Abstract

This is a RAG application in which you can ask questions about the PDF. It also retains the current chat in memory, so you can ask questions continuously in relation to previous questions. There is also an evaluation file that evaluates the answers generated by our RAG application. We use raga for evaluation purposes.

Methodology

RAG Application:

This Streamlit application allows users to upload multiple PDF documents and interact with them through a conversational interface. The application processes the PDFs, extracts text, creates text chunks, generates embeddings, and uses a conversational retrieval chain to respond to user queries based on the content of the PDFs.

The function get_pdf_text extracts text from each page of the uploaded PDFs. The get_text_chunks function splits this text into manageable chunks. The get_vectorstore function generates embeddings for the text chunks and stores them in a FAISS vector store. The get_conversation_chain function initializes a conversational retrieval chain with an LLM from HuggingFace and a memory buffer for chat history. The handle_userinput function manages user input by querying the conversation chain and updating the chat history.

The main function sets up the Streamlit app, loads environment variables, initializes session state variables, displays the interface for user interaction, handles PDF uploads, processes the PDFs, and sets up the conversational retrieval chain.

This application demonstrates a practical implementation of document-based conversational AI, utilizing PDF text extraction, text chunking, embeddings, and conversational retrieval chains for user interactions.

Thought Process

- Streamlit for Interface: Using Streamlit allows for an easy and interactive web-based interface for users to upload PDFs and ask questions.

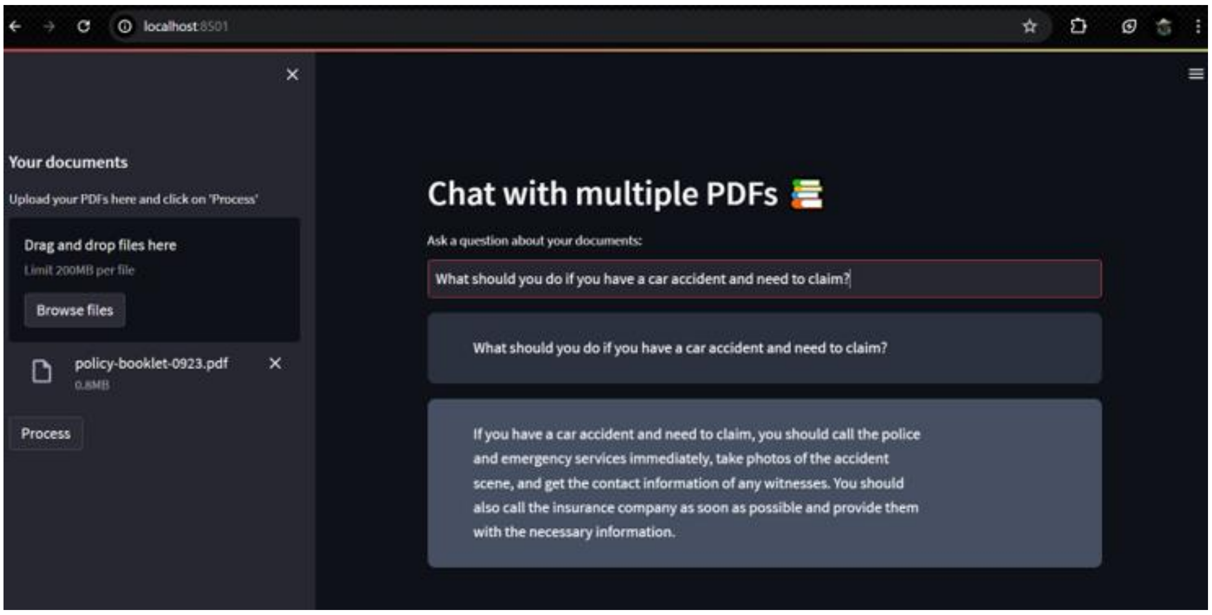
- PDF Text Extraction: PyPDF2 is utilized for robust and straightforward text extraction from PDFs.

- Text Chunking: Splitting text into chunks improves manageability and performance during embedding and retrieval.

- Embeddings and Vector Store: Embedding the text chunks and storing them in a vector store (FAISS) enables efficient and fast similarity searches.

- Conversational Chain: Using a conversational retrieval chain with an LLM and memory buffer allows for dynamic and context-aware responses to user queries.

- User Interaction Handling: Efficient handling of user inputs and updating the chat interface ensures a smooth user experience.



What should you do if you have a car accident and need to claim?

What should you do if you have a car accident and need to claim?

If you have a car accident and need to claim, you should call the police and emergency services immediately, take photos of the accident scene, and get the contact information of any witnesses. You should also call the insurance company as soon as possible and provide them with the necessary information.

Evaluation:

RAGAS:

RAGAS is chosen for evaluating the question-answering performance of the PDF processing application because of its comprehensive and precise metrics, which include:

- Faithfulness: Ensures the generated answers are factually correct and aligned with the original document content, reducing the risk of misleading or incorrect responses.

- Answer Relevancy: Evaluates the degree to which the answers directly address the questions, ensuring the responses are pertinent and useful.

- Context Recall: Assesses how much of the relevant information from the documents is captured and used to generate the answer, ensuring thoroughness and completeness.

- Context Precision: Measures the accuracy of the retrieved context, ensuring that it contains only the necessary information without extraneous details, thus maintaining focus and relevance.

Code:

The evaluation code is designed to assess the performance of a retrieval-augmented generation (RAG) model in answering questions based on PDF documents. Here's a brief explanation of the main components:

- get_pdf_text: Extracts text from a PDF document.

- get_text_chunks: Splits the extracted text into smaller chunks for processing.

- get_vectorstore: Converts text chunks into embeddings and stores them in a FAISS vector store.

- process_pdfs: Reads a PDF, extracts text chunks from it, and stores embeddings.

- evaluate_question:

- Sets up a retrieval and generation pipeline using a prompt template and an LLM.
- Defines questions and ground truth answers.
- Uses the pipeline to generate answers and contexts.
- Creates a dataset with questions, answers, contexts, and ground truths.
- Evaluates the answers using RAGAS metrics: faithfulness, answer relevancy, context recall, and context precision.
- Outputs evaluation results in a pandas DataFrame.

This process ensures a thorough assessment of how well the model retrieves and generates relevant and accurate responses based on the provided PDF content.

Thought Process

- PDF Text Extraction: PyPDF2 is utilized for robust and straightforward text extraction from PDFs.
- Text Chunking: Splitting text into chunks improves manageability and performance during embedding and retrieval.
- Embeddings and Vector Store: Embedding the text chunks and storing them in a vector store (FAISS) enables efficient and fast similarity searches.
- Retrieval-Augmented Generation (RAG): Using a RAG chain with a language model and retriever allows for dynamic and context-aware responses to user queries.
- Safe and Unbiased Responses: The prompt template ensures that the language model provides safe, unbiased, and factually correct answers.
- RAGAS Evaluation: RAGAS is chosen for its comprehensive evaluation metrics, which include context precision, context recall, faithfulness, and answer relevancy. This ensures a thorough assessment of the model's performance in generating accurate and relevant answers based on the provided documents.

Dataset:

Dataset was made by reading the pdf and making questions based on almost each topic and writing down its ground truth.

Methods to Increase Performance

- Reranking: Reranked results but found no significant improvement in accuracy for all queries.
- Embedding Models: Tested different embedding models from Hugging Face. Most models were either slower or performed worse than the current model. The Ollama embedding model was chosen for its balance of performance and speed.
- Language Models (LLMs): Experimented with several LLMs, including Gemma 2B, Gemma 7B, T5, and Llama3-7B. The T5 model provided better output but was slower. The Llama3-70B model using Groq was the fastest and produced the best results but had API limitations. Settled on the Llama3-7B model for its good performance and reasonable speed.
- Vector Databases: Tried various free vector databases but found FAISS to be the fastest and most efficient.

- Context Chunks: Experimented with different numbers of context chunks. Found that using 5 context chunks provided the best results.

- Parameter Tuning: Adjusted parameters such as chunk size and overlap during text splitting to optimize the balance between context completeness and relevance.

- Prediction-based Improvements: Future enhancements might include incorporating more advanced reranking techniques and hybrid models combining multiple embeddings to further enhance performance. Additionally, experimenting with multi-modal models that integrate visual features from the PDF pages could improve context understanding and answer generation.