RAG-Based Chatbot Project Documentation

**Problem Domain Description:**

This RAG-based chatbot project addresses the challenge of providing accurate, real-time responses by integrating retrieval-based methods with generative language models. Traditional chatbots often rely solely on pre-trained language models, which can result in outdated or inaccurate responses when queried about specific or domain-rich knowledge.

The chatbot developed in this project uses the Retrieval-Augmented Generation (RAG) approach to enhance response quality. It is designed for use cases such as customer support, academic assistants, legal document referencing, or any scenario requiring precise answers from a specific set of documents. Users can interact with the chatbot to receive factually correct, contextually relevant, and up-to-date answers drawn directly from the ingested documents.

# RAG Pipeline – Working Implementation

1. Document ingestion and preprocessing:  
    Documents are uploaded in various formats (PDF, DOCX, TXT). Each document is parsed and cleaned—removing extra whitespace, special characters, and headers/footers. The text is then chunked into manageable sizes to maintain context during embedding.
2. Embedding generation and vector store indexing:

Using models like Sentence Transformers or OpenAI embeddings, each chunk is converted into vector representations. These vectors are then stored in a vector database such as ChromaDB. This indexing facilitates quick and accurate similarity searches during retrieval.

1. Query processing and document retrieval:

When a user inputs a query, it is first embedded into a vector. A similarity search is performed against the indexed documents to retrieve the most relevant chunks. These retrieved chunks are then compiled and passed as context to the generative model.

1. Augmented prompt generation and LLM response:

The top-k relevant document chunks are embedded into the prompt as context before the user query. This augmented prompt is then fed into a large language model (e.g., OpenAI GPT, Mistral, or LLaMA via Ollama), which generates a coherent, informed answer. In this project I used the Ollama embedding models with right with the size of the data w.r.t the system size.

# Chatbot Interface:

The chatbot interface is built using Streamlit for an interactive and user-friendly experience. Users can upload documents, enter queries, and receive contextual answers in real time. The interface supports dynamic updates and allows users to view source documents for transparency.

Optional integrations with LangChain enable structured prompt templating and modular pipeline design, improving maintainability and extensibility of the chatbot.

Architecture:

* Document Ingestion layer: Handles files uploads and preprocessing .
* Embedding Layer: Convert text chunks to vectors using embedding models.
* Vector Store: ChormaDB indexes and retrieves similar documents
* LLM Layer: Generates responses using augmented prompts.
* UI Layers: Steamlit interface for user interaction.

Design Choices:

* Chosed ChormaDB for fast similarity search.
* Used Ollama Embedding Models for high quality embeddings.
* Adopted modular Langchain components for scalability.
* Selected Streamlit for rapid prototyping and clean interface.

**How to run:**

step-1: Install dependencies using ‘pip install –r requirment.txt’

step-2: Start the app with ‘Streamlit run app.py’

step-3: Upload documents and begin chatting.