Project Report: Research Paper Chatbot Using Langchain, Pinecone and OPEN AI

Team Members: Sneha Sasanapuri, Prasanna Poluru

---

1. Introduction

1.1 Objective

The project aims to create a chat bot that enables users to upload research papers in PDF format and ask questions about their content. The chat bot processes the document, indexes its content for efficient retrieval, and uses a large language model, GPT-3.5-turbo to answer user queries based on the document context.

1.2 Motivation

Research papers are often dense and complex, making it time-consuming to extract relevant information. A research paper chat bot simplifies this process by:

- Allowing natural language queries.
- Providing targeted answers based on the document content.
- Saving time for researchers and students.

1.3 Scope

The chat bot supports:

- Uploading PDF documents.
- Extracting and indexing content using a hybrid search approach (dense and sparse vectors).
- Answering questions with the context retrieved from the document.
- A user-friendly interface powered by Streamlit.

2. System Architecture

2.1 Overview

The system consists of:

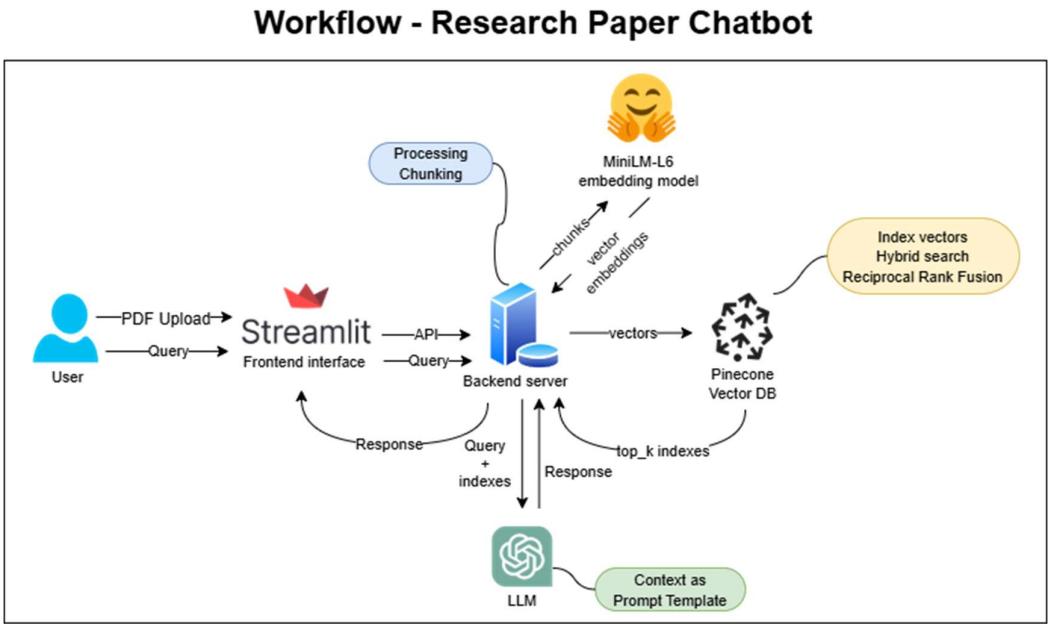
- **Frontend**: A Streamlit application for uploading files and submitting questions.
- **Backend**: Handles PDF processing, indexing, and query answering using Pinecone and OpenAI GPT APIs.

2.2 Workflow

1. The user uploads a PDF research paper.
2. The backend extracts text from the PDF and splits it into smaller chunks.
3. The chunks are indexed in Pinecone using both dense and sparse vector representations.
4. The user submits a query and chatbot retrieves relevant chunks from Pinecone and uses GPT to generate an answer.

2.3 System Diagram

The user uploads a PDF research paper and submits a query through the Streamlit frontend interface, which forwards the request to the backend server. The backend server processes the PDF by extracting its text, splitting it into manageable chunks, and sends these chunks to embedding model, then indexing these chunks in Pinecone using dense vector embedding (for semantic similarity) and sparse encodings (for keyword-based matching). When the user submits a query, the backend server retrieves the most relevant chunks from Pinecone using hybrid search, combining semantic similarity and keyword relevance, and passes these chunks along with the query to a GPT language model. The GPT model generates an answer based on the retrieved context, and the response is sent back to the backend server which sends back to Streamlit frontend, where it is displayed to the user.



3. Implementation

3.1 Technologies Used

- **Python**: Core programming language.
- **Streamlit**: User interface.
- **Pinecone**: Vector database for hybrid search.
- **HuggingFace**: Embedding generation.
- **OpenAI API**: GPT-3.5-turbo for natural language answers.
- **PyPDF2**: PDF text extraction.

4. Technical Details

4.1 Pinecone Indexing

- **Dense Vectors**:
    - Generated using Hugging Face's all-MiniLM-L6-v2 model.
    - Captures semantic meaning of sentences.

- **Sparse Vectors**:
  - Created using BM25Encoder for keyword relevance.
  - Complements dense vectors for hybrid search.
- **Metadata**:
  - Stores additional information such as the sentence context.

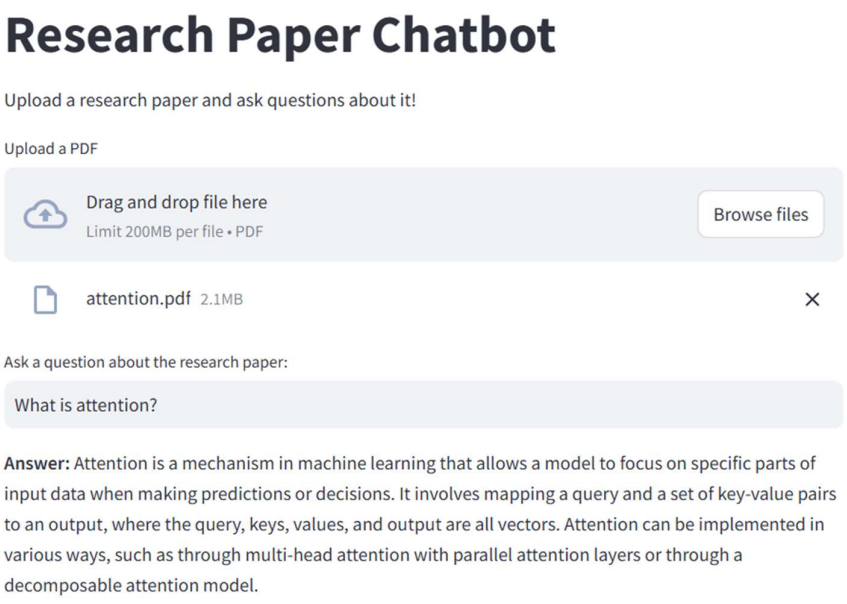
4.2 GPT Querying

- Constructs a prompt with:
  - Retrieved context from Pinecone.
  - The user's query.
- Sends the prompt to GPT-3.5-turbo for generating an answer.

5. Results

5.1 Example Query-1

- **Uploaded Paper**: Attention is all you need
- **User Query**: "What is attention?"
- **Pinecone indexes**:
- *Retrieved Docs: ['3.1 Encoder and Decoder Stacks\nEncoder: The encoder is composed of a stack of N= 6 identical layers.', 'Each position in the encoder can attend to all positions in the previous layer of the\nencoder.', 'To this end, we add "positional encodings" to the input embeddings at the\nbottoms of the encoder and decoder stacks.', 'In addition, we apply dropout to the sums of the embeddings and the\npositional encodings in both the encoder and decoder stacks.']*
- **Chat bot Response**:



5.2 Example Query-2

- **Uploaded Paper**: Cognitive Processes in the Breakfast Task
- **User Query**: "What is Shipley vocabulary?"

- **Chat bot Response**:

6. Challenges

The development of the research chatbot presents unique challenges related to processing, storing, and retrieving relevant information from research papers. These challenges stem from the nature of large and complex PDFs, the need for efficient storage and retrieval mechanisms, and ensuring the quality of the content passed to the large language model (LLM). This section highlights two significant challenges encountered during the project and their corresponding solutions.

6.1 Handling Large PDFs

Research papers are often extensive, containing many pages of dense, technical content. When these documents are uploaded, extracting and processing the entire content can lead to challenges. One primary issue arises from the token limit of LLMs like GPT-3.5/4, which restricts the number of tokens (words and symbols) that can be processed in a single query. This limitation makes it impossible to directly feed the entirety of a large document into the LLM, leading to the need for efficient content management strategies.

To address this, the text extracted from the PDF is split into smaller, manageable chunks, such as sentences or paragraphs, which are then indexed in Pinecone. This splitting ensures that individual chunks are small enough to fit within the token limits when retrieved. Additionally, when a query is made, only the most relevant chunks are retrieved based on semantic and keyword relevance. If the total retrieved content exceeds the token limit, further truncation is applied by prioritizing the highest-ranking chunks. This approach ensures that the chatbot can provide accurate answers without exceeding token constraints, while maintaining efficiency in retrieval and processing.

6.2 Redundancy in Retrieved Documents

When performing a hybrid search using Pinecone, multiple relevant chunks are retrieved for a single query. However, due to the overlapping nature of semantic and keyword-based indexing, the retrieved chunks may often include redundant content. This redundancy can confuse the LLM or lead to verbose, repetitive responses, reducing the overall quality of the chatbot's answers.

To mitigate this issue, a deduplication step is implemented in the backend. Before passing the retrieved context to the LLM, the content of the retrieved chunks is compared to identify and eliminate duplicates. Deduplication is performed by checking for identical or highly similar sentences, ensuring that only unique and diverse content is included in the context sent to the LLM. This not only enhances the clarity and relevance of the chatbot's responses but also optimizes the use of token space by removing unnecessary repetitions. As a result, the answers generated by the LLM are more concise, accurate, and directly relevant to the user's query.

By implementing text splitting and deduplication strategies, the challenges of handling large PDFs and redundant retrieved documents are effectively addressed. These solutions enhance the chatbot's ability to process large volumes of content and provide high-quality, focused responses, ensuring a seamless and efficient user experience.

7. Conclusion

This project demonstrates how modern NLP techniques, such as hybrid search and GPT-based querying, can simplify information extraction from research papers. By combining the power of vector databases with LLMs, we can provide an intuitive and efficient user experience.

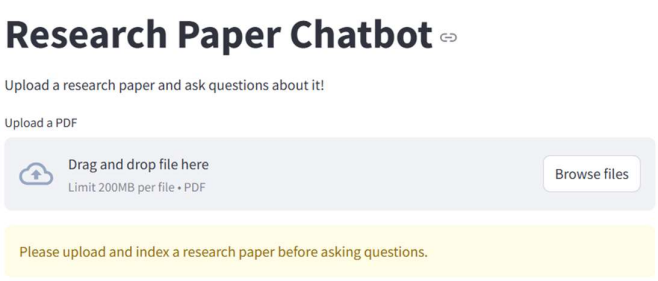
8. Accessing Chat bot

The Chabot is accessible through a Streamlit-based web application that facilitates seamless communication between the user and the backend processes. This section outlines the steps for interacting with the Chabot and accessing its features. To access the chat bot, following steps are required.

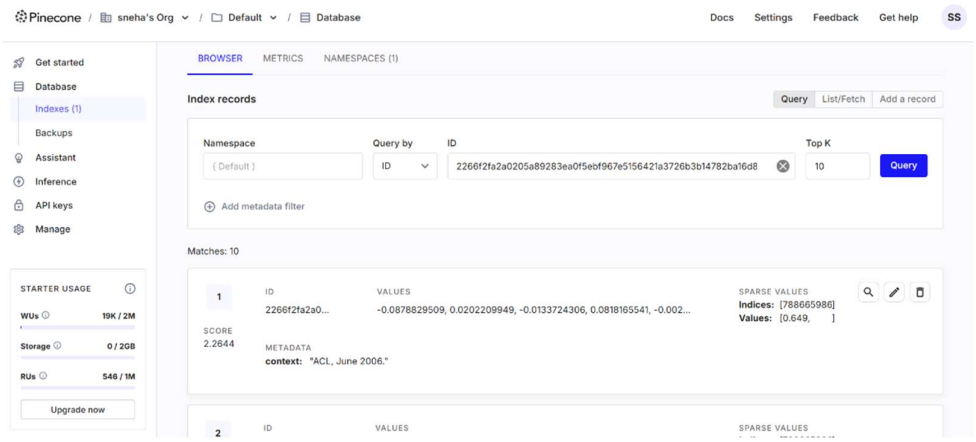
- API keys are required to run `*backend_app.py*` for Langchain, Open AI, Hugging Face, Pinecone function calls. These keys are to be loaded through `*.env*` file.
- Dependencies required to run this bot need to be installed from '*requirements.txt*' file. Following are the packages required-

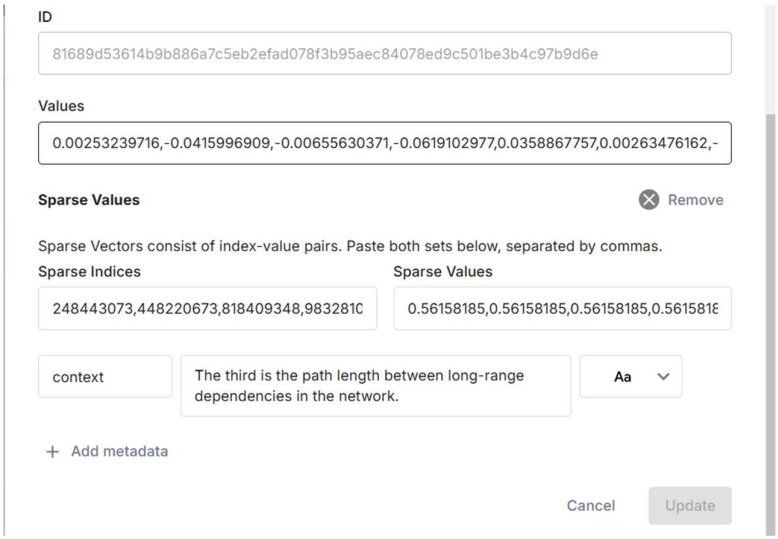
    - ✓ Python-dotenv
    - ✓ langchain-community
    - ✓ streamlit
    - ✓ langchain_openai
    - ✓ langchain_core
    - ✓ fastapi
    - ✓ uvicorn
    - ✓ sse_starlette
    - ✓ pypdf
    - ✓ PyPDF2
    - ✓ ipykernel
    - ✓ pinecone-client
    - ✓ pinecone-text
    - ✓ pinecone-notebooks
    - ✓ nltk

- Run the backend_app.py in Python environment using `*python backend_app.py*`
- Run streamlit application by starting Chabot on host PC with command `*streamlit run frontend_app.py*`. The terminal outputs local and Network URL. Access the bot at local URL.
- The landing page looks like this and prompts user to upload any research paper in pdf format.



- When the user uploads a PDF, indexes in form of vector embeddings will be stored in Pinecone vector store and its web UI appears as below.





- The bot's response is displayed directly on the Streamlit interface, providing a concise and accurate answer based on the context of the uploaded research paper. If no relevant information is found in the document, the bot politely informs the user, ensuring clarity and setting appropriate expectations.