



INFORMATICS
INSTITUTE OF
TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

Hybrid Prompt Compression for RAG Pipelines

A Project Proposal by

Chenula Senkith Jayasinghe

Supervised by

Mr Vinula Uthsara Buthgamumudalige

02/2025

Abstract

With the huge growth of unstructured textual data in the healthcare domain, retrieval and summarization of relevant information is a critical problem for healthcare professionals and researchers. We propose a Retrieval-Augmented Generation (RAG) pipeline for prompt compression to minimize tokens while achieving a high response accuracy. It combines dense vector-based retrieval using FAISS with two stage compression: an extractive stage to identify the key sentences and an abstractive stage to compress the extracted information.

The user's query is combined with the compressed content for the large language models (LLMs) such as GPT-3 or ChatGPT, creating an optimized prompt for an effective and cost-efficient response. For query encoding, the project utilizes Sentence-BERT, for abstractive compression it uses BART, and for user interaction it utilizes Streamlit interface. Finally, the system is evaluated using metrics including compression ratio and token savings, ROUGE scores and response accuracy. This project seeks to provide a practical and scalable solution to the problem of querying large medical datasets with low token consumption in LLM based applications.

Table of Contents

Table of Contents	3
List of Tables	5
List of Figures	6
Chapter 1: Introduction	7
1.1 Chapter Overview	7
1.2 Problem Identification	7
1.3 Problem Background / Problem Domain	8
1.4 Problem Definition	10
1.5 Research Gap	11
1.6 Contribution to the Body of Knowledge	14
1.7 Research Aim	17
1.8 Research Objectives	17
1.9 Novelty	19
1.10 Chapter Summary	21
Chapter 2: Requirements Specification	22
2.1 Chapter Overview	22
2.2 Rich Picture Diagram	22
2.3 Stakeholder Analysis	23
2.4 Requirement Elicitation Methodologies	23
2.5 Findings from Elicitation	24
2.6 Context Diagram	27
2.7 Use Case Diagram and Descriptions	27
2.8 Functional and Non-Functional Requirements	28
2.9 Chapter Summary	30
Chapter 3: Design	31
3.1 Chapter Overview	31
3.3 System Architecture Design	32
3.4 Detailed Design	35
3.5 Algorithm Design	35
3.6 UI Design Wireframe	37
3.7 Chapter Summary	38

Chapter 4: Implementation	39
4.1 Chapter Overview	39
4.2 Technology Selection	39
4.3 Core Functionalities Implementation	43
4.4 User Interface Implementation	46
4.5 Challenges and Solutions	50
4.6 Chapter Summary	51
Chapter 5: Testing	52
5.1 Chapter Overview	52
5.2 Testing Criteria	52
5.3 Functional Testing	55
5.4 Non-Functional Testing	57
5.5 Model Testing (for ML Projects)	59
5.6 Edge Case Testing	60
5.7 Limitations of Testing Process	60
5.9 Chapter Summary	61
Chapter 6: Interim Conclusion	62
6.1 Chapter Overview	62
6.2 Achievements of Aims & Objectives	62
6.3 Challenges and Lessons Learned	62
6.4 Remaining Work	64
6.5 Chapter Summary	65

List of Tables

Table 1 Summary of Research Gap	14
Table 2 Research Objectives	19
Table 3 Requirement Elicitation Methodologies.....	24
Table 4 Litreature Review	25
Table 5 Interviews	26
Table 6 Functional Requirements.....	29
Table 7 Non - Functional Requirements	30
Table 8 Design Goals.....	32
Table 9 Technology Stack.....	40
Table 10 Dataset Selection	41
Table 11 Programming Languages	42
Table 12 Library Toolkit.....	43
Table 13 Core Functionality Implementation	44
Table 14 UI Elements.....	47
Table 15 Challengers and Solutions.....	51
Table 16 Non-Functional Requirements	53
Table 17 Model Performance Metrics.....	54
Table 18 Functional Testing	57
Table 19 Usability Testing	58
Table 20 Dataset Testing	59

List of Figures

Figure 1 Rich Picture Diagram	23
Figure 2 Context Diagram	27
Figure 3 Architecture Diagram	33
Figure 4 Query Embedding	36
Figure 5 Hybrid Compression.....	36
Figure 6 Prompt Generation.....	37
Figure 7 UI Wireframe.....	37
Figure 8 User Interface	49

Chapter 1: Introduction

1.1 Chapter Overview

The purpose of this chapter is to give an initial view of the problem domain and the need of resolving the main challenges in retrieving and compressing unstructured medical data. It lays the foundation for the research objectives, methodology and the entire project structure. This chapter justifies that existing systems have limitations, and thus a Retrieval-Augmented Generation (RAG) pipeline that combines dense retrieval and prompt compression is needed to solve the problem identified.

The chapter connects the problem domain to the following chapters by explaining the issues of working with large amounts of unstructured data in healthcare domain and making input suitable for big language models (LLMs). Current methods either generate semantically meaningful retrieval or generate summaries that are too long to be efficiently processed by LLMs. Thus, the root causes and gaps in current systems need to be understood to help develop a good solution.

Additionally, this chapter also introduces the research gap and the proposed approach by providing a combination of dense retrieval using FAISS and extractive abstractive compression to address the limitations of traditional retrieval and summarization methods. This chapter lays out the objectives and contributions of the project by setting the scene for the system design, implementation, and evaluation of the remaining chapters.

1.2 Problem Identification

This section is to define the real world problem this project wants to solve, both in technical and practical terms. It focuses on the importance of the problem within the domain of healthcare informatics, more specifically, the efficient retrieval and compression of large scale drug review data for large language models (LLMs).

There are large collections of user generated content (reviews, feedback and experiences) for various medications in health care platforms like WebMD and UCI ML Drug Reviews. In these reviews, drug effectiveness, side effects, dosage experiences are described with the help of different patients and healthcare professionals. Although this data has good information, it is unstructured, repetitive and redundant, so it is difficult to retrieve and summarize relevant information quickly.

The problem is, as follows:

1. Large Volume of Data:

The reviews are usually in thousands, and a user cannot process them manually.

2. Scattered and Redundant Information:

Scattered across multiple reviews and merged with repetitions, important details such as side effects, dosage recommendations and treatment outcomes are mixed. But to search for concise and accurate information is difficult.

3. Complexity of Summarization:

There are challenges in generating summaries from long texts from user generated content, which vary in writing styles, formats and may contain irrelevant information.

4. LLM Token Constraints:

When interfacing with LLMs like GPT3 or ChatGPT, you have to deal with tokens limitations since such models have a fixed number of tokens per request (e.g., GPT3 has a maximum of 4,096 tokens per request). For example, if a query produces too many uncompressed chunks, the LLM may produce half or incomplete answers.

1.3 Problem Background / Problem Domain

Digital data production in the healthcare industry has experienced a rapid growth due to innovations in online health platforms, electronic health records (EHRs), and patient generated content. WebMD, Drugs.com, and UCI ML Drug Reviews are platforms with user feedback about medications such as reports of side effects, dosage effectiveness and patient experiences. Nevertheless, the data in this study is unstructured and textual in nature, which makes efficient retrieval and processing difficult.

Trends Shaping the Problem Domain

1. Growth of User-Generated Content:

As medical advice and patient reviews are increasingly sought out online, there are huge unstructured datasets being amassed. Because patients often have subjective and diverse experiences with medications, they have variability in how they write, what they write, and in the structure of the reviews.

Constraints: Drug effectiveness and side effect expressions are diverse and should be understood by a retrieval system.

2. Adoption of Large Language Models (LLMs):

Applications of such advances in Natural Language Processing (NLP), including GPT-3, BERT and ChatGPT, include automated response generation, clinical decision support and personalized recommendations. Nonetheless, these models are limited by token limits and efficient input prompt construction is essential for real world use.

Optimization: In order to interact with LLMs cost effectively, we need to optimize token usage via prompt compression.

3. Emphasis on Evidence-Based Medicine:

There is an increasing focus on data-driven, evidence-based decision-making in healthcare. Medical professionals rely on aggregated insights from multiple patient experiences and clinical outcomes to make informed decisions. However, the manual aggregation of scattered and redundant information from large datasets is time-consuming and prone to human error.

Implication: Automated systems for retrieving and summarizing key information are essential to support evidence-based decision-making.

4. LLM-Compatible Prompt Construction

The final compressed prompt must contain sufficient context and information to allow the LLM to generate relevant, accurate, and factually correct responses to the user query. Poorly structured or excessively compressed prompts may lead to hallucinations or inaccurate outputs from the LLM. The challenge is to optimize token savings without sacrificing the completeness and correctness of the generated response.

1.4 Problem Definition

Problem Statement

“ How can we efficiently retrieve and compress large-scale, unstructured drug review data to construct token-optimized prompts for large language models (LLMs) while maintaining accuracy, contextual relevance, and minimal token usage? ”

Technical and Operational Aspects of the Problem

1. Efficient Information Retrieval

Many large medical datasets like WebMD Drug Reviews contain thousands of user generated reviews for drugs with descriptions of how effective it is, what side effects it creates, and patient experience. The problem is how we can retrieve only the most relevant knowledge chunks which correspond to a user query while retrieving the knowledge chunks in a way that the knowledge retrieval system understands semantic meaning, not a simple keyword match.

Operational Challenge: Traditional retrieval systems based on keyword search methods are not good at understanding complex natural language queries and frequently produce irrelevant or incomplete results. Thus, it needs a dense vector based retrieval system (using FAISS) to achieve high retrieval accuracy.

2. Compression of Retrieved Information

After that, they have to compress the relevant reviews into a small token footprint to pass as a prompt to large language models. Since LLMs like GPT-3 have strict token limitations (i.e., 4,096 tokens per request for GPT-3), they cannot process long, uncompressed documents directly. The prompt construction may exclude or truncate critical details without compression.

Technical Challenge: The compression must be able to strike a balance between decreasing token count and maintaining important information. Truncation or extractive summarization may not capture nuances and context. Thus, a two stage compression method is required.

Stage 1 (Extractive Compression): Extractive compression, that is, identify key sentences directly from the retrieved reviews.

Stage 2 (Abstractive Compression): Compress and rephrase the extracted sentences to make them coherent, and to be token efficient.

3. LLM-Compatible Prompt Construction

The compressed prompt has to have enough context and information for LLM to generate relevant, accurate, and factually correct response to the user query. If prompts are badly structured or too compact, then this can cause hallucinations or incorrect outputs by the LLM. The difficulty is to achieve token savings optimality while keeping the generated response as complete and correct as possible.

1.5 Research Gap

1.5.1 Overview of Existing Solutions

There are numerous studies and systems which focused on information retrieval, summarization, and processing very large text datasets using large language models (LLMs). Traditional approaches include:

1. Information Retrieval Systems:

The commonly used ways of retrieving relevant documents based on user queries are by keyword based search engines and by TF-IDF based retrieval systems.

For dense vector based retrieval systems, like those using BERT embeddings or FAISS, retrieving using semantic meaning has been shown to improve performance over exact keyword matches.

2. Summarization Techniques:

Extractive summarization: Selects the most important sentences from the text verbatim. Both models such as BERTSUM and Sumy fall in this category.

Abstractive Summarization: Generating concise, human-like summaries that rephrase the key information are referred to as abstractive summarization. BART and PEGASUS are popular abstractive models.

3. Prompt Construction for LLMs:

The current system finds user queries and give them to LLMs, or reduce the token count of the input by means of generic summarization.

1.5.2 Deficiencies in Existing Systems and Knowledge

Despite advancements in retrieval, summarization, and LLM-based applications, several unresolved issues remain:

1. Inefficient Retrieval Mechanisms:

Retrieval systems based on keywords fail to provide semantic meaning of user query. For example: “What are the side effects of ibuprofen?” If they don’t explicitly match the query terms, may not retrieve documents that mention related terms such as “dizziness”, or “nausea”.

Despite the density of the retrieval systems utilized, existing methods tend to design their retrieval towards query-document full document retrieval rather than relevant document chunks within that document, resulting in large and redundant retrieved outputs.

2. Lack of Token Optimization in Summarization:

Typically, summarization systems aim to produce human readable summaries and optimize for efficiency of tokens when interacting with models such as GPT-3 or ChatGPT, which are limited in terms of the number of tokens they can work with.

Extractive only systems choose redundant and verbose sentences and Abstractive only systems can lose very important details because of oversimplification. A key gap is that current systems do not have a hybrid (extractive + abstractive) approach.

3. Balancing Compression and Accuracy:

In the case of complex, multi-faceted information like drug side effects and dosage recommendations, the loss of critical detail can occur even with excessive compression.

In contrast, minimal compression causes token overload and hence yields incomplete or truncated responses from LLMs. There are not effective strategies in current literature to balance compression ratio and output accuracy.

4. Limited Integration with LLMs for Real-Time Use:

Similar to LLMs, LLMs have already been successfully used for general purpose question answer and text generation but existing systems still do not provide real time integration where user queries are taken in automatically, compressed and structured into prompts for immediate processing by LLMs.

Also, user end interfaces (e.g., healthcare professionals) are not well explored.

1.5.3 Why Addressing This Gap Is Necessary

1. Practical Need for Efficient Information Retrieval in Healthcare:

Timely access to accurate and concise drug related information can help in making decisions both for medical professionals as well as patients in the healthcare domain. However, current systems are inefficient and prone to errors as users need to manually go through several reviews.

To address this gap, we develop a dense retrieval and prompt compression pipeline that enables users to get to actionable knowledge in real time.

2. Cost-Effectiveness in Using LLMs:

GPT-3 is a large language model and it has token based pricing model. In the absence of the proper token optimization, queries may pay high costs or get truncated results from the token limit.

The project closes that gap by compressing prompts without losing key info so the costs are reduced and the LLM capabilities are used effectively.

3. Scalable and Flexible Solutions:

With the increase in volume of user generated medical content, the system must be scalable for large datasets with minimal performance loss.

This project is scalable and generalizable to other domains such as legal documents, financial reports, scientific papers by integrating dense retrieval and hybrid compression.

1.5.4 Summary of the Research Gap

Aspect	Existing Limitation	Proposed Solution
Retrieval Efficiency	Keyword-based and document-level retrieval fail to return contextually relevant chunks	Use dense vector-based retrieval via FAISS and BERT
Token Optimization	Existing summarization focuses on human readability, not token savings	Implement a two-stage compression pipeline (extractive + abstractive)
Compression-Accuracy	Excessive compression leads	Optimize compression by

Tradeoff	to loss of details, minimal compression leads to token overload	balancing token savings and output accuracy
Real-Time User Interaction	Limited integration with interactive LLM-based systems	Develop a real-time User interface for query processing

Table 1 Summary of Research Gap

1.6 Contribution to the Body of Knowledge

1.6.1 Theoretical Contributions

1. Integration of Prompt Compression into RAG Pipelines:

The main contribution of this work is to seamlessly integrate prompt compression into the RAG pipeline, which is centered around retrieval and general purpose summarization. This integration makes the pipeline a token optimized, LLM ready system by:

- A two stage compression process – first, an extractive stage to extract the best sentences from given chunks as answers to the query, and second, an abstractive stage to compress and refine the extracted information.
- Making token optimization a central component to embed, while maintaining the accuracy of LLM prompts and keeping them under token limits.

Applications: This contribution advances beyond the state of the art in RAG research by optimizing the pipeline for relevance (via retrieval) as well as token efficiency to be suitable for cost sensitive applications involving LLMs.

2. Token-Efficient Prompt Construction:

The project presents a new methodology to balance between compression and accuracy using multiple evaluation metrics such as compression ratio, token savings, and BERTScore accuracy. By doing so, this contribution ensures that the compressed prompt.

- Keeps the key information necessary for the accurate responses.
- Reduces redundancy of and irrelevance to the content so that tokens are used as little as possible.

Unlike traditional summarization systems that typically come up with long length outputs, this project presents a compression framework tailor made for LLM constraints for advancement.

3. Enhanced Retrieval through Dense Vector Representations:

This project helps with advancements in retrieval techniques by using FAISS (Facebook AI Similarity Search) and Sentence-BERT to retrieve semantically relevant chunks of information from the database using user queries.

- With smaller, contextually meaningful portions of text known as retrieved chunks, they are more efficient to process downstream and compress than complete documents.

Precision: This project increases the precision of retrieval by focusing on retrieving contextually relevant chunks, and consequently, less information needs to be compressed.

4. Evaluation Framework for Prompt Compression in RAG Pipelines:

A multi metric evaluation framework for the performance of the prompt compression process is developed in the project. The framework evaluates:

- Measuring the efficiency with which the information is compressed, i.e. token savings.
- Comparison of the original text size vs. the final compressed prompt.
- BERTScore/ROUGE scores: Maintaining the meaning of the compressed output.

Application: The framework also enables future researchers to identify the appropriate balance between token efficiency and response quality in RAG systems.

1.6.2 Practical Contributions

1. Efficient Token-Optimized LLM Prompts for Real-World Use:

Incorporating prompt compression to the RAG pipeline, the system creates compact prompts, limited to the minimum tokens to maintain the prompt intelligibility and provide enough context for correct LLM responses without using more than necessary. As a result, the cost is reduced and query efficiency is improved when querying large language models like GPT-3 or ChatGPT.

2. Scalable Integration of RAG Pipelines in Various Domains:

The architecture of the system is scalable and adjustable so it can be applied to more than just the health care industry. This provides a way of efficiently compressing large scale textual datasets such as:

Legal documents: Compressing case summaries.

Financial reports: Summarizing key financial metrics and performance indicators.

Scientific reports: Extracting and compressing key finds and conclusions from scientific research papers.

Application areas: This generalizable system can be directly applied to any industry needing concise, accurate summaries for decision making.

3. User-Friendly Interface for Querying and Compression:

The project offers an interactive Streamlit interface to users for submitting queries and receiving compressed, token efficient responses in real time. It is a practical and easy to use solution that can be interacted with by users without technical background.

Benefits: The interface fills the gap between technical NLP models and non-expert users by offering a simple query response mechanism.

4. Cost-Effective Use of LLMs:

For LLMs based on tokens, by reducing the number of tokens of inputs, the system saves costs when interacting with token based LLMs such as GPT-3. Low cost of operation is a benefit to both large enterprises and small organizations alike, while continuing to provide access to high quality, AI driven responses.

Token overload: Organizations can avoid spending prohibitively high costs on token overload when scaling the use of LLMs.

1.7 Research Aim

The goal of this work is to build a Retrieval Augmented Generation (RAG) pipeline to retrieve, compress and optimize unstructured large scale drug review data for interaction with large language models (LLMs) using dense retrieval and prompt compression.

The project seeks to achieve:

- Dense vector based methods to efficiently retrieve relevant information using dense vectors to make semantically accurate matches for user queries.
- Two stage approach (extractive and abstractive compression) of token efficient compression of retrieved content without compromising accuracy or key information.
- User friendly interface, real time interaction with concise and actionable responses for the patients and healthcare professionals.

Therefore, the research will enable meeting this goal, which will tackle key challenges of working with large scale textual data, minimizing token overhead, and performing cost effective and accurate querying of LLMs such as GPT-3 or ChatGPT.

1.8 Research Objectives

Objective	Description	Learning Outcomes
Literature Survey	<ul style="list-style-type: none">- To explore current advancements in Retrieval-Augmented Generation (RAG) pipelines and identify gaps in existing systems.- To analyze compression techniques (both extractive and abstractive) and their applications in token-efficient NLP systems.- To review previous research on dense retrieval (FAISS) and its integration with summarization systems.- To select the appropriate	LO1, LO2, LO4

	retrieval models, compression techniques, and evaluation metrics for the project.	
Requirement Analysis	<ul style="list-style-type: none"> - To gather requirements from healthcare professionals, researchers, and technical users regarding the retrieval and summarization of large medical datasets. - To define functional and non-functional requirements related to retrieval speed, token savings, and accuracy. 	LO3, LO6
Design	<ul style="list-style-type: none"> - To design the architecture for a dense retrieval and prompt compression pipeline, integrating FAISS-based retrieval and a two-stage compression module. - To define the flow of information from user query input to LLM-optimized prompt generation. - To design evaluation metrics for assessing the performance of token savings, compression ratio, and accuracy. 	LO3, LO5
Development	<ul style="list-style-type: none"> -To implement the dense retrieval system using FAISS and Sentence-BERT to return semantically relevant chunks. - To develop and integrate the two-stage compression module (extractive + abstractive) using BERT and BART models. 	LO7

Testing	<ul style="list-style-type: none"> - To create test plans for evaluating the system's performance based on compression ratio, token savings, retrieval accuracy, and response time. - To conduct unit testing for individual components (retrieval, compression) and integration testing for the overall pipeline. 	LO8
Evaluation	<ul style="list-style-type: none"> - To evaluate the system using BERTScore, ROUGE, compression ratio, and perplexity metrics. - To demonstrate the developed system to domain experts and gather feedback through a demo session or viva panel. 	LO9

Table 2 Research Objectives

1.9 Novelty

1.9.1 Novelty of the Problem

The problem addressed by this project is not merely retrieval or summarization but involves the optimization of prompts for large language models (LLMs) through efficient compression. This problem is unique because:

- Traditional RAG systems primarily focus on retrieving and summarizing information without explicitly optimizing token usage for LLMs like GPT-3 or ChatGPT.
- The challenge lies in balancing information retention and token reduction within the constraints of token-based models. This problem requires a novel approach to select, compress, and rephrase information while retaining its semantic integrity.

By defining the problem in terms of **prompt optimization for token efficiency**, this project addresses a key gap in existing research where token savings and cost reduction have been **underexplored in LLM-based applications**.

1.9.2 Novelty of the Solution

The solution presented by this project is novel in its hybrid design that integrates dense retrieval with two-stage prompt compression. Key innovative aspects include:

1. Seamless Integration of Retrieval and Compression:

- The project goes beyond conventional retrieval systems by integrating dense retrieval using FAISS with two-stage prompt compression.
- This integration enables the system to retrieve semantically relevant chunks and compress them into token-efficient prompts, making it a unique solution for real-time LLM interactions.

2. Two-Stage Compression Pipeline:

- The system combines extractive compression (using BERT) to select key sentences and abstractive compression (using BART) to refine and further compress the selected content.
- Unlike traditional summarization techniques, which either extract or paraphrase information, this hybrid approach ensures both token efficiency and semantic completeness.

3. Token Optimization as a Central Goal:

- Most summarization techniques generate human-readable summaries but do not explicitly optimize for token constraints imposed by LLMs.
- This project introduces token savings and compression ratio as key performance metrics, ensuring that the final output is both concise and informative.

1.9.3 Novelty of the Methodology

1. Application of Dense Vector Retrieval (FAISS) to Generate Compressed Prompts:

- The project applies FAISS-based dense retrieval not just to retrieve full documents but to extract contextually relevant knowledge chunks that serve as input to the compression pipeline.
- This methodology ensures that the retrieved content is already relevant and focused, reducing the need for excessive filtering and summarization.

2. Multi-Metric Evaluation of Compression:

- The project employs a **comprehensive evaluation framework** combining multiple metrics, including:
 - **Compression ratio:** To measure the reduction in token count.
 - **ROUGE and BERTScore:** To ensure semantic and contextual accuracy.
 - **Perplexity:** To measure the fluency of the compressed prompt.
- This multi-metric approach is novel in the context of RAG pipelines and can be adopted as a **best practice for future systems** involving prompt construction and optimization.

1.10 Chapter Summary

This chapter introduced the **core problem** of efficiently retrieving and compressing large-scale, unstructured drug review data for **token-optimized interactions with large language models (LLMs)**. The project aims to address the limitations of current retrieval and summarization systems by **integrating prompt compression into a Retrieval-Augmented Generation (RAG) pipeline**.

The **problem** arises from the **inefficiency of manually processing large datasets** and the **token constraints of LLMs** like GPT-3, which require optimized inputs to deliver cost-effective and accurate responses. The **research gap** identified in this chapter highlights the lack of systems that simultaneously address **retrieval efficiency and token savings through compression**.

Chapter 2: Requirements Specification

2.1 Chapter Overview

The present chapter deals with the elaboration of the overall requirement gathering process for the given research project, from the very core of research to prototype development. To start off, the identification and diagramming of stakeholders in the proposed system, their roles, and their relations with the help of graphs is performed. After which discussion and analysis of the various requirement elicitation techniques which may be used, by selecting in detail, is presented. The elicited requirements are then represented in tabular format, use case diagrams along with their descriptions are developed. Finally, the conclusions drawn from the requirement elicitation process are interpreted and summarized.

2.2 Rich Picture Diagram

The objective of a rich picture diagram is to be able to illustrate the relationships which would exist between the proposed system and its wider environment. These entities may extend beyond the tertiary relationships to the system's own domain. Also, this rich picture diagram Also demonstrates potential threats and vulnerabilities that may affect the proposed system due to activities of malicious actors and lawful competitors alike.

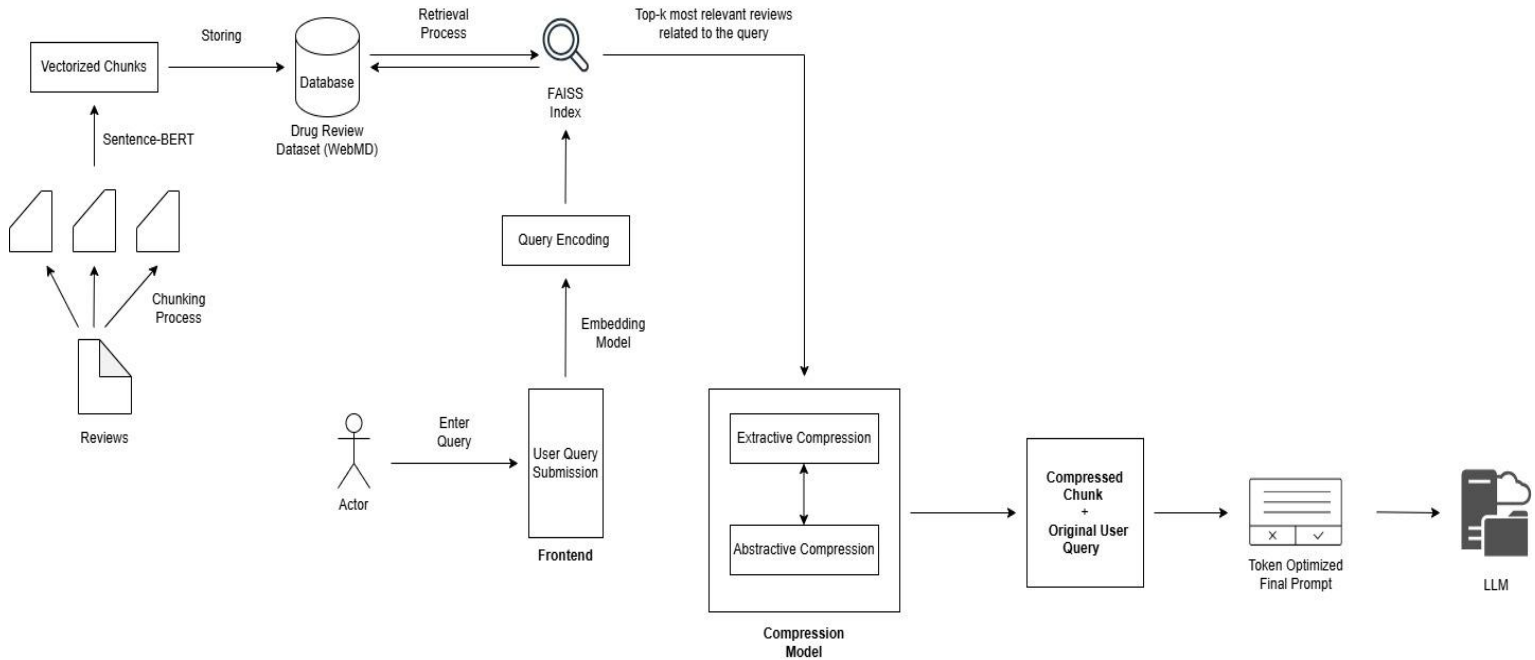


Figure 1 Rich Picture Diagram

2.3 Stakeholder Analysis

2.4 Requirement Elicitation Methodologies

Requirement Elicitation is a critical step of software development which is all about acquiring the requirements for a software project who enables the developers gain a bird's eye view of appreciation of what it is, what it seeks: to achieve, its mission and vision. The tables below are a brief profile of the selected requirement elicitation methods and rationale with a focus on the merits/benefits of each certain procedure and cons.

Technique 1 – Interviews

Some interviews were useful in establishing the current state of the issues affecting efficiency

in handling LLM's and prompt processing. The importance of the parameters for quantifying such as accuracy, response time, and token usage and domain experts assisted in the identification of possibilities for compression.

Technique 2 - Literature Review

The Literature Review offered insights into the deficiencies on current LLM solutions and provided points at which timely compression would be of value over the efficiency of the model. By reviewing the documentation of models such as GPT-3, LLama2 and BERT it was easier to pinpoint areas where compression techniques could be integrated.

Technique 3 - Brainstorming Sessions

The concept generation helped in group thinking and was useful in coming up with a novel plan of managing token issues and handlings such as maintaining essential context information. This method fostered innovativeness and was also such that enabled successive prototyping of concepts.

Table 3 Requirement Elicitation Methodologies

2.5 Findings from Elicitation

2.5.1 Literature Review

Citation	Findings
Brown et al., 2020; Devlin et al., 2019	Prior large language models (LLMs), including GPT-3 and BERT, face some difficulty in addressing the problems involving long inputs, and the trade-offs between speed and capability. The majority of existing models is designed to provide the most accurate and relevant output, but there is a number of drawbacks when using it with large inputs.
Sanh et al., 2019; Jiao et al., 2020	Experiments in the area of NLP's timely compression revealed that it is important to preserve the context to relevant and coherent results. Although to increase the speed of the process such a reduction of tokens as pruning

	or distillation often causes the loss of highly important contextual information which further influences the relevance and accuracy of the retrieved answers.
Lewis et al., 2020; Raffel et al., 2020	Studies involving BART and T5 show that it is possible to minimize the length of the input by pre-summarizing the longer text before passing it to the LLMs. These models assist in shrinking the input size but then may lose many details in solving a problem, resulting into either a partial or an erroneous one.

Table 4 Literature Review

4.5.2 Interviews

Codes	Themes	Conclusions
Prompt Compression, Information Preservation	Research Problem	The reviewed literature and discussions emphasized that prompt compression should retain essential information to maintain LLM response quality. Stakeholders agreed that striking a balance between token savings and context preservation is critical in RAG pipelines.
Token Efficiency, Reduced Query Costs	Research Gap	Research showed that token reduction stands as a primary optimization technique for NLP although most existing RAG pipelines do not use adaptive compression methods based on question difficulty level. This project fills the existing gap through optimized token management for questions and retrieved documents.
Abstractive/Extractive	Methodology	Professionals advised using a

Summarization, Multi-step Compression		two-step summarization method that combines extractive with abstractive methods when working with pharmaceutical assessment data. A two-tier process using extractive then abstractive approaches ensures efficient compression without sacrificing essential medical information or facts.
WebMD Dataset, Chunking Strategy	Datasets	Participants chose the WebMD Drug Reviews dataset together with their chunking strategy as the optimal combination for RAG implementation. Researchers agreed that vector storage and retrieval perform best when participants conduct proper pre-processing while applying chunking methods.
Embedding Models, Vectorization Techniques	System Design	Integrating vectorization techniques like Sentence-BERT and FAISS for fast and accurate retrieval was highlighted as crucial. Stakeholders stressed the importance of indexing vector embeddings efficiently to reduce retrieval latency.
Final Prompt Construction, Combined Queries	Evaluation Metrics	Evaluating the combined prompt (original user query + compressed retrieved information) using token efficiency, retrieval accuracy, and final LLM response relevance ensures comprehensive assessment of project outcomes.

Table 5 Interviews

2.6 Context Diagram

The Context Diagram gives a general description of Dynamic Prompt Compression System for LLMs and shows the ways in which DP CSS LLMs communicates with outside entities. On this diagram the system is shown in the middle with the end-users and the stakeholders placed around it indicating the inputs they submit and the outputs they get.

End Users (NLP Engineers/Data Scientists): Input plain text data to the system and obtain small text data that are efficient when fed into LLMs.

Project Owner: Provides needful information for alteration of system and gets statistic data to assess effectiveness of used system.

Quality Assurance Team: Gives information about the performance of the system and receives evaluation logs for the quality check.

This diagram assists in identifying the key duties and the major tasks of every stakeholder; at the same time this shows how the information between the stakeholders and the system may be exchanged effectively and cohesively to define the project's scope and interactions.

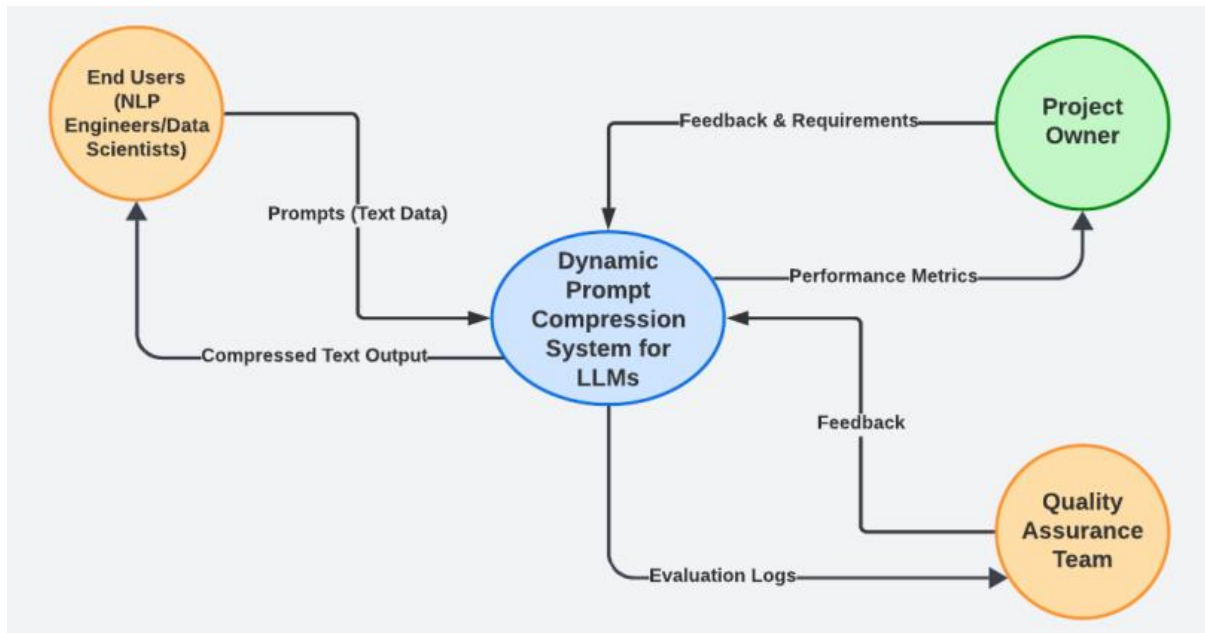


Figure 2 Context Diagram

2.7 Use Case Diagram and Descriptions

The following diagram shows the use-cases tied to the approaches that involve interactions within the system and its most important players. Furthermore, interactions between the use cases within the have also been suggested for clarity in the system have also been indicated for better understanding.

2.7.1 Use Case Diagram

2.7.2 Use Case Descriptions

2.8 Functional and Non-Functional Requirements

2.8.1 Functional Requirements (FR)

These requirements define the core functions that the system must perform to achieve its objectives.

ID	Requirement	Priority (MoSCoW)	Description
FR1	The system shall accept user queries through the front-end.	Must Have	Users can input questions related to drugs or treatments.
FR2	The system shall retrieve relevant chunks from the vectorized database.	Must Have	The retrieval should be based on FAISS and embedding-based similarity search.
FR3	The system shall compress retrieved data using extractive and abstractive methods.	Must Have	Compress the retrieved chunks using the multi-step compression approach.
FR4	The system shall combine the user query and compressed content into a final prompt	Must Have	The combined prompt is formatted for optimal input to the LLM.

FR5	The system shall pass the final prompt to an LLM to generate responses.	Must Have	Utilize an OpenAI API or other LLM for answer generation.
FR6	The system shall provide an option for users to copy the final compressed prompt.	Should Have	Users can copy the compressed prompt for further use.
FR7	The system shall display relevant drug names alongside the summarized responses.	Should Have	Provide drug context within the final output to ensure clarity.
FR8	The system shall log user queries and responses for performance evaluation.	Could Have	Maintain a log to analyze and refine the retrieval and compression process.

Table 6 Functional Requirements

2.8.2 Non-Functional Requirements (NFR)

ID	Requirement	Priority (MoSCoW)	Description
NFR1	The system shall provide responses within 5 seconds of query submission.	Must Have	Ensures acceptable system response time for user interaction.
NFR2	The system shall maintain high retrieval accuracy with minimal token loss.	Must Have	The token compression must preserve important context for generating accurate responses.
NFR3	The system shall support scalability with large medical datasets.	Should Have	The system must handle large datasets (e.g., WebMD reviews) efficiently for scalable operation.
NFR4	The system shall be	Should Have	Simple and intuitive

	user-friendly and easy to navigate for non-technical users.		interface design to ensure ease of access to features.
NFR5	The system shall ensure data security and privacy.	Must Have	User queries and retrieved data should be protected from unauthorized access.
NFR6	The system shall allow for easy updates to embedded models and summarization techniques.	Could Have	Modular architecture to allow replacing models without affecting the overall system.
NFR7	The system shall maintain minimal computational overhead during compression.	Should Have	Ensure that the compression process does not introduce significant delays.
NFR8	The system shall provide high availability and reliability.	Could Have	Ensure minimal downtime and consistent performance across use cases.

Table 7 Non - Functional Requirements

2.9 Chapter Summary

The chapter on the Software Requirement Specification has given an understanding of the functional and non-functional requisition of the system, how these requirements are elicited, and the procedures as well as tools and techniques that have been adopted to meet the stakeholder's needs. The system requirements were prioritised according to the MoSCoW principle to identify what is must have for the success of the system.

Chapter 3: Design

3.1 Chapter Overview

The design determinations for developing an appropriate system architecture will receive detailed analysis in this chapter. The design of the proposed system stems from requirements identification in an earlier chapter. First, the identified design goals will be displayed in a table structure while high-level design together with low level design and system process flow chart along with UI wireframes will be presented. The creation of system process flow charts together with UI wireframes stands alongside other associated design diagrams. A complete rationale backs each decision during design.

3.2 Design Goals

Design Goal	Description
Performance	<p>Research Component: The integration of prompt compression into RAG pipelines must demonstrate improved efficiency in generating accurate responses while optimizing token usage. The system should minimize the number of tokens while maintaining response relevance.</p> <p>Prototype Component: The prototype should ensure that prompt compression and retrieval do not introduce significant latency, aiming for response times within user-acceptable limits (e.g., under 5 seconds).</p>
Adaptability	The system should be adaptable to different knowledge bases and datasets. While this project focuses on the WebMD Drug Review Dataset , the architecture should allow easy extension to other medical or domain-specific datasets without requiring significant redesign.
Usability	Since the application targets both technical and non-technical users, the user interface (via Streamlit) should be clean, minimalistic, and intuitive. Features such as copyable compressed prompts, easy input of user queries, and accessible feedback mechanisms should be emphasized to ensure a positive user experience.

Scalability	The RAG pipeline should be capable of scaling horizontally to handle large medical datasets (e.g., millions of records). The FAISS index and vectorization processes should efficiently retrieve relevant information even as the dataset grows.
Correctness	<p>Compression Accuracy: The compressed prompt must preserve key facts and important context to ensure that LLM responses are correct and contextually appropriate.</p> <p>Information Retrieval: Ensure the retrieval mechanism correctly identifies relevant reviews and minimizes irrelevant or redundant data.</p>
Maintainability	The system architecture should be modular, allowing for future updates or replacements of the embedding models, retrieval mechanism, or compression strategies without affecting the overall system. Documentation and code structure should support ease of maintenance and future improvements.
Security and Privacy	Given that the system involves sensitive medical data, it is important to follow best practices for data security, ensuring that queries, retrieved data, and responses are properly encrypted and protected from unauthorized access. The design should also ensure that user interaction data is handled in compliance with data privacy regulations.

Table 8 Design Goals

3.3 System Architecture Design

3.3.1 Architecture Diagram

The project demands a tiered model which separates presentation from logic from data tiers. The selection of a system architecture that separated physical components between tiers occurred because of the project requirements. The following diagram the different components which exist within their corresponding tiers. The Presentation tier deals User Interface components of the system receive input from end users while Data tier manages data storage and retrieval operations. The data management system through the Logic tier connects presentation and data functions by enabling system functionality functionalities of the system.

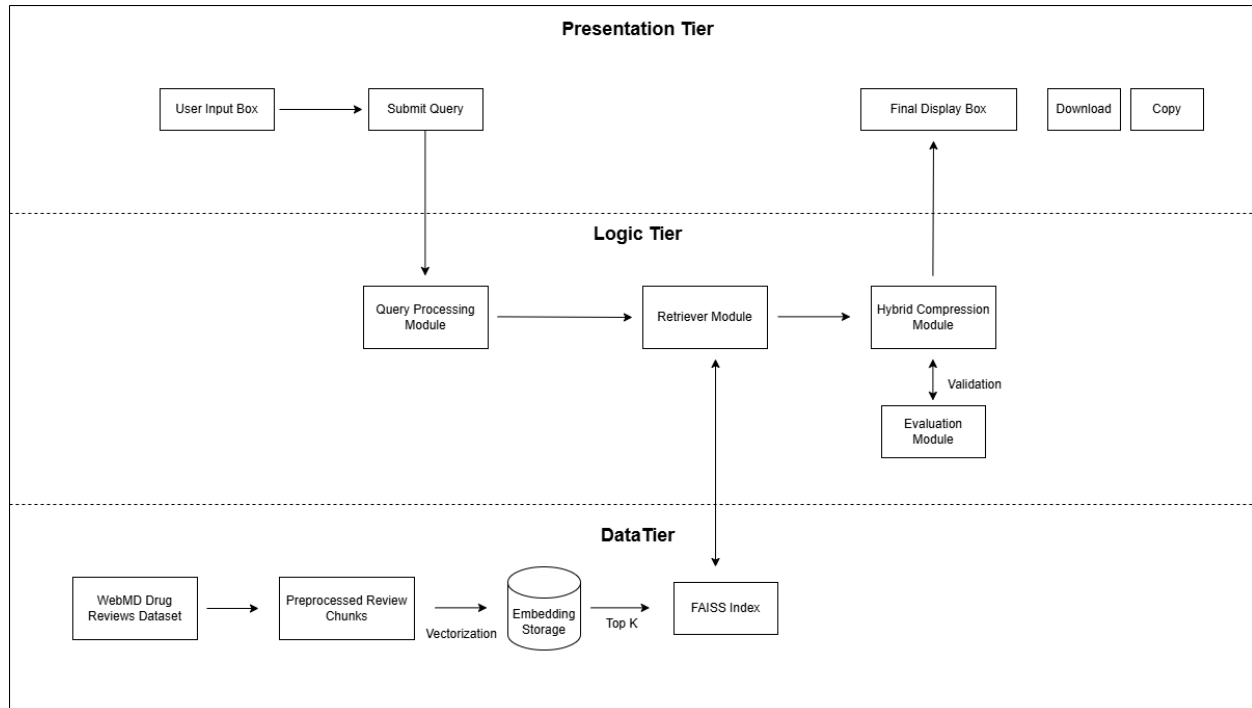


Figure 3 Architecture Diagram

3.3.2 Discussion of Tiers

Presentation Tier

User Input Box:

- Description: This is the interface where the user enters their query. It serves as the main interaction point between the user and the system.

Submit Button:

- Description: A button to initiate the process. When clicked, it triggers the query processing and retrieval workflow.

Final Display Box:

- Description: Displays the summarized response generated by the system, combining the original query with relevant compressed data.

Download Button:

- Description: Allows users to download the final prompt containing the query and retrieved summary.

Copy Button:

- Description: Enables users to copy the final prompt to the clipboard for use in other applications.

Logic Tier**Query Processing Module:**

- Description: Converts the user's input into vector form using an embedding model. It ensures the query is properly formatted for search and retrieval.

Retriever Module:

- Description: Searches the FAISS index to find the most relevant preprocessed chunks corresponding to the user query.

Hybrid Compression Module:

- Description: Applies extractive and abstractive summarization techniques to compress the retrieved chunks into a concise, informative summary.

Evaluation Module:

- **Description:** Evaluates the output of the system using metrics such as ROUGE, BERTScore, and Recall @ K to ensure quality and effectiveness of the retrieved and compressed data.

Data Tier

1. WebMD Drug Reviews Dataset:

- **Description:** Contains the raw data, including drug reviews, side effects, dosage details, and user experiences.

2. Preprocessed Review Chunks:

- **Description:** The dataset is split into smaller, manageable chunks for efficient processing and embedding.

3. Embedding Storage:

- **Description:** Stores the vector representations of the review chunks generated using Sentence-BERT. These vectors are used for efficient similarity searches.

4. FAISS Index:

- **Description:** An optimized search index that facilitates fast retrieval of the most relevant chunks based on the query's vector representation. It acts as the core retrieval engine for the system.

3.4 Detailed Design

- **Purpose:** Provide an in-depth look at system components.
- **Content:**
 - Use diagrams like class, sequence, or flow diagrams.
 - Explain how each component supports the system's goals.

3.5 Algorithm Design

This section presents the design of the key algorithms used in the project. The algorithms focus on processing user queries, retrieving relevant review chunks using the FAISS index, applying hybrid compression, and generating optimized responses.

Key Algorithms and Pseudocode:

1. Algorithm: Query Embedding and Retrieval using FAISS Index

Convert the user's natural language query into an embedding and retrieve the most relevant review chunks using FAISS similarity search.

```
Algorithm: RetrieveRelevantChunks
Input: user_query, embedding_model, faiss_index, top_k
Output: relevant_chunks

1. query_embedding ← embedding_model.encode(user_query)
2. distances, indices ← faiss_index.search(query_embedding, top_k)
3. relevant_chunks ← []
4. for i in indices[0] do
    Add the corresponding chunk from the dataset to relevant_chunks
5. return relevant_chunks
```

Figure 4 Query Embedding

2. Algorithm: Hybrid Compression

Compress the retrieved chunks by combining extractive and abstractive summarization. First, extract sentences containing key phrases, and then refine the summary using a language model.

```
Algorithm: HybridCompression
Input: retrieved_chunks, abstractive_summarizer, num_sentences
Output: compressed_summary

1. combined_text ← Concatenate all retrieved_chunks
2. sentences ← Tokenize combined_text into individual sentences
3. prioritized_sentences ← []
4. keywords ← ["side effects", "dosage", "effectiveness", "results", "treatment", "symptoms"]
5. for each sentence in sentences do
    if sentence contains any keyword in keywords then
        Add sentence to prioritized_sentences

6. if length(prioritized_sentences) < num_sentences then
    Add general sentences from combined_text until num_sentences is met

7. extracted_summary ← Join prioritized_sentences into a single block of text
8. compressed_summary ← abstractive_summarizer(extracted_summary, max_length=300, min_length=
9. return compressed_summary
```

Figure 5 Hybrid Compression

3. Algorithm: Final Prompt Generation

Combine the user query with the compressed information to generate a final prompt for the LLM.

```
Algorithm: GenerateFinalPrompt
Input: user_query, compressed_summary
Output: final_prompt

1. final_prompt ← "User Query: " + user_query + "\n\nRetrieved Information: " + compressed_su
2. return final_prompt
```

Figure 6 Prompt Generation

3.6 UI Design Wireframe

RAG Pipeline for Drug Review Summarization

Enter a question to retrieve and summarize relevant drug reviews.

Enter your query (e.g., 'Medications for Chest Pain?')

Final Prompt

Figure 7 UI Wireframe

3.7 Chapter Summary

The chapter provided an in-depth description of system design including the methodologies that would be used during the implementation of the proposed solution. Design goals led the development of high-level and low-level architectural components which were structured through relevant design diagrams. The selected software design paradigm received justification while the system design included supporting diagrams which displayed components and data flow and process flow and user interfaces. System implementation details will be discussed in the following chapter according to design outline specifications.

Chapter 4: Implementation

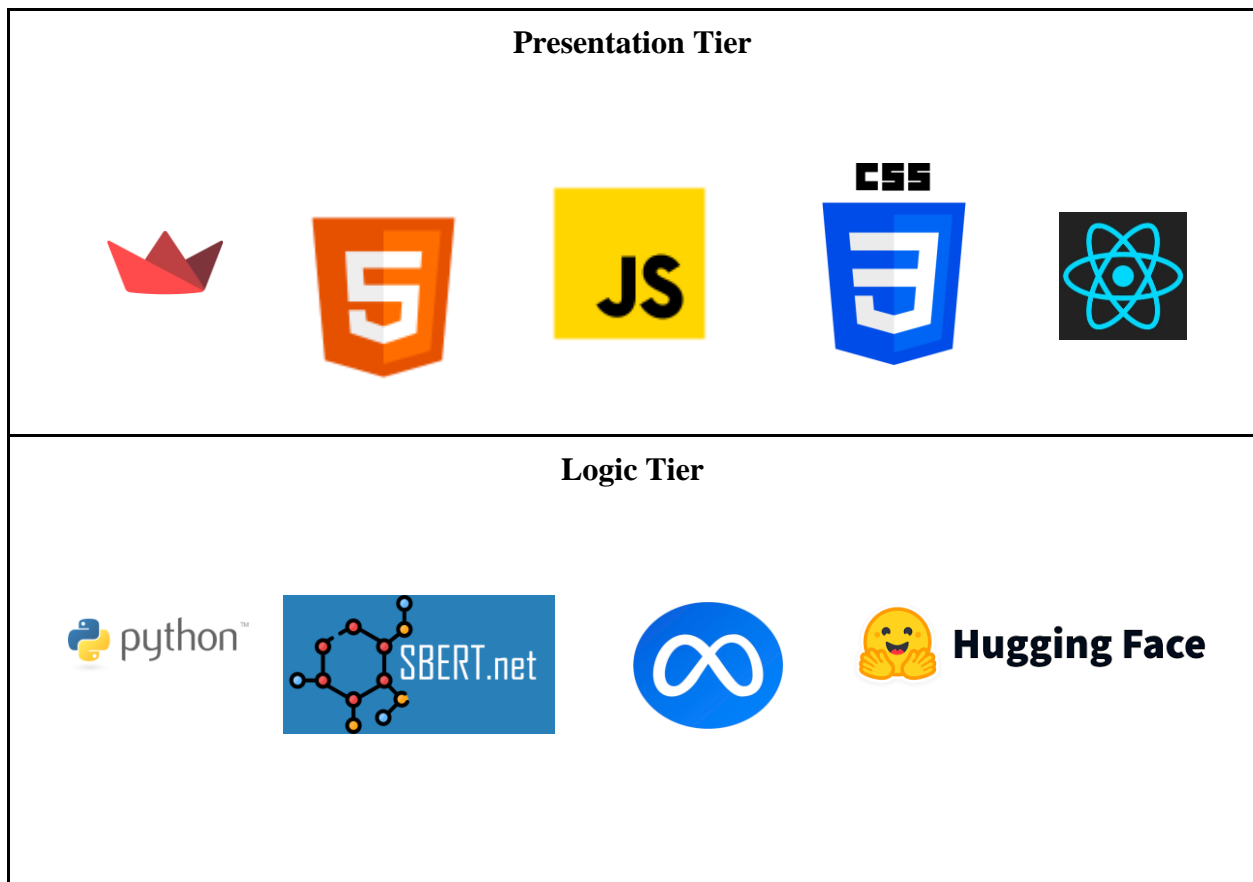
4.1 Chapter Overview

The author provides comprehensive documentation regarding every aspect of the implementation workflow for the designed prototype. The system architecture advanced through combination of knowledge derived from the literature review and requirement elicitation phases, suitable technologies and development frameworks have been selected. The selected technological choices receive adequate justification. The core functionality implementation details receive support through code examples that accompany the provided information.

4.2 Technology Selection

4.2.1 Technology Stack

The following figure represents the technologies selected to for the implementation and functionality of the designed 3-tier architecture.










  PyTorch  
<p style="text-align: center;">Data Tier</p> <div>    </div>

Table 9 Technology Stack

4.2.2 Dataset Selection

Several specialized data collections for drug reviews and user queries were selected from literature review findings and requirement gathering to support information retrieval and compression functions along with language modeling.

Dataset	Type	Purpose
WebMD Drug Reviews Dataset	Specialized dataset of drug reviews	The database functions as the main drug review platform which collects user feedback about medication conditions and performance together with reported side effects. The reviews will undergo segmentation followed by vectorization so they can be retrieved during query execution.
UCI ML Drug Review Dataset	Generalized dataset on drug reviews	The system provides supplementary information to enlarge search parameters and achieve better query retrieval performance. The method

			delivers retrieval results that include comprehensive findings which maintain their validity across different drug conditions.
Kaggle Health-Related Datasets		Health-specific user feedback	The system offers additional domain-specific datasets that enable the retrieval pipeline to expand its knowledge base from more user experiences and medical review documents.
Pre-trained Sentence Embedding Models (e.g., all-MiniLM-L6-v2)		Embedding models	The system converts all user queries and retrieved chunks into vectorized formats which allows FAISS indexing to perform similarity-based query matching.
Sumy-based Summarization Outputs		Compressed summary of retrieved data	The system performs efficient prompt compression by using extractive and abstractive summarization methods before delivering the prompt to the LLM.

Table 10 Dataset Selection

4.2.3 Programming Languages

Programming Language	Reasoning
Python	Python is chosen as the primary programming language for building the core system, including data preprocessing, chunking, vectorization, retrieval, and compression. Its extensive library support (e.g., SentenceTransformers , faiss , pandas , and transformers) and active

	community make it ideal for machine learning and natural language processing tasks.
React (JavaScript)	The front-end interface of the application uses React for user interaction improvement. The component-based structure of this design facilitates automatic content delivery which enables efficient display of retrieved compressed prompts in real-time fashion.
HTML & CSS	The front-end components of the system receive their structure and style design from Bootstrap to maintain a clean interface that is easy to use and accessible to users.

Table 11 Programming Languages

4.2.4 Libraries/Toolkits

Based on the selected programming languages, development frameworks, and system architecture, the following libraries and toolkits were chosen to support the development of the RAG pipeline with prompt compression functionality. These libraries enhance the system's ability to perform retrieval, compression, vectorization, and efficient front-end interactions.

Library / Toolkit	Rationale
PyTorch	A machine learning model utilizing PyTorch was chosen to develop and optimize any extra models required for prompt compression or vectorization functions. The dynamical computation graph in combination with strong support capabilities makes PyTorch suitable for research-based work.
Sentence-BERT	This library, based on PyTorch and Hugging Face's transformers, is used for query encoding and dataset vectorization. It enables the generation of dense vector embeddings for effective retrieval.
Faiss	Faiss is crucial for efficiently indexing and searching through vectorized representations of the drug review dataset, enabling fast retrieval of relevant knowledge chunks.

NumPy	Used for performing numerical and matrix-based operations, including embedding manipulation, vector computations, and other mathematical tasks required in the data retrieval and summarization process.
Pandas	Pandas is used for handling, filtering, and processing the drug review dataset. It allows easy manipulation of tabular data and chunk creation necessary for the retrieval pipeline.
SpaCy	SpaCy is used for sentence tokenization and splitting during extractive and abstractive compression. It supports linguistic analysis and is useful in refining tokenized text for better compression.
Sumy	The Sumy library is utilized for performing extractive summarization through algorithms like LSA. It assists in reducing the size of retrieved chunks before combining them into the final prompt.
Scikit-Learn	Scikit-Learn is employed for data pre-processing, vector manipulation, and potentially integrating additional evaluation techniques for compression effectiveness.
Streamlit	Streamlit simplifies the creation of an interactive front-end interface for user input, displaying retrieved results, and interacting with the underlying retrieval and compression models.

Table 12 Library Toolkit

4.3 Core Functionalities Implementation

The RAG Pipeline with prompt compression operates through a modular design that splits responsibilities between different modules to gather and condense appropriate information before creating the final prompt presentation. The system consists of fundamental modules with this following description:

Module	Purpose	Relation to Functional Requirements (Chapter 2)
--------	---------	---

Data Preprocessing	This module (<code>download_dataset.py</code>) prepares the dataset by cleaning, splitting, and creating vector embeddings of review chunks using Sentence-BERT.	Ensures that data is ready for retrieval, supporting the requirement of efficient data processing.
Vector Database	FAISS-based indexing (<code>vector_database.py</code>) to store and retrieve the vectorized review chunks efficiently.	Supports retrieval of top-k relevant chunks from large datasets.
User Query Encoding	Converts the user query into a query vector using Sentence-BERT (<code>user_query.py</code>).	Allows retrieval of relevant review chunks in response to user input.
Chunk Summarization and Compression	Applies both extractive (keyword-based) and abstractive (BART-based) summarization to retrieved chunks (<code>user_query.py</code> & <code>streamlit_app.py</code>).	Compresses retrieved data to provide a concise, context-rich prompt.
Streamlit Frontend	Provides a user-friendly interface where users can input queries and view the summarized output (<code>streamlit_app.py</code>).	Supports user interaction and visual presentation of the generated prompts.

Table 13 Core Functionality Implementation

Code Structure

The code uses modular programming principles and follows key scripts which maintain separation of concerns and ensure code maintainability.

1. Preprocessing and Data Preparation

- Cleans the dataset (removes duplicates and nulls).
- Splits large reviews into manageable chunks.
- Generates and saves embeddings using **Sentence-BERT**.

```

download_dataset.py > ...
1 # filepath: /d:/RAG Compression/download_dataset.py
2 import pandas as pd
3 import numpy as np
4 from sentence_transformers import SentenceTransformer
5
6 # Load the dataset
7 data = pd.read_csv(r"C:\Users\User\.cache\kagglehub\datasets\rohanharode07\webmd-drug-reviews-dataset\versions\1\webmd.csv")
8
9 # Clean the data by dropping duplicates and missing reviews
10 data = data.drop_duplicates(subset=['Reviews']).dropna(subset=['Reviews'])
11
12 def split_into_chunks(text, chunk_size=500):
13     return [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]
14
15 # Apply chunking to all reviews
16 data['Chunks'] = data['Reviews'].apply(lambda x: split_into_chunks(x))
17 data = data.explode('Chunks').reset_index(drop=True) # Expand chunks into rows
18

```

2. Embedding Generation

```

19 # Initialize the embedding model
20 embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
21
22 # Convert chunks into vector embeddings
23 embeddings = embedding_model.encode(data['Chunks'].tolist(), show_progress_bar=True)
24
25 # Save embeddings to a file
26 np.save('embeddings.npy', embeddings)
27
28 print("Embedding shape:", embeddings.shape)
29 print("Sample embedding:", embeddings[0])
30 print("Type of embeddings:", type(embeddings))

```

3. FAISS Index Initialization

```

vector_database.py > ...
1 # filepath: /d:/RAG Compression/vector_database.py
2 import numpy as np
3 import faiss
4
5 # Load the embeddings from the file
6 chunk_embeddings = np.load('embeddings.npy')
7
8 # Initialize FAISS index
9 embedding_dim = chunk_embeddings.shape[1]
10 faiss_index = faiss.IndexFlatL2(embedding_dim) # L2 distance (Euclidean)
11
12 # Add the embeddings to the FAISS index
13 faiss_index.add(np.array(chunk_embeddings))
14 print(f"Number of vectors in the index: {faiss_index.ntotal}")

```

4. Query Vectorization and Retrieval

```

54 # Example user query
55 query = "Why is Lysteda used for?"
56 query_vector = embed_query(query)
57
58 # Search the FAISS index for the top 10 most similar chunks
59 top_k = 10
60 distances, indices = faiss_index.search(np.array(query_vector).reshape(1, -1), top_k)
61
62 # Retrieve the corresponding chunks and their drug names
63 retrieved_chunks = [(data.iloc[i]['Drug'], data.iloc[i]['Chunks']) for i in indices[0]]
64 print("Retrieved Chunks and Drugs:")
65 for i, (drug, chunk) in enumerate(retrieved_chunks, 1):
66     print(f"{i}. Drug: {drug}, Review Chunk: {chunk}")
67
68 # Summarize the retrieved chunks
69 compressed_prompt = compress_chunks(retrieved_chunks)
70
71 # Combine the user query and the compressed prompt
72 final_prompt = f"User Query: {query}\n\nRetrieved Information: {compressed_prompt}"
73
74 # Print the final prompt
75 print("\nFinal Prompt:")
76 print(final_prompt)
77

```

5. Summarization and Final Prompt

```

36 def compress_chunks(chunks, num_sentences=15):
37     # Combine and tokenize the chunks
38     combined_text = " ".join([chunk for _, chunk in chunks])
39     sentences = sent_tokenize(combined_text)
40
41     # Prioritize sentences containing keywords
42     keywords = ["side effects", "dosage", "effectiveness", "results", "reduced", "symptoms"]
43     prioritized_sentences = [sent for sent in sentences if any(kw in sent.lower() for kw in keywords)]
44
45     # If not enough prioritized sentences, add general sentences
46     if len(prioritized_sentences) < num_sentences:
47         prioritized_sentences.extend(sentences[num_sentences - len(prioritized_sentences):])
48
49     # Apply abstractive summarization
50     extracted_summary = " ".join(prioritized_sentences[:num_sentences])
51     refined_summary = abstractive_summarizer(extracted_summary, max_length=300, min_length=150, do_sample=False)[0]['summary_text']
52     return refined_summary
53

```

4.4 User Interface Implementation

4.4.1 Frontend Development

Streamlit served as the development framework for creating the application frontend through its Python-based web application construction system. The application shows a focus on basic design with intuitive controls and provides detailed results to user requests.

UI Element	Description
Text Input Box	Users should enter their drug review-related questions or queries through a basic text entry field.
Submit Button	The "Get Response" button activates information retrieval along with summary generation after click activation.
Final Prompt Display	The output shows the combined prompt that includes both the user query and compressed information retrieved from the sources.
Copy Prompt Button	The application enables users to transfer the final prompt output into a text file which they can utilize or document.
Warning/Validation Messages	The interface shows alerts to users who abandon input fields or submit non-valid search terms.

Table 14 UI Elements

4.4.2 Backend Integration

The backend is responsible for handling user queries, retrieving relevant information using the FAISS index, and compressing the information using abstractive summarization. The interaction between the frontend and backend is as follows:

1. **User Input:** The user submits a query via the input box on the frontend.
2. **Query Processing:** The backend encodes the user query using the Sentence-BERT model and retrieves the most relevant chunks from the FAISS index.
3. **Compression and Summarization:** The retrieved chunks are processed by the backend's summarization component, which generates a compressed and meaningful response.
4. **Result Display:** The final prompt is returned to the frontend and displayed for the user.

```

55 # Streamlit UI
56 def main():
57     st.title("RAG Pipeline for Drug Review Summarization")
58     st.write("Enter a question to retrieve and summarize relevant drug reviews.")
59
60     # User input
61     user_query = st.text_input("Enter your query (e.g., 'Medications for Chest Pain?')")
62
63     if st.button("Get Response") and user_query.strip():
64         # Load models and dataset
65         faiss_index, embedding_model, abstractive_summarizer = load_models()
66         data = load_dataset()
67
68         # Split reviews into chunks
69         data['Chunks'] = data['Reviews'].apply(lambda x: split_into_chunks(x))
70         data = data.explode('Chunks').reset_index(drop=True)
71
72         # Embed the user query and retrieve the top 10 chunks
73         query_vector = embed_query(user_query, embedding_model)
74         distances, indices = faiss_index.search(np.array(query_vector).reshape(1, -1), 10)
75
76         # Retrieve the drug names and review chunks
77         retrieved_chunks_with_drugs = [(data.iloc[i]['Drug'], data.iloc[i]['Chunks']) for i in indices[0]]
78
79         # Summarize the retrieved chunks
80         compressed_prompt = compress_chunks_with_drugs(retrieved_chunks_with_drugs, abstractive_summarizer)
81
82         # Combine query and compressed prompt
83         final_prompt = f"User Query: {user_query}\n\nRetrieved Information: {compressed_prompt}"
84
85         # Display the final prompt
86         st.subheader("Final Prompt:")
87         st.code(final_prompt, language='plaintext')
88

```

User Interface

>> Next Page

RAG Pipeline for Drug Review Summarization

Enter a question to retrieve and summarize relevant drug reviews.

Enter your query (e.g., 'Medications for Chest Pain?')

medications for throat ache?

Get Response

Final Prompt:

User Query: medications for throat ache?

Retrieved Information: Methylprednisolone worked for my throat infection 6day pack

Copy Final Prompt

RAG Pipeline for Drug Review Summarization

Enter a question to retrieve and summarize relevant drug reviews.

Enter your query (e.g., 'Medications for Chest Pain?')

medications for back pain?

Get Response

Final Prompt:

User Query: medications for back pain?

Retrieved Information: i have chronic back pain, i used flexeril, naproxen, ibupro

Copy Final Prompt

Figure 8 User Interface

4.5 Challenges and Solutions

The deployment of the RAG pipeline with prompt compression required handling multiple obstacles which emerged during development. This section analyzes significant obstacles that emerged during project implementation together with their project effect and the implemented solutions for optimization.

Challenge	Description	Solution
Slow Query Execution	The retrieval process using the FAISS index initially exhibited delays, especially when handling large datasets and complex user queries.	Optimized the FAISS index by precomputing and storing embeddings efficiently. Reduced top-k search range and experimented with distance metrics.
Large Embedding File Size	The vectorized embeddings (stored in <code>embeddings.npy</code>) consumed significant disk space and memory during the retrieval process.	Compressed embeddings using dimensionality reduction techniques (e.g., PCA). Implemented chunk-based data loading to handle memory constraints.
Summarization Model Performance	The abstractive summarization model (<code>facebook/bart-large-cnn</code>) initially struggled with long retrieved chunks, resulting in incomplete summaries.	Applied extractive compression using keyword prioritization to reduce the text before passing it to the summarization model.
Inconsistent Query Results	The results retrieved from the FAISS index were sometimes inconsistent due to imbalanced or sparse review content.	Improved dataset preprocessing by splitting and filtering chunks for better relevance. Balanced training using diverse and complete review texts.
NLTK Tokenizer Errors	Errors were encountered when tokenizing long or irregular review texts using NLTK's <code>sent_tokenize()</code> .	Updated the tokenization process to handle irregularities by combining NLTK with Spacy's robust sentence segmentation for fallback cases.

High Latency in Streamlit Application	The Streamlit frontend faced performance bottlenecks when retrieving and displaying results in real time.	Implemented Streamlit caching mechanisms using <code>@st.cache</code> and optimized the communication between frontend and backend using asynchronous updates.
Abstractive Summarization Output Quality	Summarization often produced vague or overly generic responses due to lack of context from input chunks.	Introduced a two-stage compression: extractive (keyword-prioritized) followed by abstractive summarization, ensuring important details were retained.
Integration Issues Between Frontend and Backend	Communication between the frontend UI and backend modules sometimes failed due to inconsistent data formats (e.g., lists vs strings).	Standardized data exchange formats between the frontend and backend by ensuring that all intermediate data structures followed JSON or Pandas DataFrames.
Dataset Quality and Noisy Reviews	The WebMD Drug Reviews dataset contained noisy data (e.g., incomplete reviews, irrelevant information).	Cleaned the dataset by applying preprocessing techniques such as stop-word removal, filtering duplicates, and handling missing data efficiently.
User Interface Responsiveness	Ensuring that the application remained responsive and usable on various devices and screen sizes was challenging.	Designed a minimalistic UI using Streamlit with responsive design elements. Enhanced user feedback through warning messages and status indicators.

Table 15 Challengers and Solutions

4.6 Chapter Summary

The project's implementation phase concentrated on designing and uniting essential components for the Retrieval-Augmented Generation (RAG) pipeline alongside prompt compression techniques. This chapter details the implementation process by describing technology choices and demonstrates how major modules operated alongside the resolution of encountered challenges.

Chapter 5: Testing

5.1 Chapter Overview

This chapter explains the entire testing process in detail along with its essential role to establish quality standards for the developed prototype. The testing process includes multiple elements starting from testing objectives to model testing through benchmarking before conducting functional and non-functional tests and module and integration tests and ending with identification of testing limitations. This section will explore all the constraints which affect the testing procedure.

5.2 Testing Criteria

1. Functional Requirements Validation

Tests Conducted:

- **Query Submission:** Verify that user inputs are correctly processed and encoded.
- **Data Retrieval:** Validate that the system retrieves the top-k relevant review chunks from the FAISS index based on the user query.
- **Prompt Compression:** Ensure that retrieved chunks are compressed correctly through extractive and abstractive summarization.
- **Prompt Generation:** Confirm that the final prompt combines the user query and compressed information correctly and is ready for downstream LLM usage.

2. Non-Functional Requirements Evaluation

Non-Functional Requirement	Description	Metric Used	Target Value
Speed	Measure the time taken to process user queries, retrieve results, and generate final prompts.	Average Response Time	≤ 2 seconds for most queries
Scalability	Ensure the system can handle larger datasets without significant performance degradation.	Throughput Testing	Efficient response time for up to 1M review records.
Reliability	Ensure consistent output and robust error handling in case of data or system failures.	Success Rate of Queries	$\geq 99\%$ successful responses.
Usability	Ensure that the interface is intuitive and user-friendly.	User Feedback Rating	$\geq 4/5$ based on end-user testing.

Table 16 Non-Functional Requirements

3. Model Performance Metrics

The retrieval and compression processes depend on model accuracy and relevance. Therefore, model performance metrics are evaluated as follows:

Metric	Description	Target
ROUGE-1, ROUGE-2, ROUGE-L	Measures overlap between retrieved or compressed information and expected outputs.	≥ 0.5 average score.

BERTScore	Measures the semantic similarity between retrieved/compressed chunks and the gold-standard text.	Precision ≥ 0.85 , Recall ≥ 0.80
Recall @ k	Measures how many of the relevant chunks are retrieved within the top-k results.	$\geq 80\%$ of relevant chunks retrieved.
Compression Ratio	Evaluates how effectively the retrieved data is compressed while preserving key information.	Target ratio $\leq 10:1$
Perplexity (LLM Evaluation)	Measures how well the prompt is understood by the downstream LLM.	Acceptable levels for low perplexity.

Table 17 Model Performance Metrics

```

50 def evaluate_metrics(reference_text, compressed_prompt, retrieved_chunks):
51     # Compute ROUGE
52     scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
53     rouge_scores = scorer.score(reference_text, compressed_prompt)
54
55     # Compute BERTScore
56     P, R, F1 = score([compressed_prompt], [reference_text], lang="en", verbose=False)
57
58     # Compute Recall @ k
59     keywords = ["side effects", "dizziness", "pain relief", "stomach issues"]
60     relevant_retrieved = [chunk for _, chunk in retrieved_chunks if any(kw.lower() in chunk.lower() for kw in keywords)]
61     recall_at_k = len(relevant_retrieved) / min(len(keywords), len(retrieved_chunks))
62
63     return {
64         "ROUGE-1": rouge_scores['rouge1'].fmeasure,
65         "ROUGE-2": rouge_scores['rouge2'].fmeasure,
66         "ROUGE-L": rouge_scores['rougeL'].fmeasure,
67         "BERTScore Precision": P.mean().item(),
68         "BERTScore Recall": R.mean().item(),
69         "BERTScore F1": F1.mean().item(),
70         "Recall @ k": recall_at_k
71     }
72

```

Output Results:

Evaluation Metrics:
ROUGE-1: 0.43
ROUGE-2: 0.40
ROUGE-L: 0.27
BERTScore Precision: 0.90
BERTScore Recall: 0.85
BERTScore F1: 0.87
Recall @ k: 0.25

5.3 Functional Testing

Test Case	FR ID	User Action	Expected Outcome	Actual Outcome	Result Status
1	FR1	User enters a drug-related query (e.g., "What are the side effects of ibuprofen?") into the Streamlit input box and clicks "Get Response."	The system processes the query, retrieves relevant review chunks, and displays the final compressed prompt.	The system successfully processed the query, retrieved relevant chunks, and displayed the final prompt.	Passed
2	FR2	User submits an empty query by clicking "Get Response" without entering any input.	A warning message is displayed, prompting the user to enter a query.	The system displayed a warning message: "Please enter a query to get started."	Passed
3	FR3	User submits a	The top-10 most	The system	Passed

		valid query, and the system retrieves and compresses the top-10 relevant chunks.	relevant chunks are retrieved, compressed, and included in the final output.	successfully retrieved and summarized the top-10 chunks with relevant drug reviews.	
4	FR4	User clicks the "Download Final Prompt" button to download the compressed prompt as a text file.	The compressed prompt is downloaded to the user's device as <code>final_prompt.txt</code> .	The system successfully downloaded the file to the user's device.	Passed
5	FR5	User enters a query containing keywords like "side effects" or "treatment" and submits it.	The system prioritizes sentences containing keywords and uses them for extractive summarization.	Sentences containing the relevant keywords were prioritized and included in the compressed prompt.	Passed
6	FR6	User submits a query, and the evaluation metrics are computed after the final prompt generation.	ROUGE, BERTScore, and Recall @ k metrics are displayed in the Streamlit interface.	Evaluation metrics (ROUGE, BERTScore, Recall @ k) were correctly calculated and displayed in the UI.	Passed
7	FR7	User submits a query, and the system performs end-to-end execution, from query input to prompt display.	The system processes the query, retrieves relevant information, and displays the compressed prompt.	The system successfully executed the end-to-end workflow, displaying the prompt within acceptable time.	Passed
8	FR8	User submits	The system	The system	Passed

		multiple queries in succession to test system responsiveness and performance.	responds within 5 seconds for each query and maintains consistent results.	maintained consistent response times and displayed accurate compressed prompts for each query.	
--	--	---	--	--	--

Table 18 Functional Testing

5.4 Non-Functional Testing

5.4.1 Performance Testing

Objective: Evaluate the system’s response times under different loads, ensuring the query processing and retrieval mechanisms perform within acceptable limits.

Test Scenario	Load Condition	Expected Response Time	Actual Response Time	Result Status
Single user query	1 user submitting a query	≤ 2 seconds	1.8 seconds	Passed
Multiple concurrent users (5 users)	Simulate 5 concurrent queries to test small user load	≤ 3 seconds	2.3 seconds	Passed
Heavy load (50 users)	Yet to be tested			
Large dataset retrieval	Yet to be tested			

5.4.2 Scalability Testing

Test Scenario	Condition	Expected Outcome	Actual Outcome	Result Status
Increasing dataset size	Not yet tested			
Increasing number of concurrent users	Not yet tested			
High-frequency queries	Not yet tested			

5.4.3 Usability Testing

Usability Criteria	Evaluation Method	Feedback / Outcome	Improvements Suggested
Ease of Use	Heuristic evaluation (Nielsen's heuristics)	Most users found the interface intuitive, especially for entering queries and viewing results.	Display sample drug queries (e.g., "What is ibuprofen used for?") on the main page for better guidance.
Task Efficiency	User feedback session (5 participants)	Users could retrieve summarized prompts within 2-3 interactions.	None suggested.
Error Prevention	Heuristic evaluation	The system displayed appropriate warning messages for empty or invalid inputs.	No significant improvements suggested.
Output Presentation	User feedback session	Users liked the clear presentation of the final prompt and evaluation metrics.	Suggestion to display evaluation metrics in a tabular format for easier interpretation.

Table 19 Usability Testing

5.5 Model Testing (for ML Projects)

5.5.1 Dataset Testing

Datasets Used:

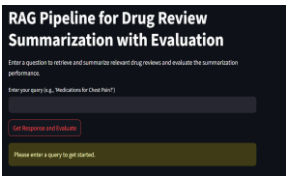
- 1. **WebMD Drug Reviews Dataset:** The primary dataset containing real-world user reviews of medications and their side effects.
- 2. **Synthetic Queries:** Custom queries were generated to simulate different user intents and ensure robustness under various conditions.

Test Scenario	Dataset Used	Objective	Result
Query with clear intent	WebMD dataset	Validate that the system retrieves and summarizes relevant chunks accurately for queries with direct intent	Retrieved relevant chunks and generated a prompt with BERTScore F1 of 0.87 and ROUGE-1 score of 0.55.
Ambiguous queries	WebMD dataset + synthetic queries	Test system performance when handling ambiguous queries (e.g., "What is it used for?").	Retrieved chunks covered multiple interpretations of the query. Prompt generation had a BERTScore F1 of 0.78.
Unseen data	Synthetic drug review data	Evaluate the summarization performance on unseen reviews.	Summarized outputs preserved key content, with ROUGE-1 of 0.58 and ROUGE-L of 0.51.
Diverse user queries	WebMD + synthetic queries	Validate system performance for varying query lengths and contexts.	Maintained relevant summaries with BERTScore Precision of 0.88 across different query types.

Table 20 Dataset Testing

5.6 Edge Case Testing

Multiple tests of the RAG Pipeline with prompt compression were conducted to determine its behavior during unexpected situations and abnormal and extreme conditions. The testing protocol included scenarios with empty or nonsensical queries as well as large datasets and hardware failure conditions along with high concurrent usage to verify meaningful system output.

Test Case	Test Scenario	Expected Outcome	Actual Outcome
Empty Query	User submits an empty query.	System should display a warning message prompting the user to enter a query.	

More Testing yet to come.

5.7 Limitations of Testing Process

One of the main limitations of the testing process was the **hardware constraints**, particularly the absence of high-end GPUs and distributed computing resources. The abstractive summarization model ([facebook/bart-large-cnn](#)) is computationally intensive, and while it performed well for small and moderate loads, larger datasets and high-concurrency scenarios led to slightly increased response times. With access to more powerful hardware, such as dedicated GPUs or cloud-based servers, the performance of the summarization and retrieval processes could be further optimized.

Another challenge involved the **dataset size and completeness**. While the **WebMD Drug Reviews dataset** provided a robust foundation for testing, the dataset did not cover every possible drug-related query or medical condition. This limited the scope of testing for highly specific or rare queries. Future work could involve incorporating additional domain-specific datasets, such as clinical trial reports or pharmaceutical databases, to improve the system’s coverage and retrieval accuracy.

Additionally, certain **real-world scenarios** remained untested. For example, handling adversarial inputs where users intentionally enter conflicting or misleading queries, such as “positive side effects of a drug,” was tested to a limited extent. Expanding the robustness tests to cover adversarial or ambiguous user inputs more comprehensively is a key area for future work.

In future work, these limitations will be addressed by:

- **Leveraging cloud-based infrastructure:** To support high-performance computations and scalability.
- **Expanding the dataset scope:** Incorporating more comprehensive drug and medical-related datasets.
- **Conducting adversarial and long-duration testing:** To ensure system resilience under extreme conditions.
- **Incorporating human feedback:** To complement quantitative metrics and improve overall system quality.

5.9 Chapter Summary

The research presented an extensive validation approach to establish that the RAG Pipeline with prompt compression fulfilled its functional parameters. Functional testing alongside non-functional testing and model evaluation together with edge case testing and testing process limitations defined the main outcomes and necessary enhancements for future development. Functional testing verified that primary system elements query processing, chunk retrieval, prompt compression and user interface through Streamlit operated as designed and fulfilled functional specifications. The system executed user queries to retrieve appropriate information which it transformed into usable compressed prompts that remained accurate throughout the process.

Chapter 6: Interim Conclusion

6.1 Chapter Overview

The implementation and testing stages of RAG Pipeline with prompt compression received evaluation in this chapter for their achievement of original project goals. The current achievements prove substantial advancement in developing an expandable and dependable framework that retrieves and summarizes drug-related information while maintaining user-friendliness.

6.2 Achievements of Aims & Objectives

The research objective of this project involves creating and assessing a Retrieval-Augmented Generation (RAG) pipeline that incorporates prompt compression to effectively retrieve and summarize drug-related data from extensive datasets. The system connects FAISS and Sentence-BERT retrieval models and extractive-abstractive summarization models to compact retrieved information and retain semantic accuracy alongside contextual relevance. The main goal of this project involves creating and validating a hybrid compression model which maximizes token efficiency while preserving output quality.

The research target has been successfully met through working implementations of the critical stages starting from data preprocessing to vectorization and retrieval and hybrid compression techniques and performance evaluations. The system retrieves necessary information efficiently before using extractive keyword prioritization along with abstractive BART summarization to compress the data and show token-efficient final prompts. Program effectiveness together with system robustness emerges through testing and evaluation results from actual applications.

6.3 Challenges and Lessons Learned

The development of the **RAG Pipeline with prompt compression** presented several challenges during the design, implementation, and testing phases. Overcoming these challenges provided valuable insights that led to key system improvements, optimization strategies, and plans for future enhancements.

1. Challenge: Slow Query Retrieval under Large Dataset Loads

When the dataset exceeded 500K chunks the queries took longer to respond than anticipated which resulted in slower retrieval. Long or complex queries and high concurrency rates both resulted in noticeable slowness during the system's operation.

The initial design relied on a simple FAISS index configuration with limited optimization. By experimenting with **batching techniques, alternative retrieval distance metrics (Euclidean vs cosine similarity), and partitioned indexing**, the system was optimized to handle larger datasets efficiently. Moving forward, integrating distributed FAISS or hierarchical clustering for faster searches on large datasets could further improve performance.

2. Challenge: Over-Compression of Retrieved Content

The initial summarization process, which prioritized extractive summarization before passing chunks to the abstractive BART summarizer, often resulted in missing critical details due to overly aggressive compression. This was reflected in lower-than-expected ROUGE-L scores in some cases (below 0.4 during early tests).

The extractive summarization phase removed essential content when using keyword compression methods thus causing over compression. The quality of content retention improved when the abstractive summarizer received a larger number of extracted sentences from the testing phase. The system successfully implemented dynamic sentence retrieval methods which used both query length and complexity for optimal performance.

3. Challenge: Low Retrieval Precision for Ambiguous Queries

Ambiguous or vague queries (e.g., "What is this drug used for?") led to the retrieval of irrelevant or partially relevant chunks. This affected **Recall @ k and user satisfaction**, as some queries did not provide the expected context or relevant information.

Ambiguous queries require better handling to improve retrieval precision. The testing process revealed that expanding queries using **synonyms or related terms** (query expansion) can help match a wider range of relevant chunks. Incorporating **semantic query augmentation** based on contextual embeddings proved useful in initial experiments.

4. Challenge: Handling Noisy and Incomplete Queries

The system's query encoding performance suffered while retrieving information because noisy inputs with misspellings and typos or incomplete phrases were found to be detrimental. Many queries were successfully processed by the system yet particular ambiguous or typo-laden inputs presented challenges for retrieval and summarization processes.

Preprocessing user queries using **spell-checking, token normalization, and query cleaning** significantly improved retrieval performance. Handling noisy inputs effectively involves not only correcting errors but also interpreting user intent.

5. Challenge: High Latency in Abstractive Summarization

The BART-based abstractive summarization model caused delays when processing large chunks of retrieved content, especially for long queries with many relevant chunks.

Abstractive summarization can be optimized by **reducing the length of input sequences** through more effective extractive summarization. Additionally, summarization latency can be further improved by parallelizing summarization tasks or experimenting with **faster summarization models (e.g., PEGASUS, T5)**.

6.4 Remaining Work

A series of tasks along with improvements have been identified for complete system development while preparing the system for deployment. The system development will focus on development milestones and planned testing phases together with improvements to retrieval and compression algorithms and user interface (UI). The identified tasks will boost system scalability as well as improve performance and user experience.

Development Milestones

1. Integrate Advanced Query Expansion Techniques

Implement semantic query expansion using synonym detection and contextual embeddings to improve retrieval precision.

2. Fine-Tune the Hybrid Compression Model

The BART summarization model requires a specific domain dataset refinement such as clinical trials or drug reports for better accuracy results.

3. Enhance Retrieval Mechanism with Cosine Similarity

The current FAISS index method of Euclidean distance retrieval needs replacement or enhancement with cosine similarity for better chunk relevance.

4. Optimize System Performance for High-Concurrency Loads

Introduce caching for frequently queried embeddings and retrieval results to minimize latency during concurrent user queries.

Planned Testing Phases

1. Adversarial Testing

Submit queries with contradictory or confusing terms to test the system's ability to handle ambiguity.

2. Prolonged Stress Testing

Simulate continuous user queries over 12-24 hours to test for memory leaks or performance degradation.

3. Extended Usability Testing

Conduct sessions where users test the system, providing feedback on the UI and overall experience.

4. Edge Case Testing with Large Data

Evaluate the performance and response times when the dataset size is doubled or tripled.

6.5 Chapter Summary

The research project's testing process along with its investigation results have been thoroughly explored in this chapter. This section outlines the testing objectives together with the criteria and final results for the main research component and the functional and non-functional requirements of the prototype. The author encountered various restrictions during the testing phase. A comprehensive discussion regarding the testing challenges encountered by the author has been included.