Case Study: Context-Aware Document Assistant

By Syed Armghan Ahmad – syedarmghanahmad.work@gmail.com

Linkedin - GitHub - Repo for this project

Overview

As a self-taught developer passionate about AI and natural language processing, I built the **Context-Aware Document Assistant**, a Retrieval-Augmented Generation (RAG) application that enables users to query large documents (PDFs or websites) using natural language. This project, developed through self-learning, integrates advanced AI tools like LangChain, HuggingFace, FAISS, and GroqLLM, with a Streamlit frontend for user interaction and a FastAPI backend for programmatic access. The application demonstrates my ability to design and implement a full-stack AI solution, addressing real-world challenges in document processing and information retrieval.

Problem Statement

In today's information-rich world, extracting specific insights from large documents, such as research papers, legal contracts, or websites, is time-consuming and error-prone. Traditional search methods often fail to provide context-aware answers, and users struggle to navigate multiple document types efficiently. My goal was to create a tool that:

- Allows users to ask questions in natural language and receive accurate, context-grounded responses.
- Supports both PDFs and website content with seamless context switching.
- Provides a user-friendly interface and a scalable API for diverse use cases.

Approach

To address this challenge, I designed a modular RAG-based system that combines document retrieval with AI-generated responses. The project consists of two main components: a Streamlit-based frontend for interactive use and a FastAPI-based backend for API access.

Key Features

- **Hybrid Document Contexts**: Users can upload PDFs or input website URLs, with the system maintaining separate vector stores for each context to ensure accurate retrieval.
- **Smart Document Processing**: Documents are split into configurable chunks (default: 1000 characters, 200 overlap), converted to embeddings using HuggingFace, and indexed in FAISS for efficient similarity search.
- **AI-Powered Responses**: Leverages GroqLLM models (e.g., Mixtral-8x7B) via LangChain to generate answers based on retrieved document chunks.
- **User-Friendly Interface**: Streamlit provides an intuitive UI with real-time chat, document upload, and settings for chunk size, overlap, and model selection.
- **Scalable API**: FastAPI endpoints enable programmatic document processing, querying, and chat history retrieval, suitable for integration into larger systems.
- **Error Handling and Feedback**: Robust error handling ensures graceful recovery from issues like invalid URLs or file uploads, with clear user feedback.

Technical Implementation

- **Frontend (Streamlit)**:
  - Built an interactive UI using Streamlit, with a sidebar for configuring input type (PDFs or websites), chunking parameters, and model selection.
  - Used `session_state` to manage chat history, processed documents, and vector stores in-memory.
  - Implemented document processing with `PyPDFLoader` for PDFs and `WebBaseLoader` for websites, splitting text with `RecursiveCharacterTextSplitter`.
  - Integrated LangChain's RAG pipeline, combining FAISS-based retrieval with GroqLLM's generation capabilities.
- **Backend (FastAPI)**:
  - Developed RESTful API endpoints (`/process-documents`, `/query`, `/chat-history`, `/health`) using FastAPI.
  - Used Pydantic models for request validation and a global state dictionary for simplicity (noted for replacement with a database in production).
  - Mirrored the frontend's RAG pipeline, ensuring consistency in document processing and query handling.
- **Core Technologies**:
  - **LangChain**: Simplified the RAG pipeline with modular chains (`create_stuff_documents_chain`, `create_retrieval_chain`).
  - **HuggingFace Embeddings**: Generated semantic embeddings for document chunks, enabling accurate retrieval.
  - **FAISS**: Provided efficient vector storage and similarity search for large document sets.
  - **GroqLLM**: Powered context-aware responses via the Groq API.
  - **Streamlit/FastAPI**: Enabled rapid prototyping (Streamlit) and scalable API development (FastAPI).

Challenges and Solutions

- **Challenge**: Learning and integrating complex AI tools like LangChain and FAISS as a self-taught developer.
    - **Solution**: Leveraged official documentation, tutorials, and experimentation to understand RAG pipelines, embeddings, and vector stores.
- **Challenge**: Managing multiple document types without mixing contexts.
    - **Solution**: Designed separate FAISS indices (`pdf_vector`, `web_vector`) and a `current_context` tracker to ensure context-specific retrieval.
- **Challenge**: Optimizing document chunking for retrieval accuracy.
    - **Solution**: Experimented with chunk sizes and overlaps, settling on defaults that balanced context retention and performance.
- **Challenge**: Ensuring robust error handling for user inputs.
    - **Solution**: Implemented try-except blocks and user feedback (e.g., `st.error`, `HTTPException`) to handle invalid URLs, file uploads, or API issues.

Impact

The Context-Aware Document Assistant successfully enables users to query large documents efficiently, delivering accurate answers with source attribution (e.g., PDF page numbers or website URLs). Key outcomes include:

- **Enhanced Productivity**: Users can extract insights from documents without manual searching, saving time in scenarios like research or legal analysis.
- **Flexibility**: Supports diverse document types and customizable settings, making it adaptable to various use cases.
- **Skill Development**: Through self-learning, I gained expertise in RAG, vector embeddings, full-stack development, and API design, preparing me for real-world AI projects.
- **Portfolio Strength**: The project showcases my ability to build end-to-end AI applications, from front-end design to back-end scalability, making it a standout piece in my portfolio.

Lessons Learned

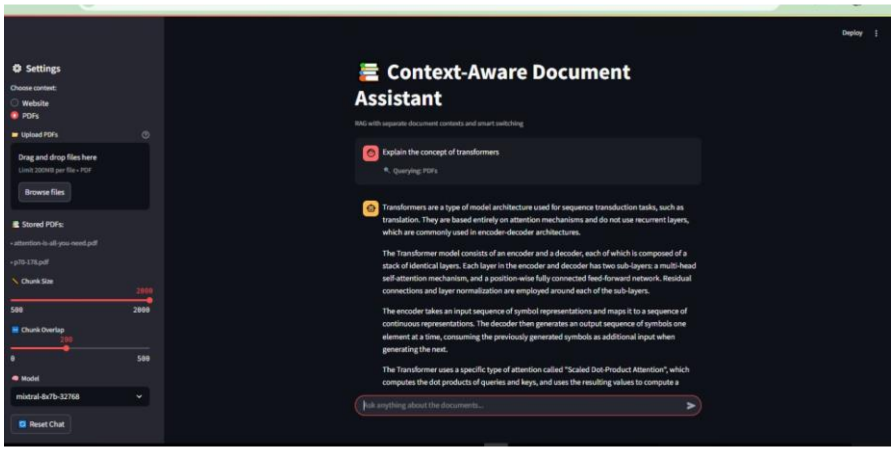
- **Modular Design**: Structuring the code into reusable functions (e.g., `process_documents`, `create_chain`) improved maintainability and extensibility.
- **Iterative Learning**: Experimentation with chunking parameters and model selection taught me the importance of tuning AI systems for specific tasks.
- **User-Centric Approach**: Prioritizing clear feedback and intuitive UI design enhanced the user experience.
- **Production Readiness**: In-memory state management works for prototyping but requires a database for scalability, informing future projects.

Future Enhancements

To productionize the application, I would:

- Replace in-memory state with a database (e.g., PostgreSQL) for persistent storage of chat history and document metadata.
- Use a managed vector database (e.g., Pinecone) for scalable, distributed vector storage.
- Implement async document processing with Celery and Redis to handle large document sets.
- Add user authentication and rate limiting to secure the FastAPI backend.
- Explore smaller models (e.g., DistilBERT) for resource-constrained environments.

Screenshots:



Choose context:
Website [✓ PDFs]

Stored PDFs:
• attention-is-all-you-need.pdf
• p70-178.pdf

Chunk Size: 2000 (500–2000)
Chunk Overlap: 200 (0–500)
Model: mixtral-8x7b-32768

Context-Aware Document Assistant

RAG with separate document contexts and smart switching

Explain the concept of transformers

Querying: PDFs

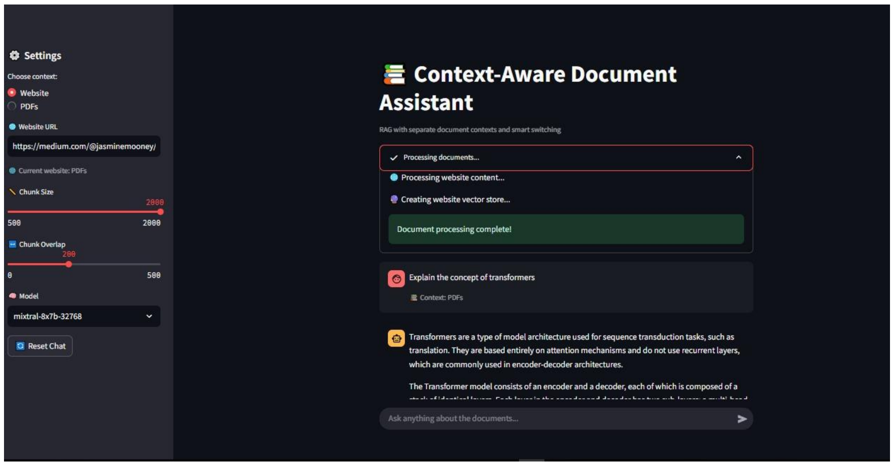
Transformers are a type of model architecture used for sequence transduction tasks, such as translation. They are based entirely on attention mechanisms and do not use recurrent layers, which are commonly used in encoder-decoder architectures.

The Transformer model consists of an encoder and a decoder, each of which is composed of a stack of identical layers. Each layer in the encoder and decoder has two sub-layers: a multi-head self-attention mechanism, and a position-wise fully connected feed-forward network. Residual connections and layer normalization are employed around each of the sub-layers.

The encoder takes an input sequence of symbol representations and maps it to a sequence of continuous representations. The decoder then generates an output sequence of symbols one element at a time, consuming the previously generated symbols as additional input when generating the next.

The Transformer uses a specific type of attention called "Scaled Dot-Product Attention", which computes the dot products of queries and keys, and uses the resulting values to compute a

Ask anything about the documents...



Choose context:
[✓ Website] PDFs

Website URL: https://medium.com/@jasminemooney/

Current website: PDFs

Chunk Size: 2000 (500–2000)
Chunk Overlap: 200 (0–500)
Model: mixtral-8x7b-32768

Context-Aware Document Assistant

RAG with separate document contexts and smart switching

✓ Processing documents...

Processing website content...

Creating website vector store...

Document processing complete!

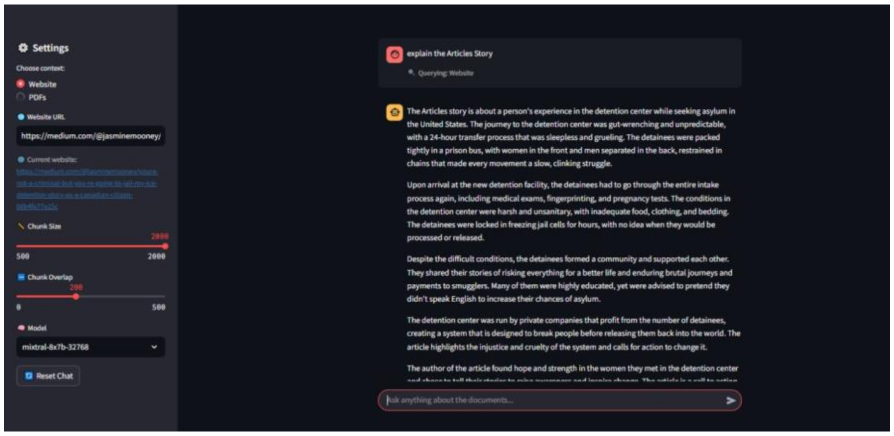
Explain the concept of transformers

Context: PDFs

Transformers are a type of model architecture used for sequence transduction tasks, such as translation. They are based entirely on attention mechanisms and do not use recurrent layers, which are commonly used in encoder-decoder architectures.

The Transformer model consists of an encoder and a decoder, each of which is composed of a

Ask anything about the documents...



Choose context:
[✓ Website] PDFs

Website URL: https://medium.com/@jasminemooney/

Current website: https://medium.com/@jasminemooney/youre-not-a-criminal-but-you-re-going-to-jail-my-ice-detention-story-as-a-canadian-citizen-[illegible]

Chunk Size: 2000 (500–2000)
Chunk Overlap: 200 (0–500)
Model: mixtral-8x7b-32768

explain the Articles Story

Querying: Website

The Articles story is about a person's experience in the detention center while seeking asylum in the United States. The journey to the detention center was gut-wrenching and unpredictable, with a 24-hour transfer process that was sleepless and grueling. The detainees were packed tightly in a prison bus, with women in the front and men separated in the back, restrained in chains that made every movement a slow, clinking struggle.

Upon arrival at the new detention facility, the detainees had to go through the entire intake process again, including medical exams, fingerprinting, and pregnancy tests. The conditions in the detention center were harsh and unsanitary, with inadequate food, clothing, and bedding. The detainees were locked in freezing jail cells for hours, with no idea when they would be processed or released.

Despite the difficult conditions, the detainees formed a community and supported each other. They shared their stories of risking everything for a better life and enduring brutal journeys and payments to smugglers. Many of them were highly educated, yet were advised to pretend they didn't speak English to increase their chances of asylum.

The detention center was run by private companies that profit from the number of detainees, creating a system that is designed to break people before releasing them back into the world. The article highlights the injustice and cruelty of the system and calls for action to change it.

The author of the article found hope and strength in the women they met in the detention center

Ask anything about the documents...

Conclusion

The Context-Aware Document Assistant is a testament to my self-learning journey and passion for AI-driven solutions. By tackling a real-world problem with a sophisticated tech stack, I demonstrated my ability to design, implement, and iterate on a full-stack AI application. This project not only solved a practical challenge but also equipped me with skills in RAG, vector search, and web development, preparing me for impactful contributions in professional settings.