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

Team Members: Ali Talasaz, Evan Parra, Bharath Sai Vallabhaneni, Ashritha Aloori

General Architecture Overview

This report details the design, modeling, and deployment of the Scientific Paper Analyzer System, an AI-driven platform for analyzing, summarizing, and interacting with academic documents. The project focuses on a comprehensive solution architecture, illustrated with use case, sequence, and class diagrams. It also incorporates cloud deployment using Google Cloud Run for scalable, serverless execution via containerization and automated service management. This demonstrates how users (students, researchers, developers, and reviewers) interact with the system in a cloud environment, how data is processed through retrieval-augmented generation (RAG) pipelines, and how the architecture ensures scalability, accuracy, and accessibility for academic research.

Solution Architecture

The system features a modular, cloud-deployed architecture integrating multiple AI components through a scalable backend on Google Cloud Run, ensuring seamless user interaction with intelligent services, high availability, and performance.

- **Users (Students, Researchers, Instructors, and Administrators):**
    - **Students and Researchers:** Upload papers, explore summaries, and ask natural-language questions for insights.
    - **Instructors:** Review materials, validate academic quality, and integrate findings into teaching.
    - **Administrators:** Manage user access, monitor performance, and oversee cloud service health.
      These roles define interaction scope and ensure an accessible, accurate, and user-focused system.
- **Chatbot Interface:** Serves as the communication layer, enabling document uploads, natural-language queries, and formatted responses via a web or chat-based interface. It's integrated with FastAPI and hosted on Cloud Run for a responsive, user-friendly experience.
- **RAG Prompt Augmentation Module:** Uses Retrieval-Augmented Generation (RAG) to enrich user queries with relevant document excerpts from the vector database, ensuring accurate, grounded, and context-aware responses.

- **LLM Backend (Language Model):** A cloud microservice acting as the system's reasoning engine. It interprets complex natural language, synthesizes information, and produces academic-quality responses (summaries, explanations, insights).
- **Response Generator:** Refines and formats the LLM's raw outputs for consistent structure, clarity, and citation formatting, delivering professional, reader-friendly responses.
- **Knowledge Base / Session Memory:** Manages session data, conversation context, and previous summaries, enabling multi-turn, stateful interactions across user sessions.
- **Cloud Deployment Layer:** The entire system is containerized and deployed on Google Cloud Run for auto-scaling, serverless management, and secure HTTPS access. It integrates with Google Cloud Build and Artifact Registry for automated builds, deployments, and monitoring, ensuring robustness, cost efficiency, and real-time updates.

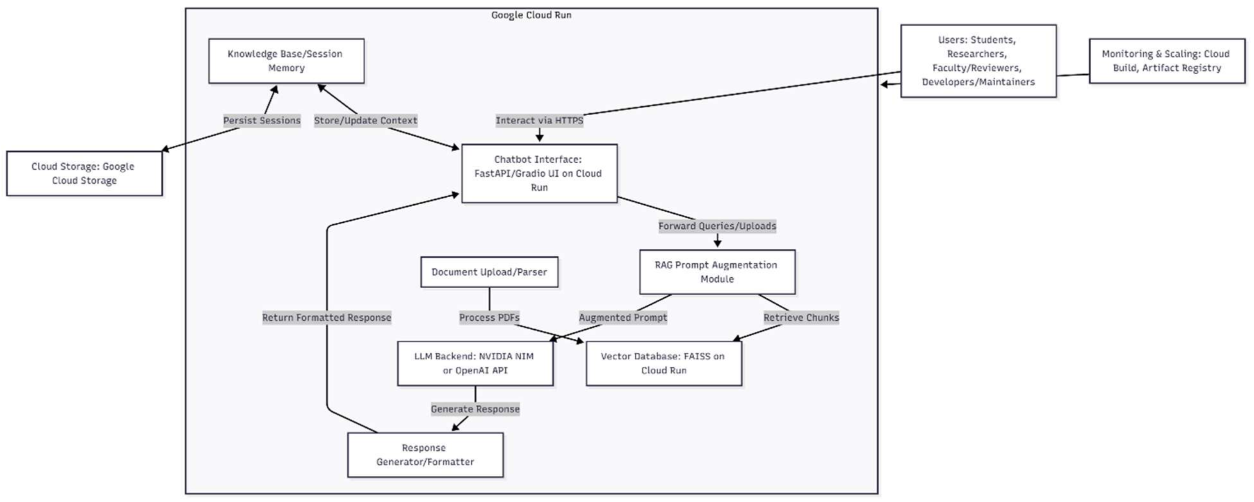
Google Cloud Run
Knowledge Base/Session Memory
Users: Students, Researchers, Faculty/Reviewers, Developers/Maintainers
Monitoring & Scaling: Cloud Build, Artifact Registry
Persist Sessions
Store/Update Context
Interact via HTTPS
Cloud Storage: Google Cloud Storage
Chatbot Interface: FastAPI/Gradio UI on Cloud Run
Forward Queries/Uploads
Document Upload/Parser
RAG Prompt Augmentation Module
Return Formatted Response
Process PDFs
Augmented Prompt
Retrieve Chunks
LLM Backend: NVIDIA NIM or OpenAI API
Vector Database: FAISS on Cloud Run
Generate Response
Response Generator/Formatter

Figure 1: High-Level Architecture Diagram. Illustrates the system's modular components, including chatbot interface, RAG modules, LLM backend, and Google Cloud Run deployment for scalability.

Use Case Diagram

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

- **Students/Researchers:** Upload papers, request summaries, and ask context-aware questions to extract insights without manual reading.
- **Faculty/Reviewers:** Review and validate generated summaries and responses for academic standards.
- **Developers/Maintainers:** Manage and optimize backend components (RAG pipelines, model updates, performance monitoring) and oversee Google Cloud Run deployment for scalability and efficiency.

Deployed through a serverless cloud infrastructure, the system enables real-time collaboration and automated scaling for concurrent sessions, visually represented in the use case diagram, emphasizing cloud deployment aspects.

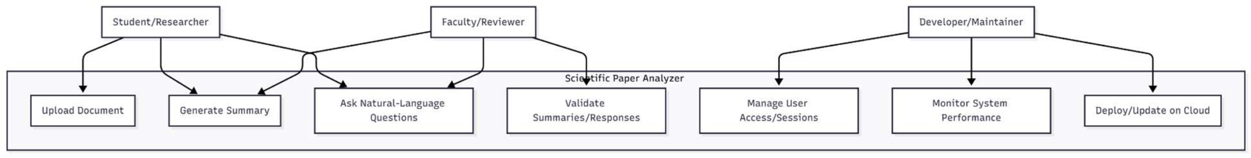
Student/Researcher
Faculty/Reviewer
Developer/Maintainer
Scientific Paper Analyzer
Upload Document
Generate Summary
Ask Natural-Language Questions
Validate Summaries/Responses
Manage User Access/Sessions
Monitor System Performance
Deploy/Update on Cloud

Figure 2: Use Case Diagram. UML depiction of user roles (Students/Researchers, Faculty/Reviewers, Developers/Maintainers) interacting with functions like upload, summarization, Q&A, and cloud management.

Sequence Diagram

This diagram illustrates the step-by-step interaction flow when a user queries a scientific paper via the chatbot interface, highlighting how components deployed on Google Cloud Run collaborate for real-time, accurate responses.

1. **User:** Initiates query via ChatAPI.
2. **ChatAPI:** Forwards request to VectorStoreManager.
3. **VectorStoreManager:** Performs semantic similarity search to retrieve relevant document chunks.
4. **PromptAugmentor:** Constructs an enriched prompt by combining the user's question with contextual excerpts.
5. **LLMService:** Uses a cloud-hosted large language model (e.g., NVIDIA or OpenAI API) to generate an answer.
6. **ResponseFormatter:** Refines the model's raw output, adding structure, formatting, and citations.
7. **ChatAPI:** Returns the final response to the User.
8. **KnowledgeBase:** Updated asynchronously to store session data and maintain conversational memory.

This sequence ensures smooth, cloud-scaled communication, delivering fast, reliable, and context-rich academic insights, even under high demand, with components running as microservices on Google Cloud Run and asynchronous knowledge base updates.

Proposed interactions ensure:

- **Grounded Responses:** Answers derived directly from the document via RAG, minimizing unsupported content.
- **Modular Design:** Distinct components for retrieval, augmentation, generation, and formatting improve scalability, maintainability, and fault isolation.
- **Context-Enriched Querying:** Query augmentation enhances factual accuracy and response depth.

- **Low-Latency Performance:** Vector similarity search ensures rapid retrieval.
- **Persistent Context:** Knowledge base enables multi-turn, session-aware conversations.
- **Seamless Scalability:** Containerized deployment allows independent upgrades without downtime.

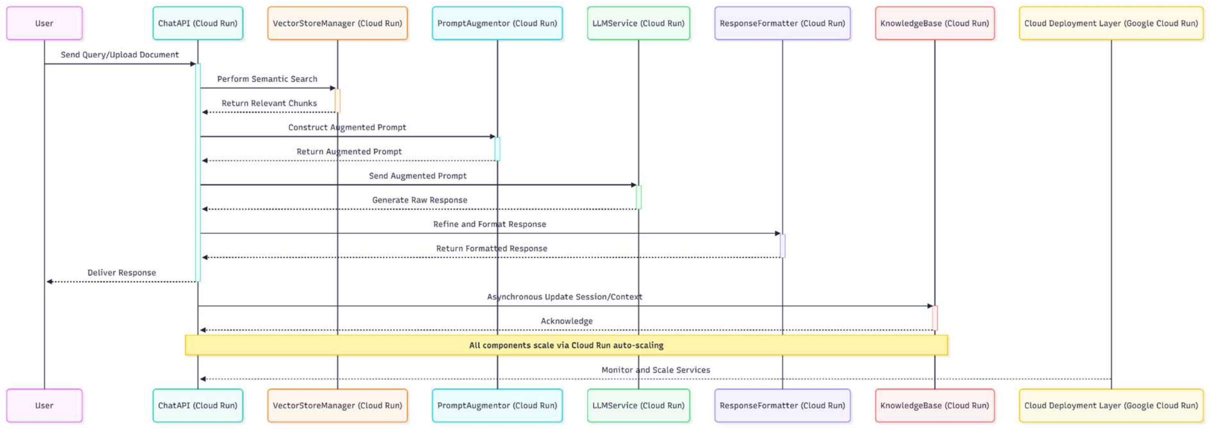
User
ChatAPI (Cloud Run)
VectorStoreManager (Cloud Run)
PromptAugmentor (Cloud Run)
LLMService (Cloud Run)
ResponseFormatter (Cloud Run)
KnowledgeBase (Cloud Run)
Cloud Deployment Layer (Google Cloud Run)
Send Query/Upload Document
Perform Semantic Search
Return Relevant Chunks
Construct Augmented Prompt
Return Augmented Prompt
Send Augmented Prompt
Generate Raw Response
Refine and Format Response
Return Formatted Response
Deliver Response
Asynchronous Update Session/Context
Acknowledge
All components scale via Cloud Run auto-scaling
Monitor and Scale Services

Figure 3: Sequence Diagram. Step-by-step query processing flow: from ChatAPI initiation to retrieval, augmentation, LLM generation, formatting, and knowledge base updates in a Google Cloud Run microservices setup.

Class Diagram

This UML class diagram represents the Scientific Paper Analyzer Bot, an AI-powered application built with LangChain, FastAPI, and deployed on Google Cloud Run. Each class plays a distinct role:

- **DocumentSummaryBase:** Pydantic model for tracking summaries, key ideas, and unanswered questions.
- **ArxivDocumentLoader:** Loads and preprocesses documents for chunking and summarization.
- **TextChunker:** Splits documents into smaller chunks using RecursiveCharacterTextSplitter.
- **RSummarizer:** Uses ChatNVIDIA to incrementally summarize chunks and update DocumentSummaryBase.
- **EmbeddingEngine:** Converts text chunks into vector embeddings using NVIDIA's nv-embed-v1 model.
- **FAISSVectorStore:** Stores and manages vector embeddings for fast similarity search.
- **ChatPromptBuilder:** Assembles prompts by combining conversation history and retrieved documents.
- **ChatEngine:** Handles chat logic, streams responses using ChatNVIDIA, and stores context via ConvStore.
- **GradioChatInterface:** Provides a user-friendly chat UI.

- **UtilityFunctions:** Helper class for formatted printing, document formatting, and FAISS setup.
- **CloudDeployment:** Deploys containers, enables auto-scaling, and monitors health for cloud integration.

This architecture ensures a modular, cloud-scalable, and maintainable system, supporting future upgrades without major refactoring.

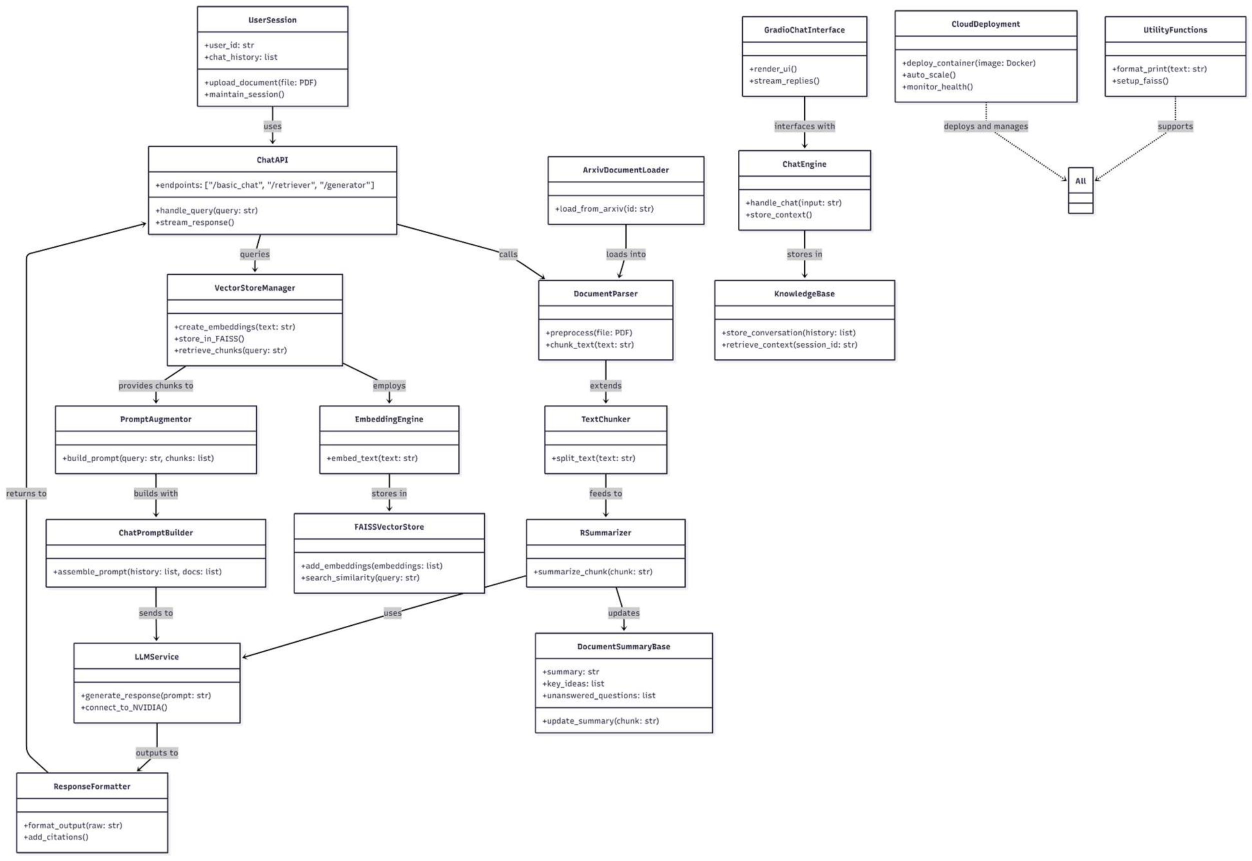

Figure 4: Class Diagram. UML model of system classes, attributes, methods, and relationships, covering document processing (e.g., ArxivDocumentLoader), RAG (e.g., FAISSVectorStore), chat (e.g., GradioChatInterface), and cloud integration (e.g., CloudDeployment).

Conclusion

This assignment successfully developed the design and modeling framework for the Scientific Paper Analyzer System. The solution architecture, use case, sequence, and class diagrams illustrate system logic, interaction, and data flow, demonstrating how various user roles engage with the platform and how components interact through RAG and large language model pipelines. The system's modular and cloud-deployed architecture ensures scalability, reliability, and accessibility, establishing a robust foundation for future implementation and enhancement.