Project Report: Chatbot_RAG

Table of Contents

Project Report: Chatbot_RAG

Abstract

Chatbot_RAG is an advanced Retrieval-Augmented Generation (RAG) based application that enables semantic interaction with documents, specifically PDFs, by combining semantic search with natural language understanding. The system accepts user queries about uploaded documents and provides accurate, contextually grounded answers by retrieving the most relevant segments from the document and augmenting them into a language model prompt. The application leverages powerful AI tools such as Hugging Face's Zephyr-7B model and Sentence Transformers, creating a robust framework suitable for academic, legal, or corporate document processing.

Introduction

In today's information-driven world, extracting relevant information from large documents remains a significant challenge. Traditional keyword-based search systems often fall short in understanding the semantic context of user queries. Retrieval-Augmented Generation (RAG) bridges this gap by combining two powerful components: a retriever that finds relevant context and a generator that synthesizes a response.

Chatbot_RAG is designed to address the problem of document comprehension by providing an intelligent interface that understands and answers user queries based on uploaded document content. It is especially useful for users seeking specific information from dense textual resources like research papers, reports, or manuals. The system is modular, scalable, and designed with extensibility in mind.

Objective

The primary goals of this project include:

- Developing a semantic QA chatbot that interacts with user-provided documents.
- Utilizing state-of-the-art NLP models for embedding and language generation.
- Implementing document ingestion, chunking, and embedding workflows.
- Creating a persistent vector storage solution for rapid and efficient retrieval.
- Designing an intuitive and responsive user interface using Gradio.
- Promoting real-world usability with a focus on modular design and easy extensibility.

System Architecture

The architecture of Chatbot_RAG is designed in a modular fashion, allowing each component to perform a distinct task within the pipeline. The major components are:

1. **Document Ingestion**: Accepts and reads user-uploaded documents in PDF format.

2. **Text Chunking**: Splits large text into manageable and context-preserving chunks.

3. **Vector Embedding**: Converts text chunks into high-dimensional vectors using the `all-MiniLM-L6-v2` model from Sentence Transformers.

4. **Vector Storage**: Stores embedded vectors in ChromaDB, a fast and lightweight vector store.

5. **Query Processing**: Encodes the user's query into a vector and uses cosine similarity to retrieve the most semantically relevant chunks.

6. **Prompt Construction**: Builds a structured prompt using the retrieved content and user query.

7. **Response Generation**: Sends the prompt to the Zephyr-7B LLM hosted on Hugging Face to generate a final answer.

Technology Stack

Technology	Purpose
Python	Primary programming language used for backend logic.
Gradio	Provides a lightweight frontend interface for user interaction.
LangChain	Orchestrates document processing, chunking, and vector retrieval.
ChromaDB	Embedding vector store to support semantic search.
SentenceTransformers	Used to convert text and queries into embedding vectors.
Hugging Face Transformers	Hosts and serves the LLM (Zephyr-7B).
PyPDF2	Parses and extracts text from PDF documents.
Scikit-learn	Computes cosine similarity between embeddings.
dotenv	Handles environment variables securely.

Code Walkthrough

`app.py`

This is the main script that powers the Gradio interface and connects all backend functionalities:

- Loads the Hugging Face LLM endpoint and environment variables.
- Uses PyPDF2 to extract text from PDF uploads.
- Applies simple regex-based text chunking to break long documents into readable segments.
- Computes semantic similarity between the user query and each document chunk.
- Constructs a prompt that includes the most relevant chunks and the user's query.
- Sends the prompt to the Zephyr-7B language model and returns the answer.

`prepare_index.py`

This script is responsible for document pre-processing and vector indexing:

- Loads various file types (`.pdf`, `.txt`, `.csv`, `.docx`) using LangChain's loaders.
- Uses a CharacterTextSplitter to create overlapping text chunks that preserve context.
- Embeds each chunk using the MiniLM model from Sentence Transformers.
- Stores these embeddings persistently in ChromaDB for fast retrieval.

`retriever.py`

This module connects to the existing ChromaDB to retrieve document segments relevant to a query:

- Initializes the same embedding model used during indexing.
- Loads the persisted ChromaDB.
- Defines a `retrieve_context()` function to extract the top-K most relevant chunks.
- Outputs these chunks as a context string to be used in prompt construction.

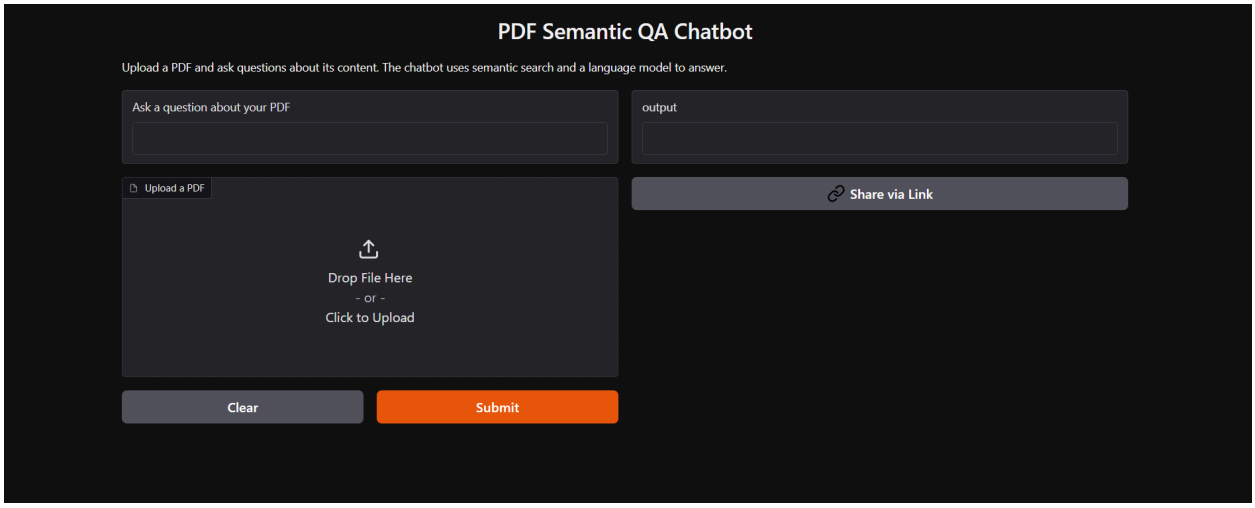
Working Workflow

1. User uploads a document and enters a query.
2. The document is parsed and chunked.
3. Each chunk is converted into a vector and stored.
4. The user query is embedded and compared to stored vectors.
5. Top matches are selected as the context.
6. A structured prompt is formed.
7. The prompt is sent to the LLM.
8. The response is returned to the user.

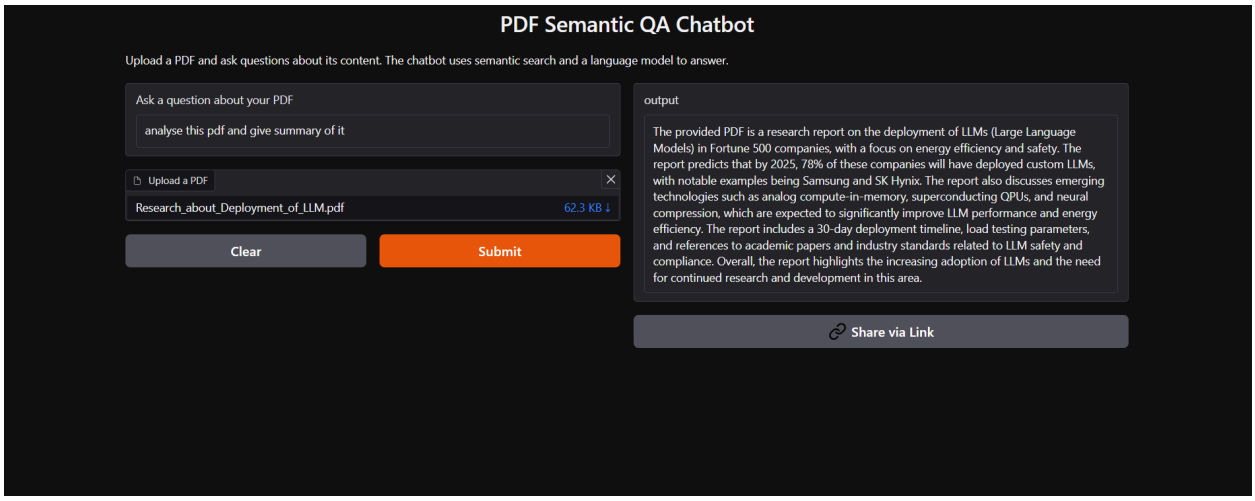
The system achieves a dynamic interaction loop between retrieval and generation, ensuring the output remains grounded in the source document.

Screenshots

- Gradio interface for PDF upload



- Sample query input and response



Conclusion

Chatbot_RAG is a modern, modular, and efficient application of the Retrieval-Augmented Generation paradigm. By combining the best practices from document processing, semantic search, and natural language generation, this system showcases the power of AI in improving information access and decision-making. It provides an excellent base for both research and production deployment in education, law, customer support, and beyond.

References

- Hugging Face Transformers: https://huggingface.co
- LangChain Documentation: https://docs.langchain.com
- Sentence Transformers: https://www.sbert.net
- ChromaDB: https://www.trychroma.com
- Gradio: https://www.gradio.app
- Python Requests: https://docs.python-requests.org
- PyPDF2: https://pypi.org/project/PyPDF2