## RAG-Powered Smart Chatbot for Any Document

## Overview

This project enables document understanding through a Retrieval-Augmented Generation (RAG) chatbot. It allows users to upload documents (PDF, DOCX, PPTX, TXT), which are processed, chunked, and embedded using Azure OpenAI’s embedding models. The chunks are stored in ChromaDB and queried using semantic similarity to generate contextual responses using GPT-4o.

## System Flow Structure

The following steps outline the processing pipeline:

* 1. Upload: User uploads documents via Streamlit UI (PDF, DOCX, PPTX, TXT).
* 2. Parsing: Extract and clean raw text from documents using unstructured loaders.
* 3. Chunking: Split the text into small manageable pieces
* 4. Embedding: Use Azure OpenAI's `text-embedding-3-small ` model to convert text chunks into high-dimensional vectors.
* 5. Storage: Save the vectors in a local ChromaDB database.
* 6. Search: Convert user query into embedding and find similar documents using vector similarity.
* 7. Response Generation: GPT-4o uses retrieved chunks as context to generate a grounded answer.
* 8. Response includes document source, page.

## Azure OpenAI Embedding: text-embedding-3-small

* Efficient and lightweight embedding model.
* Converts text into 1536-dimensional vectors.
* Used for semantic similarity matching with user queries.
* Optimized for fast computation and small memory footprint.

## GPT-4o: Advanced Chat Completion

* Multimodal and real-time generative model.
* Receives chunks and query context to produce relevant answers.
* Understands citations, formatting, and technical language.
* Excellent zero-shot reasoning and document summarization capabilities.

## Architecture Diagram

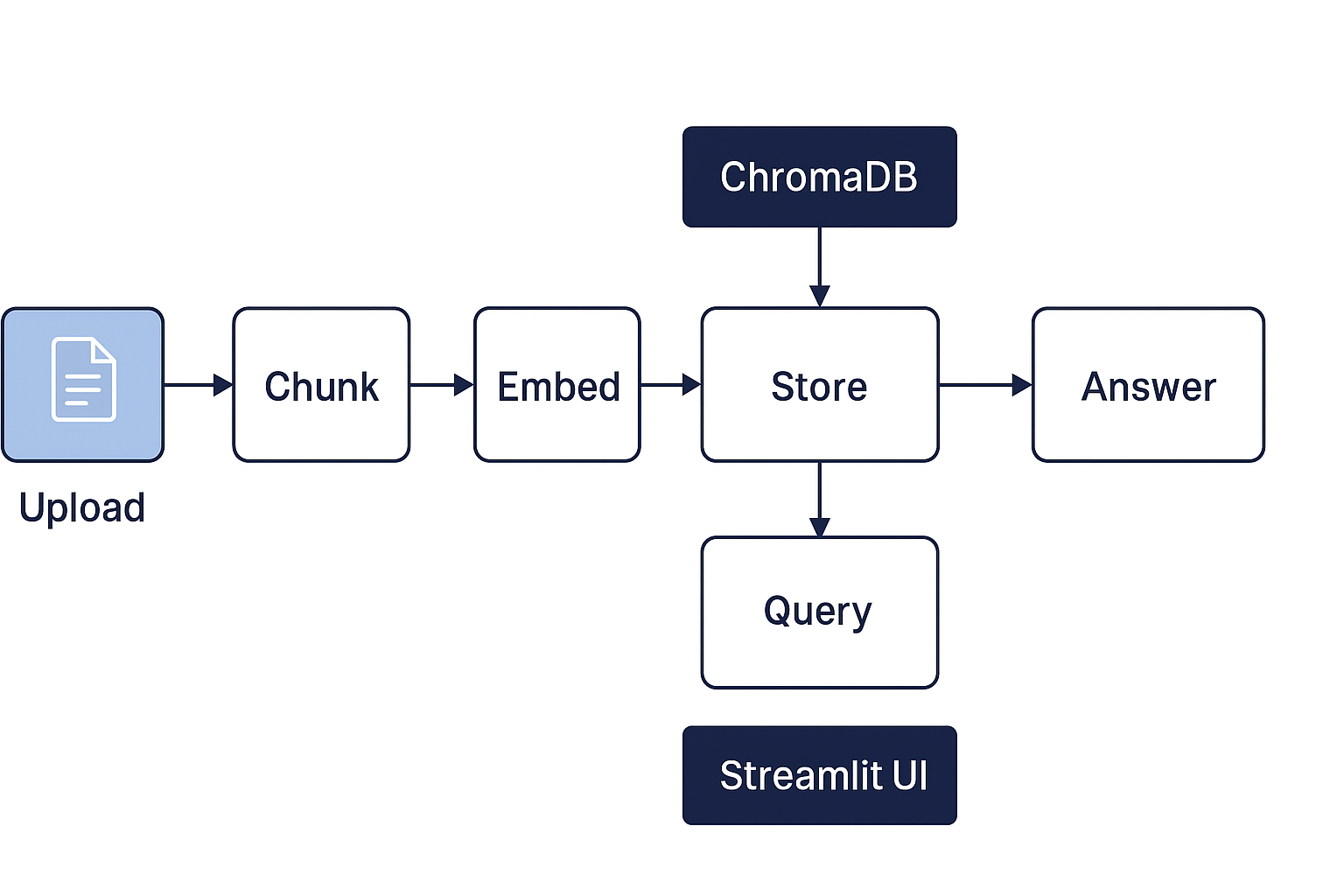


Figure: End-to-end architecture of the RAG chatbot system.

## Key Features

* Streamlit UI with file upload and reset/clear options.
* Supports PDFs, Word, PowerPoint, and plain text files.
* Local ChromaDB vector store for efficient retrieval.
* Chunked embedding for context-preserving answers.
* GPT-4o driven responses grounded in original content.

Please refer to the snippet below from the chatbot project.

I uploaded a Pptx and asked, "what are SQL Constraints ?" The chatbot responded with relevant information extracted from the file.

A screenshot of a chatbot

AI-generated content may be incorrect.

I uploaded a PDF and asked, "Who is Kalpna?" The chatbot responded with relevant information extracted from the file. And also references from a file .

A screenshot of a chatbot

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

* chatbot project was built using **FastAPI** for handling API endpoints like document upload and chat-based querying. FastAPI was chosen for its speed, simplicity, and automatic documentation support.
* The APIs are fully documented with Swagger UI (available at /docs) These tools make it easy to **test**, **integrate**, and **extend** the chatbot capabilities with minimal setup. The backend supports file uploads (PDF, DOCX, PPTX, TXT), processes them into vector embeddings, and responds to natural language questions.

A screenshot of a computer

AI-generated content may be incorrect.

A black and white striped background

AI-generated content may be incorrect.