Project Synopsis: Insurance Document Q&A (LLaMA 3.2)

1. Introduction

This project details the development of an "Insurance Document Q&A Bot," a sophisticated application designed to assist users in quickly retrieving information from insurance policy documents. Leveraging the power of Large Language Models (LLMs), specifically LLaMA 3.2, and Retrieval-Augmented Generation (RAG) techniques, the bot allows users to upload PDF insurance documents and then ask natural language questions about their content. The system processes these documents, creates a searchable knowledge base, and provides accurate answers sourced directly from the uploaded information.

2. Objectives

The primary objectives of this project are:

- To enable users to upload and process PDF-based insurance documents.
- To build a robust knowledge retrieval system that can efficiently index and store information from PDF documents.
- To develop a question-answering interface that allows users to query the indexed documents using natural language.
- To utilize the LLaMA 3.2 model for understanding user queries and generating relevant answers.
- To provide a user-friendly Streamlit interface for seamless interaction.

3. Methodology

The project is structured into three main Python components, each handling a specific part of the RAG pipeline: vector_store.py, qa_chain.py, and app.py.

3.1. vector_store.py (Document Processing and Indexing)

This module is responsible for loading and preparing the PDF document for retrieval.

- **PDF Loading:** It uses PyPDFLoader to load the content of the uploaded PDF document, which is then split into smaller, manageable pages or chunks. This segmentation is crucial for efficient processing and retrieval.
- **Embedding Generation:** OllamaEmbeddings (using the "mxbai-embed-large" model) converts the text content of each page into numerical vector representations (embeddings). These embeddings capture the semantic meaning of the text.
- **Vector Store Creation:** The generated embeddings and their corresponding document chunks are stored in a Chroma vector database. This database

(chroma_langchain_db) acts as a searchable index, allowing for quick similarity searches based on vector proximity. The persist_directory argument ensures that the vector store can be saved and reloaded.

3.2. qa_chain.py (Question Answering Chain)

This module orchestrates the retrieval and generation process to answer user queries.

- **LLM Integration:** It initializes an OllamaLLM instance, specifically loading the "llama3.2" model. This LLM is responsible for understanding the retrieved context and generating human-like answers.
- **Embedding Function:** It also uses OllamaEmbeddings ("mxbai-embed-large") to generate embeddings for the user's query, ensuring consistency with the document embeddings stored in the vector database.
- **Vector Database Loading:** It loads the pre-existing Chroma vector database from the specified persist_directory, which contains the indexed insurance document content.
- **Retriever Setup:** The vector database is configured as a retriever. When a query is received, this retriever will search the vector database for the most semantically similar document chunks.
- **RetrievalQA Chain:** A RetrievalQA chain is constructed. This chain combines the retriever (for finding relevant document chunks) and the LLM (for generating an answer based on those chunks). The chain_type="stuff" indicates that all retrieved document chunks will be "stuffed" into the LLM's context window. return_source_documents=True ensures that the original source content for the answer can also be displayed.

3.3. app.py (Streamlit User Interface)

This is the main application file that provides the graphical user interface for the bot.

- **Streamlit Setup:** Sets up the Streamlit application with a title and page configuration.
- **PDF Upload:** Provides a file uploader widget (st.file_uploader) for users to upload their insurance PDF documents. The uploaded file is temporarily saved as temp.pdf.
- **PDF Processing Trigger:** A "Process PDF" button initiates the document indexing process. Upon clicking:
    - It clears any existing Chroma database to ensure fresh indexing.
    - Calls create_vector_store from vector_store.py to process the uploaded PDF and create the vector database.
    - Initializes the qa_chain by calling load_qa_chain from qa_chain.py and stores it in Streamlit's session state for persistence across interactions.

- **Question Input and Response:** Once the PDF is processed and the QA chain is loaded:
  - A text input field (st.text_input) appears for users to type their questions.
  - When a query is entered, the qa_chain.invoke() method is called to get the answer.
  - The bot's answer (response['result']) is then displayed to the user.
- **User Feedback:** Incorporates visual feedback mechanisms like success messages, spinners for ongoing processes, and clear answer display.

4. Conclusion

This Insurance Document Q&A Bot project successfully demonstrates a practical application of LLMs and RAG for extracting specific information from large, unstructured documents like insurance policies. By combining efficient document indexing with the generative capabilities of LLaMA 3.2, the bot offers a powerful tool for quick information retrieval, significantly reducing the manual effort required to sift through complex documents. The modular design, leveraging LangChain components, ensures maintainability and scalability, making it a robust solution for similar information extraction tasks.