**Exam Assignment**

Scientific Paper Analyzer: Cloud-Deployed Summarization and Q&A Tool

Team Members:

Ali Talasaz, Evan Parra, Bharath Sai Vallabhaneni, Ashritha Aloori

**Introduction**

This report presents the design, modeling, and deployment of the Scientific Paper Analyzer System, an AI-driven platform developed to analyze, summarize, and interact with academic documents. The assignment focuses on building a comprehensive solution architecture, supported by detailed use case, sequence, and class diagrams that illustrate system functionality, data flow, and component interactions. Additionally, the project integrates cloud deployment using Google Cloud Run, enabling scalable, serverless execution of the application through containerization and automated service management. These combined efforts demonstrate how users (students, researchers, developers, and reviewers) interact with the system in a cloud-based environment, how data is processed through retrieval-augmented generation (RAG) pipelines, and how the overall architecture ensures scalability, accuracy, and accessibility for academic research at production level.

**Solution Architecture**



The system is structured as a modular, cloud-deployed architecture that integrates multiple AI-driven components through a scalable backend hosted on Google Cloud Run. The architecture ensures seamless interaction between users and intelligent services while maintaining high availability and performance.

**Users (Students, Researchers, Instructors, and Administrators)**  
Users are the primary entry point to the system, responsible for initiating all interactions.

* **Students and Researchers** upload scientific papers, explore summaries, and ask natural-language questions to extract insights.
* **Instructors** review materials, validate academic quality, and incorporate findings into teaching content.
* **Administrators** manage user access, monitor performance metrics, and oversee cloud service health for reliability and compliance.

These roles define the interaction scope and ensure that the system remains accessible, accurate, and user-focused.

**Chatbot Interface**  
The chatbot serves as the communication layer between users and backend services. It enables document uploads, natural-language queries, and formatted responses through a web or chat-based interface. Integrated with FastAPI and hosted on Cloud Run, it provides a responsive, user-friendly experience accessible from any device.

**RAG Prompt Augmentation Module**  
This module applies Retrieval-Augmented Generation (RAG) techniques to enrich user queries with contextually relevant document excerpts retrieved from the vector database. By merging retrieved content with user input, it ensures responses are accurate, well-grounded, and context-aware.

**LLM Backend (Language Model)**  
The **Large Language Model (LLM)**, deployed as a cloud microservice, serves as the system’s reasoning engine. It interprets complex natural language, synthesizes key information, and produces academic-quality responses such as summaries, explanations, and insights.

**Response Generator**  
This component refines and formats the LLM’s raw outputs, ensuring consistent structure, clarity, and citation formatting. Responses are returned in a professional, reader-friendly format, suitable for both academic and instructional use.

**Knowledge Base / Session Memory**  
The knowledge base manages session data, conversation context, and previously processed summaries. It enables multi-turn, stateful interactions that persist across user sessions, supporting a smooth and intelligent dialogue experience.

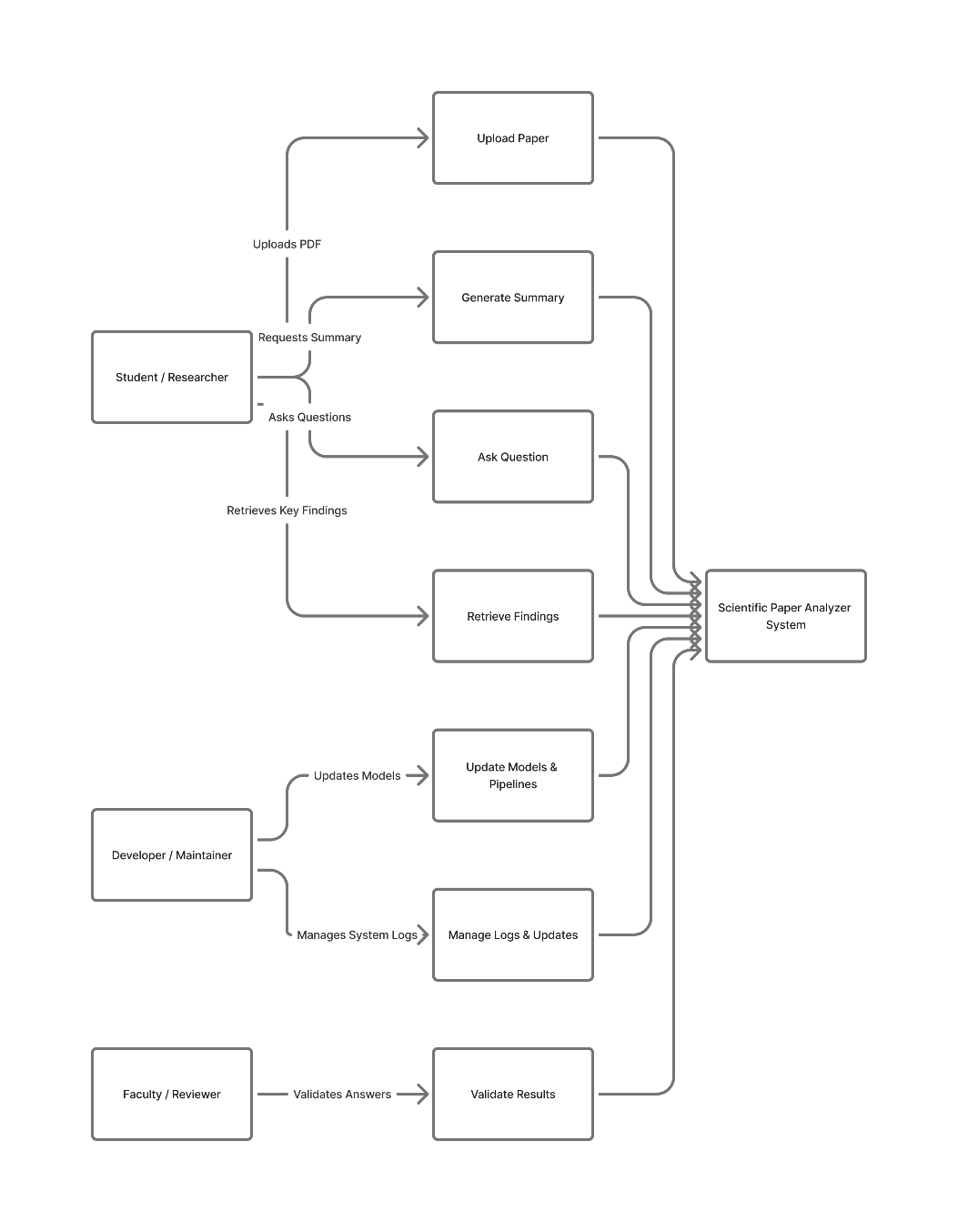
**Cloud Deployment Layer**  
The entire system is containerized and deployed on Google Cloud Run, enabling auto-scaling, serverless management, and secure access via HTTPS. This cloud deployment ensures robustness, cost efficiency, and minimal maintenance while allowing real-time updates and monitoring through integrated Cloud Build and Artifact Registry services.

**User Case Diagram**

The system connects multiple user roles to ensure both usability and accuracy in research workflows:

* **Students/Researchers** – Upload research papers, request summaries, and ask context-aware questions to quickly extract methodologies, results, and key insights without manually reading entire documents.
* **Faculty/Reviewers** – Review and validate the generated summaries and responses, ensuring the information presented is accurate, credible, and aligned with academic standards.
* **Developers/Maintainers** – Manage and optimize the backend components, including retrieval-augmented generation (RAG) pipelines, model updates, and performance monitoring. They also oversee Google Cloud Run deployment, ensuring scalability, uptime, and system efficiency.

Deployed through a serverless cloud infrastructure, the system enables real-time collaboration among users and automated scaling for multiple concurrent sessions. The use case diagram visually represents these user interactions and illustrates how each actor engages with the core system functions to support efficient, AI-assisted academic research.



**Sequence Diagram**

The sequence diagram illustrates the step-by-step flow of interactions that occur when a user submits a query about a scientific paper through the chatbot interface. It highlights how various components within the Scientific Paper Analyzer System, deployed on Google Cloud Run, collaborate to deliver accurate, context-aware responses in real time.

* The User initiates an interaction by sending a query through the ChatAPI, which acts as the primary communication gateway.
* The ChatAPI, hosted on the cloud, forwards the request to the VectorStoreManager, which performs a semantic similarity search to retrieve the most relevant document chunks from the vector database.
* The retrieved chunks are passed to the PromptAugmentor, which constructs an enriched prompt by combining the user’s question with contextual excerpts from the document.
* This augmented prompt is sent to the LLMService, which leverages a cloud-hosted large language model (e.g., NVIDIA or OpenAI API) to generate a precise and well-structured answer.
* The ResponseFormatter refines the model’s raw output, adding structure, formatting, and source citations for readability and credibility.
* The final response is returned to the User via the ChatAPI, completing the interaction loop.
* Meanwhile, the KnowledgeBase is updated asynchronously to store the session data, maintaining conversational memory and summaries for continuity.

This sequence ensures smooth, cloud-scaled communication between components—allowing the system to deliver fast, reliable, and context-rich academic insights even under high user demand.

A diagram of a diagram

AI-generated content may be incorrect.

The proposed interactions in the sequential diagram ensure:

* **Grounded Responses:** All answers are derived directly from the uploaded document through retrieval-augmented generation (RAG), minimizing the risk of hallucinated or unsupported content.
* **Modular Design:** Distinct components for retrieval, augmentation, generation, and formatting improve system scalability, maintainability, and fault isolation.
* **Context-Enriched Querying:** Query augmentation with relevant text segments enhances factual accuracy and the depth of generated responses.
* **Low-Latency Performance:** Vector similarity search ensures rapid retrieval from large document datasets, maintaining fast response times in a cloud environment.
* **Persistent Context:** The knowledge base enables multi-turn, session-aware conversations with evolving summaries and context continuity.
* **Seamless Scalability:** Component separation and containerized deployment (e.g., via Google Cloud Run) allow independent upgrades of LLMs, user interfaces, or storage modules without system downtime.

**Class Diagram**

Option 1:

A diagram of a software company

AI-generated content may be incorrect.

This UML class diagram represents a scientific paper analyzer bot built with LangChain and FastAPI.

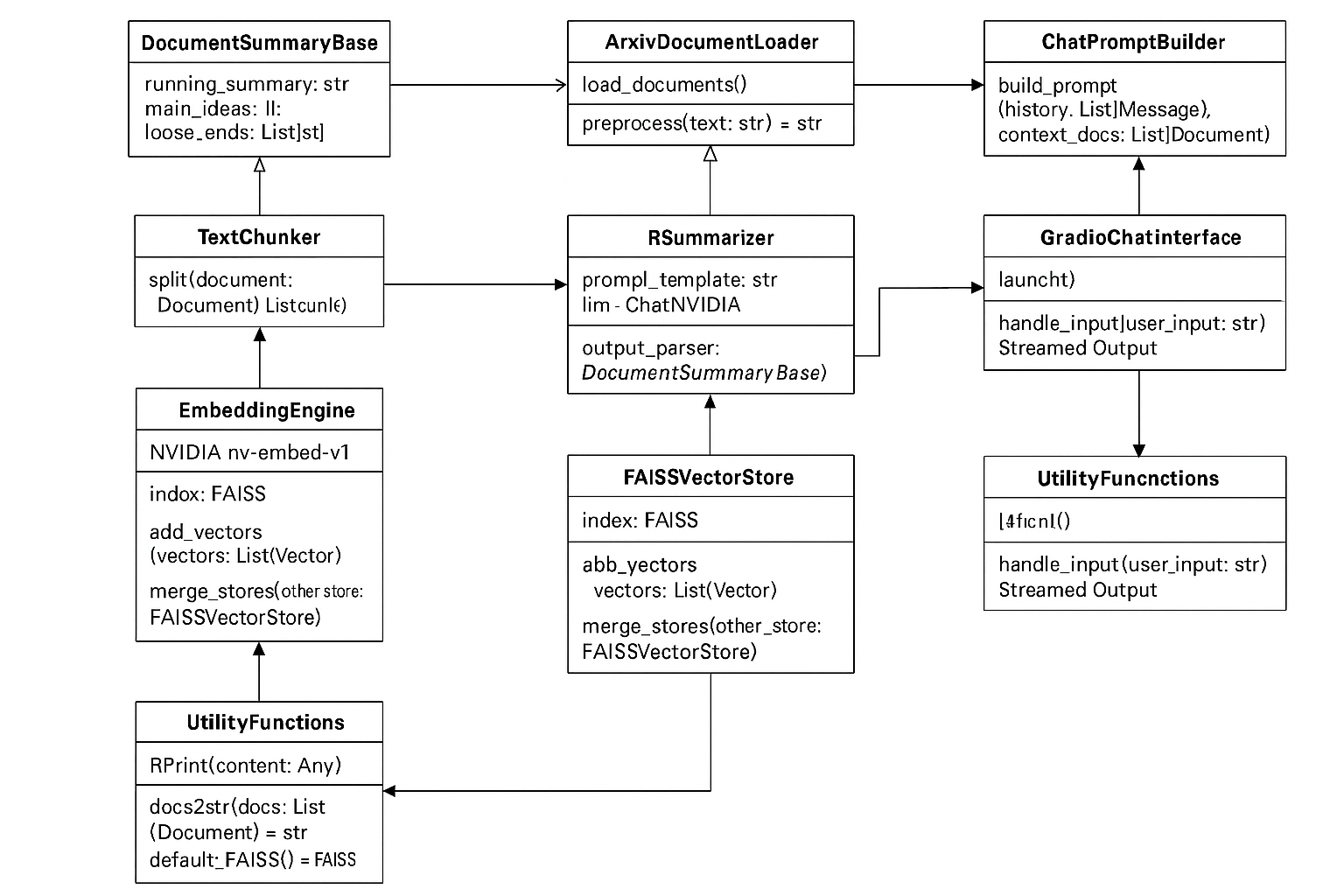
* **UserSession** manages individual user interactions, including document uploads and chat history.
* **ChatAPI** serves as the interface layer, exposing FastAPI endpoints like /basic\_chat, /retriever, and /generator.
* **DocumentParser** handles preprocessing of uploaded files, converting them into clean, chunked text.
* **VectorStoreManager** creates and stores embeddings in a vector database (e.g., FAISS) and retrieves similar chunks based on queries.
* **PromptAugmentor** builds enriched prompts using relevant text snippets retrieved from the vector store.
* **LLMService** connects to a large language model (e.g., NVIDIA) to generate answers or summaries.
* **ResponseFormatter** cleans and structures the raw model output, optionally highlighting source citations.
* **KnowledgeBase** maintains a memory of conversations and document summaries for session continuity.

Option 2:

This UML class diagram represents the Scientific Paper Analyzer Bot, an AI-powered application built with LangChain, FastAPI, and deployed on Google Cloud Run for scalable and reliable operation. Each class plays a distinct role in the document analysis, summarization, and interaction pipeline:

* DocumentSummaryBase: A Pydantic model that tracks running summaries, key ideas, and unanswered questions during document analysis.
* ArxivDocumentLoader: Loads and preprocesses documents (e.g., from arXiv) to prepare them for chunking and summarization.
* TextChunker: Splits documents into smaller chunks using RecursiveCharacterTextSplitter for efficient summarization and embedding.
* RSummarizer: Uses ChatNVIDIA to incrementally summarize each chunk while updating a shared DocumentSummaryBase state.
* EmbeddingEngine: Converts text chunks into vector embeddings using NVIDIA’s nv-embed-v1 model.
* FAISSVectorStore: Stores and manages vector embeddings using FAISS, allowing for fast similarity search and merging of vector stores.
* ChatPromptBuilder: Assembles prompts for chat by combining conversation history and relevant documents retrieved from the vector store.
* ChatEngine: Handles chat logic, streams responses using ChatNVIDIA, and stores conversation context via ConvStore.
* GradioChatInterface: Provides a user-friendly chat UI, handles input, and streams model replies using Gradio.
* UtilityFunctions: A helper class offering reusable utilities like formatted printing, document formatting, and default FAISS setup.

This architecture ensures a modular, cloud-scalable, and maintainable system—where each component operates independently yet cohesively. The design supports future upgrades to models, embeddings, or UI components without major refactoring, aligning with software engineering best practices for AI-based academic tools.

\

**Conclusion**

In conclusion, this assignment successfully developed the design and modeling framework for the Scientific Paper Analyzer System, an AI-driven platform for academic document analysis. The work encompassed the creation of a solution architecture, use case diagram, sequence diagram, and class diagram, each demonstrating a distinct layer of system logic, interaction, and data flow. These models collectively illustrate how various user roles (students, researchers, instructors, and administrators) engage with the platform, and how components interact through retrieval-augmented generation (RAG) and large language model pipelines. The system’s modular and cloud-deployed architecture ensures scalability, reliability, and accessibility across diverse environments. Overall, this structured design establishes a robust foundation for future implementation, cloud-based deployment, and continued enhancement of an intelligent, context-aware academic analysis tool.