Chat with Logistics Documents (RAG)

Course Name: Generative AI

Institution Name: Medicaps University – Datagami Skill Based Course

Student Names & Enrolment Numbers:

S.NO	STUDENT NAME	ENROLLMENT NO.
1.	AKSHAT BAJAJ	EN22CS301088
2.	AJAY PAL SINGH THAKUR	EN22CS301077
3.	AKSHITA FARKYA	EN22CS301099
4.	AASHITA CHAVHAN	EN22CS301018
5.	AKANKSHA PARIHAR	EN22CS301079
6.	AAKRATI JAIN	EN22CS301013

Group Name: 05D1

Project Number: GAI-05

Industry Mentor Name: AASHRUTI SHAH

University Mentor Name: Prof. AJAJ KHAN

Academic Year: 2025-2026

Problem Statement & Objectives

1. Problem Statement

In the logistics industry, critical operational information is stored in lengthy PDF documents such as shipment manuals, warehouse guidelines, compliance documentation, and transportation policies. Extracting relevant information manually from these large documents is time-consuming and inefficient.

Traditional keyword-based search systems fail to capture semantic meaning and contextual relevance. Additionally, generic AI chat systems may generate hallucinated responses that are not grounded in the actual document content, which is risky in professional logistics operations.

To address this problem, a Retrieval-Augmented Generation (RAG) based intelligent document assistant is required. The system should:

Accept logistics PDF documents.

Process and convert them into semantic vector embeddings.

Store embeddings in a vector database.

Retrieve contextually relevant information.

Generate grounded responses using a Large Language Model (LLM).

This project implements a complete RAG pipeline using LangChain, FAISS, HuggingFace embeddings, and Google Gemini LLM.

2. Project Objectives

The primary objectives of this project are:

1. To build an end-to-end RAG-based logistics document chatbot.
2. To implement PDF document loading using PyPDFLoader.
3. To perform intelligent text chunking using RecursiveCharacterTextSplitter.
4. To generate semantic embeddings using HuggingFaceEmbeddings.
5. To store embeddings in a FAISS vector database.
6. To retrieve relevant document chunks using similarity search.
7. To generate accurate responses using Google Gemini (models/gemini-2.5-flash).

8. To integrate the system with a Streamlit-based user interface.
9. To deploy the application using Pyngrok in Google Colab.
10. To reduce hallucination by grounding responses strictly in retrieved document context.

3. Scope of the Project

In Scope:

- PDF document upload and processing.
- Text extraction using PyPDFLoader.
- Text chunking with RecursiveCharacterTextSplitter.
- Embedding generation using HuggingFaceEmbeddings.
- Vector storage using FAISS.
- Semantic similarity retrieval.
- Response generation using ChatGoogleGenerativeAI.
- Streamlit-based user interface.
- Colab-based deployment using Pyngrok.

Out of Scope:

- Enterprise authentication system.
- Multi-user database management.
- OCR for scanned PDFs.
- Real-time logistics API integration.
- Cloud-native production deployment.

This project serves as a working prototype demonstrating practical implementation of RAG architecture.

Proposed Solution

1. Key Features

- PDF document ingestion using PyPDFLoader
- Intelligent text chunking
- Semantic embedding generation using HuggingFaceEmbeddings
- FAISS-based vector storage
- Similarity search for context retrieval
- Google Gemini (2.5 Flash) LLM integration
- Grounded response generation
- Streamlit interactive UI
- Colab + Pyngrok deployment

2. Overall Architecture / Workflow

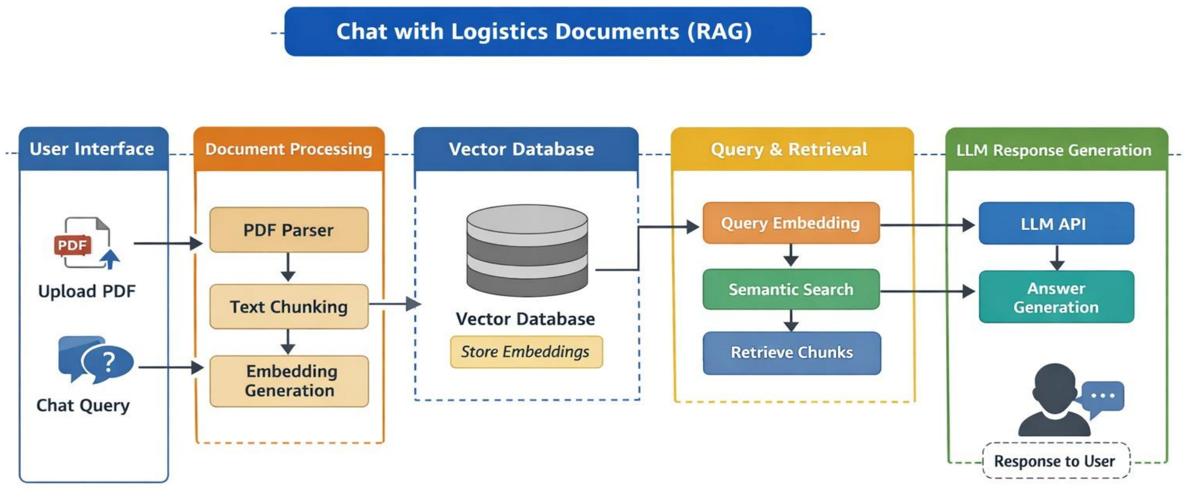

Fig: System architecture of chat with logistics

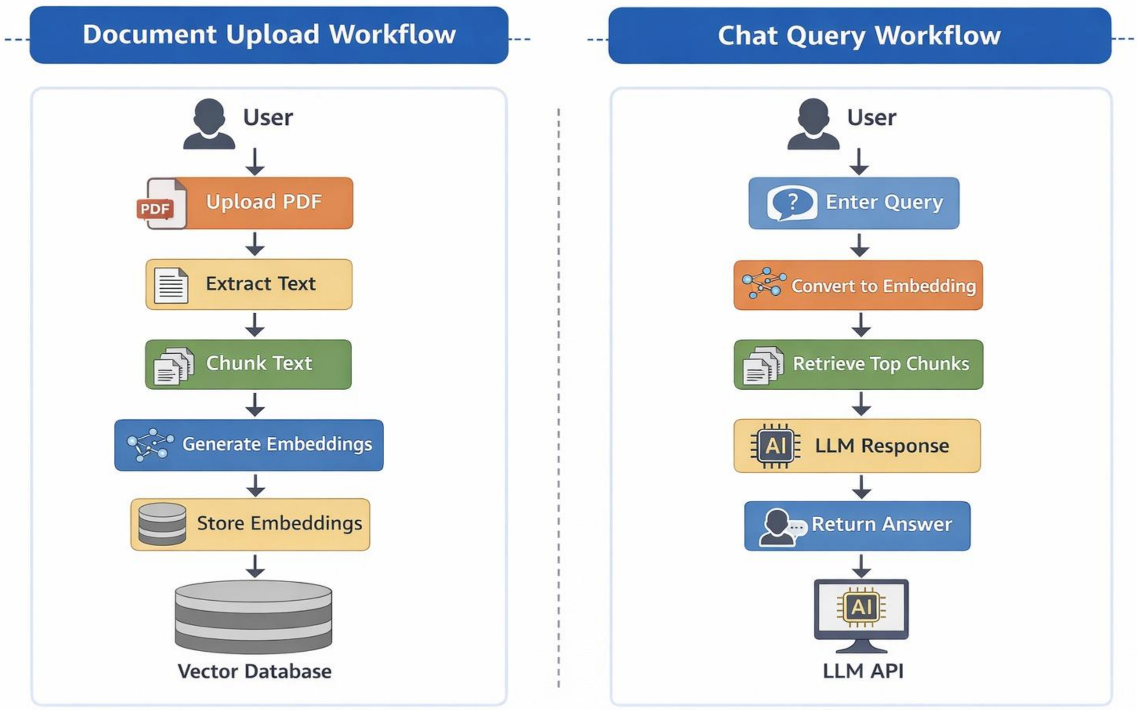

Fig: RAG system workflows for logistics documents

Phase 1: Document Processing

1. User uploads logistics PDF.
2. PyPDFLoader extracts text from the PDF.
3. RecursiveCharacterTextSplitter divides text into smaller chunks.
4. HuggingFaceEmbeddings converts each chunk into vector embeddings.
5. FAISS stores embeddings locally in memory.

Phase 2: Query Processing

1. User enters a query in Streamlit interface.
2. Query is converted into embedding using HuggingFaceEmbeddings.
3. FAISS performs similarity search.
4. Top-k relevant chunks are retrieved.
5. Retrieved chunks are passed to ChatGoogleGenerativeAI.
6. Gemini model (models/gemini-2.5-flash) generates context-aware response.

7. Streamlit displays final answer to user.

Architecture Type

- Retrieval-Augmented Generation (RAG)
- Modular LangChain pipeline
- Local vector storage (FAISS)
- External LLM API (Google Gemini)

3. Tools & Technologies Used

Programming Language:

- Python

Framework & Libraries:

- LangChain
- PyPDFLoader
- RecursiveCharacterTextSplitter
- HuggingFaceEmbeddings
- FAISS (Vector Store)
- ChatGoogleGenerativeAI
- google-generativeai
- Streamlit
- Pyngrok
- pypdf
- python-docx

LLM Used:

- Google Gemini (model: models/gemini-2.5-flash)

Development Environment:

- Google Colab
- GitHub (version control)

Results & Output

1. Screenshots / Outputs

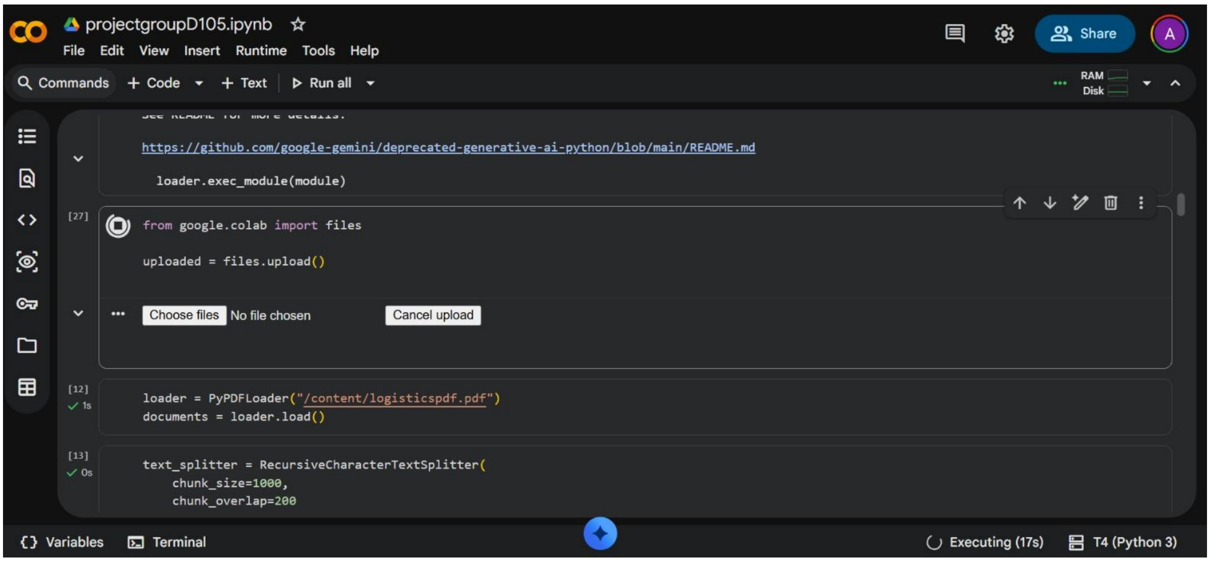

Fig: PDF upload

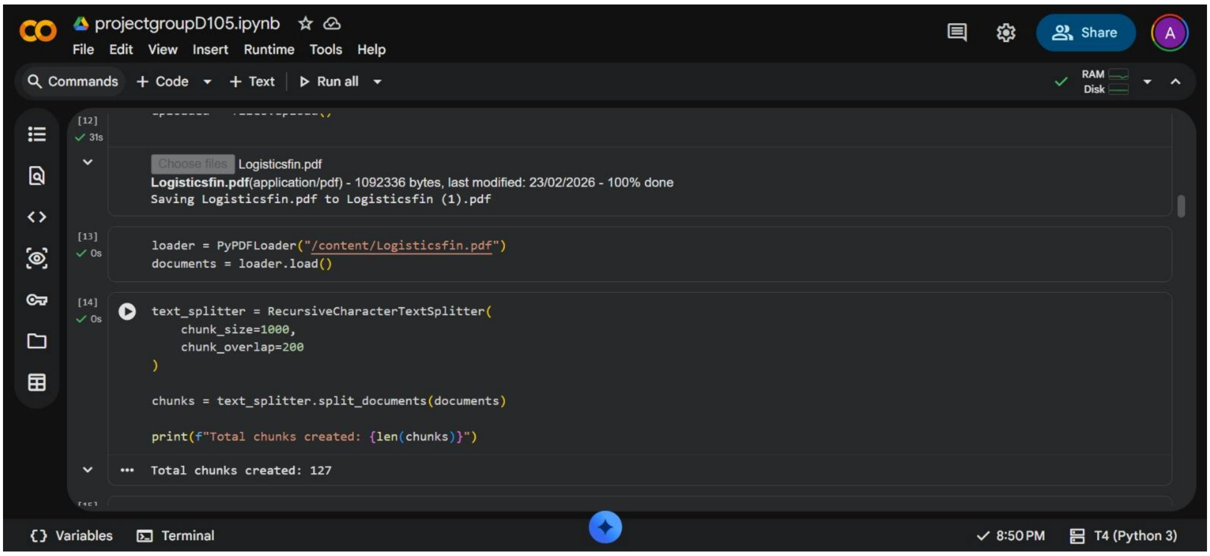

Fig: No. of Chunks created

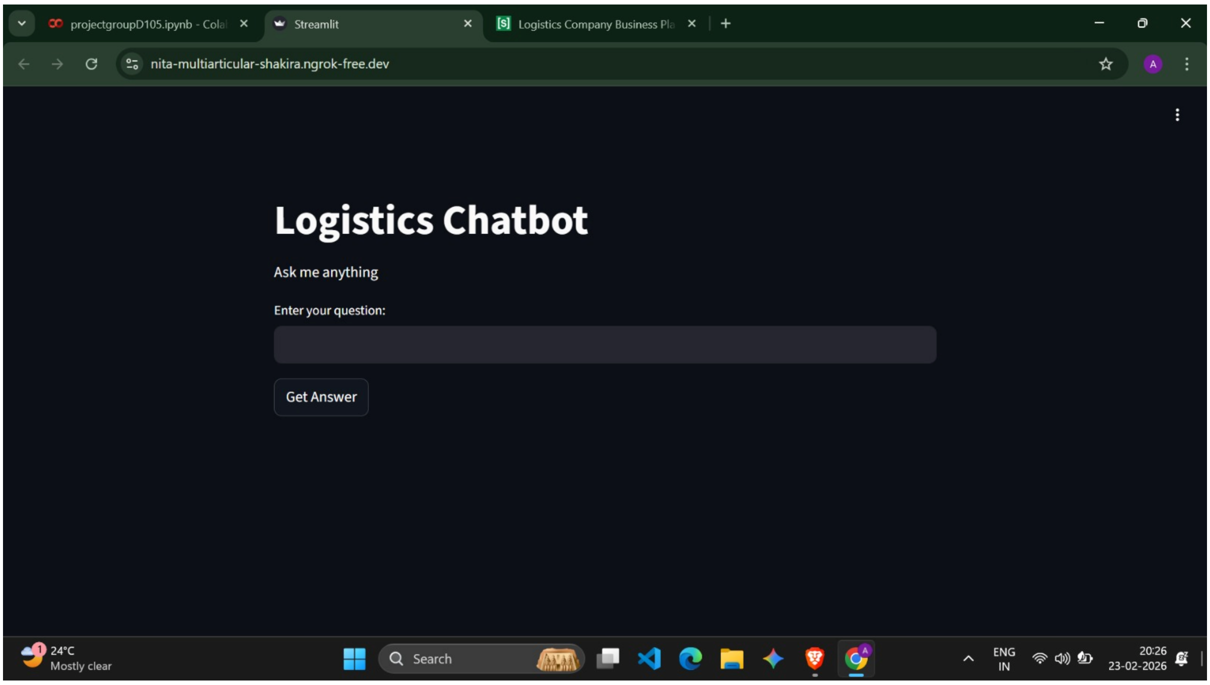

Fig: Streamlit UI interface

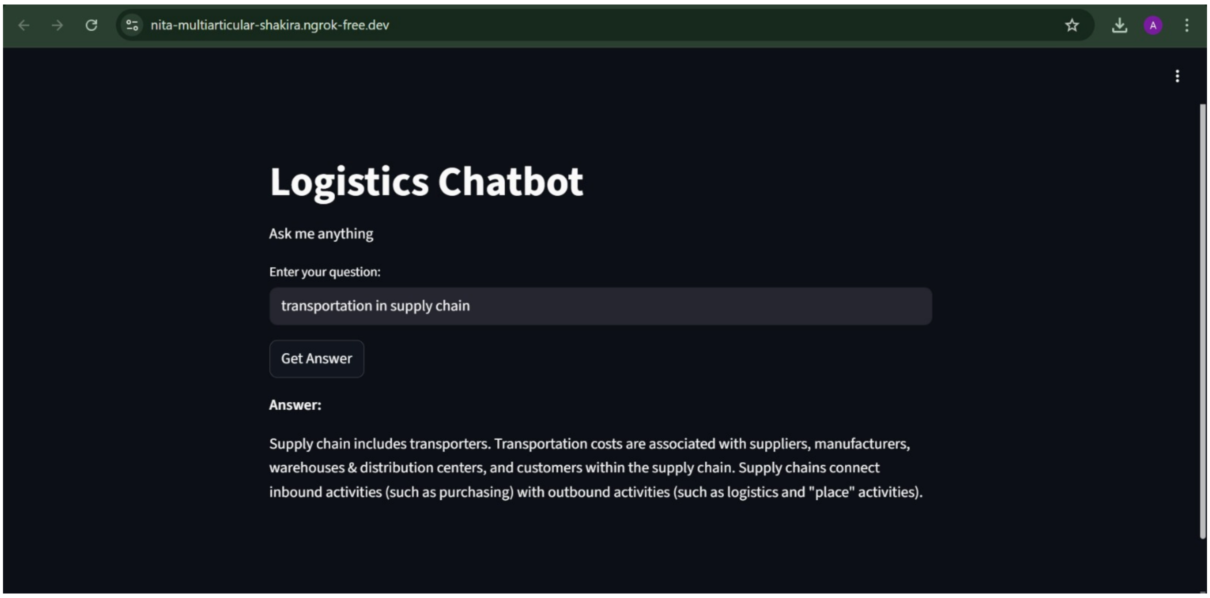

Fig: Logistics Chatbot Interface Output

System Output:

- FAISS retrieves relevant chunks.
- Gemini generates grounded answer based only on document.

- Response displayed in Streamlit chat UI.

2. Reports / Dashboards / Models

Our project demonstrates:

- FAISS vector index creation
- Embedding model (HuggingFace sentence transformer)
- Gemini LLM integration
- Query-to-context retrieval mechanism
- Streamlit-based frontend rendering
- Local in-memory vector storage model

Performance observations:

- Fast similarity retrieval via FAISS.
- Accurate contextual responses.
- Reduced hallucination due to retrieved context grounding.

3. Key Outcomes

1. Successfully implemented complete RAG architecture.
2. Integrated LangChain pipeline with Gemini LLM.
3. Built working vector search using FAISS.
4. Developed interactive UI using Streamlit.
5. Deployed Colab-based app using Pyngrok.
6. Gained practical experience in:
    - Vector embeddings
    - Semantic search
    - LLM integration
    - Prompt engineering
    - AI system architecture

Conclusion

The “Chat with Logistics Documents (RAG)” project successfully implements a complete Retrieval-Augmented Generation system using LangChain, FAISS, HuggingFace embeddings, and Google Gemini LLM.

The system enables intelligent semantic search over logistics PDF documents and generates accurate, context-aware responses through a Streamlit interface.

This project provided hands-on experience in:

- Building RAG pipelines
- Working with vector databases (FAISS)
- Integrating Google Gemini LLM
- Using LangChain framework
- Deploying AI applications using Colab and Pyngrok

Overall, the project bridges traditional document search systems with modern AI-driven semantic intelligence.

Future Scope & Enhancements

The system can be enhanced by:

1. Adding persistent vector storage instead of in-memory FAISS.
2. Supporting multiple document uploads.
3. Adding conversation memory.
4. Implementing role-based authentication.
5. Deploying on cloud platforms (AWS/GCP/Azure).
6. Adding hybrid search (keyword + semantic).
7. Supporting scanned PDF OCR.
8. Creating analytics dashboard for query insights.
9. Fine-tuning a domain-specific embedding model.
10. Implementing multi-turn conversational context retention.