Document Insight Assistant

Natural Language Query System for PDF Documents

Name: Vaidehi Patil
Class: CS 6320.001
Date: 6th May 2025.

Links

GitHub Repository: https://github.com/vaidehipatill/doc_reader_nlp

YouTube Video: https://www.youtube.com/watch?v=MWHp99dzJZc

Introduction

This project, titled Document Insight Assistant, presents a **Natural Language Query System** that allows users, students and researchers to upload academic/business/informational documents in PDF format and interactively query their content using natural language. By integrating OpenAI's language models and vector-based search with an interactive interface, the system provides precise context-aware answers to document-related queries.

The key objective of this system is to automate the process of extracting relevant information from lengthy business reports, replacing manual reading with an AI-powered question-answering interface. The application was built as part of an NLP course, with a focus on applying modern techniques such as embeddings, vector search, and retrieval-augmented generation (RAG).

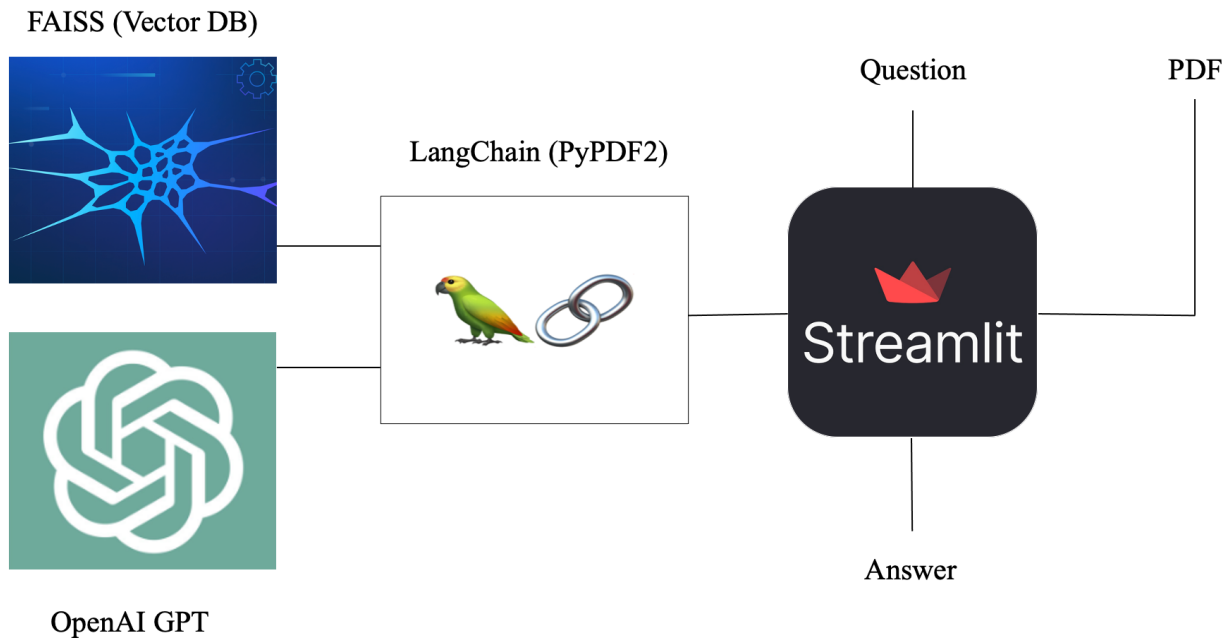
Technical Overview

- **Document Upload and Processing:** Users can upload PDF files, which are then converted into plain text, chunked into smaller sections, and converted into vector embeddings using OpenAI's embedding models.
- **Vector Indexing with FAISS:** The document chunks are stored in a FAISS index, enabling efficient similarity-based retrieval of relevant sections when a query is made.
- **Natural Language Query Handling:** When a user submits a question, the system retrieves the most relevant chunks from the FAISS index and passes them as context to an OpenAI language model (GPT) to generate a coherent, natural language response.
- **Frontend User Interface:** The user interface was developed using **Streamlit**, providing features such as document upload, a chat window, and chat history tracking.

Compared to a basic chatbot or QA system, this project demonstrates several enhancements:

- Integrated document ingestion from user-uploaded PDFs instead of relying on static datasets.
- Implemented vector search indexing using FAISS, enabling semantic search over document chunks.
- Applied retrieval-augmented generation (RAG) by fetching relevant document pieces to use as context when querying OpenAI's language model.
- Added chat history tracking to simulate an ongoing conversation and improve useability.

Technical Stack



Lessons Learned

- **APIs Evolve:** A key challenge encountered was the breaking change in the OpenAI Python API, which initially supported certain arguments (like `proxies`) that are no longer valid. This required adjustment and better version control management.
- **Retrieval Optimization:** Correctly chunking large PDFs into meaningful sections significantly improves retrieval quality. Very small chunks miss context; very large chunks dilute relevance.
- **Embedding Costs:** Using OpenAI embeddings for every document chunk adds API costs and latency. A potential improvement would be implementing local embeddings using models like Sentence-BERT.
- **Scalability:** While the system works well with small to medium documents, scaling it to hundreds of files would require persisting indexes and optimizing memory usage.

- **Session Management**: Leveraging Streamlit's session state allowed for persistent chat memory, enhancing the user experience for conversational AI applications.
- **Modular Design**: Separating concerns across distinct scripts simplified debugging and future enhancements.

Complexity Highlights

- **FAISS Indexing**: Managing large-scale document embeddings required careful memory and performance tuning.
- **API Version Control**: Upgrading the codebase to align with OpenAI's latest library changes resolved several compatibility issues.
- **Chunk Retrieval Tuning**: Iterative testing and tuning of chunk sizes, overlap, and vector search parameters significantly improved system accuracy.

Contributions

Vaidehi Patil:

- Designed and implemented the overall system architecture.
- Developed the document ingestion and processing pipeline using PyPDF2 and LangChain.
- Integrated OpenAI's embedding model and FAISS indexing for retrieval.
- Built the Streamlit frontend interface for uploading PDFs and querying documents.
- Implemented chat history tracking within the Streamlit app.
- Modularized the project into separate Python files for improved organization.
- Conducted testing using sample business reports to verify retrieval accuracy.
- Added functionality to download chat transcripts as text files.

Self-Scoring

Vaidehi Patil:

- **70 points –** significant exploration beyond baseline: The project implemented vector search and retrieval, but didn't go deep into advanced fine-tuning or custom model training.
- **20 points –** innovation and creativity: Using FAISS with RAG is effective but not novel; local embedding models could have raised this further.
- **7 points –** complexity: Faced real API issues and handled chunking strategy decisions.
- **8 points –** lessons learned and improvements: Identified real-world improvement areas like scalability and cost reduction.

Potential Improvements

- Implement local embeddings using Sentence-BERT to reduce API costs.
- Add multi-document querying and persistent indexing for scalability.
- Enhance the system with summarization capabilities to complement Q&A.
- Provide citation mapping to show which document section supports each answer

---

Conclusion

This project successfully demonstrates a working prototype of a natural language query system for business documents. It combines document processing, vector indexing, and retrieval-augmented generation in a user-friendly web application. While the system performs well on small datasets, scaling it for enterprise-level document management would require addressing indexing persistence, embedding cost optimization, and enhanced retrieval accuracy.
