**Clinical Trial Chatbot with Azure OpenAI and Azure Search**

**Overview**

This project allows users to interact with a PDF document (e.g., a clinical trial protocol or other large documents) through a chatbot interface. The system leverages Azure Cognitive Search for indexing and semantic search of document contents and uses Azure OpenAI's GPT-3.5-turbo model to provide intelligent, context-aware responses. This solution is useful for extracting valuable insights from extensive documents such as clinical trial protocols, research papers, regulatory documentation, and more.

**Use Case**

The main use case for this project is to enable researchers, medical professionals, or anyone working with large PDF documents (such as clinical trial protocols, research reports, or medical guidelines) to interact with the document using natural language queries. This chatbot can extract contextually relevant information from the document and provide intelligent answers.

**Specific Use Cases Include:**

* **Clinical Trials**: Medical professionals or researchers can interact with clinical trial protocols to quickly retrieve specific information about trial design, inclusion/exclusion criteria, methodology, and other important sections of the document.
* **Research Papers**: Academic researchers can use the chatbot to navigate lengthy research papers, extracting specific findings, methodologies, and results without having to read the entire document.
* **Medical Guidelines**: Healthcare providers can query medical guidelines and quickly retrieve key information for patient care and treatment decisions.
* **Legal and Regulatory Documents**: Legal professionals can query complex legal documents for specific sections or clauses, streamlining the review process.

**How It Works**

**Step-by-Step Flow of the Application**

1. **PDF Text Extraction**:
   * The first step involves extracting text from the PDF file using the PyPDF2 library.
   * Each page of the PDF is read and its text is extracted. If a page is empty or contains non-textual content (like images or tables), it is skipped.
2. **Text Chunking**:
   * The extracted text is split into smaller chunks using langchain.text\_splitter (specifically, the RecursiveCharacterTextSplitter).
   * This chunking ensures that the content is manageable for both Azure Cognitive Search indexing and OpenAI embeddings.
3. **Creating Azure Search Index**:
   * The project checks whether an Azure Cognitive Search index already exists for the document.
   * If the index doesn't exist, it creates a new index with fields such as id, fileName, content, and contentEmbeddings.
   * The contentEmbeddings field stores vectorized embeddings that allow semantic search.
4. **Generating Embeddings for Text Chunks**:
   * For each chunk of text, an embedding is generated using the text-embedding-ada-002 model from Azure OpenAI.
   * These embeddings represent the semantic meaning of the text, enabling efficient vector search.
5. **Indexing Text Chunks in Azure Search**:
   * Once embeddings are generated, each chunk of text (along with its embeddings) is indexed in Azure Cognitive Search.
   * The indexed documents are stored in Azure Search, which allows for fast and scalable querying.
6. **Handling User Queries**:
   * When the user inputs a query, the system generates an embedding for the query using the same model (text-embedding-ada-002).
   * This embedding is used to perform a vector search against the indexed document chunks to find the most relevant content based on the semantic meaning of the query.
7. **Generating Responses Using GPT-3.5-turbo**:
   * After retrieving relevant document chunks from the search index, a system prompt is created that includes the context from the search results.
   * This context is passed along with the user’s question to OpenAI’s GPT-3.5-turbo model to generate a response.
   * The model processes the information and provides an answer based on the content retrieved from the document.
8. **Chat Interface**:
   * The Streamlit application provides a user-friendly interface for interacting with the system.
   * Users can enter queries, view the chat history, and see answers generated by GPT-3.5-turbo.
   * The chat history is stored in the session state, allowing users to see the progression of their conversation.

**Technical Components and Libraries Used**

* **Streamlit**: For building the interactive web interface.
* **PyPDF2**: For extracting text from the PDF document.
* **LangChain**: For chunking the text into smaller pieces for efficient indexing and searching.
* **Azure Cognitive Search**: For storing and searching document content using vector search.
* **Azure OpenAI (GPT-3.5-turbo)**: For generating embeddings and providing intelligent answers to user queries.
* **Python-dotenv**: For managing environment variables securely.

**Code Breakdown**

* **extract\_text\_from\_pdf(pdf\_path)**:
  + Extracts all the textual content from the provided PDF file using the PyPDF2.PdfReader.
* **get\_text\_chunks(pdf\_text)**:
  + Uses LangChain's RecursiveCharacterTextSplitter to split the extracted text into chunks of specified size and overlap.
  + This is done to break down the document into digestible pieces that can be indexed and queried more efficiently.
* **create\_index()**:
  + Checks if the Azure Cognitive Search index already exists, and if not, it creates a new index with fields like id, fileName, content, and contentEmbeddings.
  + It also sets up vector search with HnswAlgorithmConfiguration for efficient similarity searches.
* **index\_documents\_to\_azure\_search(text\_chunks)**:
  + Takes the text chunks and uploads them to the Azure Search index.
  + For each chunk, it generates embeddings using the text-embedding-ada-002 model from Azure OpenAI and stores them in the contentEmbeddings field.
* **search\_embeddings(user\_question)**:
  + Converts the user's query into an embedding and performs a semantic search against the indexed document chunks in Azure Search.
  + Retrieves the most relevant chunks based on the similarity of the embeddings.
* **get\_chat\_response(user\_question)**:
  + Combines the search results (relevant document chunks) with the user’s query and sends them to OpenAI’s GPT-3.5-turbo model to generate a response.
* **main()**:
  + The main function that sets up the Streamlit interface, processes the PDF document, and handles user input/output.

**Expected Outcome**

* **Interactive Chatbot**: Users can ask questions related to the content of the provided PDF document (e.g., clinical trial protocol), and the chatbot will return relevant, context-aware answers.
* **Contextual Answers**: Responses are generated using GPT-3.5-turbo, with the context drawn from the document's content.
* **Efficient Document Navigation**: Instead of manually searching through lengthy documents, users can rely on semantic search to find and extract specific information quickly.
* **PDF Indexing and Search**: The project creates an index for the PDF document and stores both textual content and embeddings in Azure Cognitive Search, making it searchable and accessible for future queries.

**Benefits of the Project**

* **Increased Efficiency**: Users can easily retrieve specific information from large documents without having to read them in full.
* **Semantic Search**: Unlike keyword-based search, this solution uses embeddings for semantic search, which allows for better understanding of the content.
* **Scalable**: The system is built using Azure services, meaning it can scale to handle multiple documents and large datasets.
* **Easy to Use**: The Streamlit interface makes it simple for users to interact with the system, even without technical knowledge.
* **Supports Multiple Domains**: While designed for clinical trials, this solution can be adapted to work with any kind of structured document (research papers, regulatory guidelines, etc.).

**Outcome**

The outcome of this project is a fully functional chatbot interface that can interact with a large PDF document, extract meaningful information, and provide intelligent, context-aware responses. This is useful for anyone working with dense documents like clinical trial protocols, research papers, or medical guidelines, and it can save time by enabling users to ask specific questions and get precise answers without sifting through the entire document manually.

**Conclusion**

This project demonstrates how to integrate multiple Azure services (Azure Cognitive Search and Azure OpenAI) to create a powerful solution for interacting with and extracting information from large PDF documents. It can be applied to various domains, including healthcare, research, legal, and more, helping users quickly access critical information from lengthy and complex documents.

**Future Enhancements**

* **Expand to Multiple PDF Files**: The system currently works with a single PDF file. Future improvements could allow users to upload and interact with multiple documents.
* **Support for Other File Formats**: In the future, this can be extended to support other formats such as DOCX, HTML, or plain text.
* **Real-time Updates**: Ability to index new documents in real-time as they are uploaded or updated.