Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 10, No. 3, 585-592
2026
Publisher: Learning Gate
DOI: 10.55214/2576-8484.v10i3.12519

Software requirements analysis with large language model based artificial intelligence aid

Han Seong Son[1]*
[1]Division of Software Engineering, Joongbu University, 305 Donghun-ro, Goyang-si, Gyeonggi-do, 10279, Republic of Korea; hsson@joongbu.ac.kr (H.S.S.).

Abstract: Safety-critical systems (SCSs) are systems where failure can lead to catastrophic consequences, including loss of life, environmental damage, and significant economic loss. In such contexts, requirements analysis is a foundational phase that significantly affects the safety, reliability, and verifiability of the system. However, traditional practices for analyzing requirements often struggle with inefficiencies, ambiguity, and human error. With the advent of advanced artificial intelligence (AI) technologies—especially natural language processing (NLP), machine learning (ML), and large language models (LLMs)—novel approaches are emerging to automate and enhance the precision and scalability of requirements analysis. This work proposes a novel software requirements analysis method, which utilizes a real-time service system that enables asking questions and receiving answers about PDF documents. The service system involves document loading, chunking, embedding, storage in a vector database, and retrieval using semantic similarity. The final response is generated using an LLM. An experimental study has been performed to demonstrate the feasibility and effectiveness of the proposed method.
Keywords: *AI-assisted software development, Automated requirement extraction, Large language models (LLMs), Natural language processing (NLP), Requirements engineering.*

1. Introduction

Safety-critical systems are extensively used in domains such as aviation, automotive, nuclear energy, and medical devices. Due to their potential to cause severe harm upon failure, these systems demand rigorous engineering practices. Among them, requirements analysis plays a central role, as early-stage errors in requirement specifications can propagate through the lifecycle, leading to dangerous system behavior and costly redesigns. Defects identified during the requirements phase are significantly less costly and safer to correct than those found during later stages, such as implementation or operation. Inadequate or ambiguous requirements have been cited as major contributors to accidents involving SCSs [1]. Safety requirements are often derived from system-level hazard analyses and must be traceable back to identified hazards [2]. Safety standards such as DO-178C (aerospace), ISO 26262 (automotive), and IEC 62304 (medical) require that requirements be complete, traceable, and verifiable. Non-compliance with these standards due to poor requirements can delay certification and endanger deployment [3-5]. SCSs must operate in a predictable and deterministic manner. Ambiguous or incomplete requirements introduce uncertainty and unexpected system behavior under edge cases, undermining safety guarantees [6].

Requirements analysis traditionally involves manual interpretation of natural language documents by analysts and stakeholders. While essential, this process is often inefficient, error-prone, and unable to scale effectively for large systems. To address these limitations, researchers and practitioners are increasingly applying AI techniques to automate and optimize requirements-related tasks. The integration of AI promises improvements in accuracy, consistency, and traceability, and it supports

History: Received: 16 January 2026; Revised: 2 March 2026; Accepted: 5 March 2026; Published: 25 March 2026
* Correspondence: hsson@joongbu.ac.kr

compliance with software engineering standards and safety regulations. Recent research has leveraged LLMs such as GPT and BERT to assess the quality of natural language requirements. These models can identify ambiguity, inconsistency, incompleteness, and duplication based on linguistic and semantic features. Zhang [7] proposed the use of LLMs to automatically detect ambiguous software requirements with high precision Zhang [7]. Son [8] proposed a requirements analysis method using LLMs and a prompt pattern for detecting missing requirements and minimizing sets of requirements [8]. LLMs are also being used to generate formal requirements from unstructured inputs such as user stories, system use cases, or domain descriptions. Fujita et al. proposed a GPT-based pipeline for transforming user stories into formal requirements suitable for system design and verification [9]. AI models can help establish traceability links between requirements, design models, test cases, and source code. Semantic similarity measures powered by neural embeddings or LLMs are used to automatically map related artifacts. Microsoft's DeepDevTrace project aims to connect requirements and implementation using transformer-based embeddings [10].

This work proposes a novel software requirements analysis method, which utilizes a real-time service system that enables asking questions and receiving answers about PDF documents. To demonstrate the feasibility and effectiveness of the proposed method, an experimental study has been performed. In the study, the proposed method was applied to a part of the safety-critical software requirements of the Over Current Relay. The author organized the rest of this paper as follows: The proposed requirements analysis method is described in Section 2. The results of the experimental study are presented in Section 3. Finally, Section 4 summarizes the findings from the application and proposes potential future work.

2. Proposed Requirements Analysis Method

The core of the proposed software requirements analysis method is a real-time question-and-answer system based on PDF documents, called ChatPDF. This system enables answering user questions based on the contents of a PDF document. The process, shown in Figure 1, involves document loading, chunking, embedding, storage in a vector database, and retrieval using semantic similarity. The final response is generated using a large language model (LLM) such as OpenAI's GPT-4.

DOI: 10.55214/2576-8484.v10i3.12519

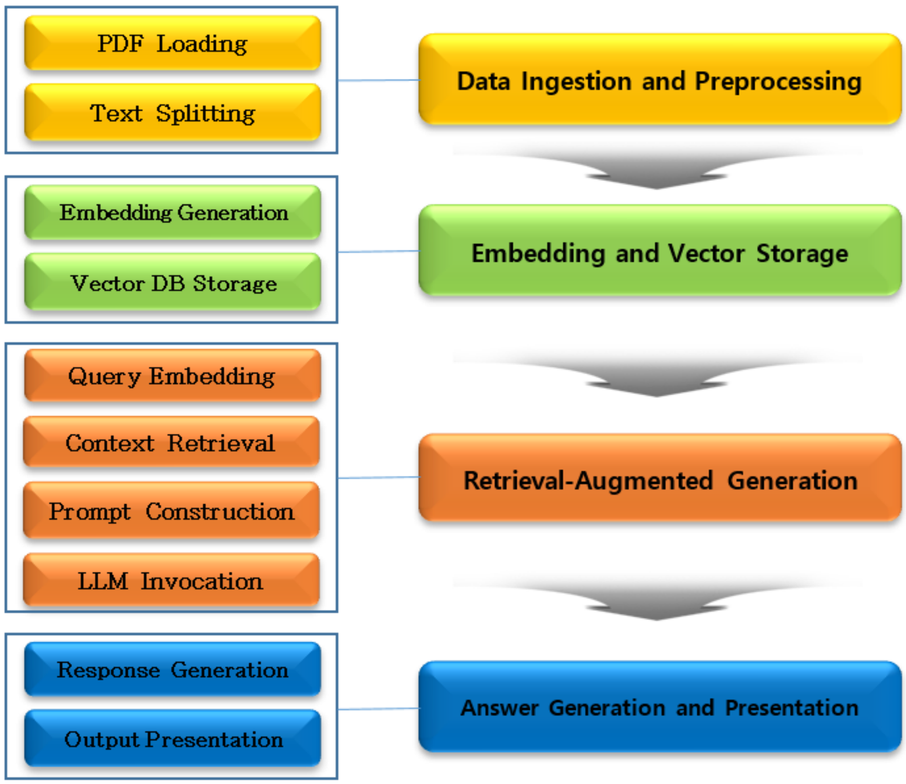

Figure 1.
Proposed Requirements Analysis Process.

2.1. Data Ingestion and Preprocessing

2.1.1. PDF Loading

The PyPDFLoader from LangChain is used to extract and convert the content of your PDF files into a format suitable for further processing [11]. PyPDFLoader is one of the most commonly used document loaders in the LangChain ecosystem. It is specifically designed to transform local PDF files into a format that LangChain can process—essentially converting static pages into a list of Document objects.

Each Document object created by the loader contains the following two main parts:

1. page_content: The raw text extracted from the PDF page.
2. metadata: A dictionary containing the source file name and the page number.

2.1.2. Text Splitting (Chunking)

The PyPDFLoader uses the ‘pypdf’ library under the hood. It follows a simple "Load and Split" logic where each page of the PDF is treated as a separate document. Because LLMs have token limitations, large PDF files can’t be processed in one piece. Therefore, the extracted content must be divided into smaller, manageable segments. A common solution is LangChain’s RecursiveCharacterTextSplitter, which enables setting the chunk size and overlap so that context is preserved between segments [12].

Chunk size is the hard limit on how large a single piece of text can be. Small chunks (e.g., 300) are great for specific fact retrieval but might lose the big picture. Large chunks (e.g., 2000) are better for summarization but might include irrelevant data.

Chunk overlap is the most important setting for maintaining contextual continuity. By overlapping chunks (e.g., 10% or 15%), you ensure that if a vital piece of information is split at the boundary, it

ISSN: 2576-8484
Vol. 10, No. 3: 585-592, 2026
DOI: 10.55214/2576-8484.v10i3.12519

appears in both chunks. This prevents the "semantic break" where a model doesn't understand what "it" refers to because the noun was in the previous chunk.

When PyPDFLoader splits a document, it intelligently clones the metadata. If Page 1 is split into 3 chunks, all 3 chunks will still have {"source": "manual.pdf", "page": 0} in their metadata. This allows tracing any specific chunk back to its exact origin in the PDF.

2.2. Embedding and Vector Storage

2.2.1. Embedding Generation

The divided text segments are transformed into numerical vectors—called embeddings—by an embedding model such as those provided by OpenAI or Hugging Face. These vectors represent the underlying semantic meaning of the content [13]. Let us assume that standard search engines look for the word "cat." Embedding generation allows the system to understand that "feline" or "kitten" are conceptually similar. For the vector representation, each chunk of text is converted into a list of numbers (e.g., a vector of 1,536 dimensions for OpenAI's text-embedding-3-small). In a vector space, two chunks with similar meanings are geometrically "close" to each other (calculated using Cosine Similarity). After that, the system sends these text strings to an embedding model (like OpenAI, Cohere, or an open-source model like Hugging Face's all-MiniLM-L6-v2). The model returns a vector for each chunk.

2.2.2. Vector Database Storage

The generated embeddings and their corresponding text segments are stored in a vector database (e.g., ChromaDB, FAISS, Pinecone). This enables efficient similarity searches for retrieval [14]. Vector databases do not just "hold" data but organize it using complex mathematical structures called indices. This allows ChatPDF to find the right answer in milliseconds, even if the PDF is thousands of pages long.

Instead of checking every single vector (which would be slow), the database uses algorithms like HNSW (Hierarchical Navigable Small World) to create a "map" of vectors. It jumps through the map to find the closest neighbors to your query. If an OpenAI model is used, each entry in the database is a list of 1,536 numbers. The database manages these high-dimensional points in a way that preserves their semantic relationships.

When we upload a PDF, the vectors are 'upserted' (updated and inserted) into the index. Unlike a Python list, these databases save to disk or the cloud so that ChatPDF remembers the files even after the application is restarted. When a question is asked, the database returns the Top K (usually the top 4 or 5) most similar text chunks to the LLM.

One of the most powerful features of Vector Storage is Hybrid Search or Metadata Filtering. If we have 100 PDFs uploaded and ask: "What were the profits in the 2024 report?", the Vector Database does not just look for "profits." It can filter the search to only examine chunks where the metadata 'year == 2024'. This significantly reduces hallucinations by ignoring irrelevant files.

2.3. Retrieval-Augmented Generation (RAG)

2.3.1. Query Embedding

When a user submits a question, it is converted into an embedding using the same model that was applied to the documents [15]. Query Embedding is the "bridge" that connects a user's natural language question to the raw data stored in the Vector Database. While the PDF was converted into vectors during the ingestion phase, the user's query must undergo the exact same transformation to make a mathematical comparison possible.

The most critical rule of Query Embedding is model consistency. To find a match, the query must be converted into the same multi-dimensional "coordinate system" as the PDF. If the PDF was embedded using OpenAI's text-embedding-3-small (1,536 dimensions) and the query is embedded using Google's Gecko model, the resulting vectors will be in different mathematical spaces. It would be like trying to find a GPS coordinate from Earth on a map of Mars. Thus, LangChain and other frameworks ensure that the same embedding object used for storage is called during the retrieval phase.

When a user types "What is the warranty period for this product?", the following happens in the background:

1. Preprocessing: The raw string is cleaned (removing unnecessary whitespace or special characters).
2. Inference: The string is sent to the Embedding Model API.
3. Vector Generation: The model returns a single high-dimensional vector (a list of floating-point numbers) representing the intent and semantics of the question.
4. Similarity Search: This "Query Vector" is sent to the Vector Database to find the k-nearest neighbors (the most similar chunks from the PDF).

2.3.2. Context Retrieval

Context retrieval is the process of selecting the most relevant snippets from your PDF and presenting them to the LLM so it can answer your question accurately. A similarity search in the vector database is performed using the query embedding to retrieve the most relevant text chunks (context) from the stored PDF content [16]. Without retrieval, an LLM would have to guess the answer based on its general training data. With retrieval, the LLM becomes an "open-book" student, looking directly at the provided text chunks.

The retrieval process happens in a fraction of a second after you submit your query. It follows a mathematical logic to bridge the gap between your question and the PDF data:

1. Similarity score calculation: The system compares your query vector against all stored document vectors in the database.

2. Top-k Selection: The database ranks each chunk based on how close it is to the query using Cosine Similarity. It typically selects the top 3 to 5 most relevant chunks; this number is known as k.

3. Context Construction: These k chunks are pulled from the database and bundled into a single block of text called the context.

2.3.3. Prompt Construction

The retrieved context is merged with the user's query to create a well-organized prompt for the Large Language Model (LLM) [17]. Prompt construction is the art of "assembling" the user's question, the retrieved document snippets, and specific instructions into a single coherent message that the LLM can understand. A well-constructed prompt usually consists of four distinct components merged into one long string: system role, context block, constraints, and user query.

2.3.4. LLM Invocation

LLM Invocation is the final "execution" step in a ChatPDF system. After the PDF has been loaded, split, embedded, and retrieved, the Invocation phase is where the Large Language Model (like GPT-4, Claude 3.5, or Gemini 1.5) processes the assembled prompt and generates a human-readable response.

In LangChain, this is the moment the "Chain" or "Graph" sends the final payload to the model provider's API. When the LLM is "invoked," it does not just see your question, but it receives the entire Constructed Prompt (System instructions + Retrieved Context + User Query). The prompt is converted into a format the API understands (usually a JSON object containing a list of "messages"). The request is sent via HTTPS to the model provider (OpenAI, Anthropic, etc.) or a local inference engine (Ollama, vLLM). The model converts the text into tokens (the basic units of text it processes). The model predicts the next tokens in the sequence, based on the patterns found in the "Context" provided.

DOI: 10.55214/2576-8484.v10i3.12519

2.4. Answer Generation and Presentation

2.4.1. Response Generation:

Response Generation is the final, user-facing stage of a ChatPDF system. It is the creative process where the LLM takes the raw, fragmented pieces of information found during "Retrieval" and weaves them into a fluid, natural language answer. While "LLM Invocation" is the technical act of calling the API, Response Generation refers to the logic and strategy used to ensure the final output is helpful, accurate, and easy for the user to read.

The LLM does not simply copy-paste text from the PDF. Instead, it performs synthesis. If the answer to a question requires information from both Page 5 (a price list) and Page 20 (a discount policy), the response generation phase combines these two disparate facts into a single, logical sentence. The model uses its internal logic to interpret the retrieved chunks. For example, if a PDF chunk says "Interest rates are up 2%," and the user asks "How is the economy doing?" the model generates a response interpreting that 2% increase as a sign of inflation or central bank policy.

2.4.2. Output Presentation

The generated answer is displayed to the user, potentially including source references or citations to the original PDF chunks for transparency and verification. Output presentation is the final layer of the ChatPDF user experience. It is where the mathematical results of the AI are translated into a visually organized, interactive, and trustworthy interface for the human user.

The most effective way to present ChatPDF outputs is through a split-screen or "slot-based" layout. A conversational window where the AI provides summaries, answers, and analysis. A live PDF renderer (using libraries like react-pdf or PDF.js) is provided. When the AI gives an answer, the document viewer automatically scrolls to the specific page or highlights the exact paragraph referenced.

3. Experimental Application

An experimental study has been conducted to demonstrate the feasibility and effectiveness of the proposed method. The method has been applied to a part of safety-critical software requirements. These requirements describe the behaviors of an Over Current Relay, a protective device that operates solely based on current without the need for a voltage coil:

- Based on active power for each phase, reactive power for each phase, apparent power for each phase, unbalanced current, unbalanced voltage, reactive positive, and reactive negative energy for each phase are calculated.
- Calculated values are saved as reactive positive and reactive negative energy.
- With the calculated reactive energies for each phase, the total reactive energy is calculated.
- By summing the calculated reactive energies for each phase, the Apparent Energy for each phase is calculated.
- With the calculated apparent energies for each phase, the total apparent energy is calculated.
- The calculated values are displayed as follows:
- P Type

Total Active Energy

- H Type

Reactive Positive Energy and Reactive Negative Energy for Each Phase

These requirements were input into ChatPDF, and the requirements analysis was performed by asking questions. The first question checked the consistency between functional and interface requirements. It asked whether all final outputs on interface requirements need to be processed in functional requirements. The question and the generated answer are as follows:

Question: "While it is specified that the total active energy is displayed on the screen, is there also a specification indicating that the total active energy is computed?"

Answer: "There is no description indicating that the total active energy is calculated."

According to the above question and answer, ChatPDF proves useful for verifying whether all final outputs on interface requirements are consistently required to be processed on function requirements. However, ChatPDF has a limitation in that it cannot generate accurate answers to abstract questions. The exemplar question and its corresponding answer are as follows:

Question: "Does the specification state that a variable which has not been computed is eventually output?"

Answer: "The specification does not state that a variable which has not been computed is eventually output."

The expression 'a variable which has not been computed' makes the question abstract, the generated answer is incorrect. The following question was asked to verify the correctness of functional requirements. It concerns whether the calculation algorithm is correct. The question and the generated answer are as follows:

Question: "Does the specification indicate that apparent energy is computed by summing active and reactive energy?"

Answer: "The specification does not state that apparent energy is computed by combining active and reactive energy. Instead, it describes a process in which apparent energy is derived by summing the reactive energy across all phases. Thus, there is no information indicating that active energy is involved in the calculation."

According to the above question and answer, ChatPDF has proven useful for verifying whether calculation algorithms are correctly described. The following question was asked to check the completeness of functional requirements. It concerns whether all functional outputs are required to be processed. The question and the generated answer are as follows:

Question: "Does the specification state that active positive energy is calculated?"

Answer: "It is not specified that active positive energy is computed. Instead, the process of calculating reactive positive energy and reactive negative energy for each phase, then computing the total energy, is described."

According to the above question and answer, ChatPDF has proved to be useful in verifying whether functional requirements are completely described or not.

4. Concluding Remarks

Since inconsistent, incorrect, and incomplete requirements may cause safety-critical failure of a software system, this work proposes a requirements analysis method to detect such issues using an LLM-based AI system. The AI system is a real-time question-and-answer system based on PDF documents, called ChatPDF. This system enables answering user questions based on the contents of a PDF document. The process involves document loading, chunking, embedding, storage in a vector database, and retrieval using semantic similarity. The final response is generated using an LLM.

An experimental study has been performed to demonstrate the feasibility and effectiveness of the proposed method. The proposed method has been applied to a part of safety-critical software requirements, which describe the behaviors of the Over Current Relay. Through the experimental study, ChatPDF has been proven to be useful to verify whether all the final outputs on interface requirements are consistently required to be processed on function requirements or not, whether calculation algorithms are correctly described or not, and whether functional requirements are completely described or not.

Accurate and comprehensive requirement specifications are critical inputs for hazard analysis procedures; thus, established hazard analysis methodologies such as FMEA, STPA, and HAZOP should incorporate the findings derived from this work. Accordingly, future research efforts will focus on applying the proposed requirements analysis method to enhance the rigor and effectiveness of hazard identification and mitigation strategies.

Transparency:

The author confirms that the manuscript is an honest, accurate, and transparent account of the study; that no vital features of the study have been omitted; and that any discrepancies from the study as planned have been explained. This study followed all ethical practices during writing.

Copyright:

References

[1] N. G. Leveson, *Safeware: System safety and computers.* Reading, MA: Addison-Wesley, 1995.
[2] National Aeronautics and Space Administration, *NASA systems engineering handbook (Rev. 2) (NASA/SP-2016-6105 Rev. 2).* Washington, DC: National Aeronautics and Space Administration, 2016.
[3] RTCA, *DO-178C: Software considerations in airborne systems and equipment certification.* Washington, DC: RTCA, Inc, 2011.
[4] International Organization for Standardization, *ISO 26262-8: Road vehicles—Functional safety—Part 8: Supporting processes.* Geneva, Switzerland: International Organization for Standardization, 2018.
[5] International Electrotechnical Commission, *IEC 62304: Medical device software—Software life cycle processes.* Geneva, Switzerland: International Electrotechnical Commission, 2006.
[6] D. L. Parnas and J. Madey, "Functional documentation for computer systems engineering," *Science of Computer Programming*, vol. 25, no. 1, pp. 41–61, 1991.
[7] Y. Zhang, "Detecting ambiguous requirements with large language models," in *Proceedings of the IEEE International Requirements Engineering Conference (RE). IEEE*, 2023.
[8] H. S. Son, "Using large language models in software requirements analysis," 2025.
[9] H. Fujita, "From user stories to formal requirements using GPT-based pipelines," in *Proceedings of the ACM SIGSOFT FSE Workshop*, 2023.
[10] Microsoft Research, "DeepDevTrace project," 2023. https://www.microsoft.com/en-us/research/
[11] LangChain Overview, "LangChain: Python tutorials and documentation," 2026. https://python.langchain.com/docs/tutorials/
[12] LangChain: Text Splitters Documentation, "In LangChain docs," 2026. https://python.langchain.com/docs/concepts/text_splitters/?utm_source=chatgpt.com
[13] LangChain, "Embedding models (concepts).", 2026. https://python.langchain.com/docs/concepts/embedding_models/
[14] LangChain, "Vector store integrations," 2026. https://js.langchain.com/docs/integrations/vectorstores/
[15] LangChain, "How to embed text," 2026. https://js.langchain.com/docs/how_to/embed_text/
[16] F. Souto, "Transforming RAG systems with enhanced context (a LangChain implementation)," 2024. https://fernandosouto.dev/blog/contexutal-retrieval/
[17] C. Y. Wijaya, "Prompt engineering patterns for successful RAG implementations. Machine Learning Mastery," 2025. https://machinelearningmastery.com/prompt-engineering-patterns-successful-rag-implementations/

ISSN: 2576-8484
Vol. 10, No. 3: 585-592, 2026
DOI: 10.55214/2576-8484.v10i3.12519