220
[16] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. arXiv:1907.10597 [cs.CY]
[17] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL]
[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems. 35 (2022), 24824–24837.
[19] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models.
[20] Zhiling Zhang and Kenny Zhu. 2021. Diverse and Specific Clarification Question Generation with Keywords. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3501–3511. https://doi.org/10.1145/3442381.3449876
[21] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*. PMLR, 12697–12706.