Innovations in Systems and Software Engineering
https://doi.org/10.1007/s11334-024-00589-8

ORIGINAL ARTICLE

# ARIA-QA: AI-agent based requirements inspection and analysis through question answering

Chitrak Biswas[1] · Souvick Das[2]

Received: 10 May 2024 / Accepted: 7 October 2024

**Abstract**
Due to their predominant use of natural language, requirements are prone to defects like inconsistency and incompleteness. Consequently, quality assurance processes are commonly applied to requirements manually. However, manual execution of these processes can be laborious and may inadvertently overlook critical quality issues due to time and budget constraints. This paper introduces ARIA, an innovative question–answering (QA) approach designed to automate support for stakeholders, including requirements engineers, during the analysis of NL requirements. The ability to ask questions and get immediate answers is extremely useful in many quality assurance situations, especially for incompleteness detection. The challenge of automating the answering of requirements-related questions is considerable, given the potential scope of the search for answers extending beyond the provided requirements specification. To overcome this challenge, ARIA integrates support for mining external domain knowledge resources like internet search results. Evaluation of seven diverse use cases drawn from the PURE dataset demonstrates ARIA's robustness and applicability across a range of real-life scenarios, highlighting its potential to significantly improve the quality and effectiveness of requirements analysis processes. This work represents one of the initial endeavors to seamlessly blend QA and external domain knowledge, effectively addressing complexities in requirements engineering through a flexible framework designed to readily integrate with evolving, advanced-level Generative LLMs.

**Keywords** AI agent · Semantic search · Knowledge graph · Question answering · Retrieval augmented generation · Prompt engineering

## 1 Introduction

The Software Requirements Specification (SRS) serves as a cornerstone in Requirements Engineering (RE), outlining the essential characteristics, functions, and qualities of a proposed system [16]. Stakeholders, including requirements engineers, frequently analyze SRS documents to ensure the quality of requirements [31]. To foster a shared understanding among diverse stakeholders from various backgrounds, requirements are commonly articulated in natural language (NL) [34]. However, the inherent challenges of NL, such as ambiguity [9, 10], incompleteness [6], and inconsistency [12], necessitate meticulous scrutiny during quality assurance processes.

Manual examination of extensive SRS documents, often spanning numerous pages, proves time-intensive and requires domain knowledge for accurate interpretation. Human reliance on domain knowledge is not always swift or efficient. Addressing these challenges, this paper introduces ARIA, an innovative Question Answering (QA) approach designed to automate support for stakeholders, including requirements engineers, during NL requirements analysis. Consider an e-commerce scenario where a developer, while examining a requirement related to the "inventory tracking" system. The developer might pose a question (Q1) regarding the specific details of how the system manages inventory during peak demand periods. The answer to such a question may not be explicitly present in the SRS. ARIA addresses this by utilizing external domain knowledge resources, such as a corpus

✉ Chitrak Biswas
chitrak@amityonline.com

Souvick Das
souvick.das@unive.it

1 Department of Computer Science and Engineering, Amity University, Kolkata 700135, India

2 Department of Environmental Science, Informatics, and Statistics, Ca' Foscari University, 30172 Venezia, Italy

Published online: 21 October 2024

automatically extracted from e-commerce-related articles or documentation.

This work presents ARIA,[1] a novel framework that introduces a question-driven, agent-based approach to requirements analysis, aiming to significantly improve the efficiency and effectiveness of identifying incompleteness and inconsistencies. Our key contributions include:

1. *Collaborative AI Agents for Requirements Analysis:* ARIA uses a unique LLM-powered Analyzer and Verifier agent interaction for iterative analysis and refinement of requirements, distinguishing it from traditional rule-based systems.
2. *Systematic Question Generation for Enhanced Detection:* ARIA uses engineered prompts to generate context-aware questions targeting potential incompleteness and inconsistency, providing a thorough analysis beyond manual inspection or less adaptable techniques.
3. *Real-Time Integration of External Knowledge in RAG:* ARIA accesses external internet knowledge to enrich analysis and detect nuanced inconsistencies and incompleteness missed by internal project data alone.
4. *Practical and Adaptable Design for Wider Adoption:* ARIA prioritizes practicality with open-source technologies and a flexible architecture that integrates with state-of-the-art LLMs, ensuring wider accessibility and cost-effectiveness for diverse development teams and projects.

Furthermore, the evaluation of the framework has been conducted on seven use cases taken from PURE [11] dataset to assess the framework's robustness and applicability across diverse real-life use cases. The rest of the paper is structured as follows:

In Sect. 7, we present the existing state-of-the-art for requirement analysis towards incompleteness detection. Section 2 provides background concepts such as RAG, vector database, and vector search. We present our proposed framework in Sect. 3. In Sect. 5, we elaborate on the experimental evaluation of the framework. Section 6 highlights some threats to the validity of the framework. We also present the comparative analysis with the proposed methods and current state-of-the-art. Finally, Sect. 8, concludes the paper by summarizing our contributions and by highlighting some future directions.

## 2 Preliminaries

This section of the research paper is an introduction to important ideas that are necessary for understanding the rest of the sections. Here, we give an overview of key concepts like RAG [17], prompt engineering, and vector search [26] mechanisms. By explaining these important ideas, we aim to build a strong foundation for the main framework discussed in the research.

### 2.1 Retrieval augmented generation (RAG)

Large Language Models (LLMs) have transformed natural language processing, showcasing impressive capabilities in generating text. However, their outputs may lack factual consistency and grounding in real-world knowledge. Retrieval Augmented Generation (RAG) [17] is an approach that enhances LLMs by incorporating external data sources to improve the accuracy, reliability, and context-awareness of their generated responses. RAG bridges this gap by seamlessly integrating external knowledge retrieval with LLM generation. It operates in three key stages:

1. *Prompt Analysis:* The LLM examines the user's prompt, extracting crucial information points, entities, and relations.
2. *Document Retrieval:* Leveraging the extracted information, RAG searches a pre-defined external knowledge source (e.g., document repository, database, API) for relevant documents. Semantic retrieval techniques ensure that retrieved documents are highly pertinent to the prompt's context.
3. *Augmented Generation:* The retrieved documents are then processed by the LLM to enhance its understanding of the prompt. Key facts, entity relationships, and document sentiment are extracted and integrated into the LLM's internal representation. This augmented knowledge informs the final generation process, resulting in more accurate, reliable, and contextually-aware outputs.

Several benefits of RAG are discussed as follows:

1. *Improved Factual Accuracy:* By grounding responses in real-world knowledge from external sources, RAG significantly reduces the risk of factual errors and biases ingrained in LLM training data [17].
2. *Enhanced Reliability:* RAG ensures consistency and repeatability in responses, crucial for tasks like question answering and report generation.
3. *Increased Context Awareness:* RAG allows LLMs to consider broader contexts not solely learned from their training data, leading to more relevant and nuanced outputs [7, 17].
4. *Domain-Specific Expertise:* Integrating domain-specific knowledge sources empowers RAG to generate responses tailored to specific fields, surpassing LLMs' general knowledge.

[1] https://github.com/svk-cu-nlp/ARIA.

5. *Reduces LLM Hallucination:* The retrieved documents act as fact-checking mechanisms. By relying on external data sources, RAG steers LLMs away from generating purely fictional content [17].

In essence, RAG acts as a knowledge amplifier for LLMs, mitigating their limitations and unlocking their full potential for accurate, reliable, and contextually relevant text generation.

## 2.2 Vector database and vector search

Vector database [28] is a specialized database designed to store and efficiently search high-dimensional vector embeddings, which are numerical representations of data entities that capture their semantic meaning and relationships. Unlike traditional databases that structure data in tables with rows and columns, vector databases use vector spaces where each object is represented as a point in the space, with its similarity to other objects determined by its proximity to those objects in that space. Instead of keyword matching or exact matches, vector databases employ similarity search algorithms like Approximate Nearest Neighbor (ANN) search to retrieve data points most closely aligned with a query vector. Ideal for applications requiring semantic search or similarity-based retrieval, such as: Recommender systems (product recommendations, content suggestions), Image and video retrieval, Natural language processing (text search, chatbots, question answering)

### 2.2.1 Vector search

The process of finding data points in a vector database that are most similar to a given query vector using specific similarity metrics. It employs ANN algorithms optimized for speed and accuracy in high-dimensional spaces, including hashing, quantization, and graph-based methods. In comparison to keyword search it offers several advantages:

- *Semantic Meaning:* Captures the underlying context and relationships within data, not just literal keyword matches.
- *Partial Matches:* Handles incomplete or imprecise queries more effectively.
- *Scalability:* Efficiently handles large datasets and real-time search demands.

***Benefits of Vector Databases and Vector Search:***

- *Relevance:* Deliver more accurate and meaningful results compared to keyword search.
- *Discovery:* Help uncover hidden patterns, relationships, and insights within data.
- *Efficiency:* Perform fast searches over large datasets, even in real-time scenarios.
- *Flexibility:* Handle diverse data types (text, images, sound, scientific data) through vector embeddings.
- *Scalability:* Can efficiently scale to accommodate growing data volumes and search demands.

## 2.3 Prompt engineering

Prompt engineering [33] stands as a pivotal aspect in the realm of natural language processing (NLP), influencing the efficacy and quality of language model outputs. It involves the deliberate crafting and formulation of prompts, the input queries or statements provided to language models, to guide and influence the nature of their responses. Key concepts of prompt engineering are highlighted as follows -

- *Prompt:* The actual text or instructions you provide to the LLM. For instance, “Translate the following Java code to Python: [Java code snippet]"
- *Zero-shot Learning:* Training LLMs on massive datasets helps them learn patterns, but prompt engineering exploits this knowledge without further training [27]. With this mechanism, NLP tasks can be achieved by giving instruction to the LLM what to do without explicitly showing it how.
- *Few-shot Learning:* When a specific task requires more tailored guidance, you can provide a few relevant examples alongside the prompt, helping the LLM grasp the nuances [25].
- *Task Context:* Setting the scene and providing background information can significantly improve the LLM’s understanding and ability to deliver relevant outputs.
- *Desired Tone and Style:* You can specify the style of the generated text, whether it should be formal, informal, funny, poetic, etc.
- *Instructions and Constraints:* Providing clear instructions and any necessary constraints helps the LLM stay focused and avoid undesirable outputs.
- *Role-playing:* You can assign specific roles or identities to the LLM, directing it to respond as a particular character or expert.

## 3 The ARIA framework

In this section, we introduce our novel framework tailored for the comprehensive analysis of requirements specification documents. Our framework features an intuitive question-answering interface, facilitating seamless interaction for diverse stakeholders, including requirements engineers. Through this interface, stakeholders can delve into various aspects of the requirements specification document,

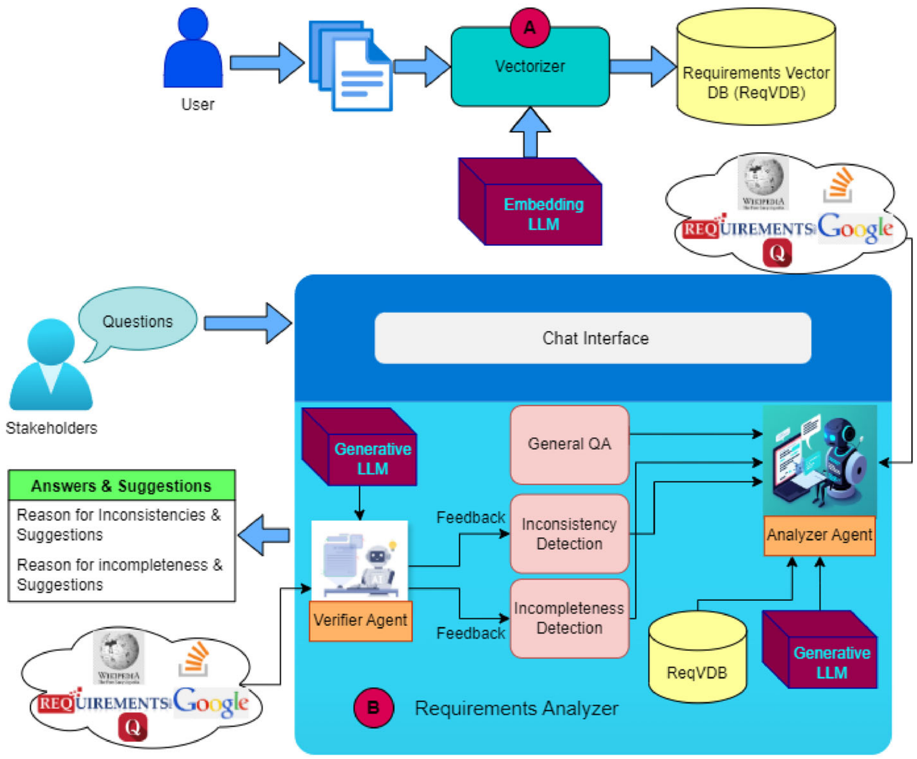

**Fig. 1** The overall architecture of ARIA

including the detection of incompleteness and inconsistency straight from the shelf. Furthermore, our framework offers robust chat support, aimed at enhancing understanding of domain-specific terminologies. This is achieved through the integration of a web search-based Retrieval Augmented Generation (RAG) mechanism.

Figure 1 illustrates the architecture of our framework, providing a visual representation of its components. Subsequent to this, we delve into an in-depth discussion of each component, elucidating their roles and functionalities comprehensively.

## 3.1 Vectorizer

The comprehension of a text is intricately tied to the efficacy of its representation and the precision with which its context is encapsulated during the embedding process. This underscores the criticality of employing a robust and meticulous embedding methodology to attain optimal performance in natural language processing endeavors. Our proposed framework has the *Vectorizer* module marked as **A** in Fig. 1, pivotal in harnessing embedding models to procure embeddings for both source codes and their corresponding explanations. Notably, this module exhibits versatility in accommodating various embedding models or providers such as OpenAI, Cohere, Hugging Face Hub, Mistral, Google Gemini, among others. Additionally, it assumes the responsibility of efficiently storing these embeddings, a task facilitated by a range of vector databases or stores, which also afford semantic search capabilities.

## 3.2 Requirements analyzer

At the heart of this module (marked as **B** in Fig. 1) lies the concept of an AI agent endowed with the ability to execute various tasks through access to pre-defined tools and the RAG mechanism. Our exploration of this section commences with an introduction to the AI agent and the diverse tools at its disposal, along with elucidation of their operational methodologies. Subsequently, we delve into the agent's decision-making process for selecting these tools based on the specific intended actions at hand.

### AI agent

Within this framework, the AI agents comprises four essential components:

1. *Powerful LLMs:* These sophisticated LLMs amplify the agent's capacity to navigate intricate documents and engage in complex interactions with humans or other entities. The framework seamlessly integrates various generative LLMs.
2. *Prompts:* In the realm of prompt engineering, prompts serve as specific instructions or queries provided to LLMs to guide their responses. We explicitly add persona to each

prompt to enhance context and focus by injecting specific background knowledge, experience, and goals, guiding the LLM towards relevant output and avoiding generic responses. This personalized approach directs LLMs toward relevant outputs, mitigating generic responses and enhancing focus on specific tasks.

3. *Memory:* Memory functionality is crucial for agents to retain conversation histories. To manage context length within predefined thresholds of LLMs, conversational summaries are employed, particularly during extensive conversations.
4. *Tools:* A repository of pre-prepared tools facilitates specific tasks, with agents accessing these tools based on their current state when initiating a task. The selection of the appropriate tool is facilitated by a semantic routing mechanism. Modules such as the *Inconsistency Detector*, *Incompleteness Detector*, and *General QA*, depicted in Fig. 1, exemplify these pre-defined tools. Detailed elucidation of these tools and their operational processes is provided in subsequent sections.

To summarize, the orchestration of the aforementioned components builds agents tailored to executing precise tasks. Instructions for subsequent actions are provided, either through predefined prompts or dynamically generated prompts from agent conversation. The semantic router facilitates the selection of suitable tools. Finally, various tools employ RAG and other external resources to analyze the text based on the instructions and generate textual responses.

## Requirements analysis tools

Within the framework, two primary tools are offered for analyzing software requirements specifications. Firstly, the framework facilitates the identification of inconsistencies within the requirements. Secondly, it aids in detecting incompleteness within the requirements. Additionally, one dedicated tool is present to support general QA on the requirements specification document.

### 3.2.1 Incompleteness detector

The core functionality of the *Incompleteness Detector* tool relies on a prompt engineering approach to generate multiple questions aimed at assessing the overall completeness of the requirements specifications based on their functionalities. In this scenario, the answers are retrieved from a vector database. The *Verifier* agent is responsible for verifying the responses provided by the *Analyzer* agent. Throughout this process, the *Verifier* agent continually encourages the *Analyzer* agent to conduct further analysis. To obtain more precise information within the RAG mechanism, a re-ranking algorithm is employed. The *Verifier* agent acts as a cross-validator throughout this entire process. The working process of the *Analyzer* Agent is elaborated as follows.

During the analysis process, the agent first identifies and recognizes the different functionalities outlined in the requirements specifications. For each functionality, it collects all related requirement statements and groups them. Next, on this set of grouped requirement statements, the agent initiates an analysis by posing various questions that are specifically designed to assess completeness. These questions are formulated from multiple perspectives as follows.

– Questions related to finding missing requirements for the functionality.
– Questions regarding domain-specific terminologies used in the requirements. (Optional)
– Questions for identifying ambiguous requirements.
– Questions for identifying incomplete inputs and outputs.
– Questions for identifying missing assumptions and dependencies.
– Questions for identifying constraints mentioned in the requirements. (Not mandatory to get an answer)
– Questions for assessing the testing and validation of the functionality. (Not mandatory to get an answer)

During the analysis process, the *Analyzer* agent employs the RAG mechanism to generate answers to the questions posed. Within this mechanism, it leverages both the vector database containing requirement specifications and internet search capabilities to gain a deeper understanding of these specifications. This is particularly important when dealing with specific functionalities and domain-specific terminologies. It is worth mentioning that the *Verifier* agent also has the capability of conducting web searches to gain additional information about a specific requirement specification. The prompts are formulated in such a manner that if there is no clear answer present within the set of requirements, it will simply respond indicating that the required information is not available. The interaction between the *Analyzer* agent and the *Verifier* agent will be further discussed in detail when exploring the role of the *Verifier* agent.

### 3.2.2 Inconsistency detector

An additional key feature of this framework is the detection mechanism for inconsistencies within the requirement specifications. Similar to the incompleteness detection module, the *Analyzer* agent generates multiple questions and retrieves answers using the RAG mechanism. Throughout this process, the *Verifier* agent continually prompts and encourages the *Analyzer* agent to conduct further analysis until it can ultimately conclude whether or not the requirements are consistent. To assess the inconsistencies, the questions are generated from the following perspectives.

- Questions regarding identifying conflicting requirements with respect to different parameters.
- Questions for logical inconsistencies.
- Questions for identifying inconsistent domain-specific terminologies.
- Questions regarding constraints and dependencies.
- Questions regarding safety and security compliance.

To acquire insights into specific features and pertinent information such as compliances, potential constraints, and dependencies, both the *Analyzer* agent and *Verifier* agent conduct web searches. By combining information extracted from internet searches and the vector database containing requirement specifications, the *Analyzer* agent can apply the RAG mechanism to obtain more precise answers.

***Enhancing Requirements Analysis Based on Communicating Agents:*** Within the *Requirements Analyzer* module, an advanced communication mechanism among AI-agents is introduced to enhance analysis capabilities. The *Analyzer* agent scrutinizes requirements, probing for incompleteness and inconsistencies within specification documents through questioning and answer extraction. Conversely, the *Verifier* agent cross-validates answers against the contextual backdrop of the requirements specification document. Leveraging external knowledge via web searches, it furnishes feedback and pivotal indicators for further analysis, guiding the *Analyzer* agent towards focused investigation in the identified direction. Ultimately, the *Verifier* agent contributes suggestions and potential modifications to rectify incompleteness and inconsistencies within the requirements specification document. These recommendations are enriched by insights sourced from external web searches, empowering the agent to offer informed guidance for refinement.

### 3.2.3 General question answering

Another noteworthy feature of this framework involves the implementation of a question-answering platform directly on the requirements specification document, tailored to meet the unique needs of various stakeholders. This platform enables targeted discussions on specific topics relevant to stakeholders' interests. Powered by the RAG mechanism and supplemented by information extracted from web searches, the question-answering system provides an efficient means of addressing inquiries and facilitating informed discussions.

### 3.2.4 Role of verifier agent

The *Verifier* agent leverages the ChatGPT [23] model to assess the response generated by the *Analyzer* agent. We have created instructions specifically designed for verifying the answers provided by the *Analyzer* agent. The following instructions can be provided in order to verify the answers to the questions posed to assess the requirements specification document, and to provide responses to an Analyzer agent for further analysis.

- *Factual Accuracy:*
  - (i) Check if the RAG response cites sources for factual claims.
  - (ii) Verify the accuracy of cited sources using independent fact-checking services or domain-specific databases.
  - (iii) If no sources are cited, assess the plausibility of the claims based on your knowledge and common sense.
- *Completeness:*
  - (i) Evaluate if the RAG response addresses all aspects of the user query.
  - (ii) Analyze if the retrieved documents provide sufficient information to answer the question comprehensively.
  - (iii) Identify any missing details or potential ambiguities in the response.
- *Consistency:*
  - (i) Ensure the answer aligns with the content and terminology used in the requirements specification document.
  - (ii) Check for contradictions or inconsistencies within the response itself.
  - (iii) Compare the response to any existing answers about the same topic to identify inconsistencies.

***Verification Report:*** The verification report includes -

- Label indicating whether the answer is verified, partially verified, or needs further analysis.
- Explanation of the verification process and reasoning behind the label.
- Identified factual errors, missing information, or inconsistencies.
- Suggestions for improvement or further investigation.

After generating the suggestions, the verifier agent can communicate them to the *Analyzer* agent. This involves providing structured data containing the identified issues, suggested revisions, and justifications for the recommendations. After going through two iterations of the conversation chain, the *Verifier* agent provides a summary of the report and suggests the requirements engineer on potential refinements to the requirements.

## 4 Illustrative example: E-commerce requirements analysis

To illustrate the framework's workflow and demonstrate its ability to detect inconsistencies and incompleteness, we consider a simplified example of requirements specifications for an e-commerce platform:

### 4.1 Initial requirements

- *Requirement 1:* Users should be able to register using a valid email address and password.
- *Requirement 2:* Registered users can browse products, add them to their cart, and proceed to checkout.
- *Requirement 3:* The system shall provide a secure payment gateway for processing transactions.
- *Requirement 4:* Users can track their order status in real-time.
- *Requirement 5:* User data shall be stored securely and in compliance with privacy regulations.

### 4.2 Requirements analysis using analyzer agent

1. *Vectorization:* The Vectorizer embeds each requirement statement into a numerical vector.
2. *Requirements Analysis:*
   - *Functionality Recognition:* The Analyzer agent, powered by an LLM like Mixtral 8x7B, identifies key functionalities like user registration, product browsing, payment processing, and order tracking.
   - *Question Generation:* For each functionality, the Analyzer agent formulates questions targeting completeness and consistency. For instance:
     - *Req. 1:* "What are the password complexity requirements?", "Can users recover forgotten passwords?"
     - *Req. 2:* "Can users filter or sort products?", "How are products displayed?"
     - *Req. 3:* "What payment methods are supported?", "What security standards apply?"
     - *Req. 4:* "How frequently are order status updates provided?", "Is real-time tracking practical?"
     - *Req. 5:* "What specific security measures are used to protect data?", "Which privacy regulations are being followed?", "How is user consent managed?"
   - *RAG-based Retrieval and Initial Analysis:* The Analyzer searches its knowledge base and the internet, finding:
     - Limited password security and no recovery mechanism specified.
     - Lack of product browsing refinements (filtering, sorting).
     - No specific payment methods or security standards were mentioned.
     - Real-time order tracking might be excessive; periodic updates are more common.
     - No mention of specific security measures, relevant privacy regulations, or consent management.

#### 4.2.1 Initial modifications

- *Requirement 1:* Users should be able to register using a valid email address and a password that is at least 8 characters long. The system shall provide a password recovery mechanism.
- *Requirement 2:* Registered users can browse products and add them to their cart. Users should be able to filter products by category and sort by price.
- *Requirement 3:* The system shall provide a secure payment gateway compliant with PCI DSS. Supported payment methods include [Specify].
- *Requirement 4:* Users can track their order status. The system will provide order status updates [Specify frequency].
- *Requirement 5:* User data shall be stored securely using encryption. The system will comply with relevant privacy regulations such as GDPR and CCPA.

### 4.3 Role of the verifier agent

1. *Feedback Loop:* The Verifier analyzes the modifications, checking for completeness, consistency, and industry alignment.
2. *Refinement and Suggestions:* The Verifier agent might suggest:
   - *Req. 1:* Suggest specific password complexity rules. Specify a password recovery method (e.g., email verification).
   - *Req. 2:* Consider adding more filtering and sorting options for better browsing.
   - *Req. 3:* Specify supported payment methods (e.g., credit cards, PayPal). Define what 'secure' entails for the payment gateway.
   - *Req. 4:* Suggest a reasonable update frequency, perhaps daily or every 12 h.
   - *Req. 5:* Specify the encryption standard to be used (e.g., AES-256). Detail how the system will obtain and manage user consent for data processing. Outline data retention policies to align with the specified regulations.

### 4.4 Revised requirements

The Analyzer Agent, taking the Verifier's feedback, refines its analysis and generates the following revised requirements:

- *Requirement 1:* Users must register with a valid email and a strong password (minimum 8 characters, including uppercase, lowercase, and a number). The system offers password recovery via email verification.
- *Requirement 2:* Registered users can browse and filter products by category, price range, and brand, sort by price or newest arrivals, and add items to their cart.
- *Requirement 3:* The system offers a PCI DSS-compliant secure payment gateway with industry-standard encryption and security protocols. Supported payment methods include Visa, Mastercard, American Express, and PayPal.
- *Requirement 4:* Users can track their order status. The system will provide order status updates at least once every 12 h.
- *Requirement 5:* User data shall be stored securely using AES-256 encryption at rest. The system will implement appropriate data access controls to restrict unauthorized access to user data.
- *Requirement 6:* The system complies with GDPR, CCPA, and other privacy regulations. Users consent to data processing upon registration and data is retained only as necessary per the privacy policy.

## 5 Experimental evaluation

This section is structured into two distinct stages. Firstly, we delve deeply into the technical aspects of the framework, providing an in-depth analysis of its components and mechanisms. Secondly, we conduct a comprehensive evaluation across various use case scenarios to extract key insights and understand its practical effectiveness.

### 5.1 Experimental setup

This section illustrates the technical details of every component of the framework.

#### Technical details of vectorizer

The vectorizer component of the framework relies on an embedding language model (LLM), specifically the BGE embedding model [5]. This choice is driven by its exceptional performance demonstrated on the Massive Text Embedding Benchmark (MTEB) [20] dataset. Furthermore, its open-source nature and light-weight design make it advantageous compared to other available embedding models. Once we obtain vectors from the requirements specification document using this LLM, we store them in a Faiss [13] vector database. Throughout different stages of the framework's operation, various agents access this database to perform semantic searches tailored to their specific roles. These searches aim to retrieve relevant information based on semantic similarity.

#### Technical details of analyzer agent

The summary of the technical details of the *Analyzer* agent is provided as follows-

- *Semantic routing:* The selection of the suitable tool is streamlined through the semantic router mechanism, which taps into the capabilities of the semantic vector space to swiftly guide decisions for both LLMs (large language models) and agents. Instead of depending on the slower process of LLM generation to make decisions regarding tool usage, this mechanism efficiently directs requests by harnessing semantic meaning.
- *Utilization of generative LLM: Analyzer* agent leverages the Mixtral 8x7B [14] a large language model (LLM) that is developed by Mistral AI, known for its sparse mixture-of-experts (SMoE) architecture [14]. Moreover, Mixtral surpasses the performance of larger models like Llama 2 70B [30] on benchmarks such as LAMBDA [29] and SuperGLUE [32], despite its smaller size (13B parameters versus 70B). Its SMoE architecture enables 6x faster inference compared to Llama 2, rendering it more efficient for real-time applications. Additionally, Mixtral offers an open-weight model with a permissive license, enabling customization and research.
- *Re-ranking for more precise retrieval:* Regarding the RAG mechanism, a commonly encountered challenge known as the “Lost in the Middle" phenomenon emerges, wherein LLMs encounter difficulties in handling tasks that involve information situated within the middle sections of lengthy contexts. To tackle this issue, we adopt re-ranking algorithms as an interim solution while simultaneously exploring improvements to LLM memory and attention mechanisms. Our strategy involves the deployment of a multi-stage re-ranking mechanism, as outlined by Nogueira et al. [21]. Our application of the multi-stage re-ranking mechanism adheres to the following process:

  (i) Initially, we execute a multi-stage re-ranking procedure on the complete vector store before conducting the initial retrieval based on the query vector.
  (ii) Subsequently, during subsequent iterations of conversation, we refine the search results by filtering out irrelevant information sources. This filtration process is guided by criteria and recommendations, such as metadata analysis, entity presence, or topic similarity, supplied by the *Verifier* agent.

– *Generated questions for incompleteness and inconsistency detection:* The questions are generated automatically by the LLM based on different perspectives presented in Sect. 3.2. Figure 2 shows some of the generated questions for assessing incompleteness in requirements specifications. Figure 3 shows the same for assessing the inconsistencies in requirements

## 5.2 Evaluation of the framework

In this section, we provide a detailed overview of the framework evaluation process. Given the foundation of our framework on the RAG mechanism and prompt engineering techniques, traditional metrics like BLEU [24] and ROUGE [18] are unsuitable due to the absence of reference text for identifying and modifying non-compliant requirements.

To address this limitation, we adopt a comprehensive two-fold evaluation strategy. The first part focuses on evaluating the retrieval and generation methods utilized within our framework. In the second approach, we assess the overall efficacy of the framework by evaluating our framework in 7 usecase requirements specification documents taken from PURE [11] dataset.

## 5.3 Evaluation of RAG pipeline

Throughout this process, we employ the RAGAS framework [7] to evaluate the effectiveness of the RAG mechanism implemented within our framework. Leveraging state-of-the-art LLMs, such as OpenAI's GPT-4 [22] or Anthropic's Claude [3], as evaluative judges have been studied by Zheng et al. [35]. The framework scrutinizes the quality of RAG components by incorporating LLM as a judge for the ground truth creation. The RAGAS framework delineates four pivotal metrics crucial for assessing the efficacy of the RAG mechanism.

i. *Context Recall:* Assesses the relevance of the context to the question, aiming to cover a higher number of attributes from the ground truth.
ii. *Context Precision:* Evaluates whether all relevant ground truth items are included in the context.
iii. *Faithfulness:* Measures the factual consistency of the generated answer with the given context.
iv. *Answer Relevancy:* Evaluates the relevance of the generated answer to the question, penalizing cases of incompleteness or redundancy rather than factuality.

### 5.3.1 Ground truth generation

The ground-truth generation process utilizes the cutting-edge LLM, Claude 3-opus [3] by Anthropic. This model surpasses GPT-4 across all benchmark datasets [3, 15]. Ground-truth data is generated by inputting small segments of a large document into the LLM, prompting it to produce question-answer pairs based on the provided segment as context. For instance, the first column of Table 1 presents the considered SRS document, and the second column indicates the total number of question-answer pairs generated for the entire document. Additionally, the document's size in terms of token count is presented in the third column. It is noteworthy that a manual verification of these question-answer pairs was conducted to ensure dataset quality, involving the removal of irrelevant questions and redundancies. Following the generation of the ground-truth dataset, we proceeded with evaluating the RAG mechanism employed in the framework using the RAGAS [7] framework.

### 5.3.2 Evaluation of RAG

Table 2 presents the evaluation results of the RAG pipeline utilized in the framework. We experimented with three LLMs: i) LLaMA-2, ii) Mixtral 8x7B, and iii) ChatGPT. In this table, we present the scores for the RAG evaluation parameters for each of the requirement specification documents. LLaMA-2, with a capability of managing the least context length of 4k tokens, scores the least among the three LLMs. Conversely, the Mixtral 8x7B model and the ChatGPT model, with context lengths of 32k and 16k respectively, score identically for the RAG evaluation metrics. The capability of managing larger context lengths markedly enhances performance, particularly in terms of context precision and recall. This enhancement arises from the ability to encompass a broader context, thus covering more attributes present in the ground-truth. The metrics of answer relevancy and faithfulness display higher scores, attributable to the re-ranking mechanism effectively identifying and providing sufficient relevant information from the document to generate meaningful pertinent answers with significantly less hallucination.

### 5.3.3 Evaluation on usecases

In Table 3, we highlight the outcomes of two iterations within the conversation chain involving the *Analyzer* and *Verifier* agents. While we have constrained the iterations to two, it is important to note that this number can be adjusted according to specific requirements.

Table 3 presents the results of the experimental evaluation. We conducted experiments with various combinations of LLMs for both the *Analyzer* and *Verifier* agents. Interestingly, utilizing solely the ChatGPT model for both agents yielded slightly superior performance. However, across different LLM combinations, we observed similar outcomes. This leads to two key observations: (i) LLMs can effectively be employed for identifying incompleteness and inconsistency, and (ii) the identified incompleteness and inconsistencies

**Fig. 2** Examples of Generated Questions for Assessing Incompleteness in Requirements

- Are there any functionalities, features, or user needs not explicitly addressed?
- Are there any requirements with unclear language or terminology that could lead to misinterpretations?
- Are all required inputs and outputs for each function clearly defined?
- Are there any underlying assumptions or dependencies not explicitly stated?
- Are there any dependencies on other systems or technologies not documented?

**Fig. 3** Examples of Generated Questions for Assessing Inconsistencies in Requirements

- Are there requirements that directly contradict each other?
- Do any requirements specify different behaviors for the same functionality or feature?
- Do any requirements violate logical reasoning?
- Are the same terms or concepts used with different meanings in different sections?
- Is there inconsistency in data formats, units, or measurement systems?

**Table 1** Specification of the SRS for RAG evaluation

| Dataset | #Questions | Size (#tokens) |
|---|---|---|
| e-store | 465 | 24k |
| ccnts | 372 | 18k |
| Tachonet | 433 | 23k |
| Video Search | 452 | 24k |
| e-procurement | 526 | 28k |
| Inventory | 416 | 22k |
| Znix | 400 | 20k |

hold validity as these LLMs consistently concur on each other's analyses, pinpointing common issues. It is worth noting that the ChatGPT model demonstrates notably superior performance for the *Verifier* agent, effectively prompting the *Analyzer* agent to conduct further analysis based on the provided markers. We observed that as the number of iterations between the Analyzer and Verifier agents increases, the likelihood of remaining inconsistencies and incompleteness in the requirements specifications decreases. This suggests that the iterative feedback loop facilitates a more thorough and targeted refinement process.

The evaluation results highlight several crucial aspects of the framework's practical value. Firstly, its ability to accurately identify incompleteness and inconsistencies in real-life use cases sourced from the PURE dataset underscores its applicability in real-world scenarios. The framework is not limited to theoretical examples but demonstrates tangible benefits in addressing challenges faced by software development teams. Secondly, the framework's flexible architecture, designed for seamless integration with various advanced-level Generative LLMs, ensures its scalability. This adaptability future-proofs the framework, allowing it to evolve alongside advancements in language models and handle increasingly complex requirements analysis tasks. Lastly, the conscious effort to prioritize cost-efficiency through the utilization of open-source models and services makes the framework accessible to a wider range of organizations. This focus on affordability, without compromising performance, makes the framework a viable solution for both resource-constrained and well-equipped development teams.

## 6 Threats to validity

The main threats to the validity of this research may be summarized as follows.

### 6.1 Threats to construct validity

The proposed framework involves two autonomous agents, each initiated through prompt engineering. While the prompts utilized in this work are carefully curated, there remains a

**Table 2** Evaluation of RAG with different LLMs. Metrics - Context Precision (CP), Context Recall (CR), Answer Relevancy (AR), Faithfulness (FF)

| Dataset | LLaMA-2 Scores | | | | Mixtral 8x7B Scores | | | | ChatGPT Scores | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CP | CR | AR | FF | CP | CR | AR | FF | CP | CR | AR | FF |
| E-store | 0.82 | 0.80 | 0.86 | 0.84 | 0.90 | 0.89 | 0.91 | 0.91 | 0.88 | 0.88 | 0.91 | 0.92 |
| CCNTS | 0.84 | 0.84 | 0.86 | 0.86 | 0.91 | 0.90 | 0.92 | 0.91 | 0.92 | 0.91 | 0.91 | 0.93 |
| Tachonet | 0.84 | 0.83 | 0.85 | 0.87 | 0.89 | 0.88 | 0.91 | 0.90 | 0.89 | 0.89 | 0.91 | 0.92 |
| Video Search | 0.83 | 0.82 | 0.86 | 0.86 | 0.90 | 0.90 | 0.91 | 0.91 | 0.91 | 0.90 | 0.93 | 0.93 |
| E-procurement | 0.79 | 0.78 | 0.84 | 0.84 | 0.90 | 0.91 | 0.92 | 0.91 | 0.90 | 0.89 | 0.91 | 0.92 |
| Inventory | 0.84 | 0.82 | 0.85 | 0.84 | 0.91 | 0.91 | 0.92 | 0.93 | 0.89 | 0.89 | 0.90 | 0.91 |
| Znix | 0.84 | 0.84 | 0.86 | 0.87 | 0.89 | 0.90 | 0.90 | 0.91 | 0.90 | 0.91 | 0.91 | 0.92 |

**Table 3** Evaluation of the Framework on Different Usecases

| Usecase | Iterations | Model for Analyzer | Model for Verifier | #incompleteness Detected | #Common Incompleteness | #Inconsistencies Detected | #Common Inconsistencies |
|---|---|---|---|---|---|---|---|
| CCNTS | 1 | LLaMA-2 | ChatGPT | 8 | 8 | 7 | 9 |
| | 2 | LLaMA-2 | ChatGPT | 0 | | 2 | |
| | 1 | ChatGPT | ChatGPT | 7 | | 7 | |
| | 2 | ChatGPT | ChatGPT | 3 | | 5 | |
| | 1 | Mixtral 8x7B | ChatGPT | 6 | | 8 | |
| | 2 | Mixtral 8x7B | ChatGPT | 3 | | 3 | |
| Tachnonet | 1 | LLaMA-2 | ChatGPT | 5 | 7 | 6 | 7 |
| | 2 | LLaMA-2 | ChatGPT | 2 | | 2 | |
| | 1 | ChatGPT | ChatGPT | 6 | | 6 | |
| | 2 | ChatGPT | ChatGPT | 2 | | 3 | |
| | 1 | Mixtral 8x7B | ChatGPT | 7 | | 6 | |
| | 2 | Mixtral 8x7B | ChatGPT | 2 | | 3 | |
| Inventory | 1 | LLaMA-2 | ChatGPT | 8 | 11 | 9 | 10 |
| | 2 | LLaMA-2 | ChatGPT | 3 | | 2 | |
| | 1 | ChatGPT | ChatGPT | 10 | | 8 | |
| | 2 | ChatGPT | ChatGPT | 3 | | 4 | |
| | 1 | Mixtral 8x7B | ChatGPT | 9 | | 8 | |
| | 2 | Mixtral 8x7B | ChatGPT | 4 | | 3 | |
| Znix | 1 | LLaMA-2 | ChatGPT | 6 | 9 | 4 | 8 |
| | 2 | LLaMA-2 | ChatGPT | 3 | | 4 | |
| | 1 | ChatGPT | ChatGPT | 8 | | 6 | |
| | 2 | ChatGPT | ChatGPT | 3 | | 3 | |
| | 1 | Mixtral 8x7B | ChatGPT | 8 | | 5 | |
| | 2 | Mixtral 8x7B | ChatGPT | 4 | | 3 | |

**Table 3** continued

| Usecase | Iterations | Model for Analyzer | Model for Verifier | #incompleteness Detected | #Common Incompleteness | #Inconsistencies Detected | #Common Inconsistencies |
|---|---|---|---|---|---|---|---|
| E-store | 1 | LLaMA-2 | ChatGPT | 8 | 12 | 6 | 11 |
| | 2 | LLaMA-2 | ChatGPT | 4 | | 5 | |
| | 1 | ChatGPT | ChatGPT | 9 | | 8 | |
| | 2 | ChatGPT | ChatGPT | 4 | | 5 | |
| | 1 | Mixtral 8x7B | ChatGPT | 9 | | 11 | |
| | 2 | Mixtral 8x7B | ChatGPT | 5 | | 4 | |
| Video Search | 1 | LLaMA-2 | ChatGPT | 10 | 14 | 7 | 11 |
| | 2 | LLaMA-2 | ChatGPT | 5 | | 4 | |
| | 1 | ChatGPT | ChatGPT | 10 | | 9 | |
| | 2 | ChatGPT | ChatGPT | 6 | | 4 | |
| | 1 | Mixtral 8x7B | ChatGPT | 13 | | 9 | |
| | 2 | Mixtral 8x7B | ChatGPT | 5 | | 4 | |
| E-procurement | 1 | LLaMA-2 | ChatGPT | 6 | 11 | 5 | 9 |
| | 2 | LLaMA-2 | ChatGPT | 5 | | 4 | |
| | 1 | ChatGPT | ChatGPT | 9 | | 7 | |
| | 2 | ChatGPT | ChatGPT | 4 | | 3 | |
| | 1 | Mixtral 8x7B | ChatGPT | 11 | | 7 | |
| | 2 | Mixtral 8x7B | ChatGPT | 3 | | 4 | |

possibility that they do not constitute the complete set of efficient prompts. It is conceivable that more optimal prompts exist, which could enhance the performance and efficacy of the framework. This potential limitation poses a threat to the construct validity of the framework's outcomes.

### 6.2 Threats to internal validity

LLMs are susceptible to extrinsic hallucination, characterized by the generation of unverifiable text, potentially leading to inaccurate suggestions. Despite employing RAG-based approaches and re-ranking mechanisms to enhance the accuracy and reliability of LLM-generated responses, there remains a risk of extrinsic hallucination occurring in LLMs.

### 6.3 Threats to external validity

The performance of the framework heavily relies on the underlying language models. Biases present in these models, stemming from training data or design choices, may lead to biased outputs or inaccurate analyses, undermining the validity of the framework's results.

## 7 Related works

The following literature review integrates recent research endeavors in the domain of leveraging NLP techniques and language models for enhancing requirements analysis. Research by Ferrari and Esuli [10] proposes a natural language processing approach to identify and rank ambiguous terms between diverse technical domains during requirements elicitation. Evaluation across multiple scenarios yielded promising results, indicating potential for further research on the interplay between domain knowledge and linguistic ambiguity. Another research [9] introduces an automated NLP-based approach for addressing requirement ambiguity, leveraging domain-specific corpora to improve accuracy. Evaluation across 20 industrial documents shows substantial enhancements in ambiguity detection and interpretation, surpassing generic corpora baselines by 33% in detection and 16% in interpretation accuracy. Arora et. al. [4] evaluates domain models' effectiveness in detecting incompleteness in natural-language requirements, finding near-linear sensitivity to both unspecified and under-specified requirements. Results suggest domain models offer valuable cues for identifying omissions, prompting further investigation into analysts' ability to leverage these cues for incompleteness detection. In another research [1], an automated question-answering (QA) approach is proposed that uses large-scale language models to extract compliance-related information from regulatory documents, achieving high effectiveness with 94% accuracy in locating relevant text passages and 91% accuracy in identifying correct answers. Another work [8] introduces QAssist, a question-answering (QA) approach that automates assistance for stakeholders, particularly requirements engineers, in analyzing natural language requirements, integrating external domain knowledge for comprehensive answers. Another research [2] proposes an AI-based automation for completeness checking of privacy policies, leveraging a conceptual model and completeness criteria derived from GDPR provisions. Through natural language processing and supervised machine learning, the approach achieves high precision (92.9%) and recall (89.8%) in detecting completeness violations, outperforming keyword search baselines by 24.5% in precision and 38% in recall. Recent work proposed by Luitel et.al. [19] investigates the utility of language models, particularly BERT, in detecting incompleteness in natural language requirements. It introduces an approach leveraging BERT's masked language model to predict missing terminology, and proposes a machine learning-based filter to enhance prediction accuracy, demonstrating the effectiveness of BERT in pinpointing missing terms and the potential of the filter to reduce noise in predictions.

Table 4 summarizes the comparative analysis of our proposed framework with existing approaches. Existing research has made significant strides in leveraging NLP techniques for requirements analysis. However, they often fall short in addressing the interconnected challenges of incompleteness, inconsistency, and the need for external knowledge integration. Existing automated approaches tend to be rule-based, focusing on specific defect types or lacking the adaptability to handle the nuances of natural language requirements. Moreover, they often operate in isolation, failing to leverage the vast amount of external knowledge available, which is crucial for comprehensive analysis. In contrast, our proposed framework, ARIA, introduces a novel agent-based approach that combines question-driven analysis, real-time external knowledge integration through RAG, and iterative refinement through agent collaboration, directly addressing these limitations and paving the way for a more robust and intelligent requirements analysis process.

## 8 Conclusion

In this research, we have presented a comprehensive framework for enhancing requirements analysis through the integration of large language models (LLMs) and AI-driven agents. Our framework leverages state-of-the-art techniques, including prompt engineering, semantic routing, and multi-stage re-ranking mechanisms, to facilitate efficient and effective analysis of software requirements specifications. We have demonstrated the efficacy of our framework in identifying and addressing incompleteness and inconsistencies

**Table 4** Comparison of existing requirements analysis methods with our proposed framework for detecting incompleteness and inconsistency

| Method/ Framework | Methodology | AI Used? | Iterative Refinement Considered? | Evaluation Strategy | Limitations |
|---|---|---|---|---|---|
| QAssist [8] | Employs a question-answering approach with external knowledge integration for analyzing natural language requirements | Yes | No | Not explicitly detailed in the paper | Scope of inconsistency detection |
| Arora et al. [4] | Uses of domain models for detecting incompleteness in natural language requirements | Yes | No | Evaluated the effectiveness of domain models in identifying omissions within requirements | Scope of incompleteness detection |
| Luitel et al. [19] | Employs BERT to predict missing terminology in requirements | Yes | No | Evaluated the effectiveness of BERT in identifying missing terms | Focuses only on missing terminology |
| Ferrari and Esuli [10] | NLP-based approach to identify and rank ambiguous terms in requirements specifications | Yes | No | Evaluated on multiple cross-domain scenarios | Primarily addresses ambiguity, not consider completeness or consistency detection |
| Ezzini et al. [9] | Automated NLP approach to detect and interpret ambiguous requirements using domain-specific corpora | Yes | No | Evaluated on 20 industrial documents | Focuses only on ambiguity resolution |
| **ARIA-QA** (Proposed Framework) | Uses AI agents with RAG and prompt engineering to analyze natural language requirements for incompleteness and inconsistency, leveraging web searches for domain knowledge | Yes | Yes, through Verifier Agent feedback | Evaluated on 7 use cases from PURE dataset using RAGAS framework and LLM judges | Reliant on LLM capabilities, potentially susceptible to LLM biases |

within requirements documents. Real-life use cases sourced from the PURE dataset served as valuable testbeds, showcasing the framework's applicability and relevance in practical scenarios. The framework can seamlessly integrate with various LLMs, offering flexibility for future enhancements and advancements in language model technology. Moreover, our cost-efficient infrastructure, combined with open-source models and services, makes the framework accessible and practical for organizations with diverse needs and budgets. Looking ahead, it will be crucial to assess the framework's robustness and applicability across diverse scenarios through extensive evaluation on real-life use cases. Additionally, we envision the framework as an efficient open-source tool, providing accessible and practical solutions to empower software development teams in tackling complex requirements analysis challenges.

**Acknowledgements** Work partially supported by SERICS(PE00000014) and iNEST (ECS 00000043) projects funded by PNRR NextGeneration EU.

## References

1. Abualhaija S, Arora C, Sleimi A, Briand LC (2022) Automated question answering for improved understanding of compliance requirements: A multi-document study. In: 2022 IEEE 30th International Requirements Engineering Conference (RE). pp. 39–50. https://doi.org/10.1109/RE54965.2022.00011
2. Amaral O, Abualhaija S, Torre D, Sabetzadeh M, Briand LC (2021) Ai-enabled automation for completeness checking of privacy policies. IEEE Trans Software Eng 48(11):4647–4674
3. Anthropic A (2024) The claude 3 model family: Opus, sonnet, haiku. Claude-3 Model Card
4. Arora C, Sabetzadeh M, Briand LC (2019) An empirical study on the potential usefulness of domain models for completeness checking of requirements. Empir Softw Eng 24:2509–2539
5. Chen J, Xiao S, Zhang P, Luo K, Lian D, Liu Z (2024) Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv preprint arXiv:2402.03216
6. Dalpiaz F, Van der Schalk I, Lucassen G (2018) Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In: Requirements Engineering: Foundation for Software Quality: 24th International Working Conference, REFSQ 2018, Utrecht, The Netherlands, March 19-22, 2018, Proceedings 24. pp. 119–135. Springer
7. Es S, James J, Espinosa-Anke L, Schockaert S (2023) Ragas: Automated evaluation of retrieval augmented generation. arXiv preprint arXiv:2309.15217
8. Ezzini S, Abualhaija S, Arora C, Sabetzadeh M (2023) Ai-based question answering assistance for analyzing natural-language requirements. arXiv preprint arXiv:2302.04793
9. Ezzini S, Abualhaija S, Arora C, Sabetzadeh M, Briand LC (2021) Using domain-specific corpora for improved handling of ambiguity in requirements. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 1485–1497. IEEE
10. Ferrari A, Esuli A (2019) An nlp approach for cross-domain ambiguity detection in requirements engineering. Autom Softw Eng 26(3):559–598
11. Ferrari A, Spagnolo GO, Gnesi S (2017) Pure: A dataset of public requirements documents. In: 2017 IEEE 25th International Requirements Engineering Conference (RE). pp. 502–505. IEEE
12. Hadar I, Zamansky A, Berry DM (2019) The inconsistency between theory and practice in managing inconsistency in requirements engineering. Empir Softw Eng 24(6):3972–4005
13. Jégou H, Douze M, Johnson J, Hosseini L, Deng C (2022) Faiss: Similarity search and clustering of dense vectors library. Astrophysics Source Code Library pp. ascl–2210
14. Jiang AQ, Sablayrolles A, Roux A, Mensch A, Savary B, Bamford C, Chaplot DS, Casas Ddl, Hanna EB, Bressand F, et al (2024) Mixtral of experts. arXiv preprint arXiv:2401.04088
15. Kevian D, Syed U, Guo X, Havens A, Dullerud G, Seiler P, Qin L, Hu B (2024) Capabilities of large language models in control engineering: A benchmark study on gpt-4, claude 3 opus, and gemini 1.0 ultra. arXiv preprint arXiv:2404.03647
16. Lamsweerde Av (2009) Requirements engineering: from system goals to UML models to software specifications. John Wiley & Sons, Ltd
17. Retrieval-augmented generation for knowledge-intensive nlp tasks (2020) Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al. Adv Neural Inf Process Syst 33:9459–9474
18. Lin CY (2004) Rouge: A package for automatic evaluation of summaries. In: Text summarization branches out. pp. 74–81
19. Luitel D, Hassani S, Sabetzadeh M (2023) Using language models for enhancing the completeness of natural-language requirements. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 87–104. Springer
20. Muennighoff N, Tazi N, Magne L, Reimers N (2022) Mteb: Massive text embedding benchmark. arXiv preprint arXiv:2210.07316
21. Nogueira R, Yang W, Cho K, Lin J (2019) Multi-stage document ranking with bert. arXiv preprint arXiv:1910.14424
22. OpenAI: Gpt-4 technical report. In: GPT-4 Technical Report (2023)
23. Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A et al (2022) Training language models to follow instructions with human feedback. Adv Neural Inf Process Syst 35:27730–27744
24. Papineni K, Roukos S, Ward T, Zhu WJ (2002) Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. pp. 311–318
25. Parnami A, Lee M (2022) Learning from few examples: A summary of approaches to few-shot learning. arXiv preprint arXiv:2203.04291
26. Platzer C, Dustdar S (2005) A vector space search engine for web services. In: Third European Conference on Web Services (ECOWS'05). pp. 9–pp. IEEE
27. Romera-Paredes B, Torr P (2015) An embarrassingly simple approach to zero-shot learning. In: International conference on machine learning. pp. 2152–2161. PMLR
28. Stata R, Bharat K, Maghoul F (2000) The term vector database: fast access to indexing terms for web pages. Comput Netw 33(1–6):247–255
29. Thoppilan R, De Freitas D, Hall J, Shazeer N, Kulshreshtha A, Cheng HT, Jin A, Bos T, Baker L, Du Y, et al (2022) Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239
30. Touvron H, Martin L, Stone K, Albert P, Almahairi A, Babaei Y, Bashlykov N, Batra S, Bhargava P, Bhosale S, et al (2023) Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288
31. Van Lamsweerde A (2000) Requirements engineering in the year 00: A research perspective. In: Proceedings of the 22nd international conference on Software engineering. pp. 5–19

32. Wang A, Pruksachatkun Y, Nangia N, Singh A, Michael J, Hill F, Levy O, Bowman S (2019) Superglue: A stickier benchmark for general-purpose language understanding systems. Adv Neural Inf Process Syst **32**
33. Wang W, Zheng VW, Yu H, Miao C (2019) A survey of zero-shot learning: Settings, methods, and applications. ACM Trans Intell Syst Technol (TIST) 10(2):1–37
34. Zhao L, Alhoshan W, Ferrari A, Letsholo KJ, Ajagbe MA, Chioasca EV, Batista-Navarro RT (2021) Natural language processing for requirements engineering: A systematic mapping study. ACM Comput Surv (CSUR) 54(3):1–41
35. Zheng L, Chiang WL, Sheng Y, Zhuang S, Wu Z, Zhuang Y, Lin Z, Li Z, Li D, Xing E, et al (2024) Judging llm-as-a-judge with mt-bench and chatbot arena. Adv Neural Inf Process Syst **36**