DocuCore: Text & Code Summariser In Cloud

Software Requirements Specification

Version <1.1>

Submitted in Partial Fulfillment for the Award of Degree of Bachelor of Technology in Information Technology from Rajasthan Technical University, Kota

MENTOR:

Mr. Praveen Kumar Yadav

(Dept. of Information Technology)

COORDINATOR:

Dr. Priyanka Yadav

(Dept. of Information Technology)

SUBMITTED BY:

Garvita Sakhrani (21ESKIT049)

Hiteshi Agrawal (21ESKIT056)

Abhishek Gupta (21ESKIT003)

DEPARTMENT OF INFORMATION TECHNOLOGY

SWAMI KESHWANAND INSTITUTE OF TECHNOLOGY,MANAGEMENT & GRAMOTHAN
Ramnagaria (Jagatpura), Jaipur – 302017

SESSION 2024-25

Revision History

Date	Version	Description	Author
17/10//24	1.0	Initial draft created with basic structure and chapter layout.	Garvita Sakhrani, Hiteshi Agrawal, Abhishek Gupta
27/11/24	1.1	Edited chapter formatting to include centered titles and updated margins and style.	Garvita Sakhrani, Hiteshi Agrawal, Abhishek Gupta

Table of Contents

1. Introduction

In today's fast-paced world, managing large documents and analyzing complex code can be challenging and time-consuming. Whether it's summarizing lengthy research papers or identifying issues in software code, traditional methods often fall short in terms of speed and efficiency. To tackle these problems, this project introduces a cloud-based system that uses advanced technologies like Large Language Models (LLMs) and code analysis tools. The system is designed to make document summarization and code analysis faster and easier, helping users save time and focus on more critical tasks.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the functional and non-functional requirements for a cloud-based system designed to automate the processes of document summarization and code analysis. By using Large Language Models (LLMs) such as GPT-3.5, and code analysis tools, the system will provide an efficient and scalable solution for summarizing large volumes of text and analyzing software code. The system will be deployed on AWS (Amazon Web Services) to ensure that it can handle multiple users and high-volume requests efficiently. This document serves as a roadmap for developers, system architects, and stakeholders to understand the key requirements and design of the project.

1.2 Scope

The system is aimed at a broad range of users, including researchers, developers, and business professionals who frequently deal with large documents and codebases. It will allow users to upload text documents such as research papers, articles, or reports for automatic summarization. Additionally, software developers can upload their code files to analyze for complexity, bugs, and overall performance. The system will output concise document summaries and detailed code analysis reports that can be further utilized. The scope of the project includes:

- Automating text summarization using LLMs.
- Providing static and dynamic code analysis functionalities.
- Hosting the system on AWS for scalability, flexibility, and real-time processing.

- Offering a user-friendly web interface for interaction.

1.3 Definitions, Acronyms, and Abbreviations

- **LLM**: Large Language Model, a type of machine learning model designed to understand and generate human language.
- **AWS**: Amazon Web Services, a comprehensive cloud platform offering compute, storage, and database services.
- **API**: Application Programming Interface, a set of functions and procedures allowing the creation of applications that access the features or data of an operating system or service.
- **NLP**: Natural Language Processing, the field of AI that enables computers to understand and process human languages.
- **SRS**: Software Requirement Specification.
- **UI**: User Interface.

1.4 References

- OpenAI GPT-3.5 API Documentation: `https://platform.openai.com/docs`
- AWS Lambda and API Gateway Documentation: `https://docs.aws.amazon.com`

1.5 Technologies to be Used

- **Large Language Models (LLMs):** GPT-3.5 for document summarization.
- **Code Analysis Tools:** Tools for analyzing code quality, bugs, and security vulnerabilities.
- **AWS Cloud Infrastructure:** Using services such as AWS Lambda, AWS API Gateway, and Elastic Beanstalk to host and scale the application.
- **Langchain:** Used to orchestrate workflows for LLMs, enabling more advanced functionalities such as retrieval-augmented generation.
- **ChromaDB:** For managing vector embeddings and fast retrieval of documents.
- **Web Framework:** Front End would be designed to make it easier for the user to interact with the environment.

1.6 Overview

The project involves building a client-server system hosted on AWS that processes document summarization requests and code analysis submissions. The system uses LLMs to generate concise summaries and code analysis tools to inspect code for performance and quality issues. Users will interact with the system via a web interface to upload their files, and they will be able to view the results. The system leverages cloud services for scalability and high availability, ensuring a seamless user experience even with large-scale operations.

2. Literature Survey

To build a robust system for document summarization and code analysis, it is essential to review existing advancements in the fields of Natural Language Processing (NLP), static code analysis, and cloud computing. This section highlights the progress made in these areas, identifies gaps, and explains how this project addresses those gaps.

2.1 Review Of Related Work

In the fields of Natural Language Processing (NLP) and Static Code Analysis, significant progress has been made. The following works are relevant to the development of this project:

- **Document Summarization:** The rise of Large Language Models (LLMs) like BERT and GPT has revolutionized text summarization tasks. Traditional methods relied on extractive summarization techniques, which select sentences directly from the source text. However, with models like OpenAI GPT-3.5, abstractive summarization is now possible, where the model generates summaries by understanding the context and key points of the document. Research by Radford et al. (2019) demonstrated how GPT models can generate highly coherent summaries across various domains.
- **Code Analysis:** SonarQube and similar tools like ESLint and PyLint are widely used for static code analysis. SonarQube's capability to identify code smells, bugs, vulnerabilities, and complexity metrics has been applied in many projects. Research by Bacchelli et al. (2012) highlighted the importance of static analysis tools in improving code quality and reducing technical debt. These tools assist developers in maintaining clean code by detecting issues early in the development process.
- **Cloud Integration:** Amazon Web Services (AWS) has become a standard for cloud deployment due to its scalability and reliability. Research conducted on serverless architectures (e.g., using AWS Lambda) shows that they enable efficient auto-scaling, reducing infrastructure costs while maintaining system performance. Studies like Baldini(2017) demonstrate how serverless computing is effective for systems requiring scalability.

2.2 Knowledge Gaps

Despite the progress in these fields, certain knowledge gaps exist that this project aims to address:

- **Lack of Integrated Platforms:** Many existing platforms focus on either document summarization or code analysis, but few offer both in a unified solution. The integration of LLM-based summarization with automated code analysis remains underexplored, especially within a cloud-based, real-time environment.
- **Summarization Accuracy for Complex Documents:** While GPT-3.5 is proficient in summarizing general texts, its performance on highly technical documents (e.g., research papers, software documentation) needs more evaluation. The lack of domain-specific fine-tuning for such models may result in summarization inaccuracies or oversimplified content.
- **Real-Time Code Analysis Feedback:** Static analysis tools like SonarQube are powerful, but they typically work offline and asynchronously. Integrating real-time code analysis into a live development environment remains a challenge, especially in a cloud infrastructure where delays can hinder developer productivity.

2.3 Comparative Analysis

The current solutions—GPT-3.5, SonarQube, and AWS cloud services—each offer powerful capabilities in their respective areas but also present challenges:

- GPT-3.5 shines in general document summarization but may require further optimization for domain-specific texts.
- SonarQube is a strong player in static code analysis but lacks real-time processing, which can hinder developers' productivity when integrated into a live environment.
- AWS infrastructure ensures scalability but requires careful management to balance cost and performance, especially when dealing with large data volumes and complex processing.

2.4 Summary

The Cloud-Based Document Summarization and Code Analysis System aims to integrate these technologies to provide a unified, real-time platform, addressing the limitations and leveraging the strengths of these existing solutions.By leveraging AWS Cloud Services and serverless architecture, the system can dynamically scale based on user demands, ensuring that even large datasets and complex codebases are processed efficiently. This approach improves upon existing systems by making summarization and code analysis faster, more accessible, and available as a unified service, making it ideal for researchers, developers, and businesses that work with vast amounts of text and code.

3. Specific Requirements

To deliver a seamless and efficient solution, the system's requirements are categorized into functional and non-functional aspects. The functional requirements define the core features and modules necessary for the system's operation, such as document summarization, code analysis, and cloud-based deployment. Meanwhile, non-functional requirements ensure the system meets performance, scalability, security, and usability standards. This section outlines the technical and operational needs to guide the development and deployment of the cloud-based document summarization and code analysis system.

3.1 Functional Requirements

- **Text Summarization Module**: Utilizes LLMs for generating summaries from large text datasets. Users can input long documents and receive concise, contextually accurate summaries.
- **Cloud Infrastructure Module**: Deploys the summarization and code analysis system on the cloud using AWS for scalability and availability. Provides secure and real-time access to the services.
- **Code-Based Analysis Module**: Uses code analysis tools to perform static and dynamic code examination. Identifies code complexity, bugs, and provides recommendations for code optimization.
- **Web-Based User Interface**: The system should provide a web-based user interface for interacting with the document summarization and code analysis tools.

3.2 Non-Functional Requirements

- **Performance**: The system should perform efficiently, handling multiple document and code submissions concurrently.
- **Scalability**: The system should scale to accommodate an increasing number of users and data.
- **Reliability**: The system should be highly available and reliable, with minimal downtime.
- **Security**: The system must ensure secure access to data and prevent unauthorized access.

- **User Experience**: The system should provide an intuitive, user-friendly interface with an emphasis on ease of use.
- **Maintainability**: The system should be easy to maintain and update.
- **Compliance**: The system should comply with relevant data privacy and protection regulations.

3.3 Hardware Requirements

- **Development Environment:** A system with a minimum of 8GB RAM and 500GB hard disk.

3.4 Software Requirements

- **Operating System:** Windows 10/11, Linux.
- **Languages:** Python 3.9 or higher.
- **Cloud Resources:** Cloud Resources: AWS resources such as EC2 instances or Lambda functions for processing, S3 for storage, and ChromaDB for data retrieval.
- **Visual Studio Code:** Text Editor.

3.5 Agile Methodology

The project will follow an Agile development methodology, allowing iterative improvements and continuous feedback. Sprint planning will focus on developing one feature at a time—starting with the document summarization module, followed by code analysis, web interface development, and finally cloud deployment. Regular standups, reviews, and retrospectives will ensure that the project meets its goals within each sprint.The Agile methodology is implemented in the following steps:

- **Project Initiation**
  This is the foundation phase where the project goals, objectives, and scope are defined. The main focus is on building a cloud-based platform to automate document summarization and code analysis. During this phase, the project team is assembled, consisting of developers, UI/UX designers, and domain experts in NLP and code analysis.
- **Product Backlog Creation**
  The product backlog is a list of all desired features and functionalities for the system. These include document upload, summarization, code analysis, and cloud integration. The features are prioritized based on their impact on the project and the users' needs.

- **Sprint Planning**

  In this phase, the prioritized features from the backlog are broken down into smaller, manageable tasks. Each task is estimated in terms of effort and assigned to the upcoming sprint. For instance, document upload functionality might be scheduled for the first sprint, while code analysis could be tackled in the next.

- **Sprint Execution**

  During sprint execution, the development team works on the assigned tasks, such as integrating the GPT-3.5 model for document summarization or setting up AWS for cloud deployment. Daily stand-up meetings are held to track progress and address any challenges.

- **Continuous Integration and Testing**

  Throughout the sprint, developers continuously integrate their code into a shared repository. Automated and manual tests are run to ensure new code is free from errors and meets quality standards. This ensures that both the document summarization and code analysis features are functioning correctly.

- **Sprint Review**

  At the end of each sprint, the team showcases the completed features. For example, they might demonstrate the document summarization feature, where users upload a document, and a summary is generated. Stakeholders provide feedback on the output, which will be used in the next sprint.

- **Sprint Retrospective**

  The team reflects on the sprint's successes and challenges, discussing what worked well and what could be improved. This helps improve the process in future sprints, such as addressing any delays in integrating external APIs or optimizing team communication.

- **Incremental Deployment**

  Once features are fully developed and tested, they are deployed incrementally in the live environment. Users can start uploading documents or code which helps identify further improvements.

- **Continuous Feedback and Adaptation**

  The system collects feedback from developers about the accuracy of the summaries or the usefulness of the code analysis reports. Based on this feedback, the team adapts the system by refining existing features or reprioritizing new functionalities.

- **Iterative Development**

  The cycle of development, review, and feedback repeats with each sprint, gradually adding new features or improving the existing ones. This ensures the system evolves continuously, incorporating new functionalities like domain-specific summarization or security-focused code analysis as needed.

3.6 Business Process Model

- **Document Summarization Module**
  - Objective: Provide automatic summarization of uploaded documents.
  - Process: User uploads a document via the web interface.Document is sent to the GPT-3.5 LLM model for summarization.The LLM processes the document and generates a concise summary.
- **Code Analysis Module**
  - Objective: Analyze uploaded code for quality, performance, and bugs.
  - Process: User uploads code for analysis.The code file is processed by integrated code analysis tools.The system checks for bugs, complexity, and security vulnerabilities.A detailed analysis report is generated.
- **Web-Based User Interface Module**
  - Objective: Enable interaction between the user and the system through a user-friendly interface.
  - Process: Users access the platform via a web browser.The interface allows users to upload documents or code files, view summaries, and analysis reports.Provides access to document management, and report viewing. Notifications or error messages are shown in case of failed uploads or processing.
- **Cloud Integration Module**
  - Objective: Ensure scalability and availability using cloud services.
  - Process: System components are deployed on AWS services like Lambda and Elastic Beanstalk.User requests are processed in real time by the backend on AWS.The system dynamically scales based on the number of user requests and usage load.

The business process model shows the end-to-end process flow of the system:

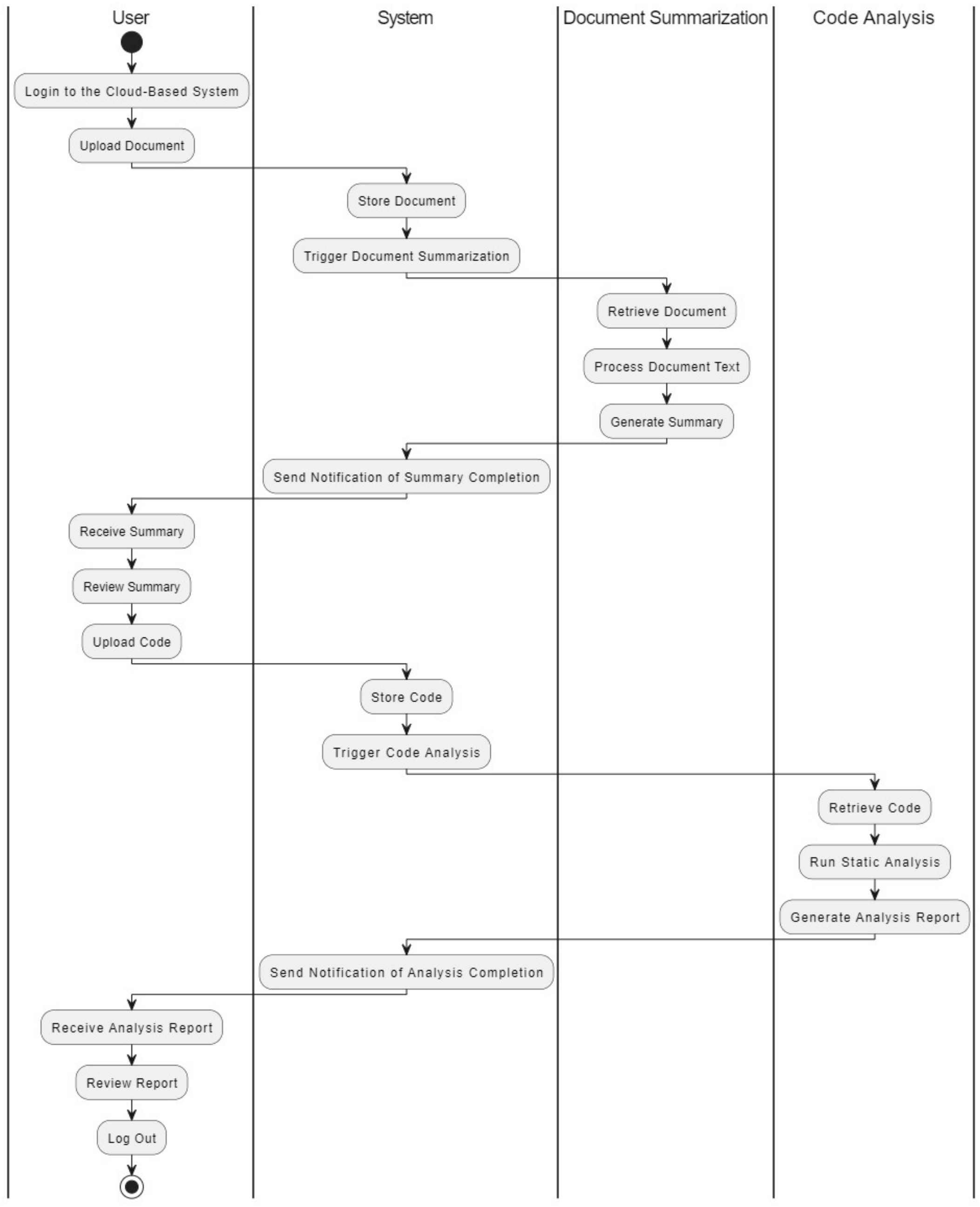


Figure 3.1: Business Process Model

3.7 Supplementary Requirements

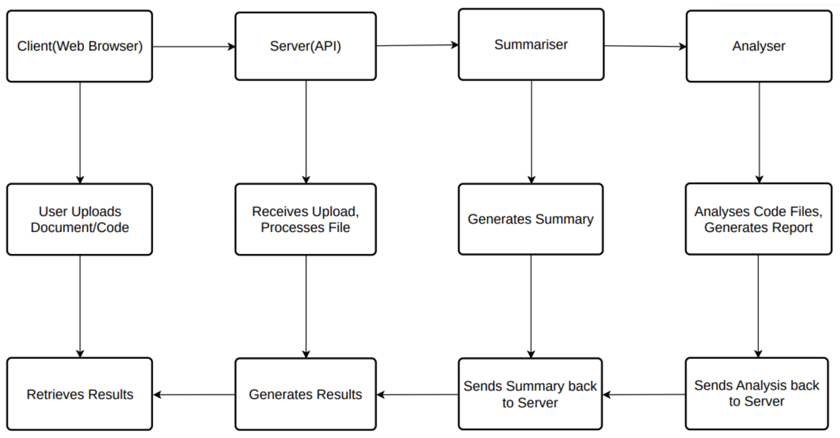
- **System Availability:** The system should ensure 99.9% uptime by utilizing cloud services for high availability.
- **Monitoring Mechanism:** Monitoring tools like AWS CloudWatch should be used to track system performance, ensuring early detection of issues like high server load.

4. System Architecture

The system is designed to handle user requests efficiently and securely. It uses a client-server setup where users interact with the system through a web browser to upload documents or code. The server processes these inputs and sends back the results. This structure ensures smooth communication and reliable performance.

4.1 Client-Server Architecture

The system follows a client-server architecture where the client (browser) interacts with the server to upload documents and code. The backend processes the inputs using LLMs and code analysis tools, returning results to the client



Client(Web Browser)
Server(API)
Summariser
Analyser
User Uploads Document/Code
Receives Upload, Processes File
Generates Summary
Analyses Code Files, Generates Report
Retrieves Results
Generates Results
Sends Summary back to Server
Sends Analysis back to Server

Figure 4.1: Client Server Architecture

4.2 Communications Interfaces

- **RESTful APIs:** Facilitates communication between the client (web interface) and the server (AWS-based backend).
- **HTTPS:** Ensures secure transmission of data between clients and the server.

5. Design and Implementation

This section outlines the detailed design of the system, showcasing how various components work together to deliver the functionality described in earlier sections. The following diagrams provide a visual representation of the system's structure, interactions, and flow of data, which are essential for understanding the technical implementation of the project.

5.1 Product Feature

- **Document Summarization:** The system allows users to upload documents in formats like PDF, DOCX, and TXT. Using OpenAI GPT-3.5, the platform generates concise summaries of lengthy documents, making it easier for users to grasp the main points quickly.
- **Code Analysis:** Users can upload code files for analysis. The system uses SonarQube to provide a detailed static code analysis report, identifying bugs, code smells, and vulnerabilities across multiple programming languages such as Python, Java, and JavaScript.
- **Cloud Integration:** Hosted on AWS, the system leverages AWS Lambda and Elastic Beanstalk for scalability and high availability. AWS S3 is used for secure storage of uploaded files, while auto-scaling ensures smooth performance during heavy traffic.
- **User-Friendly Web Interface:** The platform features a responsive web interface that allows users to upload files, view results,summaries or code analysis reports. The interface is designed for ease of use.

5.2 Data Flow Diagram (DFD)

The data flow diagram outlines how data moves between various components of the system:

- User Inputs: Users upload documents through the web interface.
- Data Processing: The system routes the inputs to LLMs for document summarization or code analysis tools for processing code.
- Storage: Processed summaries and analysis reports are stored in AWS S3.
- Output: The summarized documents and analysis reports are sent back to the user.

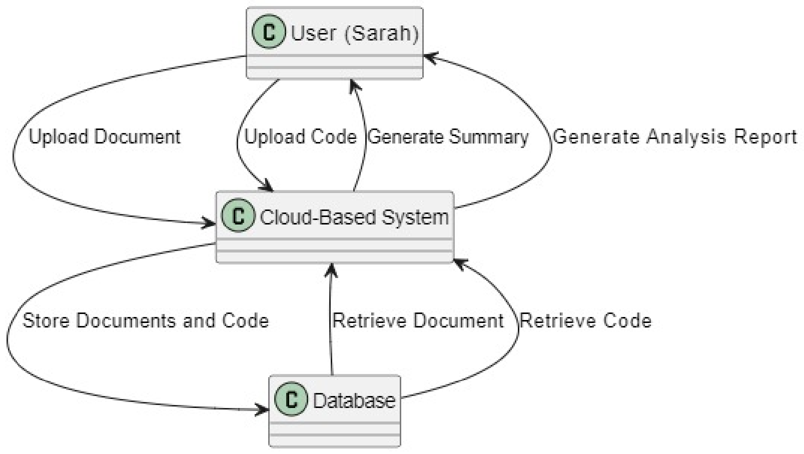


Figure 5.1: Data Flow Diagram (DFD) Level 0

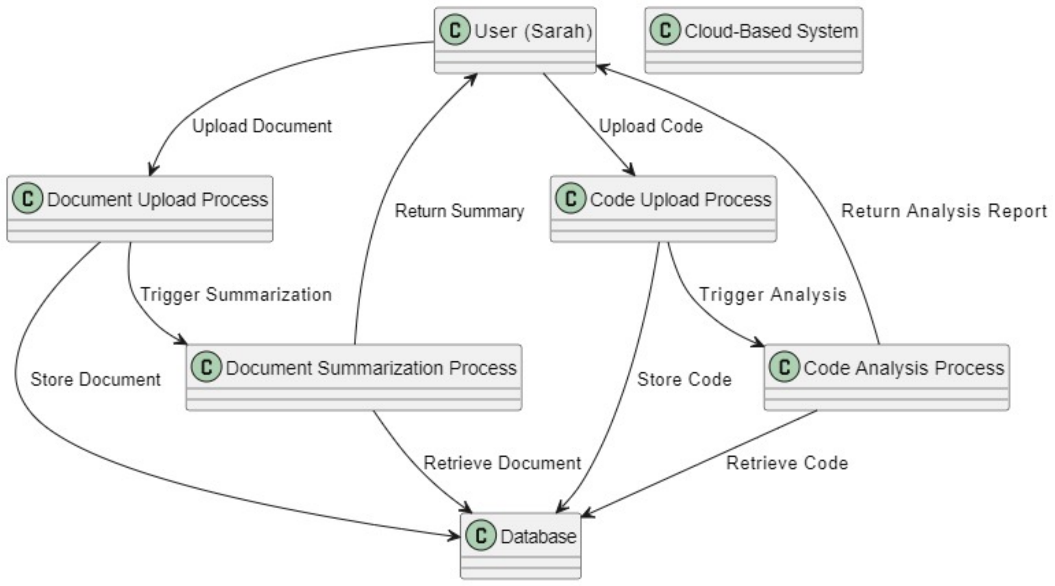


Figure 5.2: Data Flow Diagram (DFD) Level 1

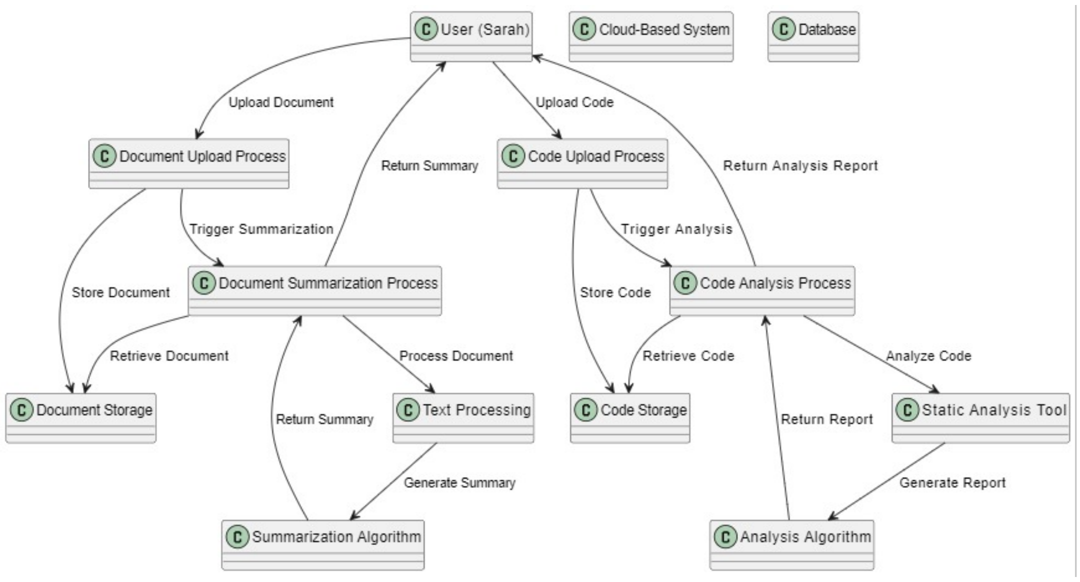


Figure 5.3: Data Flow Diagram (DFD) Level 2

5.3 ER Diagram

The ER diagram illustrates the relationships between the various entities in the system. The main entities include:

- User: Represents a user who uploads documents or code for analysis.
- Document: Contains metadata for uploaded documents (e.g., title, file type, upload date).
- Summary: Stores the generated summary of a document or code analysis result.
- Code Analysis Report: Contains details about complexity, bugs, and other code metrics.

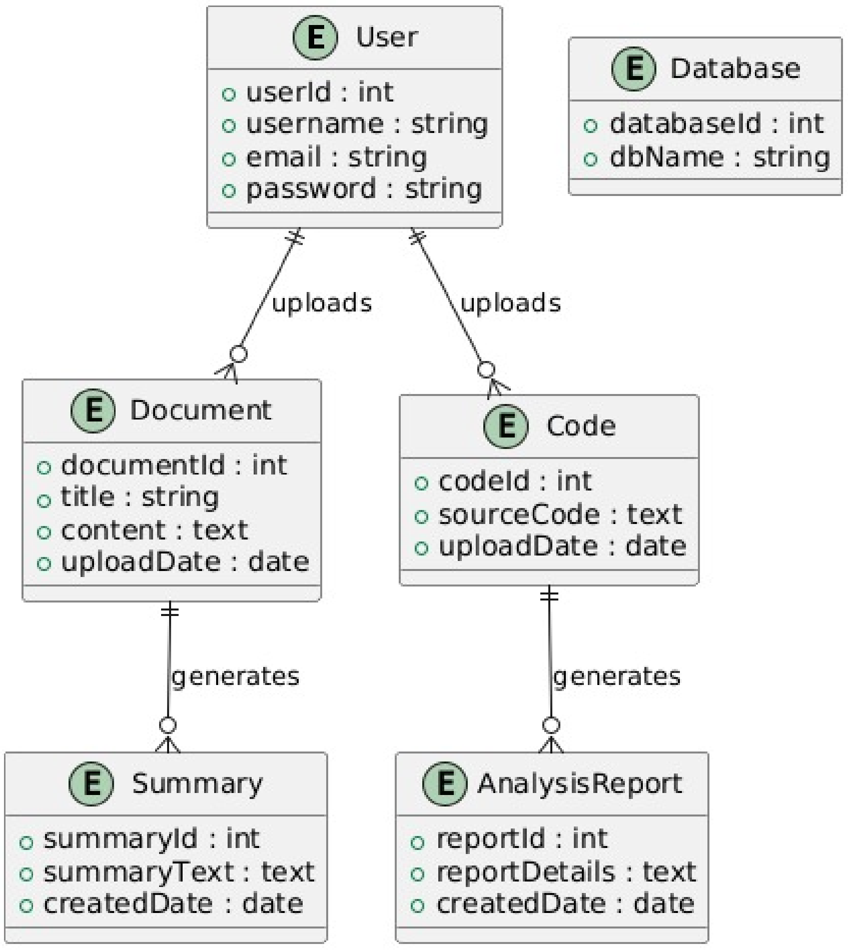


Figure 5.4: ER Diagram

5.4 Class Diagram

The Class Diagram shows the structure of the system in terms of classes, their attributes, methods, and relationships (associations).

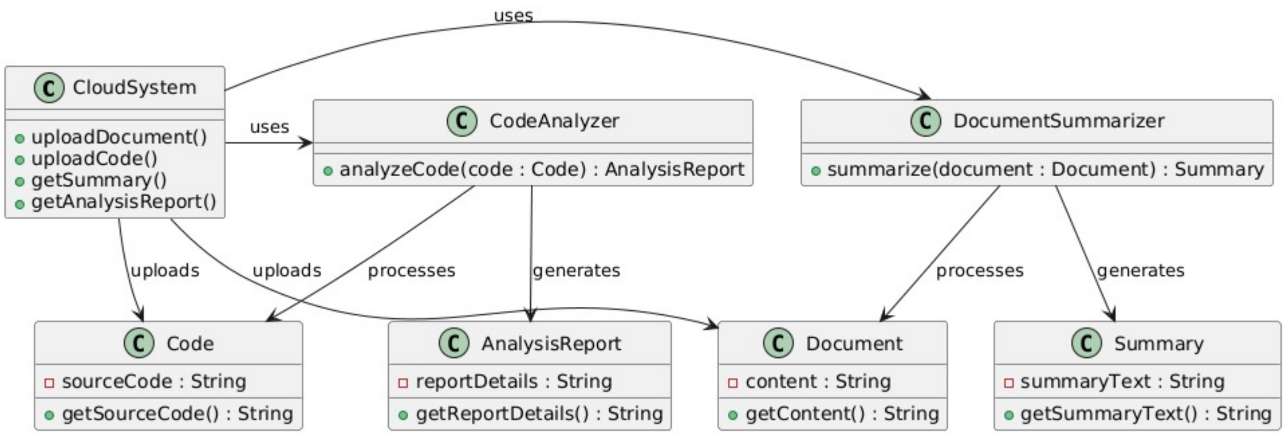


Figure 5.5: Class Diagram

5.5 Use-case Model Survey

The Use Case Diagram represents the interactions between the actors (users and system components) and the use cases (specific functionalities of the system).

Actors:

- **End User:** The user who uploads documents and code files for summarization or analysis.
- **Admin:** Manages the system, users, and database.

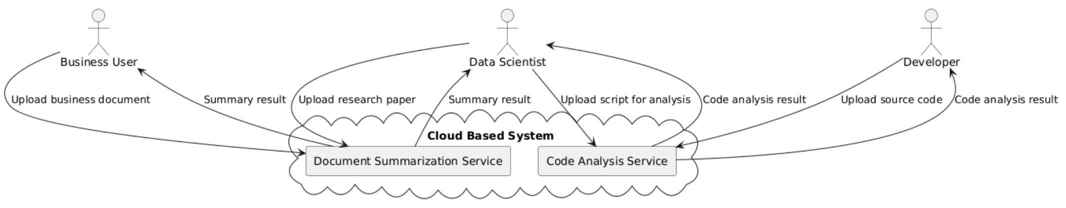


Figure 5.6: Use Case Diagram

5.6 Behavior Diagrams

5.6.1 Sequence Diagram

The sequence diagram demonstrates the communication flow between users and the system for key functionalities, such as uploading a document or code file, processing it using LLMs, and returning the summary or analysis.

- User Action: The user initiates a request by uploading a document.
- System Process: The system interacts with AWS services (e.g., Lambda, API Gateway) to process the file.
- LLM Summarization/Code Analysis: The LLM model or code analysis tools perform the summarization or analysis.
- Response: The summarized document or analysis report is returned to the user through the web interface.

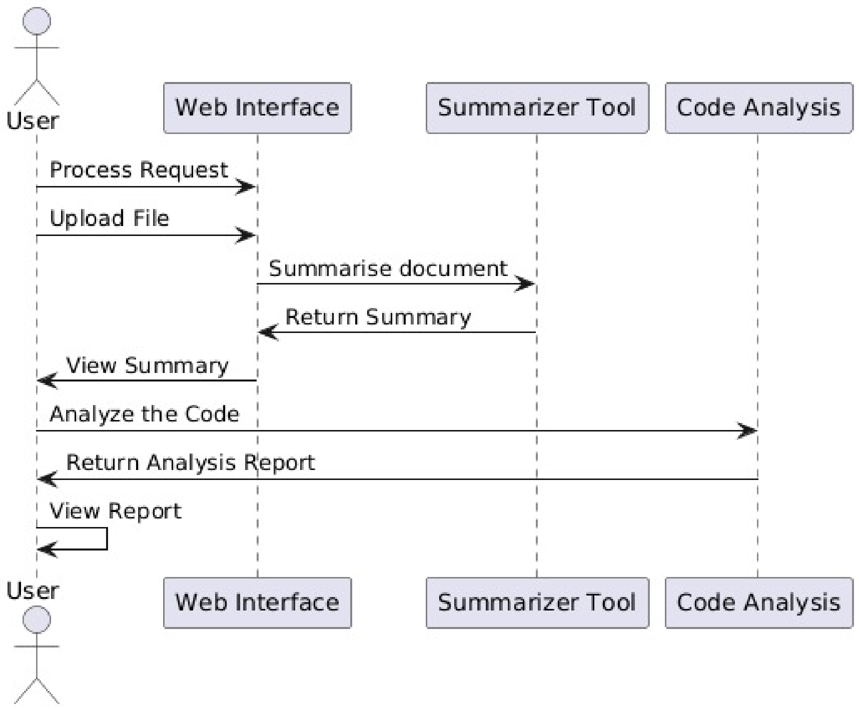


Figure 5.7: Sequence Diagram

5.6.2 Activity Diagram

The activity diagram outlines the steps involved in the summarization and analysis processes:

- Upload Action: The user initiates the process by uploading a document.
- Data Processing: The system processes the uploaded file using the appropriate module (LLM or code analysis).
- Result Generation: Summaries or code analysis reports are generated.
- Storage: The results are stored in the cloud (AWS S3).
- Result Delivery: The results are returned to the user for viewing. .

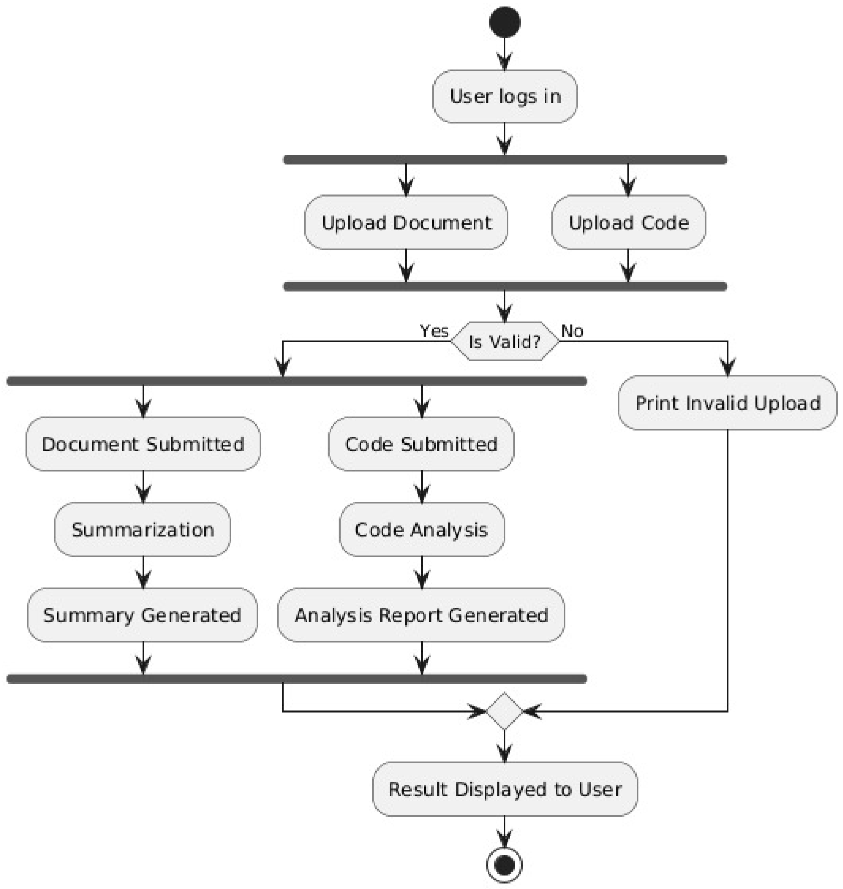


Figure 5.8: Activity Diagram

5.6.3 Communication Diagram

The Communication Diagram shows how objects (such as users, the system, and external components) interact with each other through messages.

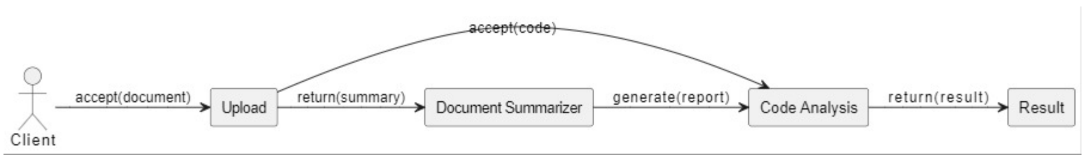


Figure 5.9: Communication Diagram

5.7 Structure Diagram

5.7.1 Component Diagram

The component diagram provides an overview of the system's structure and shows the relationship between different modules:

- Web Interface: Allows users to upload documents and code files.
- Summarization Module: Receives the input and uses LLMs (GPT-3.5) to process text.
- Code Analysis Module: Utilizes tools to perform code analysis.
- AWS Infrastructure: Hosts the web interface, summarization engine, and analysis tools using services such as AWS Lambda, EC2, API Gateway, and S3 for data storage.
- ChromaDB: Manages the vector embeddings for fast retrieval and analysis.

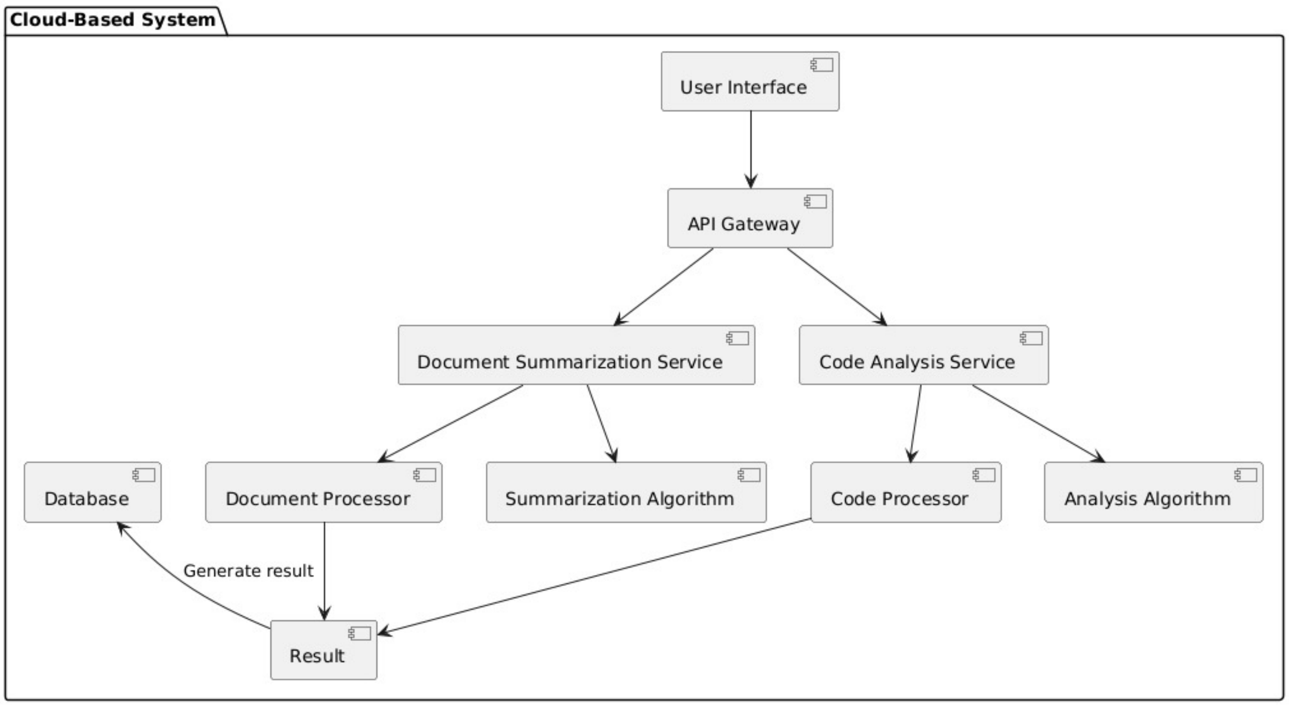


Figure 5.10: Component Diagram

5.7.2 Deployment Diagram

The deployment diagram illustrates how the system is deployed across AWS services:

- Web Server: Hosted on AWS EC2, which provides the user interface for document and code uploads.
- Lambda Functions: Execute LLM summarization and code analysis in a serverless environment.
- API Gateway: Facilitates communication between the client and backend services.
- S3 Storage: Stores uploaded files and analysis/summarization results.
- Database (ChromaDB): Manages data retrieval for summaries and analysis.

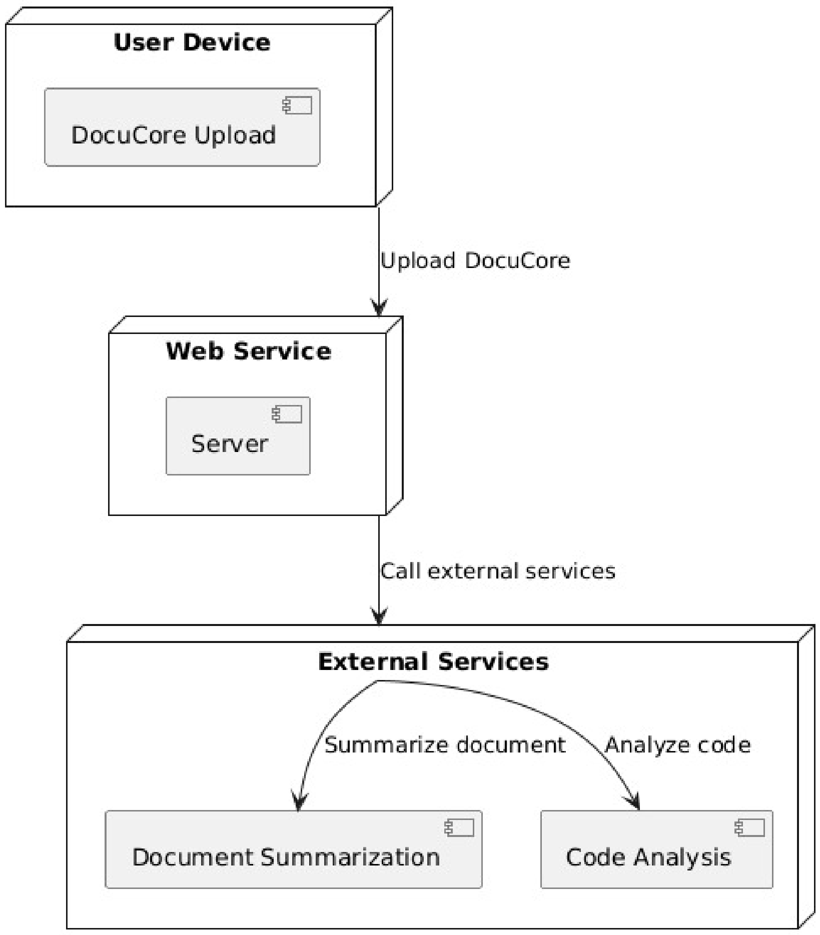


Figure 5.11: Deployment Diagram

5.8 Assumptions and Dependencies

5.8.1 Assumptions

- **Internet Connectivity:** It is assumed that users will have a stable and fast internet connection to interact with the system, as the document uploads, code analysis, and cloud-based operations rely on internet connectivity.
- **User Technical Knowledge:** It is assumed that the users (researchers, developers, etc.) have basic knowledge of uploading documents or code files and interpreting the results provided by the system (e.g., summaries or code analysis reports).
- **Availability of AWS Services:** The system assumes that AWS services, such as Lambda, API Gateway, and Elastic Beanstalk, are available and functioning properly. Any service downtime could impact the system's ability to process requests in real-time.
- **Document and Code Formats:** It is assumed that the uploaded documents will be in standard formats like .txt, .pdf, or .docx, and the code files will follow commonly used programming languages.

5.8.2 Dependencies

- **Summarisation Modules:** The system depends on various modules for generating document summaries. If the API service experiences any downtime or changes in access policies, it may disrupt the summarisation functionality.
- **Code Analysis Tools:** The code analysis module relies on code analysis tools to evaluate uploaded code files for bugs, complexity, and vulnerabilities. The performance of the system is dependent on the availability and accuracy of these tools.
- **Cloud Infrastructure (AWS):** The entire system is hosted on AWS, meaning it depends on AWS services like S3 for storage, EC2/Lambda for compute resources. Any issues with these services could lead to performance degradation or downtime.
- **Browser Compatibility:** The system's web-based interface depends on modern web browsers for proper functioning. It assumes compatibility with popular browsers like Chrome, Firefox, and Edge.

6. Supporting Information

6.1 List Of Figures

7. Conclusion and Future Scope

7.1 Conclusion

The Cloud-Based Document Summarization and Code Analysis System successfully integrates advanced technologies for document summarization and code analysis, offering users a unified platform for processing large volumes of text and code. By leveraging AWS cloud services, the system ensures scalability, high availability, and secure data storage. The platform streamlines workflows for researchers and developers by automating tasks that would otherwise be time-consuming, improving both productivity and code quality.

7.2 Future Scope

- **Support for Multiple Languages:** Expanding document summarization to support multilingual documents would make the system accessible to a global user base.
- **Real-Time Collaboration Features:** Adding real-time collaborative code analysis features would allow development teams to work together more effectively, enhancing productivity.
- **Mobile Integration:** Developing a mobile-friendly version of the platform would enable users to upload and analyze documents or code from their smartphones or tablets.
- **Advanced Summarization Models:** Future enhancements could include fine-tuning GPT models for specific domains (e.g., legal, medical) to improve summarization accuracy for technical documents.

8. Concerns/Queries/Doubts if any

8.1 Current Constraints

- **Lack of Customization for Summaries and Reports:**
  - Concern: Users have expressed a desire for more customization options in generating document summaries or code analysis reports.
  - Impact: The lack of flexibility in customizing outputs may limit its appeal.