### 1. Introduction

The purpose of this project is to implement a Retrieval-Augmented Generation (RAG) system that answers user queries based on the contents of a PDF document. The system uses LangChain, Google Gemini API, and Streamlit to provide an interactive chatbot experience.

### Key capabilities:

- Answers user queries from text and tables inside PDFs.
- Uses embeddings + retrieval for context-aware answers.
- Provides a web-based interface for users via Streamlit.

### 2. Problem Statement

Traditional chatbots rely solely on their pretrained knowledge and cannot answer questions about specific documents.

This project solves that problem by:

- Loading a PDF into a retrievable format.
- Creating embeddings of document chunks.
- Using a retriever + LLM pipeline to answer user queries.

# 3. System Architecture

### 3.1. Components

- 1) Frontend (UI):
- Implemented with Streamlit.
- Provides a chat-style interface for user queries and bot responses.

### 2) Backend (RAG pipeline):

- · LangChain for chaining steps.
- Google Gemini (Generative Al API) as the LLM.
- Vector index built from document embeddings for retrieval.

### 3) Data Source:

- PDF file(s) uploaded into the system.
- Example: <u>AAPL Q3 2022 Report</u>.

### 3.2. Workflow

### Step 1: Load Document

PDF is read and split into text chunks.

### Step 2: Embed & Index

- Embeddings are generated for each chunk.
- Stored in a vector index for similarity search.

### **Step 3: User Query**

User inputs a question in Streamlit.

### Step 4: Retrieval

• The retriever finds the most relevant document chunks.

### Step 5: LLM Response

- Retrieved context + user question are sent to Gemini LLM.
- The LLM generates a final answer.

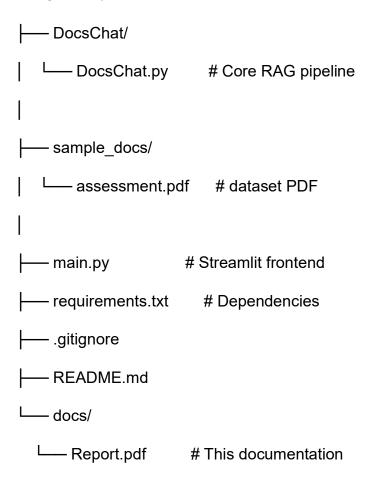
### Step 6: Display

Response shown in Streamlit chat UI.

## 4. Implementation

# 4.1. Directory Structure

RAG-DocsChatBot/



### 4.2 Core Code Functions

- initial\_load(): Loads PDF, splits into chunks, creates embeddings + retriever.
- query(question): Handles user queries by passing them through the retrieval + LLM pipeline.
- main.py: Implements Streamlit UI, calls query() to display results interactively.

# 5. Installation & Usage

# **5.1. Clone Repository**

```
git clone https://github.com/siikum/RAG-DOC-CHAT.git cd RAG-DOC-CHAT
```

### **5.2. Create Virtual Environment**

```
python -m venv venv

source venv/Scripts/activate # Windows

source venv/bin/activate # Linux/Mac
```

## 5.3 Install Dependencies

pip install -r requirements.txt

# 5.4 Add API Key

- Get a free **Google Gemini API Key** from <u>Al Studio</u>.
- Save it in .env file:
- GOOGLE\_API\_KEY=your\_api\_key\_here

# 5.5 Run the Application

streamlit run main.py

## 6. Limitations

- Works only with text-based PDFs (not scanned images without OCR).
- Limited to the quality of the retriever and embeddings.
- Currently processes one PDF at a time.
- Figures/images are not processed.

## 7. Future Improvements

- Support multiple PDFs at once.
- Improve table extraction (current table parsing depends on PDF text encoding).

## 8. Conclusion

This project demonstrates how RAG systems can effectively answer domain-specific questions using text and table information from PDF documents. It combines LangChain, Google Gemini, and Streamlit to create a practical, interactive chatbot for document-based QA.