Project Report: RAG-Powered PDF Chatbot

Prepared by: Yashraj Singh Rathore **Date:** August 14, 2025

1. Introduction & Project Objective

The primary goal of this project was to develop a sophisticated, AI-powered chatbot capable of answering user queries based on the content of a provided PDF document. The solution required the implementation of a **Retrieval-Augmented Generation (RAG)** pipeline, integrating a vector database and an instructed language model, all served through a user-friendly and interactive Streamlit interface with real-time streaming responses.

The final application successfully meets these objectives, providing users with a powerful tool to quickly and accurately extract information from dense documents.

2. System Architecture and Flow

The application is built on a modern RAG architecture, which ensures that the AI's responses are grounded in the provided text, minimizing inaccuracies and "hallucinations."

Architectural Flow

The user's interaction with the chatbot follows a five-step process:

1. **Ingestion & Chunking**: The user uploads a PDF document. The system uses PyPDFLoader to extract the raw text, which is then intelligently split into smaller, overlapping chunks of about 1000 characters using the RecursiveCharacterTextSplitter.
2. **Embedding**: Each text chunk is converted into a high-dimensional vector (an "embedding") using the HuggingFaceEmbeddings library with the all-MiniLM-L6-v2 model. This process runs locally on the user's machine.
3. **Indexing & Storage**: The generated embeddings and their corresponding text chunks are loaded into an in-memory **ChromaDB** vector database. This creates a searchable index of the entire document.
4. **Retrieval**: When a user asks a question, the application embeds the query and uses ChromaDB to perform a similarity search, retrieving the most relevant text chunks from the database.
5. **Generation**: The retrieved text chunks (the "context") and the user's original question are inserted into a carefully designed prompt. This complete prompt is then sent to the **Google Gemini API**, which generates a coherent, context-aware answer.

3. Document Structure and Chunking Logic

The system is designed to handle unstructured text from standard PDF documents. The core of the preprocessing is the **chunking strategy**, which is vital for effective retrieval.

- **Document Loading:** The application uses the PyPDFLoader library to extract raw text content from the uploaded PDF file.

- **Chunking Logic:** The extracted text is then passed to LangChain's RecursiveCharacterTextSplitter. This splitter was chosen for its robustness in handling document structure. It intelligently tries to split text along sentence boundaries and paragraphs before resorting to a hard character limit.
    - **Chunk Size:** We use a chunk_size of **1000 characters**. This is large enough to capture meaningful context but small enough for efficient processing.
    - **Chunk Overlap:** A chunk_overlap of **200 characters** is configured. This ensures that a sentence or idea that spans the end of one chunk and the beginning of the next is not lost.

4. Explanation of Embedding Model and Vector Database Used

- **Embedding Model (all-MiniLM-L6-v2):** For the critical task of converting text chunks into numerical vectors, we use the all-MiniLM-L6-v2 model from Hugging Face. This model was selected because it offers an excellent balance of performance and resource efficiency. It runs entirely on the user's local machine, which means the embedding process is **100% free and private**.
- **Vector Database (ChromaDB):** ChromaDB was chosen as the vector database for its simplicity and speed. It runs entirely **in-memory**, which means there is no need to set up or manage a separate database server. This makes the application self-contained and easy to run.

5. Example Queries with Responses

Performance demonstrated using the AI Training Document.pdf

- **Query 1 (Success Case - Factual Recall):**
    - **Question:** "Can I use a web scraper to access eBay's services?"
    - **Response:** *"No, you cannot. The User Agreement states that you will not use any robot, spider, scraper, data mining tools, data gathering and extraction tools, or other automated means to access the Services for any purpose, except with the prior express permission of eBay."*
    - **Analysis:** This is a success. The model correctly identified the relevant policy and provided a direct, accurate answer.
- **Query 2 (Success Case - Synthesis):**
    - **Question:** "What are the main responsibilities when listing an item for sale?"
    - **Response:** *"When listing an item for sale on our Services, you agree to comply with eBay's Listing policies and Selling practices policy and also agree that: You assume full responsibility for the item offered and the accuracy and content of the listing... Your listing may not be immediately searchable... Content that violates any of eBay's policies may be modified, obfuscated, or deleted..."*
    - **Analysis:** This is also a success. The model synthesized information from the "Listing Conditions" section to provide a comprehensive summary.
- **Query 3 (Failure Case - Out of Context):**

- **Question:** "What is the return policy for items bought on Amazon?"
- **Response:** *"I don't have enough information to answer that."*
- **Analysis:** This is a "successful failure." The model correctly recognized that the document does not contain information about Amazon and followed its instructions to not invent an answer.

6. Notes on Hallucinations, Model Limitations, or Slow Responses

- **Hallucinations:** The RAG architecture, combined with our strict prompt, significantly minimizes the risk of the AI inventing facts ("hallucinations"). However, no model is perfect. The "View Sources" feature was implemented as a critical tool for users to cross-reference the AI's claims with the original text.
- **Model Limitations:** The quality of the answer is entirely dependent on the quality of the provided document. If the PDF contains vague, contradictory, or poorly written information, the AI's answers will reflect that.
- **Slow Responses:** The primary bottleneck in the application is the **initial document processing step**, where embeddings are created locally on the user's CPU. This is a one-time cost per document and can be slow on older machines. The actual question-answering process is very fast, as it leverages the powerful Google Gemini API.