Project Report

On

PDF Insighter: ChatBot for Document Intelligence

IE7945 –Master’s Project

Submitted by:

Mrudula Challagonda (002765266)

Term and Year: Spring 2024

Submitted to: Prof. Sivarit Sultornsanee

Submitted Date: April 25th, 2024

Table of Contents

1. Introduction

In our modern world, where information is abundant and constantly evolving, the ability to find the right piece of information from an existing document at the right time is invaluable. Whether it's for academic research, professional tasks, or personal endeavors, having access to relevant information quickly can make all the difference. This capability has been greatly enhanced by advancements in technology, which have revolutionized the way we store, access, and retrieve information.

One of the key advantages of finding information from documents promptly is the efficiency it brings to various tasks. For example, in a business setting, being able to quickly access relevant data from a report or a presentation can help executives make informed decisions swiftly. Similarly, in academic research, the ability to find relevant studies or articles can significantly expedite the research process and lead to more impactful findings.

Real-time technologies have played a crucial role in enabling this efficiency. For instance, search engines like Google have revolutionized information retrieval by providing instant access to a vast array of documents and resources. Tools like Google Scholar have made academic research more accessible by indexing scholarly articles and journals. Additionally, cloud storage services such as Google Drive and Dropbox have made it easier to store and access documents from anywhere, at any time.

Another benefit of finding information from documents promptly is the empowerment it provides to individuals. In today's digital age, people have access to a wealth of information at their fingertips, allowing them to learn new skills, stay informed about current events, and make better decisions in their personal and professional lives. For example, a student researching a term paper can quickly find relevant information from academic journals, while a professional preparing for a presentation can access industry reports and data to enhance their work.

Furthermore, finding information from documents promptly can lead to innovation and creativity. By having access to a wide range of information, individuals can draw inspiration from various sources and develop new ideas and solutions to complex problems. This has been particularly evident in fields such as science and technology, where researchers and innovators rely on access to vast amounts of information to drive advancements and discoveries.

“Pdf Insighter” is basically a chatbot that can answer questions based on multiple uploaded PDF documents. The user can upload PDF files containing information, and the chatbot will use natural language processing to understand and respond to questions about the content of these documents.

The application interface includes a main area for asking questions and viewing answers, a sidebar for uploading PDF files, and an expandable section for displaying previous questions and answers. The chatbot uses advanced language models and document processing techniques to provide accurate and relevant responses to user queries.

Overall, this project aims to demonstrate the practical application of natural language processing and document analysis in creating a useful and interactive tool for accessing information stored in PDF documents.

2. Literature Review

After going through various articles and research papers related to chatbots, Natural Language Processing, Large Language Models(LLM’s), Generative AI, OpenAI below are the concepts that have been used in this particular project. There were different kinds of chatbots with different functionality that have evolved over the last couple of years. Understanding the base concepts are of great importance when it comes to building chatbots.

2.1 Introduction to Chatbots and NLP:

Chatbots are computer programs designed to simulate conversation with human users, typically over the internet. They are used in various applications, including customer service, information retrieval, and entertainment. Chatbots can be simple, following predefined rules, or more complex, using artificial intelligence (AI) and natural language processing (NLP) to understand and respond to user inputs.

NLP plays a crucial role in enabling chatbots to understand and generate human-like responses. NLP allows chatbots to interpret the meaning behind user messages, even if they are phrased in different ways or contain typos or grammatical errors. NLP techniques such as named entity recognition (NER), sentiment analysis, and syntactic parsing help chatbots extract relevant information from user inputs and generate appropriate responses. Below is a brief of few important concepts of NLP, they show how the sentences are broken and understood by the computer.

- **Tokenization:** Breaking text into smaller units such as words or phrases. Example: Sentence: "Hello, how are you?" Tokenized: ["Hello", ",", "how", "are", "you", "?"]

- **Part-of-Speech (POS) Tagging:** Assigning grammatical tags to words in a sentence. Example: Sentence: "She is reading a book." Tags: [("She", "PRP"), ("is", "VBZ"), ("reading", "VBG"), ("a", "DT"), ("book", "NN")]

- **Named Entity Recognition (NER):** Identifying named entities such as persons, organizations, or locations in text. Example: Text: "Apple is a technology company." Entities: [("Apple", "ORG")]

- **Sentiment Analysis:** Determining the sentiment (positive, negative, or neutral) expressed in a piece of text. Example: Text: "I love this product!" Sentiment: Positive

- **Word Embeddings:** Representing words as numerical vectors in a continuous space, capturing semantic relationships. Example: Word: "king" - Embedding: [0.23, -0.45, 0.87, ...]

- **Text Summarization:** Generating a concise summary of a longer text while preserving its key information. Example: Original Text: "The quick brown fox jumps over the lazy dog." Summary: "A fox jumps over a dog."

- **Stemming and Lemmatization:** Stemming reduces words to their root form (stem), while lemmatization reduces them to a dictionary form (lemma). Example (Stemming): "Running" -> "run", Example (Lemmatization): "Running" -> "run"

- **Syntax Parsing:** Analyzing the grammatical structure of a sentence to identify relationships between words.Example: "The cat chased the mouse" -> [(ROOT, chased), (nsubj, cat), (det, the), (dobj, mouse)]

- **Dependency Parsing:** Analyzing the grammatical structure of a sentence to determine the relationships between words. Example: "The cat chased the mouse" -> [(ROOT, chased), (nsubj, cat), (det, the), (dobj, mouse)]

- **Topic Modeling:** Identifying topics present in a collection of documents and assigning them to each document. Example: Documents about sports may be assigned the topic "Sports".

- **Coreference Resolution:** Identifying all expressions in a text that refer to the same entity. Example: "The car hit the tree. It was damaged." -> ("car" -> "It")

- **Word Sense Disambiguation:** Determining the meaning of a word based on the context in which it is used. Example: "I saw a bat." -> (bat as a flying mammal) or (bat as a sports equipment)

By leveraging NLP, chatbots can engage in more meaningful and contextually relevant conversations with users, leading to improved user experiences. NLP also enables chatbots to handle more complex queries and tasks, such as answering questions, providing recommendations, and even conducting transactions on behalf of users.

2.2 Previous Work on Chatbots and Document Analysis:

Research and projects involving chatbots for document analysis have evolved significantly, moving from simple question-answering systems based on fixed templates to more sophisticated models capable of understanding and generating human-like responses.

Initially, chatbots were limited to answering questions based on predefined rules and templates, with limited flexibility. However, advancements in natural language processing (NLP) and machine learning have led to the development of more advanced chatbots. They have various applications:

Information Extraction: Chatbots have been used to extract specific information from documents, such as key dates, names, and locations. This is often done using natural language processing (NLP) techniques to understand the content of the document and extract relevant data.

Summarization: Chatbots can generate summaries of documents to provide users with a quick overview of the content. This is useful for quickly understanding the main points of a document without having to read the entire text.

Question-Answering: Chatbots can answer questions about the content of documents, similar to how a human would. This involves understanding the question, finding the relevant information in the document, and providing a coherent answer.

Sentiment Analysis: Chatbots can analyze the sentiment of a document to determine the overall tone or mood. This is useful for understanding the general sentiment of a piece of text, such as whether it is positive, negative, or neutral.

Research and projects in this area have focused on various applications, including information extraction, summarization, question-answering, and sentiment analysis. Recent advancements in NLP, particularly with deep learning models like BERT (Bidirectional Encoder Representations from Transformers), have further improved the accuracy and effectiveness of chatbots for document analysis. These models can understand the context of words and phrases in a document, leading to more accurate information extraction and summarization.

One significant advancement is the use of large language models (LLMs) such as GPT (Generative Pre-trained Transformer) models developed by OpenAI. These models are trained on vast amounts of text data and are capable of understanding and generating human-like text. By leveraging the power of LLMs, chatbots for document analysis can now analyze and understand the content of documents more effectively, leading to more accurate information extraction, summarization, and question-answering. These models can generate coherent and contextually relevant responses, even for complex questions or scenarios, and provide more accurate and informative answers than earlier rule-based systems.

Overall, the evolution of chatbots for document analysis from simple rule-based systems to advanced LLM-powered models has significantly improved their capabilities and effectiveness. Continued research and development in this area are likely to lead to further advancements, making chatbots even more useful for handling textual data.

2.3 Use of Language Models in Chatbots:

Language models play a crucial role in the development of chatbots for document analysis, enabling them to understand and generate human-like text for tasks such as information extraction, summarization, question-answering, and sentiment analysis. Several advanced language models have been developed, each with unique characteristics and capabilities:

a. BERT (Bidirectional Encoder Representations from Transformers): Developed by Google, BERT is a transformer-based model designed to understand word context in sentences by considering both left and right contexts. It is pretrained on a large text corpus and can be fine-tuned for specific tasks, making it effective for document analysis requiring deep language understanding.

The choice of using BERT embeddings in this application is motivated by the following reasons:

1. **Superior Performance**: BERT has consistently demonstrated state-of-the-art performance across a wide range of NLP tasks, including text classification, question answering, and semantic similarity detection.
2. **Transfer Learning**: By leveraging pre-trained BERT models, the application can benefit from the vast knowledge and language understanding capabilities encoded in these models, which have been trained on massive amounts of text data.
3. **Contextual Understanding**: BERT's ability to capture contextual information is particularly beneficial for understanding the meaning and relationships within the PDF document content, enabling more accurate retrieval and generation of relevant information.
4. **Ease of Integration**: The Hugging Face Transformers library and the langchain integration provide a convenient and well-documented interface for incorporating BERT embeddings into the application, reducing the overhead of implementing and fine-tuning the model from scratch.
5. **Scalability**: BERT embeddings can be efficiently stored and queried in vector stores like FAISS, allowing for scalable and high-performance retrieval as the number of documents or text chunks increases.

By leveraging the power of BERT embeddings, this application can provide more accurate and contextually relevant responses to user queries, enhancing the overall quality of the conversational experience and the effectiveness of the document intelligence capabilities.

b. GPT (Generative Pre-trained Transformer): OpenAI's series of transformer-based language models are trained to generate human-like text, used for tasks like text generation, summarization, and question-answering. GPT models are known for producing coherent and contextually relevant responses.

c. XLNet: Another transformer-based model from Google, XLNet addresses some limitations of BERT by modeling bidirectional contexts. It uses a permutation language modeling approach to capture word dependencies in both directions, leading to improved performance on NLP tasks.

d. RoBERTa (Robustly optimized BERT approach): Facebook's variant of BERT, RoBERTa, is pretrained on a larger text corpus and uses different training techniques to enhance performance. It has demonstrated superior performance over BERT on various NLP benchmarks.

e. T5 (Text-to-Text Transfer Transformer): Google's T5 is trained in a text-to-text manner, converting input text into output text. This makes it more versatile than traditional models and has achieved state-of-the-art performance on tasks like text summarization and question-answering.

f. Hugging Face Hub Model: Hugging Face offers a model on its hub that utilizes the google/FLAN-T5-XXL model. This model, based on the T5 architecture, is designed for generating responses in chatbots. It is trained on a large text corpus and is capable of generating human-like responses to user inputs, enhancing chatbot interactions.

These language models have significantly advanced chatbots' capabilities for document analysis, enabling them to understand and generate text with unprecedented sophistication.

Continued research and development in this field is expected to further enhance chatbots' effectiveness in handling textual data.

2.4 Document Processing and Text Extraction:

Any chatbot must first take the user input. There are different input options, it could be direct texts, or pdf’s or word documents etc.

In this project we are creating an application that takes multiple pdf’s as input as it’s a chatbot for document intelligence. Below are the different methods for extracting text from pdf documents.

PyPDF2: PyPDF2 is a Python library for reading PDF files and extracting text. It allows you to extract text, metadata, and other information from PDF documents. PyPDF2 is useful for basic text extraction tasks but may not handle complex PDF structures or encrypted files.

PDFMiner: PDFMiner is another Python library for extracting text from PDF documents. It offers more advanced features compared to PyPDF2, such as the ability to extract text with formatting information. PDFMiner is capable of handling a wide range of PDF files, including those with complex structures.

Tabula: Tabula is a tool for extracting tables from PDF documents. It can extract tables into CSV or Excel formats, making it useful for extracting structured data from PDF files.

Apache Tika: Apache Tika is a content analysis toolkit that can extract text and metadata from various file formats, including PDF. It uses advanced algorithms to extract text accurately and can handle a wide range of PDF files.

OCR (Optical Character Recognition): OCR technology can be used to extract text from scanned PDF documents. Tools like Tesseract OCR can convert scanned images of text into editable text, allowing you to extract text from PDFs that do not contain selectable text.

2.5 Vectorization and Text Representation:

Vectorization is the process of converting text data into numerical vectors that can be used as input for machine learning models. This process is essential because most machine learning algorithms require numerical input. There are several techniques for vectorizing text data, including:

Bag of Words (BoW): This approach represents text as a multiset of words, disregarding grammar and word order. Each document is represented by a vector where each element corresponds to the frequency of a word in the document.

Term Frequency-Inverse Document Frequency (TF-IDF): TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It assigns higher weights to words that are frequent in a document but rare in the corpus.

Word Embeddings: Word embeddings are dense, low-dimensional representations of words that capture semantic relationships between words. These embeddings are learned from large text corpora using techniques like Word2Vec, GloVe, or fastText.

Contextual Embeddings: Contextual embeddings, such as those provided by OpenAI's GPT models or HuggingFace's transformers, capture the context of a word in a sentence or document. These embeddings are pre-trained on large text corpora and can be fine-tuned for specific tasks.

- **OpenAI Embeddings:** OpenAI offers pre-trained language models like GPT (Generative Pre-trained Transformer) models, which are capable of generating human-like text. These models can also be used to obtain embeddings for text, which represent the semantic meaning of words or sentences in a high-dimensional space. OpenAI's embeddings are known for their effectiveness in capturing context and meaning in text, making them suitable for a wide range of NLP tasks, including chatbots, language translation, and text summarization.

- **Hugging Face Embeddings:** Hugging Face provides a library called Transformers, which offers a wide range of pre-trained models for various NLP tasks, including embedding models like BERT (Bidirectional Encoder Representations from Transformers) and RoBERTa (A Robustly Optimized BERT Approach). These models are trained on large amounts of text data and can generate embeddings that capture rich semantic information. Hugging Face's embeddings are widely used in research and industry for tasks such as text classification, sentiment analysis, and question answering.

Faiss: Faiss is used to convert text embeddings or vectors into a format that can be easily compared and searched. For example, after using techniques like Word Embeddings or Contextual Embeddings (e.g., OpenAI Embeddings or Hugging Face Embeddings) to represent text as numerical vectors, Faiss can be used to quickly find similar documents or clusters of documents based on these embeddings.

In document analysis, vectorization and text representation play a crucial role in tasks such as document classification, clustering, and information retrieval. By representing documents as numerical vectors, machine learning models can effectively process and analyze textual data.

2.6 Conversational AI and Memory Management:

Conversational AI models, such as ChatOpenAI and HuggingFaceHub, are used in chatbots to generate responses based on user input by leveraging natural language understanding and generation capabilities. Here's how they work:

Natural Language Understanding (NLU): When a user inputs a message, the chatbot's NLU component processes the text to understand the user's intent, context, and any relevant entities mentioned in the message. This step involves tasks like tokenization, part-of-speech tagging, named entity recognition, and syntactic parsing.

Conversational AI Model: The NLU output is passed to the conversational AI model, such as ChatOpenAI or HuggingFaceHub. These models use sophisticated deep learning architectures, such as transformers, to generate a response that is contextually relevant to the user input. The models are typically pre-trained on large text corpora and fine-tuned for specific chatbot applications.

Natural Language Generation (NLG): The generated response from the conversational AI model is passed to the NLG component of the chatbot, which converts the model's output into natural language text that can be understood by the user. This step involves selecting the best response from the model's output and ensuring that it is grammatically correct and coherent.

Response to User: Finally, the generated response is sent back to the user, completing the conversational loop. Memory management techniques, such as ConversationBufferMemory, are used in chatbots to store and retrieve conversation history. These techniques are important for maintaining context across interactions and providing more personalized responses. Here's how they work:

Memory Storage: As the chatbot interacts with users, the ConversationBufferMemory stores the conversation history, including user inputs and bot responses. This memory can be structured as a list or a database, depending on the complexity of the chatbot.

Contextual Retrieval: When the chatbot receives a new user input, the ConversationBufferMemory can be queried to retrieve relevant information from past interactions. This information can include user preferences, previous queries, or any other context that may be useful for generating a response.

Dynamic Memory Management: ConversationBufferMemory can be designed to dynamically adjust the amount of history it retains based on factors like conversation relevance or memory constraints. This helps the chatbot maintain a balance between context retention and resource efficiency.

Overall, conversational AI models and memory management techniques are key components of chatbots that enable them to provide more engaging and contextually relevant interactions with users.

Langchain Pipeline: The langchain package, simulates a comprehensive language processing pipeline for a chatbot application. It includes components like text splitters for segmenting text, embeddings for converting text into numerical representations, a vector store for efficient storage and retrieval of embeddings, and a memory component for maintaining conversation history. These components work together to process user queries, generate responses, and maintain context, showcasing a typical workflow for building a chatbot system

2.7 <u>Future Directions:</u>

a. Improved User Interaction

User Interface Enhancements: Enhance the user interface to provide a more intuitive and engaging experience. This could include better organization of the chat history, visual cues for document processing progress, and clearer prompts for user interactions.

Real-Time Updates: Implement real-time updates for the chat interface to reflect the processing of documents and provide immediate feedback to the user.

b. Advanced Document Processing

Document Summarization: Integrate document summarization techniques to provide users with concise summaries of each uploaded document. This could help users quickly grasp the key points of the documents.

c. Enhanced Chatbot Capabilities

Contextual Understanding: Improve the chatbot's ability to maintain context across interactions. This could involve implementing a more sophisticated memory mechanism or using a contextual embedding model like BERT for better context retention.

d. Integration with External Systems

Data Integration: Integrate the chatbot with external data sources such as databases or APIs to provide users with access to additional information or services. For example, the chatbot could retrieve relevant data from a healthcare database for medical-related queries.

Workflow Automation: Explore the possibility of integrating the chatbot into existing workflows to automate tasks such as document processing, data analysis, or report generation. This could improve efficiency and reduce manual effort.

e. Performance Optimization

Efficiency Improvements: Optimize the codebase and algorithms for better performance, especially when processing large documents or handling multiple user interactions simultaneously.

Scalability: Design the system to be scalable, allowing it to handle a growing number of users and documents without compromising performance.

f. User Feedback and Iteration

Feedback Loop: Implement a feedback mechanism to gather user feedback on the chatbot's performance and use it to improve the system iteratively.

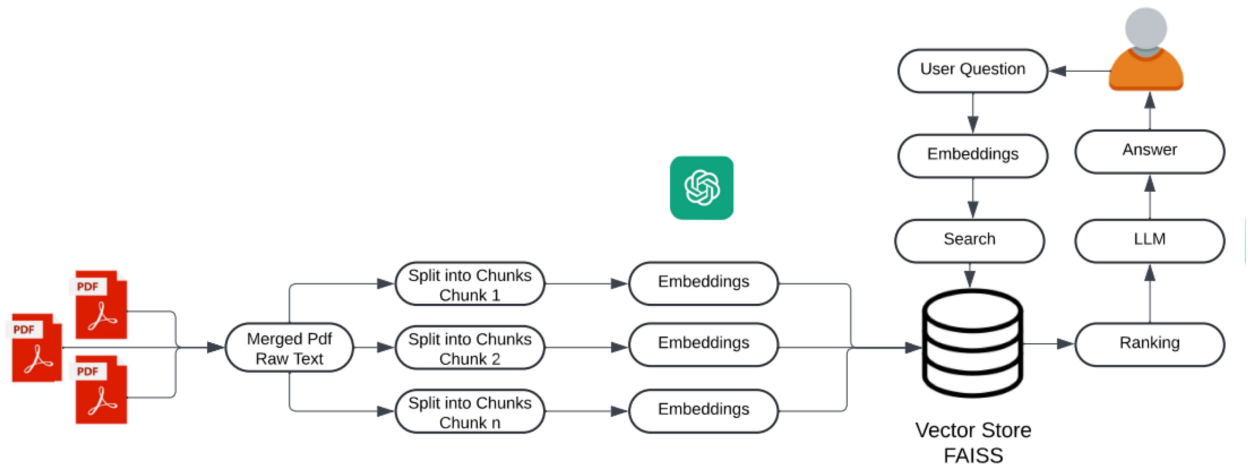
User Testing: Conduct regular user testing to identify usability issues, gather suggestions for improvements, and ensure that the chatbot meets user needs effectively.

Overall, these future directions aim to enhance the functionality, usability, and performance of the chatbot, making it a more valuable tool for users interacting with multiple PDF documents.

3. Proposed Method

In this project, a streamlit application has been created. This Streamlit application utilizes the langchain package, specifically OpenAIEmbeddings models, to create a PDF chatbot. The application's design is managed in a separate file, Template.py. The OpenAI API key is stored in a **.env** file, ensuring secure access. The application's main functionality involves uploading multiple PDFs, processing them to extract text, and using the extracted text to power the chatbot's responses. The chatbot maintains context and conversation history, providing a seamless and interactive experience for users interacting with the PDFs.

Below is the architecture for the application:



The process starts with the user uploading a desired number of PDFs and clicking on the "Process" button. The PDFs are then processed, and their texts are merged into one long essay, which is referred to as the "Merged PDF Raw Text." This merged text is then split into smaller chunks for easier processing.

The text chunks are converted into embeddings, numerical representations that capture the semantic meaning of the text. The application leverages the power of Google's BERT (Bidirectional Encoder Representations from Transformers) model, a state-of-the-art language representation model. The HuggingFaceEmbeddings class from LangChain is used to generate high-quality embeddings for each text chunk using the pre-trained 'bert-base-uncased' model from the Hugging Face Transformers pipeline. BERT's bidirectional nature and ability to capture contextual information result in accurate embeddings that represent the text's nuanced meaning.

The generated embeddings and their corresponding text chunks are stored in a FAISS vector store, a library for efficient similarity search and clustering of dense vectors. This vector store allows for fast retrieval of relevant embeddings based on their vector representations, enabling the chatbot to quickly identify and retrieve the most pertinent information from the PDF content based on user queries.

Once the relevant vectors are retrieved, the chain uses a ranking algorithm to rank these vectors based on their relevance to the user question. This ranking is based on the similarity between the vectors and the question. The chain then generates a response based on the highest-ranked vectors and any additional logic or models it incorporates.

The application sets up a conversational retrieval chain to facilitate natural language interactions with the chatbot. An instance of ChatOpenAI, a wrapper around OpenAI's language models, is created for conversational interactions. A ConversationBufferMemory instance maintains the conversation history, providing context for coherent and relevant responses. The ConversationalRetrievalChain combines the ChatOpenAI language model, the FAISS vector store retriever, and the conversation memory, enabling the chatbot to retrieve relevant information and generate contextually appropriate responses.

Users can input their questions through a text input field in the Streamlit application. The ConversationalRetrievalChain retrieves the most relevant text chunks from the vector store based on the question and conversation context, generates a response leveraging the language model, and displays it to the user. A feedback system is implemented, allowing users to indicate if the response was helpful or not, with the option to perform a Google search for alternative information if the response is deemed unhelpful.

The application incorporates several additional features to enhance the user experience and document exploration capabilities. The get_suggestion_prompts function generates prompts based on the most common words in the text chunks, displayed in the sidebar as starting points for content exploration. The get_document_summaries function generates summaries for the first few text chunks, providing users with a high-level overview of the document content. Custom CSS styles and background images create a visually appealing user interface, while stop word removal improves the relevance of suggestion prompts by focusing on meaningful words. Parallel processing is implemented in this code to improve the performance and efficiency of certain operations, particularly when dealing with large PDF files or a large number of files. It is achieved using the concurrent.futures module in Python, which provides a high-level interface for asynchronously executing callables (functions or methods) using thread or process pools.

The implementation of parallel processing in this code is beneficial for the following reasons:

1. **Performance Improvement**: By utilizing multiple threads or processes, the application can take advantage of modern multi-core CPUs to perform CPU-bound tasks concurrently. This can significantly reduce the overall execution time, especially when dealing with computationally intensive operations like merging and splitting large text files.
2. **Efficient Resource Utilization**: Parallel processing allows for better utilization of available system resources, such as CPU cores. Instead of a single thread or process

executing tasks sequentially, multiple threads or processes can work on different tasks simultaneously, leading to improved efficiency and throughput.

3. **Scalability**: As the number of PDF files or the size of the files increases, parallel processing becomes even more crucial. By distributing the workload across multiple threads or processes, the application can scale more effectively, ensuring that the processing time does not increase linearly with the input size.
4. **Separation of Concerns**: By separating the merging and splitting operations into independent tasks, the code becomes more modular and easier to maintain. Each task can be executed concurrently without affecting the others, promoting code reusability and testability.

It's important to note that parallel processing can improve performance, but it also introduces additional complexity and potential issues, such as thread safety and synchronization. In this code, the operations being parallelized (merging and splitting) are independent and do not require synchronization, making it a suitable case for parallel processing using the concurrent.futures module.

Overall, the implementation of parallel processing in this code is a performance optimization technique that leverages the capabilities of modern hardware and the Python ecosystem to provide a more responsive and efficient document processing experience, especially when dealing with large amounts of data.

This Streamlit application leverages LangChain, OpenAIEmbeddings, BERT, and parallel processing to provide an interactive PDF chatbot experience. It processes PDF documents, generates embeddings with BERT, stores embeddings in a FAISS vector store, and utilizes a ConversationalRetrievalChain with the ChatOpenAI language model to deliver relevant and contextual responses to user queries, enhanced by additional features for a comprehensive document exploration experience.

4. Experiments and Results

4.1 <u>Code Explanation:</u>

Data Collection: Users can upload PDF documents using the Streamlit file uploader widget (st.file_uploader). The PyPDF2 library (PdfReader) is used to extract text from each uploaded PDF document in the MergePdf function.

Text Chunking: The extracted text is split into chunks using the CharacterTextSplitter class in the SplitPdf function. This class splits the text into chunks of a specified size, with an overlap to ensure continuity

Embedding Generation and Chatbot Initialization:

In this code, the FaissVectorstore(text_chunks) function is responsible for generating embeddings for the text chunks extracted from the PDF files. Here's how this works:

Embedding Generation: The HuggingFaceEmbeddings class from the langchain library is used to generate embeddings for each text chunk. Specifically, the 'bert-base-uncased' model from Hugging Face's Transformers pipeline is employed. This pre-trained BERT model is capable of generating high-quality embeddings that capture the semantic meaning of the text.

Vector Store Creation: After generating embeddings for all the text chunks, the FAISS class from the langchain.vectorstores module is used to create a vector store. FAISS is a library for efficient similarity search and clustering of dense vectors. The FAISS class creates an index for the embeddings, allowing for fast retrieval of similar embeddings based on their vector representations.

Return Value: Finally, the FaissVectorstore function returns the vectorstore, which contains the embeddings for all the text chunks. This vectorstore is then used in the ResponseChain function to create a conversational retrieval chain for the chatbot.

Conversational Retrieval Chain and User Interaction:

The Streamlit library is used to create a user-friendly interface where users can upload documents and ask questions.User questions are submitted through a text area in the interface (st.text_area("Ask a question about your documents:", key="user_question")).

The ResponseChain(vectorstore) function sets up the conversational retrieval chain for the chatbot. Here's how it works:

1. An instance of ChatOpenAI is created, which is a wrapper around OpenAI's language models, specifically designed for conversational interactions.
2. A ConversationBufferMemory instance is created to maintain the conversation history, providing context for the chatbot's responses.
3. A ConversationalRetrievalChain is created using the from_llm method, which combines the ChatOpenAI language model, the vectorstore retriever, and the conversation memory.

The handle_userinput(user_question) function is responsible for handling user input and displaying the chatbot's responses. It passes the user's question to the conversation chain (created by ResponseChain), retrieves the response, and displays it along with the user's question in the Streamlit app. Additionally, it implements a feedback system where users can indicate if the response was helpful or not. If the response was not helpful, the app performs a Google search and displays the top result as an alternative source of information.

Chatbot Response:

A ConversationalRetrievalChain is used to handle the conversation flow. This chain uses the vector store to retrieve relevant text chunks for the chatbot's responses.The chatbot uses the loaded language model to generate responses based on the retrieved text chunks and the chat history stored in memory.

Feedback Loop:

The feedback loop is implemented as part of the handle_userinput function, which is called when a user interacts with the chatbot. Here's a more detailed explanation of how the feedback loop works:

User Interaction: When a user enters a question, the handle_userinput function is called with the user's question as an argument.

Chatbot Response: The chatbot uses the conversation_chain to generate a response based on the user's question. The response is then displayed to the user.

Feedback Prompt: After displaying the chatbot's response, the code checks if the response was helpful to the user. If the response was not helpful, the code uses the googlesearch library to perform a Google search using the user's question. This is done by constructing a search query that includes the user's question and the site:wikipedia.org parameter to limit the search results to Wikipedia.

Google Search: The googlesearch.search function is called with the search query, and the code attempts to retrieve the first search result using next(search(query, num=1, stop=1)). If a search result is found, it is displayed to the user as an alternative source of information.

Feedback Handling: If the user indicates that the response was helpful, the code skips the ineffective response handling and continues with the conversation. If the user indicates that the response was not helpful, the Google search result is displayed as an alternative source of information.

By using the feedback loop, the chatbot can learn from user interactions and improve its responses over time. If a user indicates that a response was not helpful, the chatbot can use external sources, such as Google search results, to provide more relevant information to the user. This iterative process helps the chatbot become more effective in providing useful and accurate information to users.

Chat History Display:

The chat interface displays the user's questions and the chatbot's responses in a chronological order. This is implemented in the for loop that iterates over st.session_state.chat_history to display previous questions and answers.

Importing Libraries

- streamlit is a Python library for building interactive web applications.
- dotenv is used to load environment variables from a .env file.
- PyPDF2 is a library for reading and manipulating PDF files.
- transformers is a library from Hugging Face for natural language processing tasks, specifically for tokenization and embedding models.
- langchain is a framework for building applications with large language models, providing utilities for text splitting, embeddings, vector stores, and conversational chains.
- Template is a custom module containing HTML templates for styling the user interface.
- base64 is used for encoding images as base64 strings.
- googlesearch is a library for performing Google searches.
- concurrent.futures is a Python module for parallel execution of tasks.
- collections provides data structures like Counter for counting occurrences.
- nltk (Natural Language Toolkit) is a library for natural language processing tasks, used here for stop word removal.

Helper Functions

- MergePdf(pdf_docs): This function takes a list of PDF documents and merges their content into a single text string.
- SplitPdf(text): This function splits the input text into smaller chunks using the CharacterTextSplitter from LangChain. The chunks are split based on newline characters, with a maximum chunk size of 1000 characters and an overlap of 200 characters.

- FaissVectorstore(text_chunks): This function creates a FAISS vector store from the text chunks using the Hugging Face BERT embeddings.
- ResponseChain(vectorstore): This function creates a ConversationalRetrievalChain from LangChain, which combines a language model (ChatOpenAI), a vector store for retrieval, and a conversational memory buffer.store and ChatOpenAI model.

Update Model Function

- update_model(response, user_question, feedback): Placeholder function for updating the model based on user feedback. Currently, it prints the user question, bot response, and feedback.

User Input Handling

- handle_userinput(user_question): This function handles user input by passing the question to the conversation chain, retrieving the response, and displaying it along with the user's question in the Streamlit app. It also implements a feedback system where users can indicate if the response was helpful or not. If the response was not helpful, it performs a Google search and displays the top result.

Other Functions

- get_img_as_base64(file): Encodes an image file as a base64 string.
- get_suggestion_prompts(text_chunks): This function generates suggestion prompts based on the most common words (excluding stop words) in the text chunks.
- get_document_summaries(text_chunks): This function generates summaries for the first few text chunks by asking the conversational chain to summarize each chunk.

Main Function

- Loads environment variables from a .env file.
- Sets the page configuration and applies custom CSS for styling.
- Initializes session state variables for conversation and chat history.
- Handles user input and file uploads for PDF processing.
- Processes uploaded PDFs by merging, splitting, and generating a vector store.
- Interacts with the chatbot based on user questions and displays responses.

Report Details

- Technology: Python, Streamlit, PyPDF2, Transformers (BERT), Langchain, dotenv, Googlesearch.

- APIs/Services: Hugging Face Transformers pipeline, Google Search API.

- Functionality: The app allows users to upload multiple PDF files and ask questions about the content. It processes the PDFs to extract text, generates embeddings using BERT and FAISS vector store, and uses a ChatOpenAI model powered by LangChain for responding to user queries. A feedback system is implemented, where users can indicate if the response was helpful or not. If not helpful, the app performs a Google search and displays the top result as an alternative source of information.

- Design: Custom CSS styling is applied to enhance the visual appeal of the user interface, including background images and styling for various Streamlit components.

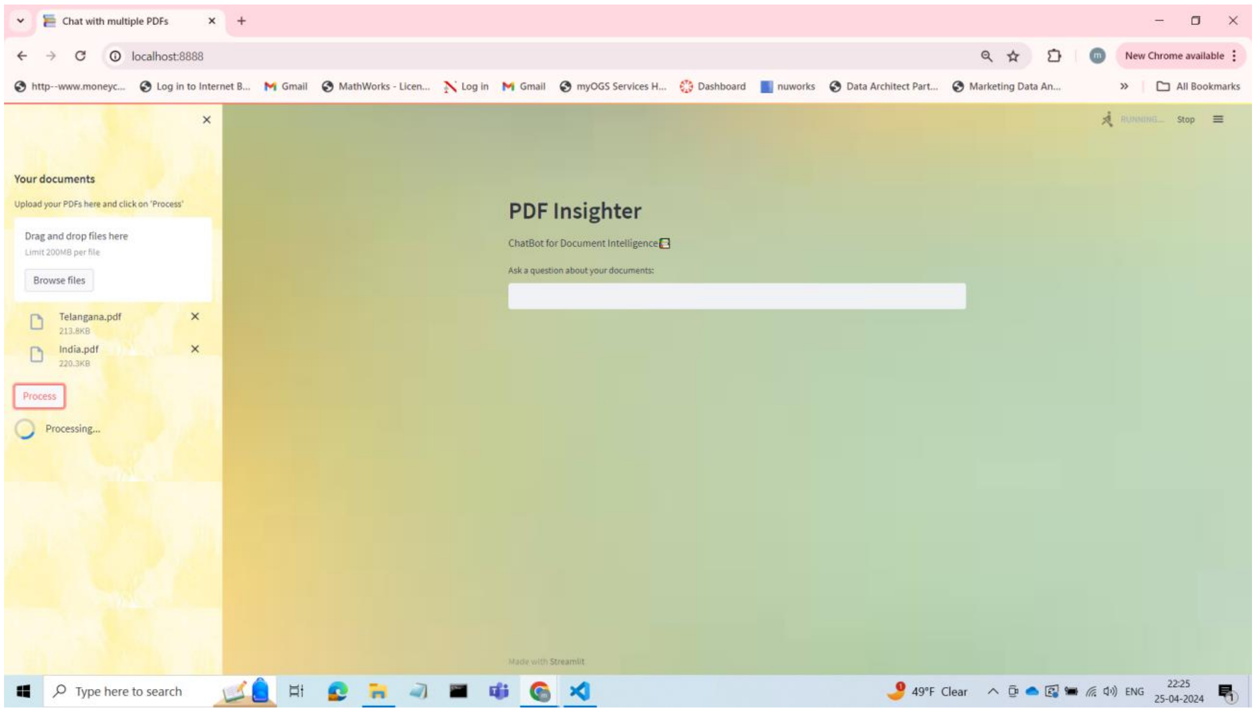
- User Interaction: Users can interact with the chatbot by asking questions in a text input field. They can provide feedback on the chatbot's responses by selecting "Yes" or "No" in a selectbox. The app also displays suggestion prompts based on the most common words in the documents, allowing users to explore the content further.

4.2 How to Run the Code:

1. create a virtual environment →python -m venv .venv
2. if already exists then activate → .venv\Scripts\Activate
3. might need to install these → pip install streamlit pypdf2 langchain python-dotenv faiss-cpu openai huggingface_hub/ pip install tiktoken/pip install langchain/pip install InstructorEmbedding sentence_transformers/pip install instructor
4. Run the file using → streamlit run --server.port 8888 App.py

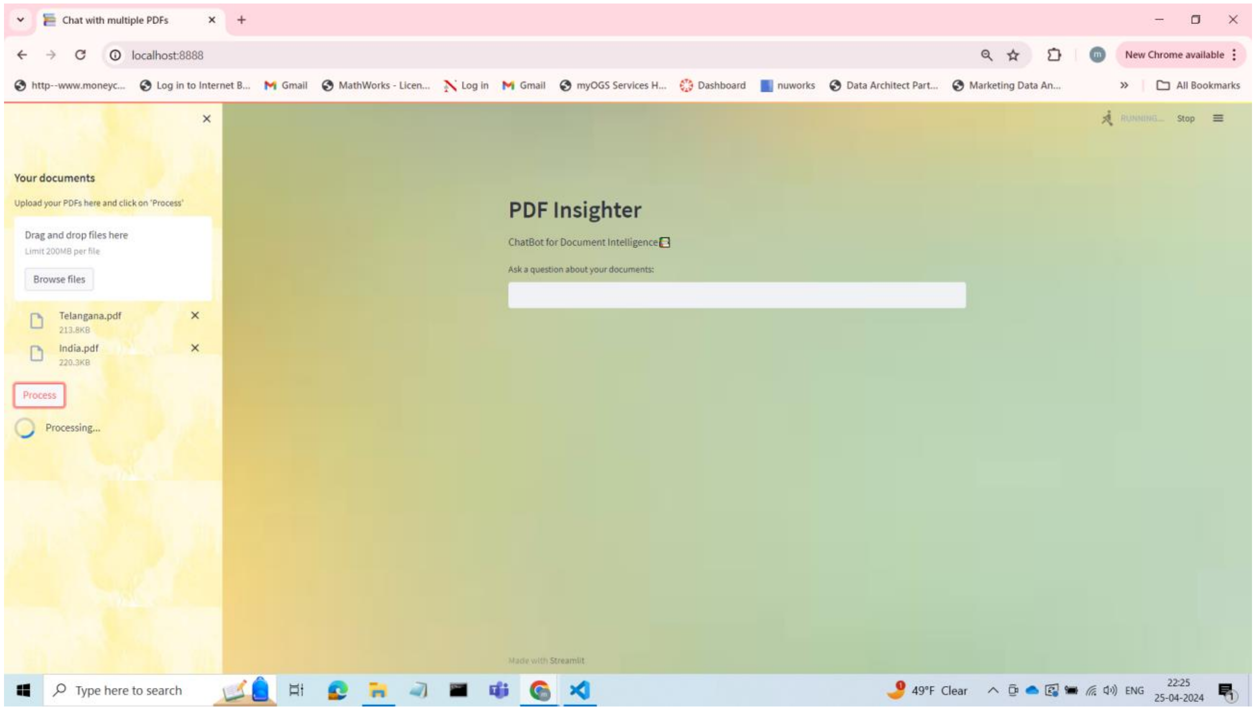
4.3 Results:

The interface and the results of a case where two pdf's were uploaded are below. Documents "India.pdf" and "Telangana.pdf" were uploaded, and few questions related to information in both were asked. The first snippet just shows the look of the streamlit application. Later is the snippet that shows how it looks when pdf's are uploaded

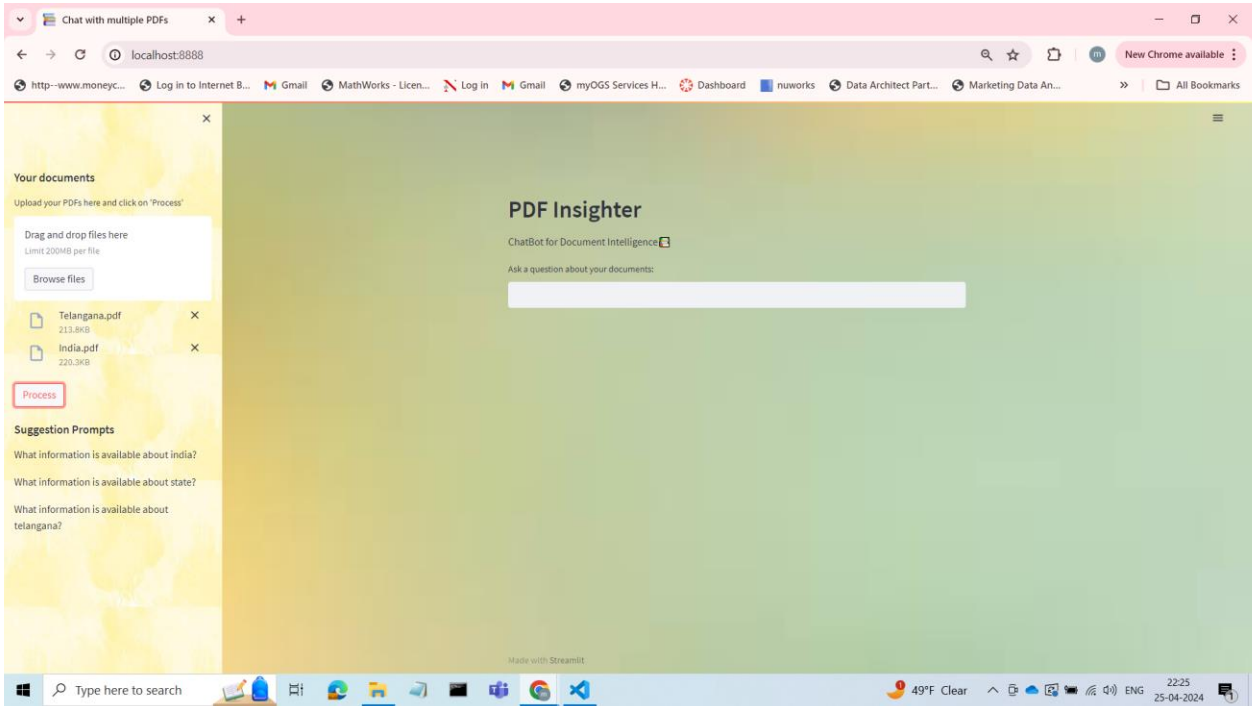


This is the base outlook of the application. It allows user to upload pdf's by clicking on the

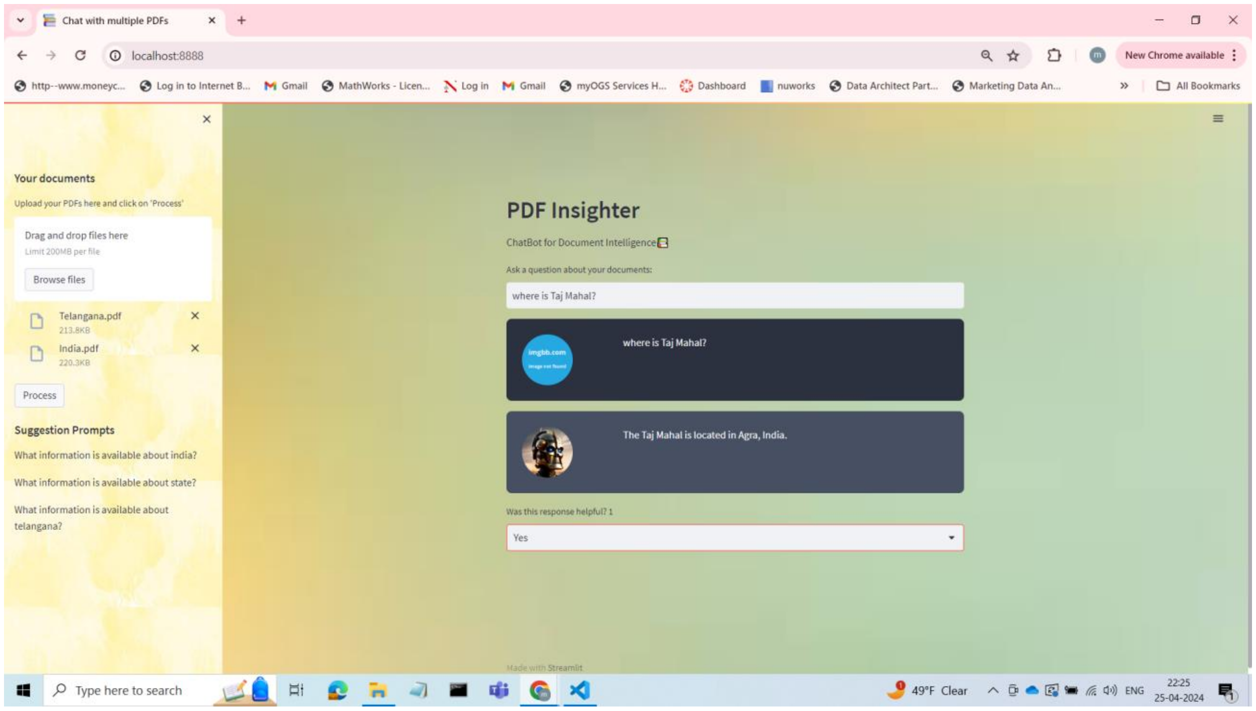
“Browse Files” option. Multiple pdf’s can be uploaded. Once files are uploaded, clicking on “Process” starts the application. The content s merged, separated into chunks and embeddings are formed.



You can also see that few Suggestion Prompts are also available for the user to ask.



Here are snippets of questions being asked and the chatbot responding to them.



Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here
Limit 200MB per file
Browse files

Telangana.pdf 213.8KB
India.pdf 220.3KB

Process

Suggestion Prompts

What information is available about india?
What information is available about state?
What information is available about telangana?

PDF Insighter

ChatBot for Document Intelligence

Ask a question about your documents:

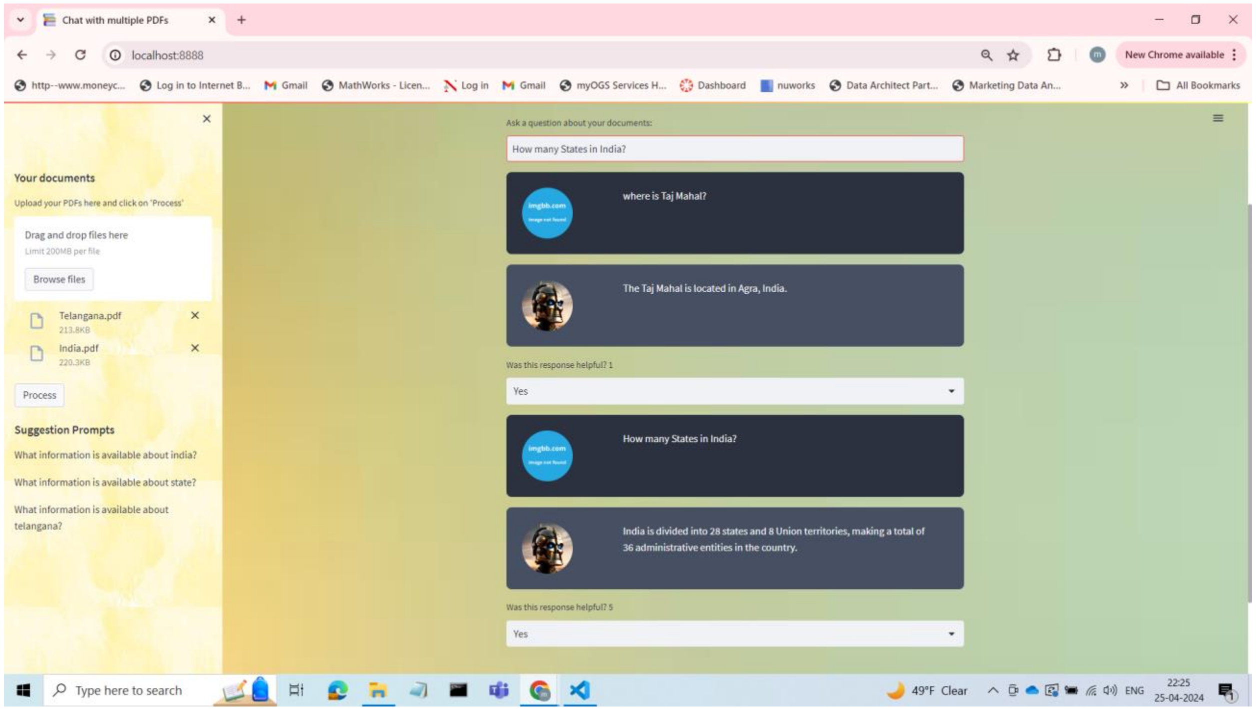
where is Taj Mahal?

where is Taj Mahal?

The Taj Mahal is located in Agra, India.

Was this response helpful? 1

Yes



Ask a question about your documents:

How many States in India?

where is Taj Mahal?

The Taj Mahal is located in Agra, India.

Was this response helpful? 1

Yes

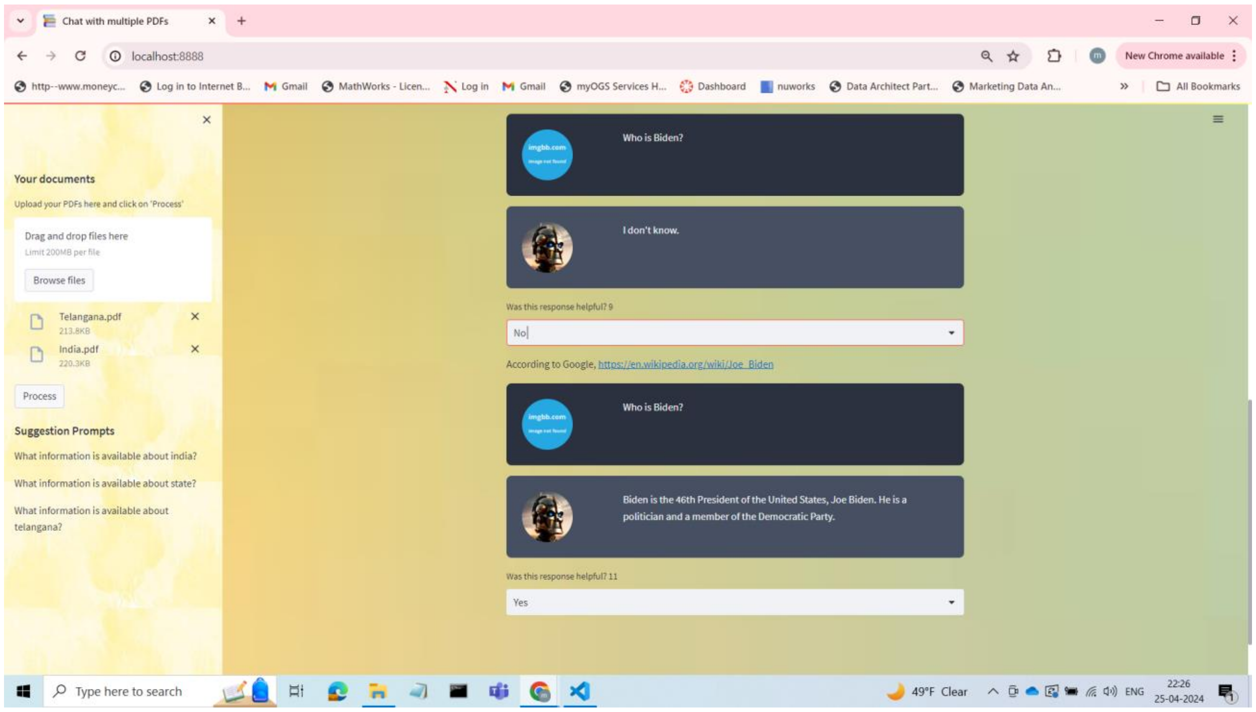
How many States in India?

India is divided into 28 states and 8 Union territories, making a total of 36 administrative entities in the country.

Was this response helpful? 5

Yes

From the below images you can see how the feedback loop works.



Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here

Limit 200MB per file

Browse files

Telangana.pdf 213.8KB

India.pdf 220.3KB

Process

Suggestion Prompts

What information is available about india?

What information is available about state?

What information is available about telangana?

Who is Biden?

I don't know.

Was this response helpful? 9

No

According to Google, https://en.wikipedia.org/wiki/Joe_Biden

Who is Biden?

Biden is the 46th President of the United States, Joe Biden. He is a politician and a member of the Democratic Party.

Was this response helpful? 11

Yes

5. Conclusion

The project "**Pdf Insighter - Chatbot for Document Intelligence**" demonstrates the integration of several technologies to create a novel application for interacting with PDF documents through natural language processing (NLP) and machine learning (ML) techniques. The application combines various techniques and libraries to process multiple PDF documents, generate embeddings using the powerful BERT model, store the document content in an efficient vector store, and facilitate natural language conversations with users through a conversational retrieval chain.

One of the key features of this application is its ability to handle multiple PDF files in parallel, leveraging Python's concurrent.futures module to distribute the workload across multiple threads or processes. This approach significantly improves performance and efficiency, especially when dealing with large or numerous PDF files. The application utilizes the PyPDF2 library to merge and split the PDF content into manageable chunks for further processing.

The application takes advantage of Hugging Face's BERT model to generate high-quality embeddings for the text chunks. BERT, a state-of-the-art language representation model, is renowned for its ability to capture rich semantic and contextual information. By using BERT embeddings, the application can more accurately retrieve relevant information based on the user's queries, as the embeddings encode the meaning and relationships between words and phrases in the text.

The FAISS vector store, provided by the LangChain library, is employed to efficiently store and retrieve the text chunks and their corresponding embeddings. This vector store enables fast and accurate retrieval of relevant information based on semantic similarity, a crucial aspect of providing contextually relevant responses to user queries.

The application integrates OpenAI's language model through the ChatOpenAI class from LangChain. This powerful language model, trained on vast amounts of data, enables the application to engage in natural language conversations with users. By combining the OpenAI language model, the vector store retriever, and a conversation memory component, the application can maintain conversational context and provide coherent and contextually relevant responses.

One notable feature of this application is its feedback system, which allows users to indicate whether the chatbot's response was helpful or not. If the response is marked as unhelpful, the application performs a Google search using the user's query and the site:wikipedia.org filter, displaying the top result as an alternative source of information. This feedback mechanism not only enhances the user experience by providing additional resources but also serves as a valuable input for potential model improvement or refinement.

Furthermore, the application generates suggestion prompts based on the most common words (excluding stop words) in the text chunks. These prompts, displayed in the sidebar, serve as starting points for users to explore the document content, fostering a more interactive and engaging experience.

Overall, this project highlights the potential of NLP and ML technologies to revolutionize document analysis and information retrieval. The application's ability to process multiple PDFs and engage in meaningful conversations with users opens up new avenues for research, knowledge discovery, and document exploration. The integration of parallel processing, BERT embeddings, vector stores, and OpenAI's language model demonstrates the power of combining NLP and ML techniques to create innovative applications for document analysis and interaction. The feedback system and suggestion prompts further enhance the user experience and provide opportunities for continuous improvement. Ultimately, this project showcases the potential of natural language interaction with documents, opening up new possibilities for information retrieval and knowledge exploration.