information

MDPI

Article

A Framework for Structurally Deterministic Pipeline Based Drafting and Quality Improvement of Software Requirements Specifications Using Language Models and Reinforcement Learning

Muhammad Ali Akhtar [1,*] and Raheela Asif [2]

[1] Department of Computer and Information Systems Engineering, NED University of Engineering and Technology, University Road, Karachi 75270, Pakistan
[2] Department of Computer Science and Information Technology, NED University of Engineering and Technology, University Road, Karachi 75270, Pakistan; rahmed@neduet.edu.pk
* Correspondence: aliakhtar@neduet.edu.pk

Abstract

The process of authoring a Software Requirements Specification (SRS) document is a resource-intensive task in software development that requires coordination among multiple stakeholders and is often time-consuming, costly, and prone to human error. Latest advancements in artificial intelligence have enabled the generation of specification documents using Large Language Models (LLMs). However, such approaches still depend on manual prompt engineering and prompt optimization to extract relevant knowledge and do not consistently ensure structural coherence, completeness, and reliability. This paper presents a systematic approach to SRS generation in which input requirements from stakeholders are classified into semantically meaningful topics, followed by the construction of an initial skeleton document based on these topics. The document is then incrementally expanded using reinforcement learning to improve consistency, completeness, and coverage. The proposed approach also improves the dependability of the output by reducing hallucinations that may arise from the unstructured, raw nature of user inputs. The experimental evaluation of the proposed framework increases topic classification accuracy from 0.40–0.70 to 0.75–0.95 across six requirement topics, thereby improving document structure and generation quality. Compared with a ChatGPT Model 5.2 baseline, the framework achieved significant improvements in key text-generation metrics, including a 19.8% increase in ROUGE-L and an 11.8% increase in METEOR, while maintaining contextual relevance with an average semantic cross-similarity score of 0.615. These results indicate that the proposed method can produce SRS documents that are contextually reliable and structurally coherent while requiring less manual prompts such as requirements from analysts or stakeholders.

Keywords: software requirements specification (SRS); large language models (LLMs); reinforcement learning (RL); topic classification; document skeleton generation; continuous quality improvement (CQI); hallucination control; deterministic automation

Check for updates

Academic Editors: Leon Sterling and Xuan Bach D. Le

Received: 22 May 2026
Revised: 3 July 2026
Accepted: 4 July 2026
Published: 9 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Software engineering requires a software requirements specification (SRS). However, manual preparation of SRS documents is a complex and error-prone process that requires close collaboration between technical and non-technical stakeholders. In particular, SRS

generation with LLMs can reduce development cycles, costs, and improve the uniformity of documents in the case of large and complex projects.

Despite their potential, LLMs' non-deterministic and hallucinatory behavior limits their direct use in structured document generation [1,2]. Fine-tuning enables models to learn structural patterns but does not provide strict output control, whereas few-shot prompting can provide some structural guidance, but predefined templates are necessary. In a production environment, such limitations compromise reliability. Therefore, while LLMs can assist in drafting, structuring, and refining SRS documents, stakeholders must still provide the original requirements, define constraints, validate the generated content, and approve the final specification.

This study proposes a systematic approach for deterministic SRS generation in order to address these issues. Unstructured requirements are categorized into semantically coherent clusters in the first step of the process, known as topic classification. These clusters are then used to generate a document skeleton that is aligned with a predefined Topics Table. The basic structure is gradually expanded using a reinforcement learning-based method inspired by DocGen-RL [3], and structural consistency, completeness, and coverage are improved by continuous quality improvement (CQI) measurements. The main contributions of this study are as follows: (i) A skeleton document generation that uses topic-based chunking and embedding-based clustering. (ii) Incremental document expansion supported by reinforcement learning with CQI integration. (iii) Deterministic approach for creating a reproducible document structure. (iv) Empirical evaluation that improves structural scalability and hallucination control.

Research Questions

To generate Software Requirements Specifications (SRS) automatically while maintaining structural dependability and consistency, this research investigates the use of large language models (LLMs) and reinforcement learning. This study is based on the following research questions.

RQ1: What role can LLMs play in SRS generation, and what are their limitations in production environments?

Particulary, in software development applications, LLMs' inherent non-deterministic behavior poses significant application challenges [4]. The probabilistic nature of LLMs, which estimates the probability of a word or symbol in a context, causes nondeterminism, leading to different responses for identical prompts because of sampling methods such as top-k sampling or nucleus sampling [5,6]. Inconsistent responses might arise from this variation, which are problematic in software development processes where reliability and consistency are required.

RQ2: How does a systematic approach introduce structural determinism to document generation through LLMs?

By nature, LLMs are nondeterministic due to their probabilistic behavior [4]. Setting the temperature parameter to zero increases determinism and makes responses more predictable. However, determinism may still not be absolute when prompts vary [7]. A systematic approach can remove randomness in areas where consistency is required. In document generation, the document structure must remain consistent even when prompts vary. To remove this inconsistency, the LLM generates responses for different sections separately, while the structure generation is handled through another mechanism.

RQ3: How does the use of Language Models improve the SRS generation process?

Language models improve document generation through advanced language understanding and text generation capabilities. Large Language Models are capable of generating

coherent and meaningful documents and have evolved through advancements in natural language processing techniques [8]. These models are also used for document understanding and information extraction, enabling improved document processing workflows [9].

RQ4: How does context affect response generation?

In-context learning allows LLMs to generate responses using relevant contextual information. This study improves the coherence and relevance of generated responses for complex tasks. The proposed system provides the LLM with structured context to improve response generation [10].

RQ5: How does reinforcement learning affect response generation?

LLMs can be fine-tuned using reinforcement learning with a reward function. This allows the model to learn from feedback on its outputs and adjust its parameters to improve performance. The reward function guides the model to generate outputs aligned with the desired results [11,12].

2. Related Works

2.1. Literature Search and Selection Methodology

A structured literature search was conducted to identify studies related to SRS generation, requirements engineering, LLM-based document generation, deterministic structuring, hallucination control, and reinforcement learning-based improvement. Candidate studies were collected from scholarly search engines, academic indexing services, publisher repositories, and recent preprint sources where the work was directly related to this study.

Keywords related to software requirements specification, such as "SRS generation", "requirements engineering", "large language models", "structured document creation", "hallucination control", "deterministic generation", and "reinforcement learning", were used in the literature search. Although previous research from 2010 onward was retained as historical baselines for rule-based, template-based, or interactive SRS automation, studies from 2023 to 2026 were prioritized for LLM-based approaches.

Studies were included when they focused on automated SRS generation, LLM-assisted requirements engineering, structured document generation, document understanding, hallucination reduction, deterministic generation, topic classification, or reinforcement learning for text generation. Studies that were weakly related to requirements engineering or structured document production were excluded. After screening, the selected articles were analyzed according to their focus, method, output, evaluation approach, limitation, and relevance to the proposed framework.

2.2. Role of LLMs in Requirements Engineering

According to recent research, large language models are increasingly being used in requirements engineering because requirements are mostly written in natural language. Zadenoori et al. reviewed recent LLM-based requirements engineering studies and showed that LLMs have been applied to elicitation, validation, classification, and analysis of requirements [13]. However, many studies still depend on prompting and do not provide strong control over output structure and reliability.

Vogelsang and Fischbach discussed the use of LLMs for natural language processing tasks in Requirements Engineering [14]. Their work supports the use of LLMs for requirement understanding and classification. However, it does not focus on complete SRS document generation or deterministic document structure. BERT fine-tuning has also been applied to software requirements classification (SRC), achieving strong results on the PROMISE and FR_NFR datasets [15]. However, that work focuses on classification, while this study connects requirement classification with deterministic SRS structure generation

and incremental expansion. A systematic literature review in [16] focuses on the need for structured and adaptable requirement engineering frameworks capable of balancing business context, project management, and agile development techniques. The significance of predictable and structured SRS generation techniques is further supported by these results.

These studies show that Large Language Models are useful for requirements-related tasks, but they do not fully address deterministic SRS generation. The proposed work extends this direction by combining topic classification, skeleton document generation, incremental expansion, and reinforcement learning-based quality improvement.

2.3. LLM-Based SRS and Structured Document Generation

The application of LLMs to the creation of software requirements specifications has also been investigated in recent research. LLMs can generate useful SRS drafts, according to the evaluation of GPT-4 and CodeLlama by Krishna et al. for SRS document generation [17]. However, the generated structure still depends mainly on the prompt and may change when either the input requirements or the prompt format change.

ReqInOne [18] proposes an LLM-based agent for SRS generation. It divides the generation process into summary generation, requirement extraction, and requirement classification. This modular design improves the quality of generated SRS documents. However, it does not explicitly generate a deterministic document tree or perform CQI-based incremental expansion.

These studies are closely related to the proposed research because they also focus on LLM-based SRS generation. However, the proposed framework differs by separating topic classification, skeleton document generation, and section-wise incremental expansion. This separation helps to keep the generated SRS structure more stable and reproducible.

2.4. Document Intelligence, RAG, Hallucination Control, and RL-Based Refinement

Previous studies explored the creation of synthetic documents through reinforcement learning. The DocGen pipeline [3] creates synthetic documents by using reinforcement learning to optimize queries and improve query-document relevance. Although this approach improves document quality, it does not explicitly address the generation of deterministic structured documents. The proposed framework, on the other hand, emphasizes structured text-based SRS production with explicit structural control.

There are two main ways in which the proposed framework differs from DocGen-RL's flat, single-pass generating method. First, rather than submitting the raw query directly to the policy, K-Means is used to group the input after it has been divided into chunks and transformed into embeddings. Each cluster reflects a unique intent or topic within the user requirements. This converts an unstructured prompt into topic-based chunks that show what the SRS document should cover. Second, the proposed framework introduces the Topics Table, which maps these discovered clusters to the IEEE-style SRS section space. The Topics Table is constructed from the hierarchical JSON representation of SRS documents in the corpus and is merged during inference with a predefined IEEE 830-based skeleton [19]. Cluster centroids are matched with topics table entries using cosine similarity, and the matched sections are then used to retrieve relevant domain chunks for section-wise response generation.

To improve retrieval efficiency, the question-answering approach for long structured documents in [20] segments documents on a per-page basis. It works well for querying. However, it does not address controlled, structured document production.

The extraction of knowledge from LLMs has been optimized through the application of information retrieval techniques [3,21]. Although they do not impose hierarchical

document structure, these methods enhance answer quality through fast optimization and reinforcement learning.

Recent work on document information extraction from visually rich documents [22] highlights challenges such as the absence of layout encoding and grounding mechanisms to avoid hallucination. While these strategies enhance extraction accuracy, they focus mostly on extraction rather than structured production. In order to enhance long-document understanding and extraction performance, recent work on structure-aware generative information extraction has investigated the use of heterogeneous document graphs, semantic alignment, and topology-aware encoding [23]. These approaches show the significance of maintaining structural and semantic relationships in document processing. However, their primary focus is on information extraction rather than deterministic hierarchical document generation for software requirements specifications (SRS). Although RAG-based approaches improve document understanding and retrieval [24], they do not ensure structured and hierarchical document generation, which is addressed in the proposed approach.

2.5. Classical SRS Automation as Historical Baseline

To speed up SRS development, the SRS Automator framework [25] offers grammar checking and organized templates. Likewise, NALASS [26] automates requirements elicitation and specification using predefined syntactic and semantic patterns. In order to improve usability and ease of use, tool-based boilerplate techniques have also been proposed to help identify and translate requirements into standardized SRS statements [27]. However, these systems depend on predefined structures and rule-based methodologies, restricting adaptability. The study in [28] presents an early NALASS-based approach for automatically generating a Software Requirements Specification document. It mainly focuses on the SRS Documentation component, where formalized requirements, Information Objects, attributes, an SRS template, and predefined rules are used to produce a structured natural language SRS document.

The later work in [26] extends this idea by presenting NALASS as a complete Requirements Engineering tool. It supports question generation, requirement formalization, classification, diagrams, use cases, and SRS document generation. To improve requirement analysis, another method [29] suggests automated SRS generation using knowledge graphs and an S/U/C matrix. Although it improves requirement visualization and evaluation, it does not introduce incremental refinement or reinforcement learning-based expansion.

However, both [26,28] depend on predefined questions, templates, and rule-based transformation. They do not use LLM-based generation, topic classification, reinforcement learning, or CQI-based incremental expansion. These older studies are retained as historical baselines because they demonstrate early SRS automation through templates, rules, and user interaction. However, they are not representative of the current LLM-based state of the art and do not provide deterministic LLM-based document generation.

2.6. Synthesis and Research Gap

Overall, existing methods address document generation, extraction, retrieval, or SRS automation individually. However, they lack an integrated deterministic pipeline that combines topic-based structural control, incremental expansion, reinforcement learning optimization, and Continuous Quality Improvement (CQI) for reproducible SRS generation. The proposed framework addresses this gap by introducing structured skeleton generation with reinforcement learning-driven refinement and determinism enforcement. We have identified a significant gap in the existing literature concerning document generation, especially in the context of language models. There is a noticeable absence of research on this important use of language models. For a variety of business applications, the ability to

generate documents that demonstrate both valid content and structure while maintaining consistency for use in production environments is essential.

Previous work, such as [25,26,28], adheres to strictly defined output formats and significantly depends on user interactions. Moreover, they often lack generalizability across diverse types of software. This study bridges this gap by using Large Language Models (LLMs).

Recent approaches to document generation with LLMs use straightforward methods; They may, however, produce outputs with incorrect content and structure. Even minor changes in user requirements might result in significantly different document outputs, leading to inconsistency and nondeterminism, which are unacceptable in production environments [2].

In order to address these challenges, this research introduces a three-step deterministic method that ensures the production of valid and consistent documents. The proposed methodology presents several novel features and innovations, including LLM-based generation, document embedding (Topic Classification step), KNN embedding-to-section classifier (Skeleton Generation step), distance-based relevance function (Response Generation), and RL-based incremental document expansion. These contributions differentiate this study from previous approaches and offer new insights into SRS generation as shown in Table 1.

Table 1. Feature comparison of selected papers based on the identified research gap.

Feature	[20]	[25]	[26]	[28]	[3]	[17]	[18]	[13]	[14]	This Study
LLM-based generation	✓	×	×	×	✓	✓	✓	✓	✓	✓
Determinism	×	Partial	Partial	Partial	×	×	Partial	×	×	✓
SRS document generation	×	✓	✓	✓	×	✓	✓	×	×	✓
Document structure generation	×	✓	✓	✓	×	Partial	Partial	×	×	✓
Generalization for different types of software	Partial	×	×	×	Partial	Partial	Partial	Partial	Partial	✓
Structure control during generation	×	Partial	Partial	Partial	×	Partial	Partial	×	×	✓
Hallucination and reliability handling	Partial	×	×	×	Partial	Partial	Partial	Partial	Partial	✓

3. Methodology

This section explains the methodological framework used in the proposed SRS generating strategy. The process begins with data collection and preparation, followed by topic classification, skeleton document generation, response generation, and reinforcement learning-based refinement. Each stage is designed to transform unstructured user requirements into an SRS document that is organized and semantically aligned while adhering to the target document format. In order to provide reliable, reproducible, and context-aware document generation, the methodology also includes evaluation and quality improvement steps.

Reproducibility means that initially the SRS will be developed from the input of stakeholders. However, improvements provided by the user, analyst, or any other stakeholder can also expand or improve the document in a structurally deterministic way. In short, the stakeholder will use this framework, which uses LLMs, and additional input from any stakeholders can address document improvement through the proposed framework.

3.1. Data Collection and Preparation

PURE (Public Requirements dataset) [30] is a collection of 79 publicly available natural language requirements documents gathered from the web, containing 34,268 sentences. It is designed for NLP tasks in requirements engineering, such as model synthesis, abstraction identification, ambiguity detection, and requirements categorization [31]. The dataset provides all documents in PDF format, along with a subset of 19 documents converted to XML format. However, due to its limited size, additional SRS documents were col-

lected from educational institutions, multidisciplinary corporate sectors, and open public repositories. Only documents with explicit public or open-use permissions, such as the GNU general public license, creative commons licenses, public-domain declarations, or equivalent institutional permission statements, were retained. Documents with unclear, restricted, or unverifiable license status were excluded from the corpus.

The documents were selected carefully and were required to satisfy the following conditions: (1) must be a document, (2) must adhere to IEEE standards, and (3) sections must contain textual content.

During preprocessing, sections and subsections were extracted from the documents as shown in Figure 1. A sections dataset was constructed containing the content of each section labeled with its section name. Sections were summarized to extract key information, and at least eight sentences were selected from each section.

A Sentences dataset was then created using the selected sentences where section names served as labels. The Sentences dataset and the Sections dataset are the two datasets used by the system. The Sections dataset has labeled sections, while the Sentences dataset contains labeled sentences for each section.

Since the sentence embedding model is trained with SetFit, eight examples per section are sufficient for fine-tuning [32]. The sentence embedding model used in Topic Classification is improved by the Sentences dataset, whereas the Sections dataset is used to refine the large language model (LLM) utilized in the Response Generation phase.

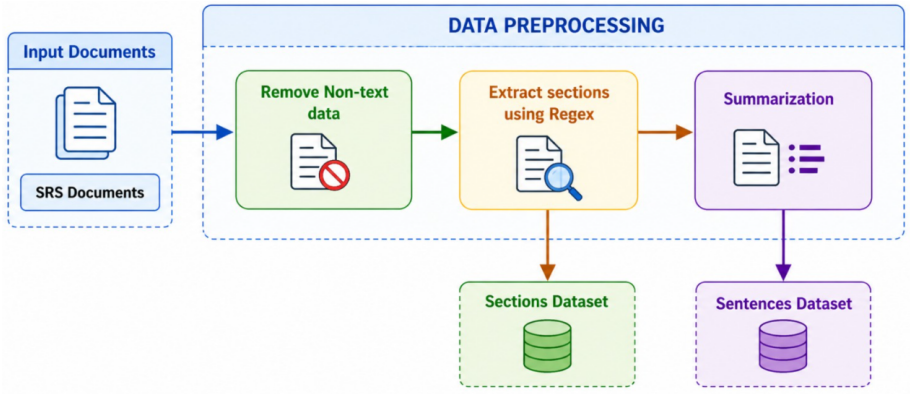


Figure 1. Data preprocessing methodology for building datasets from SRS documents.

3.2. Proposed Framework

Large language models are used in this proposed systematic approach to address the challenges of consistency and validity in document generation. This approach combines the strengths of fine-tuning and few-shot learning while reducing their drawbacks to produce valid documents with the required structure.

Sentence embedding, clustering, topic-based chunking, skeleton document generation, missing-node expansion, and incremental document expansion are all parts of the proposed methodology pipelines. First, a sentence embedding model is used to transform each requirement sentence into a vector representation. Next, requirement sentences that are semantically similar are grouped using clustering. Each cluster represents a topic-level group that may correspond to an SRS section or subsection. Topic-based chunking is performed by assigning each requirement sentence to the most relevant cluster. The SRS skeleton is then constructed using these topic-based chunks, and the system places new or modified requirements into the appropriate document section. The proposed framework has three stages: Topic Classification, Skeleton Document Generation, and Incremental Document Expansion as shown in Figure 2.

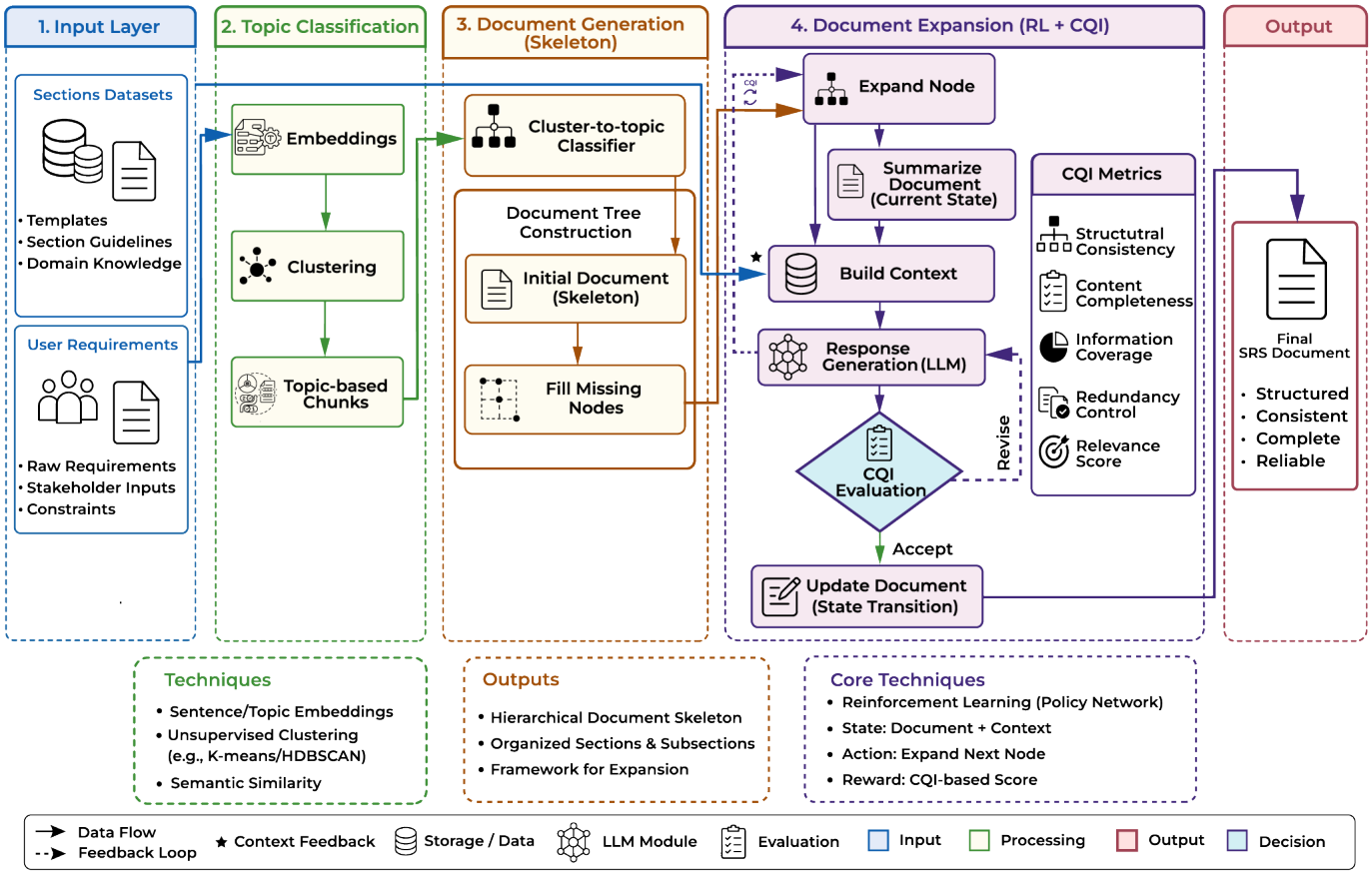


Figure 2. Architecture of the system that takes user requirements as input and produces a final document as a result.

3.2.1. Topic Classification

Topic classification is first performed to classify the input document into different topic-based chunks. In this step, requirement sentences are first converted into embeddings, and clustering is applied to group semantically similar sentences. Each cluster represents a topic or candidate SRS section. Topic-based chunking then places each requirement sentence into the most relevant cluster. This process extracts general and key information from the document. The information contained in a topic-based chunk may overlap with other chunks, and these chunks represent the different sections of the generated document. This classification step structures the content and preserves structural consistency when using LLMs with unstructured input.

Topic-based chunking is a technique used in [33] to break down long texts into manageable chunks based on their semantic relevance, as shown in Figure 3.

It begins by sequentially building sentence-level blocks within provided lengths and representing them as nodes in a graph. The text is parsed into different sentences or subsentences based on punctuation marks, and these sentence-level blocks are filled until they exceed a certain length before moving on to the next block [33]. Once the sentence-level blocks are obtained, they are connected in a graph representation of the long text based on a pre-trained sentence embedding model, such as Mini-LLM. The weight representing the semantic similarity between the i-th and j-th sentence-level chunks is calculated using cosine similarity, which calculates the cosine of the angle between two embeddings. A greater degree of resemblance between two chunks suggests that they are closer in topic [33]. Clustering algorithms are then applied to the graph to identify the underlying topic structure. Within each cluster, the sentence-level blocks are grouped sequentially to obtain

the topic-based chunks [33]. The purpose of clustering is to group related requirement sentences so that each topic-based chunk can be mapped to the appropriate SRS section.

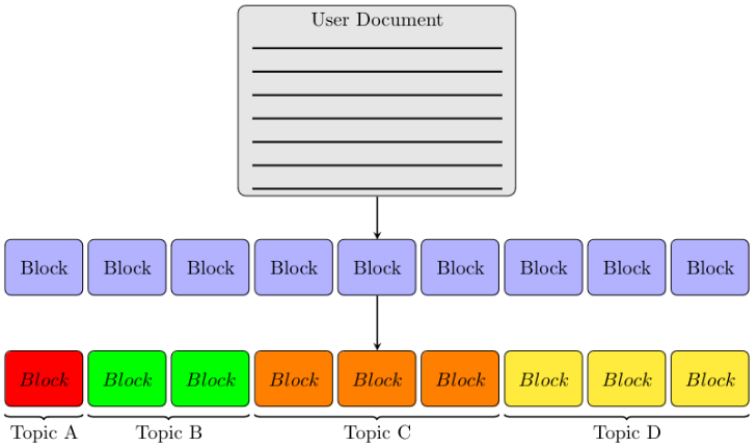


Figure 3. Topic-based Chunking.

The Topics Table is a simple registry of unique SRS section identities. It is constructed by traversing the hierarchical JSON representation of each SRS document in the corpus. Each section is stored using a canonical section identifier based on the IEEE-style decimal numbering scheme. During inference, this table is merged with a predefined IEEE 830 -based SRS skeleton, which provides fallback section titles for sections that are not present in the training corpus. The final output of the system is a structured SRS document, not a ranked list of retrieved chunks.

This study uses the Topic-based chunking technique to embed requirement sentences in the embedding space using a sentence embedding model. Clustering algorithms are applied to cluster embeddings of sentences. These clusters represent the sections and subsections of the document. The sentences within each cluster are grouped together to form a topic-based chunk. Figure 3 shows the Topic-based chunking, and the colors represent the different topic-based chunks. For K-Means clustering, the value of k is selected dynamically. In this study, k is set as the smaller value between the number of query chunks and 7. This ensures that the number of clusters does not exceed the number of available chunks, which is required for K-Means. The upper limit of 7 is used to keep the topic groups close to the main requirement intents and the common high-level organization of SRS documents. This setting prevents repeated K-Means iterations during inference while making the Topic Classification step more efficient for document generation.

In this research, clustering is used as an initial organization stage rather than the only option for final section placement. This is important because some requirements can be ambiguous in a particular domain and appear semantically close to more than one topic. After clustering, the topic-based chunk is compared with the Topics Table, and during expansion, supporting information is extracted from relevant section samples and document context. This helps mitigate the effects of imperfect clustering because the final generated section is determined by the cluster, the predetermined structure of the SRS section, and the contextual information obtained. Additionally, domain-specific examples can be used to improve the sentence embedding model in order to better accurately represent specialized terms.

The Topic-based chunk contains all the sentences in its cluster and the embedding, which is the centroid of the cluster.

3.2.2. Skeleton Document Generation

Following topic classification, a skeleton document is generated. The skeleton document consists of the structure and initial content for each chunk or topic identified in

the first step. The structure of the document is defined, and the initial content is inserted, creating a basic outline of the document.

Combining fine-tuning and few-shot learning helps address the limitations of each approach. Fine-tuning enables the LLM to learn syntactic patterns and valid document structures from a dataset of relevant documents. Few-shot learning provides control over the generated structure by presenting examples of the desired structure as context.

However, fine-tuning alone does not provide control over the generated structure, and few-shot learning requires the structure to be known beforehand. This limitation can be overcome by the systematic approach, which generates the document structure before text expansion through topic classification and skeleton document generation.

The document is modeled as a tree data structure, with each child node representing a section of the document and the root node representing the entire document. These sections, in turn, can be further divided into subsections, also modeled as child nodes within the section nodes. The idea of a tree data structure in computer science is comparable to this hierarchical structure, where each node (a section or subsection of the document in this case) can have multiple child nodes (subsections) but only one parent node (the section to which it belongs) [34].

For every identified topic-based chunk, a corresponding section is established within the document. Using the K-nearest neighbor (KNN) embedding-to-topic classifier, the topic of each chunk is determined. The classifier works by computing the distance between "all topics listed in the Topics Table" and the "embedding of the topic-based chunk". Subsequently, it selects the nearest neighbor to the topic-based chunk on the basis of these distances. The system maps each topic-based chunk to an SRS section. The cluster-to-topic mapping determines where each topic-based chunk should appear in the SRS hierarchy. If a recognized topic requires parent sections or related nodes that are not yet present in the document tree, the missing nodes are added from the Topics Table. In this way, the document skeleton is completed before detailed text expansion is performed.

The robustness of topic-to-section mapping is supported by using the same sentence embedding model, normalization process, and cosine similarity measure across all datasets. The embeddings of topic-based chunks and Topics Table entries are normalized before comparison, and cosine similarity is used because it compares vector direction rather than raw magnitude. This reduces sensitivity to feature scaling. In addition, topic-to-section mapping is used mainly as an organizational step. The generated section content is still supported by retrieved section examples, document context, and the CQI quality loop, which helps reduce the effect of occasional mapping uncertainty.

Once the sections are delineated, the initial document tree is constructed. This process entails the creation of a root node that represents the entire document, followed by the establishment of child nodes for each section. Furthermore, each section may contain its own set of child nodes, representing sub-sections or subtopics within that particular section [34].

Each node contains its respective topic-based chunk, as shown in Figure 4 and if the topic is identified, then the node also contains the information about the topic it belongs to. If the hierarchy in the document tree is incomplete, then the hierarchy is completed for recognized topics by adding nodes from the Topics Table, shown in Figure 5, until the ancestral relationship is complete for all recognized topics. This step introduces determinism at the document-structure level. Once a requirement sentence is embedded, clustered, and mapped to a topic, it is inserted into the corresponding position in the predefined SRS hierarchy. Therefore, adding or modifying a requirement affects only the related topic-based chunk or section, rather than regenerating the entire document.

This step reduces structural variation and helps maintain a stable SRS organization across several updates.

Every topic that the system has recognized is listed in the Topics Table. The use of the Topics Table helps to structure the document around the recognized or learned topics. It has information such as topic name, position in hierarchy, and embedding, which is the cluster's centroid containing the sentence embeddings for the particular topic.

The tree structure organizes and presents the final SRS document, but it does not strictly limit how requirements are processed. Topic classification and retrieval is responsible for cross-cutting requirements, where the requirement level and document level context are both considered. As a result, requirements related to security, performance, usability, or compliance can still have an impact on several sections via retrieved context, even if they are eventually placed under one primary section in the document tree. The structure remains adaptable because sections can be added, removed, or modified during document generation. Where possible, overlapping content is consolidated to reduce redundancy. This allows the framework to capture requirements that affect more than one section while maintaining an organized structure.

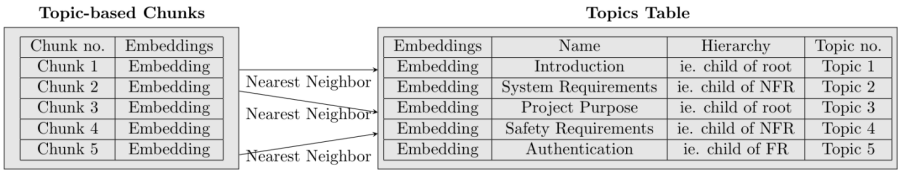


Figure 4. Topic-based Chunks and Topics Table.

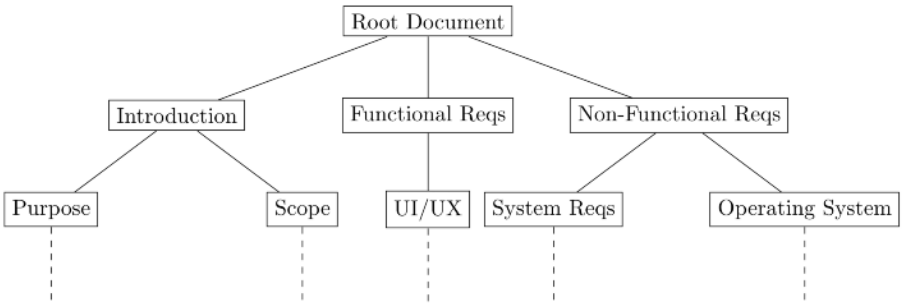


Figure 5. Document Tree.

3.2.3. Incremental Document Expansion

The process of adding new data to the nodes of a document tree is known as document expansion. The document can expand incrementally while maintaining structural stability because each node expands individually. Each topic-based chunk is expanded in its corresponding section once the skeleton document has been constructed. The information in the chunk, the topic selected, and the document's context are all used in the expansion. Since every section or node is expanded individually, The entire document does not need to be regenerated in order to include additional requirements. This helps in maintaining the document structure developed in the skeleton document generation phase.

Instead of generating the entire document at once, incremental expansion expands each node independently, providing determinism in the generation process. This step ensures that the output document maintains consistency and avoids changes to the document structure.

The proposed system uses an approach to expand the information present within the nodes of a document tree. Maintaining consistency and preventing the development of invalid document structures depend on this, especially when the entire context is given

to a large language model (LLM) and asked to produce both the document structure and its contents. In contrast, Document Expansion works incrementally, gradually adding new sections and information to the document as needed and verifying its validity at each expansion.

In addition, The concept of incremental expansion has been used in several domains, including information retrieval systems and search engine queries. For example, query expansion techniques aim to improve search results by adding relevant terms to the original query, thereby broadening the scope of the search to include more relevant documents [35,36]. This concept is similar to incremental document expansion in that it involves incrementally enhancing the original query or document structure to improve the quality of the output.

With the initial document built in the previous step, we incrementally expand the different sections or nodes of the document tree using the information that the nodes contain. The Response Generation Pipeline is used to expand the nodes. If the output sentences generated during expansions lie in a different cluster of topics, then a new sub-section is created. New sub-sections also go through expansion and are split into sub-sections depending on the expansion output. The incremental document expansion continues until a maximum depth of the document is reached, or a desired number of maximum sub-sections are generated.

A response is generated through the response generation pipeline with a (Command, Context) pair as input, which then replaces the information contained in the node. Three pieces of context are provided to the pipeline. The first piece is retrieved from the Sections dataset based on the node under expansion, which serves as an example that guides the LLM to generate responses pertinent to the node. The second piece is the summarized key information from the most recently generated document, which allows the LLM to access the key information that is present in other nodes. The third piece is the information contained within the node itself, which serves as the main text to be expanded. The Command tells the LLM to expand the third piece of context, considering the information contained in the second piece, and following the style of the first piece of context.

The built context integrates the Sections dataset, the current expansion node, missing nodes, and summarization to produce deterministic results for new requirements.

3.3. Fine-Tuning and Deep Reinforcement Learning

The process of training a pre-trained Large Language Model (LLM) on a particular task or domain is known as LLM fine-tuning. The goal is to leverage the pre-training phase, where the model was trained with a significantly large text corpus, and adapt it to perform better on a specific task [37,38].

The system incorporates a sentence embedding model for topic classification and a large language model (LLM) for Document Expansion. In the proposed system, we use a sentence embedding model in topic classification and a downstream LLM in incremental document expansion.

Sentence embedding models transform sentences into numeric vectors that represent their semantic meaning. These models enable sentence comparison based on embeddings and are frequently used in natural language processing operations such as semantic search, text classification, and paraphrase detection. Using a sentence embedding model, sentences are converted by the system into numerical vectors that capture their semantic meaning [39].

The use of the Mistral-7B model yields high-quality output in LLM-based generation. Mistral-7B performs better than Llama2-13b and Llama2-34b for tasks such as reasoning, math, and code generation [40]. Mistral-7B was selected because of its robustness as a

base model, which we further refine to enhance performance in the generation of software requirements specification (SRS) documents.

To refine the performance of the sentence embedding model in capturing the similarity between unlabeled input sentences and those found in various SRS sections [41], model fine-tuning is performed using few-shot learning examples. Each example contains a sentence paired with its corresponding section label [32]. Few-shot learning allows the system to understand the relationship between sentences and sections [42] on a small number of training samples.

The Response Generation Pipeline is further improved using reinforcement learning. The proposed system continually fine-tunes the LLM using Proximal Policy Optimization (PPO) [43]. The process of fine-tuning a Large Language Model (LLM) can be formulated as a markov decision process (MDP) [44], where the trained weights of the LLM serve as the reinforcement learning (RL) policy network. This network predicts the probability distribution over possible next tokens (actions) given the current input (state):

$$a_t \sim \pi(a_t \mid s_t) \tag{1}$$

$$s_{t+1} = s_t \cup \{a_t\} \tag{2}$$

Here, s_t represents the current sequence of tokens (state), and a_t represents the next predicted token (action). The vocabulary of the LLM defines the action space, and the sequence $\{s_0, s_1, s_2, \ldots, s_n\}$ constitutes the state transitions.

A reward model [3] guides the learning process by assessing the quality and usefulness of the produced content. The goal of training is to learn an optimal policy π^* that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \tag{3}$$

In this expression:

- $\tau = \{(s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T)\}$ represents the sequence of generated states and actions.
- $R(s_t, a_t)$ is the reward assigned to each step.
- $\gamma \in [0, 1]$ is the discount factor that controls the importance of future rewards.

The proposed formulation allows LLMs to be fine-tuned through reinforcement learning in a structured way, where every token generation event is interpreted as an action in a sequential decision process.

The LLM utilized in Response Generation undergoes continuous refinement using proximal policy optimization (PPO) [43]. PPO directly optimizes the policy network, which corresponds to the trained weights of the LLM. In contrast, value-based methods such as Q-learning estimate expected returns through a Q-function rather than directly optimizing the policy network [45].

Significant enhancements have been observed in Topic Classification and Response Generation. The sentence embeddings generated by the sentence embedding model result in more accurately classified sentences, while the performance of the LLM in Response Generation is notably improved, as depicted in Figures 9 and 10.

3.4. Response Generation

The large language model (LLM) is utilized to generate responses based on a provided (Command, Context) pair. A function is established to maximize the relevance between the original context (C) and the generated response (R), as shown in Equation (4). Additionally, the function minimizes the distance between the vector embedding of the generated

response (ER) and that of the topic-based chunks (ET), as shown in Equation (5). This minimization effectively enhances the relevance of the generated response to the topic or section of the node. This function is referred to as the distance-based relevance function.

$$\underset{R}{\text{Maximize: }} Relevance(C, R) \tag{4}$$

$$\underset{D}{\text{Minimize: }} Distance(ER, ET) \tag{5}$$

Topic-based chunks are identified by looking up the topic of the node affected by the command in the topics table. The embedding field of the topic-based chunk is used to calculate distance in the reward function.

In this study, we refer to the (Command, Context) pair as a prompt. This prompt serves as input to the downstream LLM, which generates a response. The reward function described earlier serves to assess the quality of the response, effectively measuring the performance of the LLM. We utilize this reward function as a performance metric to improve the LLM's performance through reinforcement learning.

The process of generating a response can be seen as a markov decision process (MDP). This process consists of states, actions, a transition model, and a policy. In this manuscript, states are the generated tokens, actions are the complete vocabulary of the LLM, the transition model is the probability distribution of transitions between states, and the LLM itself is the policy network [3].

DocGen presented in [3] consists of three components. We focus on its few-shot guided document generation component, which consists of three steps: (i) Query Expansion; (ii) Query Highlight; (iii) Document Generation. The enhanced version of DocGen, known as DocGen-RL, uses reinforcement learning to optimize the query used as a prompt to the LLM to generate synthetic documents.

In the current investigation, we provide a (Command, Context) pair instead of a query as a prompt to the LLM, and then optimize the generated response using reinforcement learning.

In instances where the response quality is deemed low, the LLM is treated as a policy network. We continuously refine the generated response through online reinforcement learning, utilizing the aforementioned reward function to assign a scalar value to the output responses. Until an optimal response is achieved, this iterative optimization process continues. Through fine-tuning of the LLM across diverse topics or sections, the LLM gradually acquires the ability to generate responses of adequate quality across various subjects, as evidenced by the quality assessment displayed in the Results section.

3.5. System Interaction

Users can optimize the generation process and modify the generated document by sending actions to the system. The system supports predefined interactions through commands. Interaction with the system through prompting in natural language is currently not supported because it introduces additional complexity and uncertainty to the system. The supported actions are "Add Context", "Add Section", "Remove Section", "Rename Section", and "Add Learning Examples".

"Add Context" adds additional context to the system. For example, if the user wants to add more requirements to the document, then this action can be used.

The "Add/Remove/Rename Section" action, as the name suggests, is used for making changes to the structure of the document by adding, removing, or renaming sections.

"Add Learning Examples" action allows the user to update the dataset that the system uses to learn about the different sections in the software requirements specification document.

The document analysis system is implemented using a Python–based framework (Python 3.11 or later). At its core is a fine-tuned Sentence-Transformer model (from the `Hugging Face Transformers` library), which generates dense embeddings. The most significant topic-based chunks for a particular requirement are found by comparing these embeddings using `scikit-learn`'s cosine similarity.

What makes the system unique is a technically advanced, customized data preprocessing pipeline that overcomes the limitations of raw input text. In particular, we created techniques to:

- Fill missing nodes: Automatically reconstruct incomplete or fragmented data, ensuring semantic completeness.
- Expand nodes: Enrich key concepts by programmatically pulling context from surrounding text.
- Build rich context: Merge relevant entities and sections to generate coherent, context-aware inputs for the model.

The limitations of simple techniques (such as SetFit) are overcome by this comprehensive pre-processing, enabling the fine-tuned model to produce more semantically accurate search results. Using the Proximal Policy Optimization (PPO) technique, the system integrates Reinforcement Learning (RL) to further improve the generation phase. A large language model (LLM) is interactively trained using a customized reward function based on distance-based semantic alignment (CQI). Over time, this loop ensures that the responses become more precise and context-aware, transforming the architecture into an intelligent, self-optimizing system.

4. Experimental Setup

Documents from a variety of open-source platforms were gathered for this study, and only documents clearly labeled with open licenses, such as Creative Commons or public domain declarations, were included in the dataset, including the PURE dataset. A labeled dataset was then constructed with section names as labels and the section contents as the corresponding textual instances.

4.1. Dataset Composition and Characteristics

Count: The final corpus consists of 176 structured requirements documents, including the 79 open-licensed documents from the PURE dataset and 97 additional Software Requirements Specification (SRS) documents collected from publicly accessible sources with explicit open-use permissions. A labeled dataset was constructed using section names as labels and the corresponding section content as textual instances. For the Topic Classification stage, these labels were grouped into six topic classes (Topics 1–6).

Requirement Specification Structure: The documents generated and evaluated by the proposed pipeline follow an IEEE 830/ISO 29148-compliant structure. Each document contains up to eight top-level clauses and covers 108 distinct hierarchical section identifiers across 119 unique section types. Among these, 55 section types represent atomic (leaf-level) specification elements.

Requirement Types: The corpus contains both functional and non-functional requirements. Of the 119 section types, 91 correspond to functional requirements (e.g., authentication, data management, and transaction processing), 9 correspond to non-functional requirements, and 19 represent front matter or structural sections. The non-functional categories include performance and load requirements, security requirements, usability and interface requirements, and software quality attributes.

Other Characteristics: The Topic Classification component operates over six topic classes. The experimental results show that the mean classification accuracy improved

from approximately 0.52 using the baseline embedding model to approximately 0.87 using the SetFit fine-tuned model across the six topics. Additionally, five documents were held out as a test set containing ground-truth reference texts for baseline comparison and evaluation purposes. The corpus composition used in the proposed pipeline is summarized in Table 2.

Table 2. Corpus composition per source artifact.

Source Artifact	Format	Role in Pipeline	Unit of Indexing	Count
Structured hierarchy	Nested JSON numbered sections, titles, and body text	Defines the canonical section catalog; flattened into per-section documents	One document per section node	1002 sections across 176 documents
Requirements text	Plain text	Requirement-level grounding for retrieval	Sentence-level requirement records	4425 sentences
Full document text	Plain text	Fallback and broad-context grounding	Full-document textual content	877,056 characters across 176 documents
Topics table	Derived	Canonical, numerically ordered section catalog	Unique section identifiers and section types	108 section identifiers; 119 section types

All values reported in Table 3 are generated directly by the ingestion pipeline document discovery → recursive hierarchy flattening → chunked source loaders → topics-table construction), ensuring that the statistics are fully reproducible from the released parsing code.

Table 3. Per-Document and Dataset Split Summary.

Attribute	Description
Documents (D)	Number of distinct source SRS documents
Mean/median sections per document	Computed from the flattened structured hierarchy
Mean hierarchy depth	Average nesting depth of the section tree
Requirement chunks per document	Generated after recursive chunking
Train/evaluation split	Documents reserved for reference-based evaluation are disjoint from those used during the learning loop

We assess the effectiveness of the proposed approach by comparing it with baselines. Instead of training the proposed sentence embedding model on labeled data, which can be costly and time-consuming in terms of data annotation, we used few-shot learning methods as an alternative.

For fine-tuning this selected sentence embedding model, which is utilized in the Topic Classification phase, we employ SetFit (Sentence Transformer Fine-tuning). SetFit is a straightforward and efficient prompt-free framework designed for few-shot fine-tuning of Sentence Transformers [32]. The fine-tuned model was evaluated on a labeled dataset comprising labeled document sections, and the resulting accuracy improvement is reported in the Results section. Recent research shows that larger models do not always outperform smaller models for specific NLP tasks [46]. In this work, we utilize Mistral-7B in combination with the structured document generation pipeline (topic classification, skeleton document generation, and incremental document expansion) to achieve efficient and high-quality SRS generation.

In the fine-tuning process of the LLM, we utilize reinforcement learning. Specifically, we employ TRLX (transformer reinforcement learning X) by CarperAI, which is a distributed framework tailored for fine-tuning large language models using reinforcement

learning with either a reward function or a reward-labeled dataset. As the number of epochs increases, a decrease in loss is observed, as discussed in the Results section.

4.2. Model and Training Hyperparameter Configuration

Additional replication resources supporting reproducibility were uploaded to Zenodo at [47]. These materials include the data-processing and analytical scripts used for the given results, as well as the pseudo-code of the suggested framework. Table 4 summarizes the main model, training, and evaluation hyperparameters used in the proposed framework.

Table 4. Model and training hyperparameters.

Component	Hyperparameter	Value
Embedding Model	Backbone	Sentence-Transformer, MiniLM-class bi-encoder (384-d)
Embedding Model	Similarity	Cosine, L2-normalized embeddings
Topic Clustering	Algorithm/Clusters (k)	KMeans, $k = \min(\text{query chunks}, 7)$
Policy Model	Base Model	7B instruct LLM (Mistral-7B-Instruct), optional 4-bit NF4
Policy Model	LoRA (r, α, dropout)	$r = 32$, $\alpha = 64$, dropout $= 0.05$
Policy Model	LoRA Target Modules	All attention and MLP projections
Policy Model	Decoding	max_new_tokens ≈ 640
Evaluation/Scoring	Default Scorer	Deterministic rubric (grounding, lexical recall, structure)
Evaluation/Scoring	Rubric Dimensions	Factual adherence, completeness, structural quality, specificity
Evaluation/Scoring	CQI Loop	Max 3 re-drafts; approval threshold 4.0/5.0
RL Optimizer (PPO)	Algorithm	PPO (clipped surrogate) with value head + frozen reference
RL Optimizer (PPO)	Learning Rate	1×10^{-5}
RL Optimizer (PPO)	KL Coefficient	0.04 (adaptive)
RL Optimizer (PPO)	Clipping Range	0.2 (policy and value)
RL Optimizer (PPO)	Value Coefficient	0.1
RL Optimizer (PPO)	Advantage Estimation	GAE
RL Optimizer (PPO)	Epochs/Temperature	1 epoch per round; sampling temperature 0.8
RL Optimizer (PPO)	Reward	Bounded hybrid reward in $[-1.5, 1.5]$, EMA-normalized

4.3. Baseline Evaluation Setup

A baseline evaluation was performed using five SRS-related documents. Each document had a ground-truth reference text, and all generated outputs were compared with these references. The evaluation was completed in two phases. First, outputs were generated using ChatGPT Model 5.2 and scored against the reference texts to form the baseline. Second, the same documents were processed using the proposed trained framework and evaluated under the same conditions.

ROUGE-1, ROUGE-2, ROUGE-L, BLEU, METEOR, and BERTScore F1 were used in the evaluation. These metrics measure lexical overlap, semantic similarity, and response quality. To reduce the impact of any one document, reported scores are calculated as the arithmetic mean of the five documents.

4.4. Evaluation Mapping to Research Questions

Table 5 connects each research question to the technique, metric, and section in order to make the connection between the research questions and the evaluation evident.

Topic classification accuracy, section-assignment behavior, section-classification performance metrics, section count, type diversity, and structure score were used to measure structural consistency. These indicators measure whether requirement content is assigned to appropriate SRS sections and whether the generated document maintains a stable section organization.

Contextual relevance was evaluated using cosine similarity, F1, cross-similarity, semantic similarity heatmaps, and overlap analysis. These indicators assess how well the

generated content maintains semantic similarity with the source requirements and contextual information retrieved.

Table 5. Research questions mapping to evaluation methods and results.

RQ	Evaluation Focus	Metrics / Evidence	Related Results
RQ1	Suitability of LLMs for production use	Baseline comparison, consistency, and reliability issues	Section 5.1, Table 6
RQ2	Effect of systematic structure generation	Topic classification, skeleton generation, and structure stability	Figures 7 and 8 ; Tables 7, 13 and 14
RQ3	Improvement in SRS generation using language models	ROUGE, BLEU, METEOR, Cosine Similarity, and BERTScore F1	Section 5.1
RQ4	Effect of context on response generation	Semantic similarity, cross-similarity, and relevance of generated output	Figure 12; Tables 10–12
RQ5	Effect of reinforcement learning on response generation	Reward improvement and loss reduction during training	Figures 9 and 10

5. Results

The results are organized in accordance with the main evaluation objectives and connected to every research question in this study.

5.1. Baseline Comparison with Standard LLM Prompting

5.1.1. Average Metric Scores

Table 6 displays the average score for each metric across all five documents, which compares our trained framework with ChatGPT Model 5.2 (baseline), along with the absolute and percentage change. Positive values indicate improvement; negative values indicate regression. This subsection compares the proposed framework with a standard LLM prompting baseline in order to address RQ1 and RQ3.

Table 6. Performance comparison between ChatGPT Model 5.2 baseline and the proposed framework.

Metric	ChatGPT Model 5.2 Baseline	Our Framework	Absolute Delta	% Change
ROUGE-1	0.2901	**0.3386**	+0.0485	+16.7%
ROUGE-2	0.0999	**0.1182**	+0.0183	+18.3%
ROUGE-L	0.0907	**0.1086**	+0.0180	+19.8%
BLEU	0.0278	**0.0270**	−0.0008	−3.0%
METEOR	0.2231	**0.2495**	+0.0264	+11.8%
Cosine Sim.	0.5815	**0.6015**	+0.0200	+3.4%
BERTScore F1	0.7759	**0.7915**	+0.0156	+2.0%

Note: Bold values reflect proposed framework values.

The assessment metrics were selected and interpreted according to their specific purposes and limitations. ROUGE, BLEU, and METEOR were used only to measure lexical overlap with the reference documents, whereas semantic similarity was calculated using cosine similarity and BERTScore. Classification accuracy was applied only to the topic classification task, and document-level characteristics, including readability, structure, and completeness, were summarized using quality-score indicators.

5.1.2. Side-by-Side Metric Comparison

The average scores for all seven metrics for both systems are directly compared in Figure 6, which makes the relative strengths of each system immediately apparent.

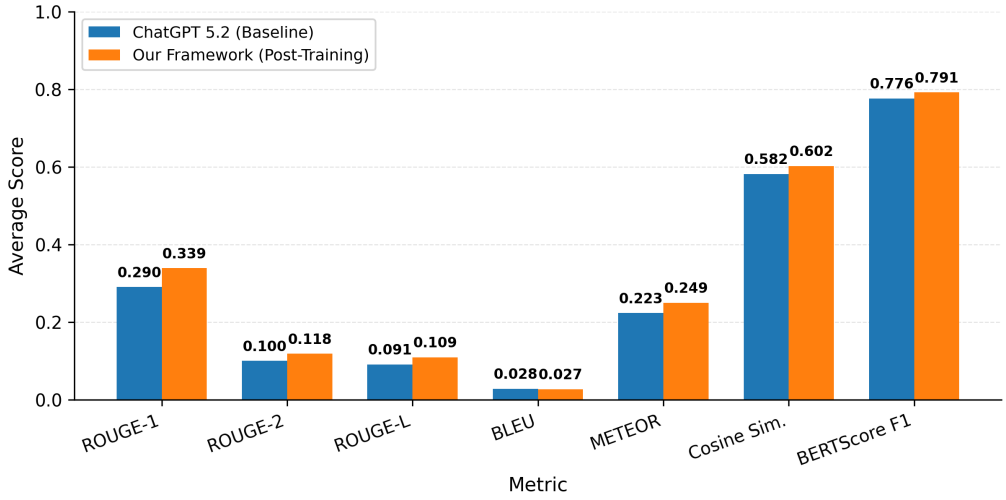


Figure 6. Grouped bar chart comparing average ChatGPT Model 5.2 baseline scores against our framework's post-training.

5.2. Topic Classification Results

In the proposed approach, topic classification is an essential first step in ensuring coherence and consistency in document generation. We consistently achieved an accuracy of more than 0.8 for recognized topics by optimizing the sentence transformer using few-shot learning techniques. Figure 7 illustrates the accuracy enhancement achieved by the refined sentence embedding model in comparison to the baseline model without any fine-tuning. Depending on the complexity and type of data from various domains, the effectiveness of different baseline models can vary. In order to address RQ2, these findings demonstrate how the Topic Classification step supports structured document generation.

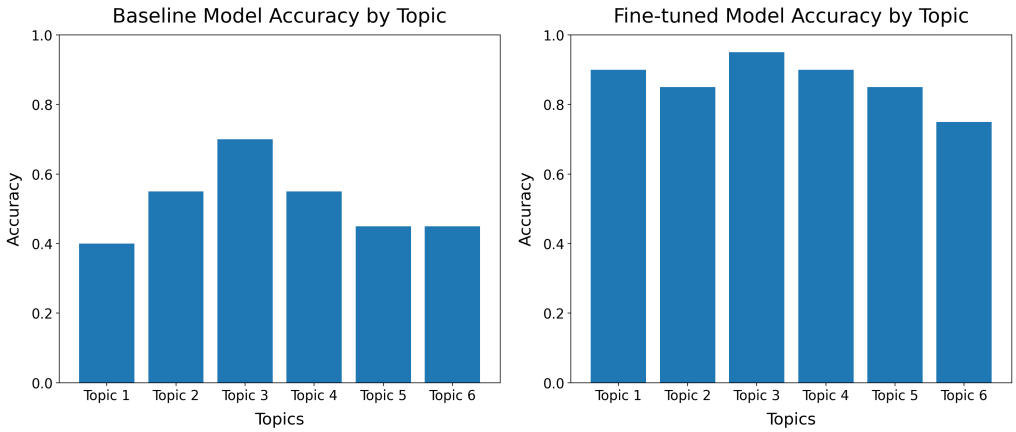


Figure 7. Histogram presenting Classification accuracy improvement of fine-tuned model over the base model.

The comparative histograms show how effective the fine-tuning process is, with noticeable increases in accuracy across all topic classes. Table 7 provides a detailed breakdown of these improvements, demonstrating enhanced topic classification performance after SetFit-based fine-tuning.

Table 7. Base model and fine-tuned model accuracy with absolute difference and percentage improvement for each topic.

Topic	Baseline	Fine-Tune	Difference	Improvement (%)
Topic 1	0.40	0.90	0.50	125.00
Topic 2	0.65	0.85	0.20	30.77
Topic 3	0.70	0.95	0.25	35.71
Topic 4	0.65	0.90	0.25	38.46
Topic 5	0.45	0.85	0.40	88.89
Topic 6	0.65	0.75	0.10	15.38

Accuracy is measured as topic-wise classification accuracy, calculated as the number of correctly classified requirement sentences for a topic divided by the total number of evaluated sentences for that topic. Baseline Accuracy refers to the accuracy before fine-tuning, while Fine-tuned Accuracy refers to the accuracy after SetFit-based fine-tuning.

$$\text{Accuracy}_{\text{topic}} = \frac{\text{Correctly classified sentences in the topic}}{\text{Total evaluated sentences in the topic}} \quad (6)$$

Figure 8 presents training and validation accuracy and loss over time, showing whether the model learns effectively or overfits.

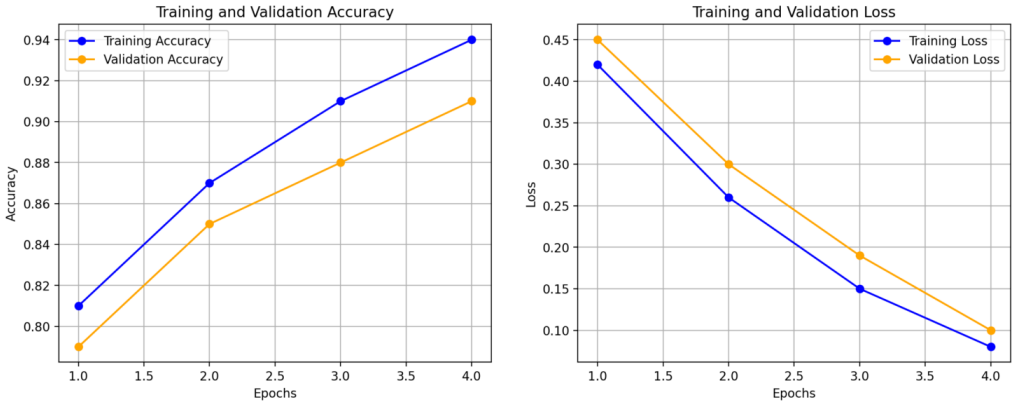


Figure 8. Model Training Progress Over Epochs.

Training accuracy/loss: measures the model's performance on the training data.

Validation accuracy/loss: measures the degree to which the model generalizes or predicts its accuracy against unseen data.

These results support RQ2 by showing that the Topic Classification model becomes more stable after training.

5.3. Reinforcement Learning Results

Reinforcement learning plays a pivotal role in optimizing the relevance of the generated response to both the original context and the respective topic or document section. This optimization is realized by defining a reward function, as outlined in the Response Generation Pipeline section of this manuscript, and subsequently utilizing reinforcement learning to maximize the reward obtained for the generated responses. Figure 9 depicts the decrease in loss over steps, and Figure 10 illustrates the mean reward value over steps.

Furthermore, this proposed system has demonstrated robust performance even in scenarios with limited data, which is an application of the implementation of reinforcement learning and the distance-based relevance function.

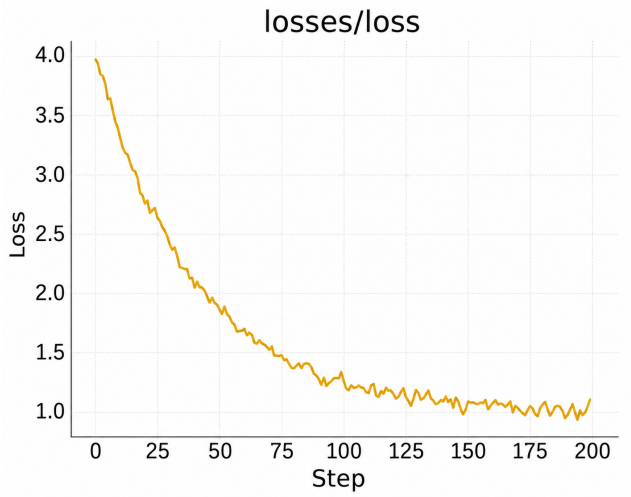


Figure 9. Decrease loss over steps.

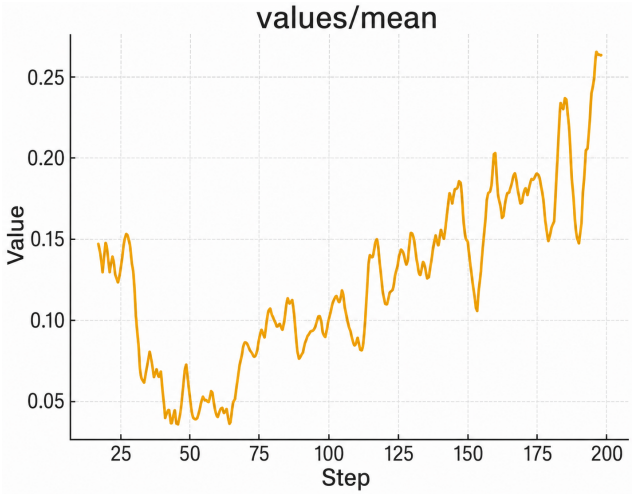


Figure 10. Increase reward values over steps.

By showing the effect of reinforcement learning on response generation through loss reduction and reward improvement, these results address RQ5.

5.4. Generated Document Quality

The quality of the generated documents was evaluated using metrics for accuracy, completeness, and relevance. These results address RQ3 by evaluating the quality of generated SRS documents using automatic text generation metrics.

The system applies the three-step method to produce a final document. The produced document is assessed using metrics for accuracy, completeness, and relevance. By applying these quality metrics, we demonstrate the significance of the three-step method in producing a high-quality document. The metrics used here for quality assessment are BERTScore (Relevance), ROUGE (Accuracy), and Completeness (checking for the presence of key terms or concepts in the output).

Various key dimensions are evaluated, particularly Content Richness, Structural Complexity, Technical Depth, and Completeness. As mentioned above, the generated document's overall Quality Score 0.532 indicates a moderate level of quality. This chart displays the model's ability to handle high-quality technical content and shows the document's suitability for structured SRS generation. Table 8 contains the summary.

Table 8. Document Quality Score Summary.

Metric	Value
Average Score	0.532
Highest Score	1.000
Lowest Score	0.083

Figure 11 presents the document quality radar chart for the uploaded document with DOCID: 52a955787c12, providing a visual summary of the quality of the generated document in the selected quality dimensions. It shows how the generated document performs in terms of content richness, structural complexity, technical depth, readability, and completeness.

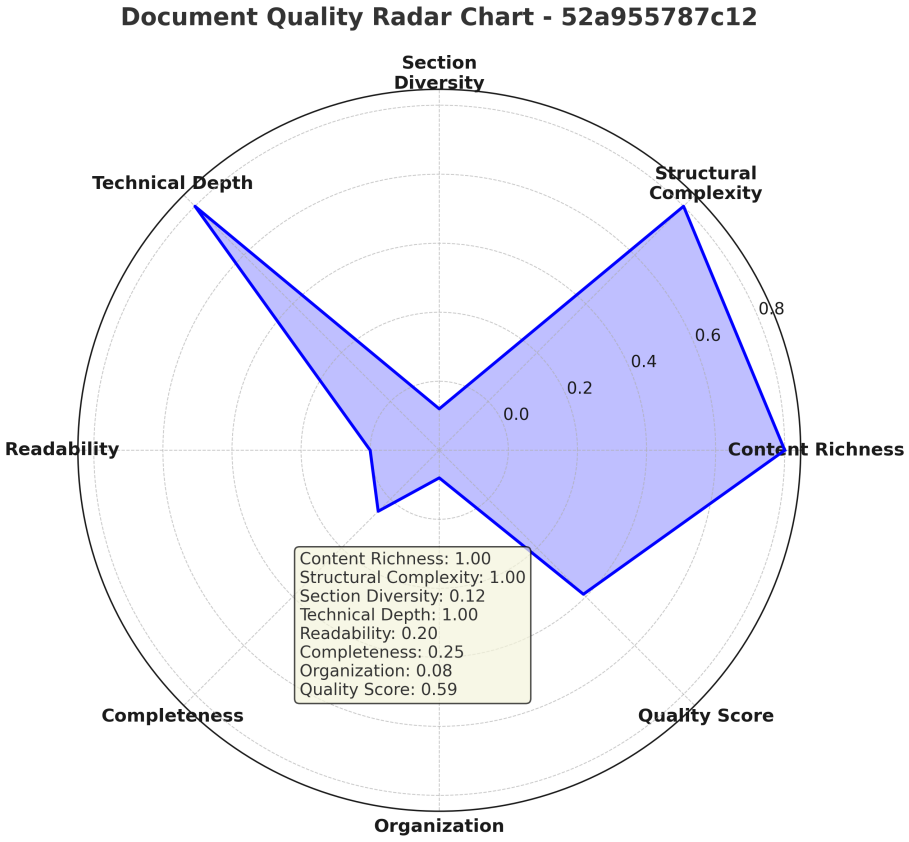


Figure 11. DocumentQuality Radar Chart.

Table 9 indicates high scores for Content Length, Complexity, Processing Quality, and Readability, indicating strong technical content readiness [48]. To better support downstream structured document generation, the lower Section Count, Type Diversity, and Structure scores reveal areas where content segmentation and variety could be improved. Scores are normalized from 0 to 1, with the corresponding grades.

Grades were calibrated from the normalized score values: A for scores $\geq$ 0.80, B for scores from 0.50 to <0.80, and C for scores $<$ 0.50. Thus, each grade provides a concise qualitative interpretation of the corresponding normalized score. Therefore, the overall normalized score of 0.5333 is interpreted as a moderate overall quality level and is assigned grade B.

Table 9. Comparison Matrix (Scoring Table).

Metric	Underlying Metric Used to Calculate the Value	Raw Value	Normalized Score (0–1)	Grade
Content Length Score	Total number of characters in the generated document	93,740 chars	1.0000	A
Section Count Score	Number of detected sections in the generated document	1 section	0.0500	C
Complexity Score	Textual/structural complexity of the generated content	1.0000	1.0000	A
Type Diversity Score	Number of different detected sections/content types	1 types	0.1250	C
Processing Quality Score	Processing status of the document, mapped from labels such as High/Medium/Low	High	0.9000	A
Readability Score	Readability status of the generated text, mapped from labels such as Good/Moderate/Poor	Good	1.0000	A
Structure Score	Structural completeness status, mapped from labels such as Complete/Moderate/Basic	Basic	0.1000	C
Overall Quality Score	Composite quality score calculated from the normalized quality indicators	Moderate	0.5333	B

5.5. Contextual Relevance and Semantic Similarity

The semantic similarity between sections of the uploaded document and the existing model knowledge base is shown in Table 10. The table indicates how closely the generated content aligns with the original source material. A stronger semantic match is indicated by higher cosine similarity values, which are closer to 1. This represents that the generated document is closely aligned with the actual requirements.

Table 10. Semantic similarity between uploaded document sections and model sections.

Model Section	Cosine Similarity
Sec 3—Requirements	0.702
Sec 2—Requirements	0.687
Sec 10—Security	0.654
Sec 1—Requirements	0.629
Sec 7—Introduction	0.618
Sec 5—Interface	0.616
Sec 8—Logical A	0.569
Sec 4—Interface	0.567
Sec 6—Internal	0.565
Sec 9—Logical B	0.539

The table indicates how accurately the system generates content that has semantically aligned information. It shows the system's expertise in maintaining contextual relevance between generated and original documents.

Run A and Run B represent two controlled processing stages used to assess the stability of the proposed document ingestion and structuring pipeline. Run A was conducted in a smaller corpus as an initial validation stage, while Run B was conducted on an expanded corpus to examine whether the same pipeline remained stable with increased document

content and section coverage. Therefore, these runs were used for staged pipeline validation rather than repeated random trials.

Additionally, the calculated cross-similarity value for Run A is shown in Table 11. These results address RQ4 by demonstrating how the resulting output's semantic relevance is improved when structured context is used.

Table 11. Cross-Similarity Summary (Run A).

Metric	Value
Average Cross-Similarity	0.615
Max Cross-Similarity	0.702
Min Cross-Similarity	0.539

This similarity matrix provides content overlap analysis, which is a detailed analysis of the content duplication between the uploaded document's sections and a sample of the data currently included in the model. As shown in Figure 12, it evaluates similarity using multiple metrics such as Jaccard Similarity (word overlap), Semantic Similarity (embedding-based), Length Similarity, and a Combined Similarity Score. This helps identify redundant content in the two inputs.

The proposed research outcome confirms whether the generated or classified content, based on model output, semantically aligns with the source document. A high degree of similarity indicates that the model comprehends and maintains the original objective. Based on these results, a summary of overlaps is provided in Table 12.

Table 12. Overlap Analysis Summary.

Metric	Score
Average Word Overlap	0.011
Average Semantic Similarity	0.550
Average Length Similarity	0.007
Average Combined Score	0.280

Figure 12, the final heatmap, illustrates content overlap and semantic similarity between the generated document and source requirements. It helps identify clusters of semantically close information within or between documents. The heatmap is based on one of the 30 sampled parts in the dataset and shows that the model can identify related information, supporting improved content retrieval and organization in large-scale document systems.

5.6. Structural Stability and Section Assignment

In this case, the topic embedding and section classifier in the architecture operate deterministically, proven by statistics of the classification confusion matrix generated by the system, which is consistent with the framework design objective of reliable structure assignment.

Downstream classification performance is enhanced by an increase in the classification scores for a particular document, which presents early document structuring. Run B's classification findings in Table 13 significantly outperformed the initial results of Run A, according to the reported accuracy, precision, recall, and F1-score values. These results further support RQ2 by showing how the proposed topic embedding and section classifier improve stable section assignment.

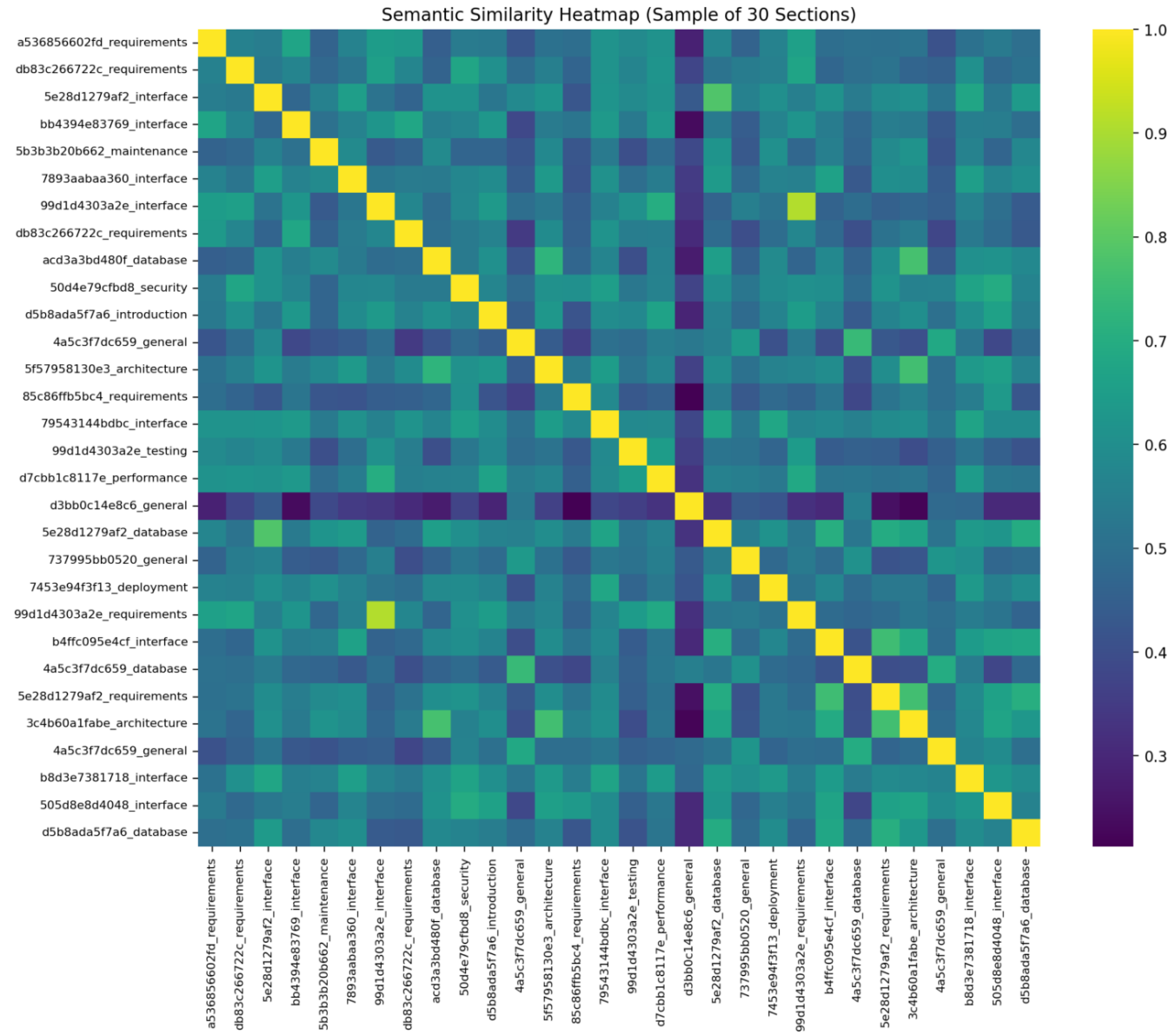


Figure 12. Documents Semantic Similarity.

Table 13. Performance metrics for Run A and Run B.

Metric	Run A	Run B
Accuracy	0.980	0.985
Precision	0.990	1.000
Recall	0.992	1.000
F1-score	0.940	0.986

The metrics in Table 14 suggest the degree to which this model can estimate the confidence as well as the correctness of classification [49]. As this framework incorporates hierarchical classification of documents and association with topics/sections, a strong correlation in terms of confidence can be observed:

1. Classification of sections/topics is usually dependent on the downstream structuring of documents.
2. Reinforcement learning fine-tuning produces meaningful confidence outputs.

3. When creating structured content, predictions can be reliably used, leading to increased determinism and greater certainty.

Table 14. Classification Confidence Metrics.

Metric	Value
Spearman Correlation	0.6405
R-squared	0.2945
Mean Squared Error	0.0169
Mean Absolute Error	0.1016

Spearman correlation was used to examine the monotonic association between score-based indicators, while R-squared, Mean Squared Error, and Mean Absolute Error were used to assess fit and prediction error. These measures do not prove SRS correctness. Instead, they serve as indicators of similarity, relevance, structural consistency, and model behavior.

5.7. Additional Corpus and Structural Analysis

As shown in Figure 13, we first performed two rounds of document fetching to validate the system's first phase. In the first run, a total of 92 documents were processed with a success rate of 97.8%, resulting in an average document length of 5323 characters and a total content length of about 794,696 characters. The second run processed 176 documents. It had a 98.8% success rate. This yielded 877,056 characters and a higher average document length.

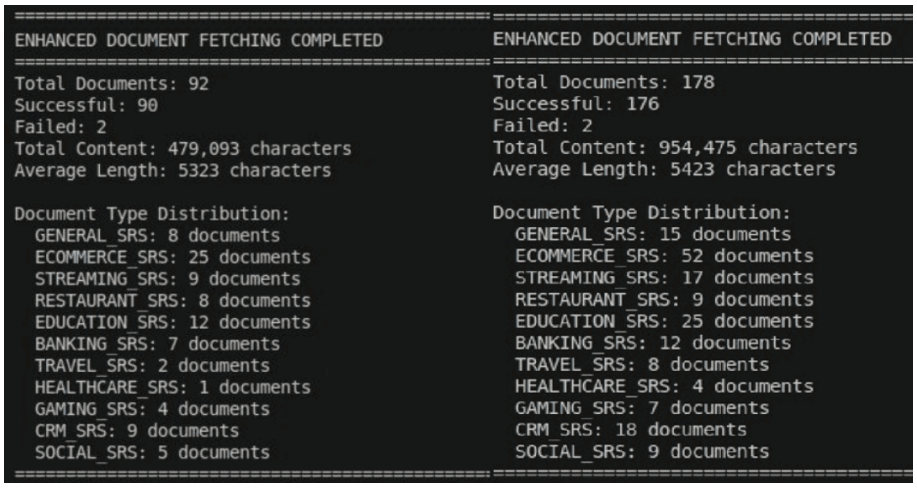


Figure 13. Documents Fetching summary.

To ensure efficient model training, we classify the retrieved documents into structured types. This covers various domains like e-commerce, education, CRM, and healthcare.

This system was trained on 176 structured documents. These covered various SRS topics during the enhanced training phase. A section-level dataset with 1002 sections and an enriched sentence-level dataset with 4425 sentences were created as part of the process. Furthermore, 1354 of the 1594 advanced training examples were used for instruction, and 240 were used for evaluation. The MiniLM (L6-v2) sentence transformer was pre-trained with the help of this systematic approach, providing a strong basis for producing a consistent, context-aware text. Figure 14 presents a representative system log showing the document ingestion and dataset creation process.

```
2025-07-31 21:21:55,041 - INFO - Loaded document 41cc2a53edff: 2873 chars, 7 sections, type: TRAVEL_SRS
2025-07-31 21:21:55,044 - INFO - Loaded document 9e2f884b3580: 4614 chars, 5 sections, type: GENERAL_SRS
2025-07-31 21:21:55,053 - INFO - Loaded document d027b72cf0e1: 8269 chars, 14 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,065 - INFO - Loaded document 1a802ce1a229: 8611 chars, 17 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,073 - INFO - Loaded document 4d12e1a60f13: 5224 chars, 5 sections, type: EDUCATION_SRS
2025-07-31 21:21:55,081 - INFO - Loaded document 5ca201f24987: 5108 chars, 8 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,090 - INFO - Loaded document 61266e3a67f8: 7001 chars, 13 sections, type: EDUCATION_SRS
2025-07-31 21:21:55,096 - INFO - Loaded document 631478148a06: 7785 chars, 11 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,107 - INFO - Loaded document b11922cebf34: 8049 chars, 16 sections, type: TRAVEL_SRS
2025-07-31 21:21:55,110 - INFO - Loaded document c7cf5bc4c412: 1737 chars, 3 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,117 - INFO - Loaded document 39d370cc1426: 4662 chars, 6 sections, type: GAMING_SRS
2025-07-31 21:21:55,127 - INFO - Loaded document 0206e3d00993: 8883 chars, 13 sections, type: ECOMMERCE_SRS
2025-07-31 21:21:55,139 - INFO - Loaded document ce8db45da130: 9720 chars, 9 sections, type: CRM_SRS
2025-07-31 21:21:55,139 - INFO - Successfully loaded 176 training documents
2025-07-31 21:21:55,139 - INFO - Creating enhanced sentence-level dataset...
2025-07-31 21:21:55,465 - INFO - Created enhanced sentence dataset with 4425 sentences
2025-07-31 21:21:55,526 - INFO - Creating enhanced section-level dataset...
2025-07-31 21:21:55,638 - INFO - Created enhanced section dataset with 1002 sections
2025-07-31 21:21:55,691 - INFO - Creating advanced training examples...
2025-07-31 21:21:55,806 - INFO - Created 1594 advanced training examples
2025-07-31 21:21:55,857 - INFO - Training set: 1354 examples
2025-07-31 21:21:55,857 - INFO - Test set: 240 examples
2025-07-31 21:21:55,857 - INFO - Training enhanced sentence_transformer...
2025-07-31 21:21:55,857 - INFO - Training enhanced model: sentence_transformer
2025-07-31 21:21:55,888 - INFO - Use pytorch device_name: cpu
2025-07-31 21:21:55,888 - INFO - Load pretrained SentenceTransformer: all-MiniLM-L6-v2
2025-07-31 21:22:03,176 - INFO - Creating comprehensive evaluation dataset...
2025-07-31 21:22:03,177 - INFO - Starting enhanced model training...
```

Figure 14. Documents Ingestion and Dataset creation.

A consolidated view of the document retrieval and model training results is displayed in Table 15. With a high success rate, 176 structured documents representing different domain documents were processed.

Table 15. Summary of Document Fetching and Model Training.

Document Fetching Summary	Model Training Results
Total Documents: 176 Successful: 174 Failed: 2	Documents Processed: 176 Sentences Extracted: 4425 Sections Extracted: 1002
Total Content: 877,056 characters	Training Examples: 1354 Test Examples: 240
Average Length: 5423 characters	Model: `all-MiniLM-L6-v2` Status: SUCCESS
Document Types: ECOMMERCE, EDUCATION, CRM, etc.	Spearman Corr.: 0.6405 R-squared: 0.2945 MSE: 0.0169

The advanced training phase processed 4425 sentences and 1002 sections, generating 1354 training examples. The sentence transformer (all-MiniLM-L6-v2) achieved impressive evaluation metrics, including a Spearman Correlation of 0.6405 and a mean squared error of 0.0169, confirming that the system's architecture for structured document generation is initially effective.

The above training examples help the sentence transformer in learning contextual patterns and producing embeddings with meaning. Performance in a generalization test set of 240 examples was used as a measure of performance. These metrics are used to calculate the effectiveness of the model by computing metrics used to measure its efficiency, such as Spearman correlation, R-squared, and Mean Squared Error.

Table 16 compares the main statistics of two batch processing runs of documents, showing a substantial increase in the number of documents, total content length, and extracted sections from Run "A" to Run "B".

Table 16. Comparison of Document Statistics Between Run A and Run B.

Metric	Run "A"	Run "B"
Total Documents	92	176
Total Content Length	794,696 chars	877,056 chars
Total Sections	693	1367

Figure 15 illustrates the clustering procedure by showing how semantically similar sections are grouped into clusters and how topic distributions assist the fine-tuning process.

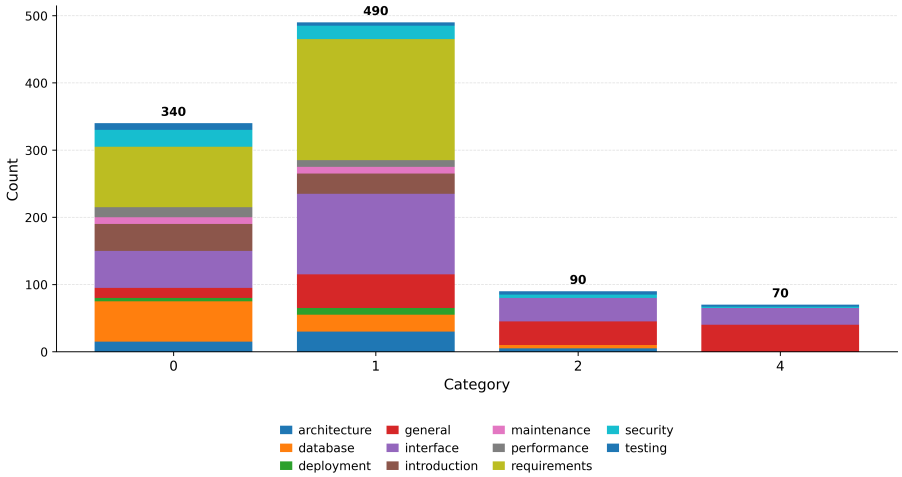


Figure 15. Section Type distribution by cluster.

Complexity scores are represented by average metrics (content length, number of sections, complexity score) grouped by document type across all processed documents in Table 17, with the plot in Figure 16 providing information on typical characteristics of different categories of documents.

$$\text{Avg Complexity Score}_{\text{type}} = \frac{\sum_{i=1}^{n} \text{Complexity Score}_i}{n} \tag{7}$$

where n is the number of documents of a certain type, and $\text{Complexity Score}_i$ refers to the complexity score for each document in that type.

Table 17. Average Metrics by Document Type.

Document Type	Avg Content Length	Avg Sections	Avg Complexity Score
BANKING_SRS	4300.43	8.86	0.24
CRM_SRS	6907.22	10	0.30
ECOMMERCE_SRS	4812.44	6.64	0.23
EDUCATION_SRS	4462.67	6.58	0.21
GAMING_SRS	4177.75	10	0.24
GENERAL_SRS	4328.75	5.38	0.15
HEALTHCARE_SRS	4704	12	0.38
RESTAURANT_SRS	4211.75	7.38	0.21
SOCIAL_SRS	5573.40	8.80	0.29
STREAMING_SRS	4257.56	8.11	0.22

Figure 16 displays a matrix of common structural patterns in documents that increase structural determinism and reusability. In this approach, the co-occurrence matrix helps us see how often “interface” and “requirements” appear together. As shown in Figure 17, this information helps in adapting templates for many document kinds, including financial and healthcare.

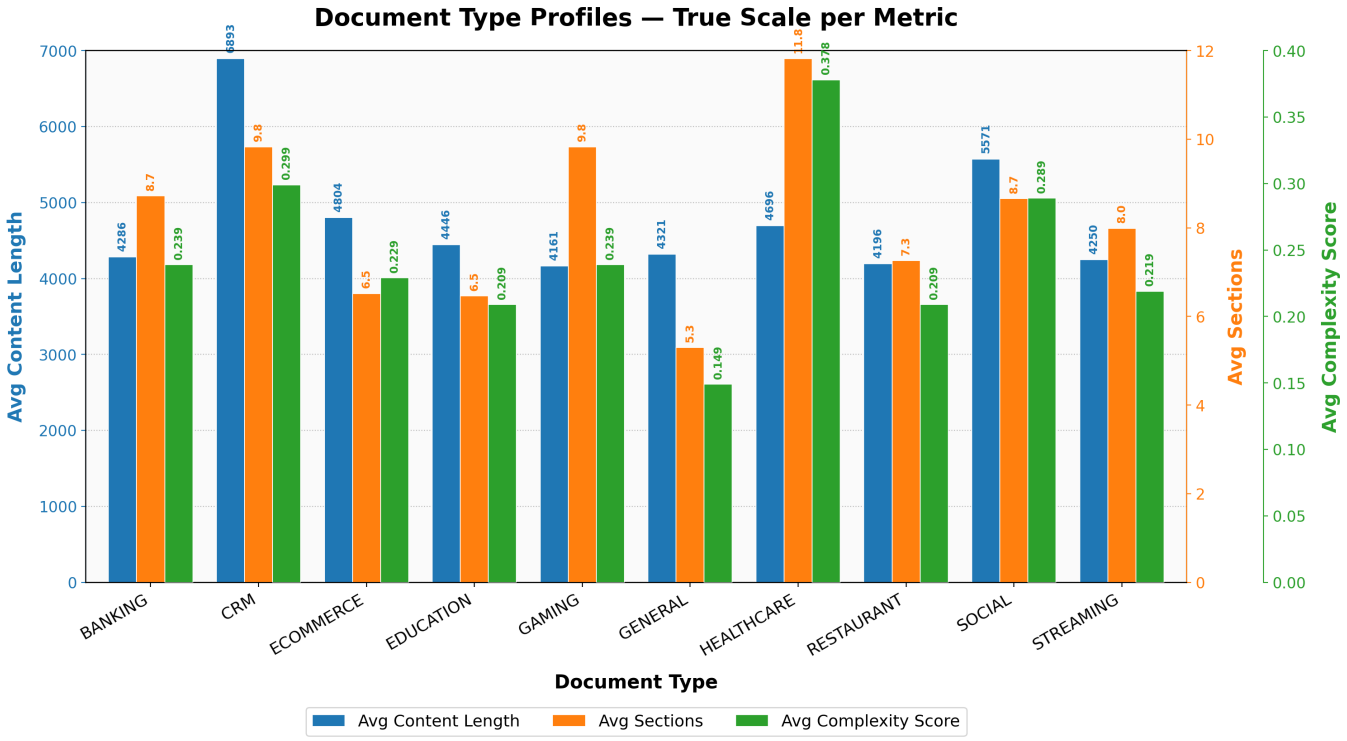


Figure 16. Average Metrics by Document Type Plot.

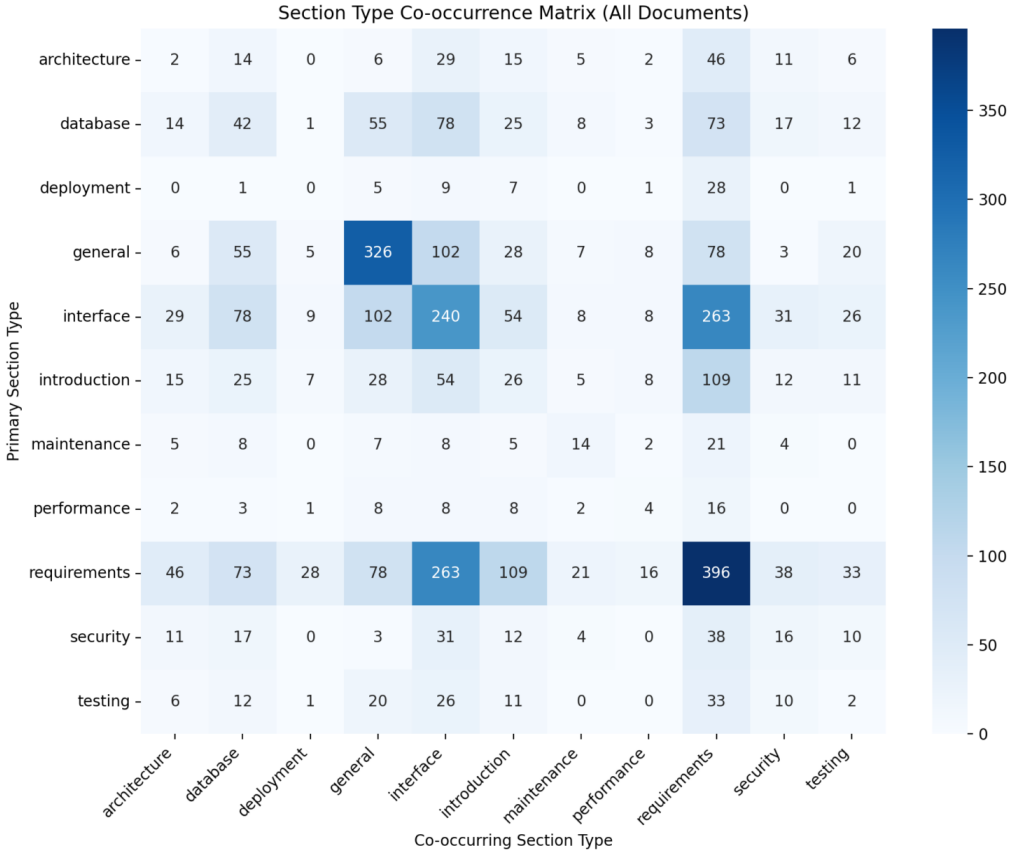


Figure 17. Section Type Co-occurrence Matrix (All Documents).

6. Practical Application of the Proposed Framework

Software Requirements Specification: The founder of a small food delivery business can explain the system in simple terms, including rider monitoring, client ordering, pay-

ment, and delivery updates. These inputs are organized into topic-based chunks by the framework, maps them to IEEE-style SRS sections through the Topics Table, and generates a structured SRS document without skipping important sections.

Legal Contract Drafting: For a short-term fintech consulting job, the system can identify topics including jurisdiction, intellectual property, payment terms, confidentiality, and termination. These topics are mapped to suitable contract sections and supported by corpus examples, producing a draft that is more comprehensive than flat, single-pass generation.

Healthcare Procurement Proposal: The framework identify patient data handling, access control, audit history, reporting, and compliance-related requirements for an electronic health record system. These topics are then organized into the appropriate document sections, assisting in the transformation of incomplete user input into a structured technical proposal.

Financial Feasibility Report: For a bakery loan application, the framework can identify business background, cost estimation, financial projections, repayment plan, and risk assessment. These topics are arranged into a professional structure, reducing the gap between informal user input and formal document requirements.

7. Discussion

This section discusses the experimental results with respect to the defined research questions. The findings demonstrate the effectiveness of the proposed deterministic framework for structured and reliable SRS generation. The results of this study are linked with the research questions through the evaluation mapping presented in Table 5.

RQ1: What role can LLMs play in SRS generation, and what are their limitations in production environments?

The proposed system addresses this problem by using a structured deterministic document generation pipeline. In a production environment, reliability and repeatability are ensured by embedding, clustering, and mapping the same input requirements to the same topic-based chunks. The process of generating text embeddings with the fine-tuned Sentence-Transformer model, applying clustering, and mapping chunks to SRS topics is predictable, reducing the variability associated with direct generative models. The system provides controlled document generation employing skeleton-based structuring and incremental section enlargement in addition to deterministic topic-based structuring. This reduces structural inconsistencies commonly observed in fully generative LLM-based systems.

RQ2: How does a systematic approach introduce structural determinism to document generation through LLMs?

By externally enforcing the structure, this methodology achieves systematic determinism. The system's output is a structured SRS document generated from topic-based chunks, not only a list of preexisting document sections ranked by relevance. This eliminates any variability. Technically, vector embedding and similarity calculation are predictable math operations that control the pipeline. This helps ensure that changes in user requirements do not randomly alter the complete document structure. Existing information is preserved, and new or modified requirements are mapped to the relevant topic-based section, making the document structurally deterministic.

RQ3: How does the use of Language Models improve the SRS generation process?

Our system uses a fine-tuned Sentence-Transformer model's ability to understand language extensively to semantically analyze requirements. This makes it much easier to find information for tasks like SRS generation. Through the processing and embedding

of requirement sentences, it is possible to find and retrieve all applicable text based on a particular SRS requirement efficiently. This approach focuses on structured SRS generation by using embedding, clustering, and topic-based organization before content expansion.

RQ4: How does context affect response generation?

This system provides the response generation process with specific contextual input. The requirement sentence is used as the main context, and a dense vector embedding is constructed for it. The requirement embedding is then used to identify the most semantically relevant topic-based chunks. These chunks provide structured context for section-level generation and help ensure that the generated response remains related to the assigned SRS topic. This also prevents unrelated sections of the document from being modified when a new or changed requirement is added.

RQ5: How does reinforcement learning affect response generation?

This novel proposed architecture goes beyond supervised fine-tuning by integrating a feedback loop using Reinforcement Learning (RL) to enhance semantic search and response generation. Using a distance-based relevance function (CQI) as a reward function, we have applied proximal policy optimization (PPO). This CQI function can identify the semantic closeness of generated answers with the most pertinent documents, allowing the model to adapt to user feedback in real time. The proposed system continually improves the quality and relevance of its outputs through adaptive learning by utilizing this RL-based optimization in the response generation and retrieval processes.

The findings provide two main implications:

(i) For researchers, the study shows that reliable SRS generation requires more than direct prompting; structure control, topic-based organization, and evaluation against research questions are necessary to make LLM-based document generation more reproducible.

(ii) For practitioners, the results imply that, rather than depending solely on an unstructured prompt, LLMs can be utilized more safely for SRS drafting when generation is driven by a predefined structure, section-wise context, and quality checks. In the proposed system, LLM supports requirement organization, drafting, completeness checking, consistency checking, and structured document generation, while human experts remain responsible for validating intentions, resolving ambiguity, and approving the final SRS.

8. Comparative Analysis

In references [26,28], software tools designed to automate requirement engineering are described. These tools facilitate the creation of Software Requirements Specification (SRS) documents through question and template engineering, where users are prompted to respond to pre-defined questions. Similarly, the 'SRS Automator' tool, as detailed in [25], guides users through a series of structured questions via an input form. These questions address a variety of software requirements, such as general descriptions, detailed specifications, initial queries, and room for miscellaneous requirements.

However, previous research indicates a limitation in which the output documents are constrained to a strict format, and the reliance on user interactions required by such tools proves to be resource-intensive in terms of time and effort. As a result, there is an obvious need for a system that runs independently and derives all of its functionality from an initial user requirements document.

To address the difficulties caused by manual querying, our approach uses LLMs to reduce reliance on template engineering and time-intensive user interaction. This approach is based on recent advances in Artificial Intelligence, especially the strong language under-

standing capabilities of LLMs. The efficiency and accuracy of the document-generating process are greatly improved by these models' ability to read user needs and produce software requirements specifications appropriately.

This system uses large language models (LLMs) to overcome the limits of traditional document generation. It moves beyond manual questioning and template engineering. This approach aims to alleviate the issues associated with strict output document formats and time-consuming user interaction prompted by predefined questions. The two widely used methods for document generation with LLMs are:

- Fine-tuning: This trains the LLM on a diverse dataset of relevant documents. It allows the model to learn the syntactic patterns and valid structures. However, fine-tuning does not provide control over the generated structure.
- Few-shot learning: This provides the LLM with a few examples of the desired structure along with the prompt. While it offers control over the generated structure, it requires that you provide an example of structure beforehand, which might not be known at the time.

Both approaches provide no direct control over the generated contents and may generate invalid and widely varying contents over little change in user requirements, making the output inconsistent [2].

Limitations and Threats to Validity

The study proposes a structured framework for deterministic SRS generation using topic classification, skeleton document generation, and reinforcement learning-based refinement. However, there are several limitations that should be considered while interpreting its results. The dataset may not accurately reflect all industrial domains, organization-specific SRS formats, or safety-critical systems, even though it was expanded beyond the PURE dataset. The evaluation primarily employs automatic measures such as ROUGE, BLEU, METEOR, cosine similarity, BERTScore, classification accuracy, and quality scores. While these metrics indicate relevance to the source documents, they cannot fully evaluate the accuracy, completeness, ambiguity, or practical usefulness of the requirements.

Several factors may vary the results. These include prompts such as requirements from analysts or stakeholders, models, retrieval settings, embedding models, clustering parameters, and preprocessing choices. Human experts remain responsible for validating the intention of the requirements. Run A and Run B were used to compare initial and expanded corpus settings, not as repeated stochastic trials with multiple random seeds. Thus, additional robustness testing is necessary. Key threats to validity include dataset bias, metric selection bias, and implementation dependency. Future work should address these issues by using larger industrial datasets, expert evaluation, more baselines, and repeated robustness tests.

The goal of streamlining and automating software requirements specification (SRS) documents has been achieved on the basis of results. By simply providing the system with raw, unstructured user requirement documents or other text-based documents pertaining to software product development, end users can save a substantial amount of time and resources when creating SRS documents. The system then produces the desired document as output. Users can further optimize the generation process by providing the system with actions that aim to modify the document according to their preferences.

This study presents an integrated document analysis system built on a Python-based framework that combines advanced semantic search and reinforcement learning techniques. The system achieves reliable, contextually aware retrieval by fine-tuning a Sentence-Transformer model and applying a customized preprocessing to raw text data. Reinforcement Learning with proximal policy optimization (PPO) significantly improves output

quality through a dynamic feedback loop. The main objective of this architecture is to provide a deterministic and adaptive response system appropriate for activities involving the generation of structured documents.

Overall, this system aims to improve the generation of software requirements specification documents while also reducing costs. By allowing developers to focus on high-level software planning and delegating formalization to the system, it facilitates a more efficient and streamlined workflow.

9. Conclusions

This research shows that when a large language model (LLM)-generating text is incorporated into a structured workflow instead of being employed through independent prompting, reliable software requirements specification (SRS) generation may be obtained. The suggested methodology combines retrieval-grounded section-wise generation, deterministic skeleton generation, embedding-based topic classification, and reinforcement learning-based refinement into a single agentic pipeline. Together, these components fix common issues with direct prompting, including structural inconsistencies, unsupported content, and incorrect section assignment.

The framework consistently outperforms conventional single-pass LLM prompting across lexical and semantic evaluation standards, such as ROUGE, METEOR, cosine similarity, and BERTScore, according to experimental results. The improvement in both content relevance and semantic coherence across these complementary metrics suggests better alignment with reference requirements. Section-wise generation based on retrieval, which restricts content to relevant source material, is the primary motivation of these advances, along with the continuous quality improvement (CQI) loop, which iteratively improves sections according to specificity, factual adherence, completeness, and structural quality.

The results emphasize the importance of topic classification and deterministic skeleton generation for sustaining document structure. While the deterministic skeleton guarantees a consistent and valid document organization regardless of model variability, appropriate requirement assignment to sections minimizes the need for redrafting and enhances retrieval quality. Furthermore, the reinforcement learning stage demonstrates efficient feedback-driven optimization, with increasing rewards and decreasing training loss indicating gradual improvements in the level of generated sections.

Overall, the findings show that combining topic-aware organization, deterministic structuring, retrieval-based generation, and reinforcement learning-based refinement produces SRS documents that are more reliable, structurally consistent, and reproducible than those generated through unconstrained prompting alone. The primary contribution of this work is therefore an integrated generation framework that combines these complementary mechanisms to support high-quality automated requirements engineering.

Author Contributions: Conceptualization, M.A.A. and R.A.; methodology, M.A.A.; software, M.A.A.; validation, M.A.A. and R.A.; formal analysis, M.A.A.; investigation, M.A.A.; resources, M.A.A.; data curation, M.A.A.; visualization, M.A.A.; writing—original draft preparation, M.A.A.; writing—review and editing, M.A.A. and R.A.; supervision, R.A.; project administration, R.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data sources used in this study were collected from publicly accessible repositories and datasets with explicit open-use permissions, including GNU General Public License, Creative Commons licenses, public-domain declarations, or equivalent institutional permission statements. The PURE dataset is available through Zenodo at https://doi.org/10.5281/zenodo.1414117, accessed on 1 January 2026. Documents without clear license information or reuse permission were excluded from the corpus.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Huang, L.; Yu, W.; Ma, W.; Zhong, W.; Feng, Z.; Wang, H.; Chen, Q.; Peng, W.; Feng, X.; Qin, B.; et al. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans. Inf. Syst.* **2025**, *43*, 1–55. [CrossRef]
2. Ouyang, S.; Zhang, J.M.; Harman, M.; Wang, M. An Empirical Study of the Non-Determinism of ChatGPT in Code Generation. *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–28. [CrossRef]
3. Askari, A.; Aliannejadi, M.; Meng, C.; Kanoulas, E.; Verberne, S. Expand, Highlight, Generate: RL-driven Document Generation for Passage Reranking. In *Proceedings of the ResearchGate*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023. [CrossRef]
4. Lee, M.; Liang, P.; Yang, Q. CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*; Association for Computing Machinery: New York, NY, USA, 2022; pp. 1–19. [CrossRef]
5. Krishna, K.; Chang, Y.; Wieting, J.; Iyyer, M. RankGen: Improving Text Generation with Large Ranking Models. *arXiv* **2022**, arXiv:2205.09726. [CrossRef]
6. Mitchell, E.; Lee, Y.; Khazatsky, A.; Manning, C.D.; Finn, C. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. *arXiv* **2023**, arXiv:2301.11305. [CrossRef]
7. Baldini, I.; Wei, D.; Ramamurthy, K.N.; Yurochkin, M.; Singh, M. Your fairness may vary: Pretrained language model fairness in toxic text classification. *arXiv* **2022**, arXiv:2108.01250. [CrossRef]
8. Naveed, H.; Khan, A.U.; Qiu, S.; Saqib, M.; Anwar, S.; Usman, M.; Akhtar, N.; Barnes, N.; Mian, A. A Comprehensive Overview of Large Language Models. *arXiv* **2024**, arXiv:2307.06435. [CrossRef]
9. Wang, D.; Raman, N.; Sibue, M.; Ma, Z.; Babkin, P.; Kaur, S.; Pei, Y.; Nourbakhsh, A.; Liu, X. DocLLM: A layout-aware generative language model for multimodal document understanding. *arXiv* **2023**, arXiv:2401.00908. [CrossRef]
10. Liu, Y.; He, H.; Han, T.; Zhang, X.; Liu, M.; Tian, J.; Zhang, Y.; Wang, J.; Gao, X.; Zhong, T.; et al. Understanding LLMs: A Comprehensive Overview from Training to Inference. *arXiv* **2024**, arXiv:2108.01250. [CrossRef]
11. Sun, H. Reinforcement Learning in the Era of LLMs: What is Essential? What is needed? An RL Perspective on RLHF, Prompting, and Beyond. *arXiv* **2023**, arXiv:2310.06147. [CrossRef]
12. Li, Z.; Yang, Z.; Wang, M. Reinforcement Learning with Human Feedback: Learning Dynamic Choices via Pessimism. *arXiv* **2023**, arXiv:2305.18438. [CrossRef]
13. Zadenoori, M.A.; Dąbrowski, J.; Alhoshan, W.; Zhao, L.; Ferrari, A. Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review. *arXiv* **2025**, arXiv:2509.11446. [CrossRef]
14. Vogelsang, A.; Fischbach, J. Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline. *arXiv* **2024**, arXiv:2402.13823. [CrossRef]
15. Eltahier, S.; Dawood, O.; Saeed, I. BERT Fine-Tuning for Software Requirement Classification: Impact of Model Components and Dataset Size. *Information* **2025**, *16*, 981. [CrossRef]
16. Hoy, Z.; Xu, M. Agile Software Requirements Engineering Challenges-Solutions—A Conceptual Framework from Systematic Literature Review. *Information* **2023**, *14*, 322. [CrossRef]
17. Krishna, M.; Gaur, B.; Verma, A.; Jalote, P. Using LLMs in Software Requirements Specifications: An Empirical Evaluation. In Proceedings of the 2024 IEEE 32nd International Requirements Engineering Conference (RE), Reykjavik, Iceland, 24–28 June 2024; pp. 475–483. . [CrossRef]
18. Zhu, T.; Cordeiro, L.C.; Sun, Y. ReqInOne: A Large Language Model-Based Agent for Software Requirements Specification Generation. *arXiv* **2025**, arXiv:2508.09648. [CrossRef]
19. *IEEE Std 830-1998*; IEEE Recommended Practice for Software Requirements Specifications. IEEE: New York, NY, USA, 1998; pp. 1–40. [CrossRef]
20. Saad-Falcon, J.; Barrow, J.; Siu, A.; Nenkova, A.; Yoon, D.S.; Rossi, R.A.; Dernoncourt, F. PDFTriage: Question Answering over Long, Structured Documents. *arXiv* **2023**, arXiv:2309.08872. [CrossRef]
21. Bonifacio, L.; Abonizio, H.; Fadaee, M.; Nogueira, R. InPars: Data Augmentation for Information Retrieval using Large Language Models. *arXiv* **2022**, arXiv:2202.05144. [CrossRef]

22. Perot, V.; Kang, K.; Luisier, F.; Su, G.; Sun, X.; Boppana, R.S.; Wang, Z.; Wang, Z.; Mu, J.; Zhang, H.; et al. LMDX: Language Model-based Document Information Extraction and Localization. *arXiv* **2024**, arXiv:2309.10952. [CrossRef]
23. Li, Y.; Tao, C.; Zhang, B.; Zhang, W. Structure-Aware Generative Information Extraction via Feature Space Alignment. *Information* **2026**, *17*, 409. [CrossRef]
24. Muludi, K.; Fitria, K.M.; Triloka, J.; Sutedi. Retrieval-Augmented Generation Approach: Document Question Answering using Large Language Model. *Int. J. Adv. Comput. Sci. Appl.* **2024**, *15*, 776–785. [CrossRef]
25. Duggal, M.; Saxena, N.; Gurve, M. SRS Automator—An Attempt to Simplify Software Development Lifecycle. In Proceedings of the 2020 6th International Conference on Signal Processing and Communication (ICSC), Noida, India, 5–7 March 2020; pp. 278–283. [CrossRef]
26. Andreou, A. A Novel Software Tool for Supporting and Automating the Requirements Engineering Process with the Use of Natural Language. *Int. J. Comput. Theory Eng.* **2013**, *5*, 443–449. [CrossRef]
27. Anuar, U.; Ahmad, S.; Emran, N.A. An Empirical Investigation on a Tool-Based Boilerplate Technique to Improve Software Requirement Specification Quality. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 397–401 [CrossRef]
28. Georgiades, M.G.; Andreou, A.S. Automatic generation of a Software Requirements Specification (SRS) document. In Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, Cairo, Egypt, 29 November–1 December 2010; pp. 1095–1100. [CrossRef]
29. Wei, X.; Wang, Z.; Yang, S. An Automatic Generation and Verification Method of Software Requirements Specification. *Electronics* **2023**, *12*, 2734. [CrossRef]
30. Ferrari, A.; Spagnolo, G.O.; Gnesi, S. PURE: A Dataset of Public Requirements Documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)* ; IEEE: New York, NY, USA, 2018; pp. 502–505. [CrossRef]
31. Ferrari, A.; Spagnolo, G.O.; Gnesi, S. PURE: A Dataset of Public Requirements Documents. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, 4–8 September 2017; pp. 502–505. [CrossRef]
32. Tunstall, L.; Reimers, N.; Jo, U.E.S.; Bates, L.; Korat, D.; Wasserblat, M.; Pereg, O. Efficient Few-Shot Learning Without Prompts. *arXiv* **2022**, arXiv:2209.11055. [CrossRef]
33. Fei, W.; Niu, X.; Zhou, P.; Hou, L.; Bai, B.; Deng, L.; Han, W. Extending Context Window of Large Language Models via Semantic Compression. *arXiv* **2023**, arXiv:2312.09571. [CrossRef]
34. Buchmann, J.; Eichler, M.; Bodensohn, J.M.; Kuznetsov, I.; Gurevych, I. Document Structure in Long Document Transformers. *arXiv* **2024**, arXiv:2401.17658. [CrossRef]
35. Weller, O.; Lo, K.; Wadden, D.; Lawrie, D.; Durme, B.V.; Cohan, A.; Soldaini, L. When do Generative Query and Document Expansions Fail? A Comprehensive Study Across Methods, Retrievers, and Datasets. *arXiv* **2024**, arXiv:2309.08541. [CrossRef]
36. Billerbeck, B.; Zobel, J. Techniques for Efficient Query Expansion. In *Proceedings of the String Processing and Information Retrieval*; Apostolico, A., Melucci, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 30–42. [CrossRef]
37. Zhu, L.; Wang, X.; Wang, X. JudgeLM: Fine-tuned Large Language Models are Scalable Judges. *arXiv* **2025**, arXiv:2310.17631. [CrossRef]
38. Liu, Y.; Singh, A.; Freeman, C.D.; Co-Reyes, J.D.; Liu, P.J. Improving Large Language Model Fine-tuning for Solving Math Problems. *arXiv* **2023**, arXiv:2310.10047. [CrossRef]
39. Jiang, T.; Huang, S.; Luan, Z.; Wang, D.; Zhuang, F. Scaling Sentence Embeddings with Large Language Models. *arXiv* **2023**, arXiv:2307.16645. [CrossRef]
40. Jiang, A.Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D.S.; Casas, D.d.l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. Mistral 7B. *arXiv* **2023**, arXiv:2310.06825. [CrossRef]
41. Schopf, T.; Schneider, D.N.; Matthes, F. Efficient Domain Adaptation of Sentence Embeddings Using Adapters. In Proceedings of the Conference Recent Advances in Natural Language Processing—Large Language Models for Natural Language Processings, Varna, Bulgaria, 4–6 September 2023; pp. 1046–1053. [CrossRef] [PubMed]
42. Parnami, A.; Lee, M. Learning from Few Examples: A Summary of Approaches to Few-Shot Learning. *arXiv* **2022**, arXiv:2203.04291. [CrossRef]
43. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
44. Ye, Y.; Cong, X.; Tian, S.; Qin, Y.; Liu, C.; Lin, Y.; Liu, Z.; Sun, M. Rational Decision-Making Agent with Internalized Utility Judgment. *arXiv* **2025**, arXiv:2308.12519. [CrossRef]
45. Zheng, R.; Dou, S.; Gao, S.; Hua, Y.; Shen, W.; Wang, B.; Liu, Y.; Jin, S.; Liu, Q.; Zhou, Y.; et al. Secrets of RLHF in Large Language Models Part I: PPO. *arXiv* **2023**, arXiv:2307.04964. [CrossRef]
46. Korga, A.M.; Wefers, S.; Hanken, K.; Tareaf, R.B.; Steemers, B.; Avvad, H. Does Size Matter? Examining Sentence Similarity Performance in Large Language Models. In Proceedings of the 2025 International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, 15–17 January 2025; pp. 595–600. [CrossRef]

47. Akhtar, M.A. Supplementary Reproducibility Materials for Deterministic SRS Generation Using LLMs and Reinforcement Learning. *Zenodo* **2026.** [CrossRef]
48. Duarte, A.V.; Marques, J.; Graça, M.; Freire, M.; Li, L.; Oliveira, A.L. LumberChunker: Long-Form Narrative Document Segmentation. *arXiv* **2024**, arXiv:2406.17526. [CrossRef]
49. Leng, J.; Huang, C.; Zhu, B.; Huang, J. Taming Overconfidence in LLMs: Reward Calibration in RLHF. In *International Conference on Learning Representations*; OpenReview.net: Online, 2025; Volume 2025, pp. 16484–16517.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.