DOCUCHAT

- (a private chatGpt)

A PROJECT REPORT

Submitted by

Sandeep Gajula - B181024

Of

Bachelor of Technology

Under the guidance of

Mr. Revya Naik

Assistant Professor

Dept. Of CSE, RGUKT- Basar

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Rajiv Gandhi University of Knowledge Technologies
BASAR, NIRMAL (DIST.)
TELANGANA – 504107

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies

Basar

CERTIFICATE

This is to certify that the **"Project Report"** entitled "**DOCU CHAT"** submitted by, **Sandeep Gajula - B181024** Department of Computer Science and Engineering, Rajiv Gandhi University of Knowledge Technologies, Basar; for partial fulfillment of the requirements for the degree of **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING**, is a bonafide record of the work and investigations carried out by him/ her under my supervision and guidance.

PROJECT SUPERVISOR		**HEAD OF DEPARTMENT**
Mr. Revya Naik		**Mr. Revya Naik**
Assistant Professor	**EXTERNAL EXAMINER**	**Assistant Professor**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies

Basar

DECLARATION

We hereby declare that the work which is being presented in this project entitled, **"DocuChat"** submitted to **RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES, BASAR** in the partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING is** an authentic record of our own work carried out under the supervision of **"Mr. Revya Naik", Assistant Professor in the Department of Computer Science And Engineering, RGUKT, Basar.**

The matter embodied in this project report has not been submitted by me/us for the award of any other degree.

Place: Basar

Date:

Name of the Student (ID. No)

Sandeep Gajula - B181024

ACKNOWLEDGEMENT

First we would like to thank the management of RGUKT-Basar for giving us the opportunity to do major projects within the organization. We also would like to thank all the people who worked along with us in RGUKT- Basar, with their patience and openness they created an enjoyable working environment.

We would like to thank our project guide **Revya Naik, Assistant Professor , RGUKT Basar** for his guidance and help throughout the development of this project work. Without his guidance, cooperation and encouragement, we couldn’t learn many new things during our project tenure.

We would like to thank our Head of the Department**, Mr. Revya Naik, Assistant Professor, RGUKT Basar** for his support and encouragement in completing our project.

We would like to thank our department Project Coordinator **Mrs. B. Latha**, **Assistant Professor, RGUKT Basar** for their support and advice to get and complete our project within the time.

We are extremely grateful to our department staff members, family members, and friends who helped us in the successful completion of this major project.

Student Names:	**Signatures:**
Sandeep Gajula	

ABSTRACT

In the realm of academic exploration, our project endeavors to redefine the way individuals engage with PDF documents through a document query answering system. Traditional keyword-based searches often lead to frustrating inefficiencies, hindering productivity for researchers, students, and professionals. Our solution introduces a conversational AI system that allows users to pose natural language queries, extracting precise responses directly from relevant document passages.

Powered by the advanced LaMini-Flan-T5-248M language model and supported by sophisticated vector search mechanisms, our system facilitates rapid and accurate information retrieval. This transformative approach accelerates research processes, fostering a deeper understanding of complex topics beyond the constraints of conventional keyword searches.

The impact of our project extends to four key areas. Firstly, it accelerates research and learning by eliminating the need for time-consuming manual exploration of PDF documents. Secondly, it promotes a deeper understanding by unveiling nuanced connections and insights within intricate subjects. Thirdly, it enhances productivity by redirecting focus from outdated search methods to analysis and creative thinking. Lastly, it fosters inclusive knowledge sharing by making research accessible to a broader audience and breaking down technical barriers.

In conclusion, our document query answering system goes beyond being a tool; it serves as a gateway to expedited, in-depth, and universally accessible knowledge. By seamlessly connecting individuals with information, this technology empowers users to learn, research, and create with enhanced efficiency and impact. Our project stands as a testament to the transformative potential of conversational AI in reshaping information retrieval and making it more intuitive and impactful for all.

Keywords:
Conversational AI, LaMini-Flan-T5-248M Language Model, Vector Search Mechanisms.

Contents

CHAPTER 1: INTRODUCTION

1.1 Background

Information Overload in PDFs: Individuals and organizations across various industries frequently encounter the challenge of extracting relevant information from lengthy PDF documents. This process often involves time-consuming manual reading and keyword searches, which can lead to inefficiencies and potential oversights.

Limitations of Traditional Methods: Traditional methods of information retrieval, such as keyword searches and table of contents, often fall short in handling complex queries, understanding context, and providing concise summaries and translation.

The Promise of Large Language Models: Large language models (LLMs) have emerged as a powerful tool for natural language processing, capable of generating human-quality text and comprehending complex relationships within text.

1.2 Objectives

Develop a Conversational Interface for PDFs: To create a user-friendly conversational interface that allows users to interact with PDF documents naturally, using their own language to ask questions and receive informative responses.

Leverage LLM for Enhanced Information Retrieval: To utilize the capabilities of LLMs to extract relevant information from PDFs, generate comprehensive answers, and provide contextual insights beyond simple keyword matching.

Improve Efficiency and Reduce Cognitive Load: To streamline the process of information extraction from PDFs, saving users time and effort, and reducing the cognitive burden associated with manual reading and search.

1.3 Purpose, Scope, and Applicability

1.3.1 Purpose

The primary purpose of DocuChat is to provide a more intuitive and efficient way to interact with PDF documents, enabling users to find the information they need quickly and easily without having to read through entire documents manually.

It aims to enhance comprehension and knowledge retention by presenting information in a conversational format, aligning with natural human communication patterns.

1.3.2 Scope

- DocuChat currently focuses on interacting with PDF documents in English.
- It supports a wide range of question types, including factual, analytical, and inquisitive queries.
- It can handle multiple PDFs simultaneously, allowing users to cross-reference information from different sources.

1.3.3 Applicability

DocuChat has potential applications across various domains where PDFs are prevalent, such as:

- Academic research and education
- Legal and financial industries
- Business and corporate settings
- Healthcare and medical documentation
- Government and administrative agencies
- Personal knowledge management and productivity

CHAPTER 2: SURVEY OF TECHNOLOGIES

2.1 Core Technologies:

- **LangChain:** An open-source framework designed to empower LLM-powered applications. Imagine it as a bridge between your application and the LLM, handling heavy lifting like document retrieval, text processing, and memory management. This frees you to focus on the core logic of your application. LangChain boasts several modules tailor-made for document query answering:
  - DocumentLoaders: These modules seamlessly integrate with different document formats, like PyPDFLoader for PDFs, extracting the raw text needed for further processing.
  - TextSplitters: Splitting massive text into manageable chunks, as CharacterTextSplitter does, improves search efficiency and feeds information to the LLM more easily.
  - Embeddings: Modules like SentenceTransformerEmbeddings convert text into numerical representations called embeddings, enabling efficient similarity comparisons and retrieval of relevant passages.
  - Vector Stores: FAISS serves as an example, efficiently storing and retrieving these embeddings, allowing LangChain to quickly find relevant information within your documents.
  - LLMs: HuggingFace Pipeline facilitates smooth interaction with LaMini-Flan-T5-248M, integrating its text generation and reasoning capabilities seamlessly.
  - Chains: ConversationalRetrievalChain is a dedicated LangChain chain for conversational interactions. It orchestrates information retrieval based on user queries and the conversation history, feeding it to the LLM for insightful responses.
  - Memory: ConversationBufferMemory remembers past user queries and LLM responses, fostering a more natural and contextual conversation flow.
- **LaMini-Flan-T5-248M:** This factual language model, developed by Google and trained on a vast dataset of text and code, serves as the project's brain. Its knowledge and understanding of the real world empower it to:
  - Generate natural language text: Whether summarizing information, providing creative

formats, or simply answering questions, LaMini-Flan-T5-248M excels at crafting high-quality, informative responses.
    - Translate languages: Break down language barriers and access information from various sources by leveraging LaMini-Flan-T5-248M's multilingual capabilities.
    - Write different creative text formats: Need a poem, script, email, or even code snippet? LaMini-Flan-T5-248M taps into its creative side to generate diverse text formats based on your prompts.
- **Streamlit**: This Python library simplifies building interactive web applications. It takes away the web development hurdles, allowing you to focus on your application's logic and functionality. Streamlit benefits this project by:
    - Rapid prototyping and development: Build a functional web app quickly and iterate on its design easily without extensive web development experience.
    - Interactive user interface: Create dashboards, charts, and interactive elements that make your application engaging and user-friendly.
    - Easy deployment: Share your Streamlit app with others by deploying it online with minimal configuration.

2.2 Supporting Technologies:

- Transformers: This popular open-source library provides pre-trained models and pipelines for various NLP tasks. It offers the tools and infrastructure needed to work with cutting-edge NLP models like LaMini-Flan-T5-248M.
- PyPDF2: This Python library specifically designed for PDFs enables reading and parsing documents, extracting text, images, and other elements. In this project, PyPDF2 preprocesses the uploaded documents by extracting the raw text.
- FAISS: Standing for "Facebook AI Similarity Search," this powerful library speeds up similarity search and retrieval using dense vectors. It plays a crucial role in:
    - Storing text embeddings: FAISS efficiently organizes embeddings created by SentenceTransformerEmbeddings, enabling swift retrieval of relevant document passages when a user asks a question.
    - Accelerating retrieval: The system can quickly identify the most similar text chunks using

FAISS, significantly boosting the speed and accuracy of the document query answering process.

- HuggingFace Pipeline: This platform hosts and shares machine learning models, including LaMini-Flan-T5-248M. The HuggingFace Pipeline interface conveniently interacts with these models within Python code. In this project, it's used to:
    - Load the LaMini-Flan-T5-248M model: The pipeline function from the Transformers library, powered by Hugging Face, effortlessly loads the pre-trained LaMini-Flan-T5-248M model, making its capabilities available for text generation and other NLP tasks.
    - Simplify model interaction: Hugging Face Pipelines abstract away the complexities of loading and managing model weights and configurations, allowing developers to focus on using the model's functionalities

- Translate : It is a simple but powerful translation tool written in python with support for multiple translation providers. By now we offer integration with Microsoft Translation API, Translated MyMemory API, LibreTranslate, and DeepL’s free and pro APIs

    The biggest reason to use translate is to make translations in a simple way without the need of bigger effort and can be used as a translation tool like command line

- TexSoup: It is a Python library designed for parsing and manipulating LaTeX documents, commonly used for technical and scientific writing. It facilitates the extraction of structured content from LaTeX files, enabling access to elements like sections, equations, figures, and tables. TexSoup offers functionalities for navigating the document tree, searching and filtering content, and programmatically modifying document elements. With seamless integration into the Python ecosystem, TexSoup empowers users to automate tasks related to LaTeX document processing, from data extraction to analysis, streamlining workflows for researchers, writers, and data scientists alike.

CHAPTER 3: REQUIREMENTS AND ANALYSIS

3.1 Problem Definition:

In today's information-rich world, navigating through complex documents like PDFs to find specific answers can be time-consuming and frustrating. Current search methods often rely on keyword matching, leading to irrelevant results and missed opportunities for crucial insights. This inefficiency impacts researchers, students, and professionals across various fields, hindering their productivity and knowledge discovery.

Requirements:

HARDWARE REQUIREMENT:

- Processor –Core i5
- Hard Disk – 4 GB
- Memory – 8 GB RAM

SOFTWARE REQUIREMENTS:

- Python: 3.7 or later.
- LangChain: Framework for LLM-powered applications (document retrieval, text embedding, memory management)
- Instructor-xl: Finetuned text embedding model
- LaMini-Flan-T5-248M: Google's factual language model for generating responses
- Streamlit: Python library for building interactive web apps (user interface)

Existing System:

- Users rely on manual keyword searches within documents, often leading to incomplete or irrelevant results.
- Information retrieval is a linear process, lacking context-awareness and user-driven exploration.

- Existing systems lack user-friendly interfaces, making document exploration cumbersome and unintuitive.

Proposed System:

- **Conversational search interface**: Users can ask questions about documents in natural language, mimicking a conversation for a more user-friendly experience.
- **Advanced information retrieval:** Utilizes language models and document embeddings to understand user queries and retrieve relevant information accurately.
- **Contextual understanding:** Tracks past interactions and builds context to provide more relevant responses and recommendations.
- **Enhanced user experience:** Provides a visually appealing and intuitive interface for easy navigation and exploration of documents.

Benefits:

- **Improved retrieval accuracy:** Find relevant information quickly and efficiently, eliminating the need for tedious keyword searches.
- **Deeper understanding of documents:** Extract insights and connections beyond simple keyword matches, leading to richer knowledge discovery.
- **Enhanced user experience:** Interact with documents naturally and intuitively, making the process enjoyable and productive.
- **Increased productivity:** Save time and effort spent on traditional search methods, allowing for more focused research and analysis.

CHAPTER 4: SYSTEM DESIGN

4.1 Component Breakdown:

1. Data Acquisition:
   - Streamlit: User interface for uploading and interacting with PDFs.
   - PyPDF2: Extracts text content from uploaded PDF files.
2. Text Processing:
   - CharacterTextSplitter: Splits the extracted text into manageable chunks for efficient processing.
3. Embedding Generation:
   - SentenceTransformers: Generates semantic representations (embeddings) of text chunks using the "Instructor-xl / all-MiniLM-L6" model.
4. Vector Store:
   - FAISS: Stores and retrieves text embeddings efficiently for fast document indexing and search.
5. Conversation Engine:
   - LaMini-Flan-T5-248M: A fine-tuned Flan-T5 model powering both text generation and retrieval.
   - ConversationalRetrievalChain: Manages the conversation flow by retrieving relevant text chunks based on user queries and conversation history.
   - ConversationBufferMemory: Maintains a buffer of past interactions to provide context-aware responses.
6. User Interface:
   - Streamlit: Renders the chat interface and displays responses generated by the conversation engine.

4.2 Component Interactions:

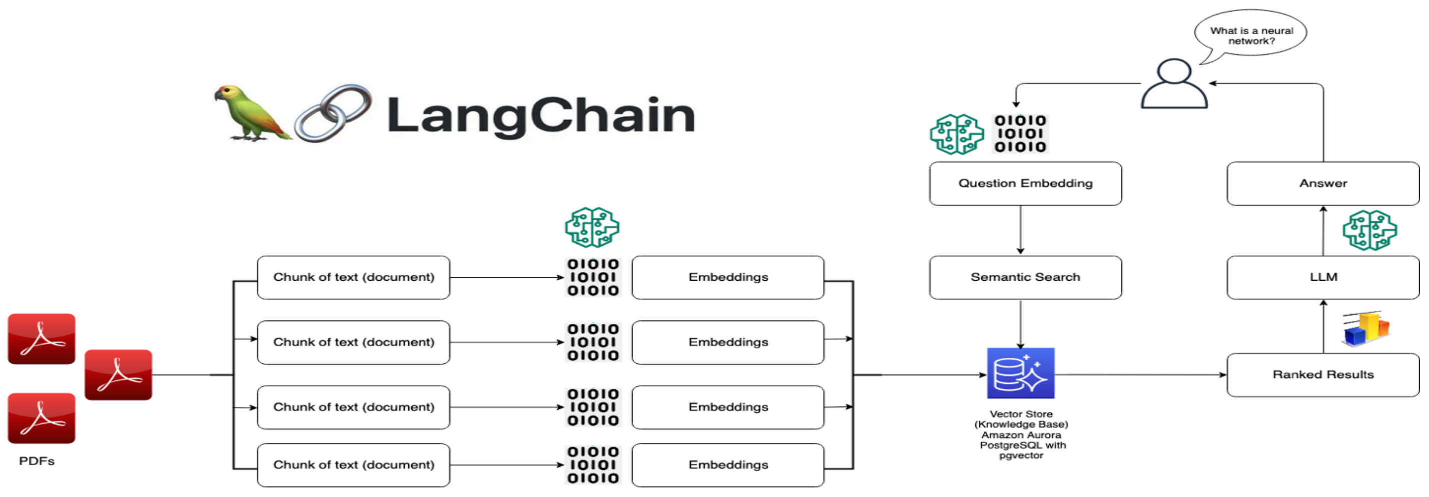
1. User Input: User submits a question through the chat interface.
2. Text Processing: CharacterTextSplitter segments the uploaded PDF text into chunks.

3. Embedding Generation: SentenceTransformers creates vector representations of each chunk.
4. Vector Store Retrieval: FAISS retrieves relevant text chunks based on the user's question and past conversation (via ConversationalRetrievalChain).
5. Language Model Integration: LaMini-Flan-T5-248M analyzes the retrieved chunks and conversation history to generate a response.
6. Conversation Buffer Update: ConversationBufferMemory stores the recent interaction for future context.
7. Response Display: Streamlit renders the generated response in the chat interface.

4.3 Design Advantages:

- Modular architecture: Facilitates independent development and testing of individual components.
- Efficient storage and retrieval: FAISS enables fast access to relevant information based on semantic proximity.
- Context-aware conversation: ConversationalRetrievalChain and ConversationBufferMemory ensure responses are consistent with past interactions.
- Fine-tuned language model: LaMini-Flan-T5-248M offers advanced text generation and retrieval capabilities.
- User-friendly interface: Streamlit provides a simple and intuitive chat interface for user interaction.

4.4 Data Flow Diagram



Data Flow:

1. Users upload PDFs via web interface.
2. PDF text extraction using PyPDF2.
3. Text chunking into smaller segments.
4. Embedding generation for text chunks using Sentence Transformers.
5. Storage of embeddings in FAISS vector store for efficient retrieval.
6. User asks questions in natural language.
7. Conversation chain retrieves relevant text chunks from FAISS.
8. LLM generates responses based on retrieved chunks and conversation history.
9. Responses displayed to the user in the chat interface**.**

CHAPTER 5: SYSTEM IMPLEMENTATION

5.1 Code Flow

Screen-1:

Function-1: Loads the core language model and tokenizer for text processing and generation.

Steps:

1. Specifies the model checkpoint: Sets `checkpoint` to "LaMini-Flan-T5-248M", a fine-tuned model with strong text capabilities.
2. Prints the checkpoint path: Confirms the model being loaded for debugging purposes.
3. Loads the tokenizer: Fetches the tokenizer associated with the model to break down text into tokens for processing.
4. Loads the model: Loads the LaMini-Flan-T5-248M model itself, ready for language tasks.
    - `device_map="auto"`: Optimizes performance by choosing the best device (CPU or GPU) for computations.
    - `torch_dtype=torch.float32`: Sets the data type for tensors to the common 32-bit floating-point format.

Key Points:

- Loads the essential language model and tokenizer for DocuChat's functionality.
- Employs LaMini-Flan-T5-248M for its text generation and understanding capabilities.
- Optimizes performance through device selection and data type configuration.

Function-2: Creates an efficient, cached text-generation pipeline for language model interactions.

Steps:

1. Caches for performance: Uses @st.cache_resource to avoid re-creating the pipeline for every user interaction.
2. Sets up Hugging Face pipeline:
    - Specifies 'text2text-generation' for text-based tasks.
    - Connects to the LaMini-Flan-T5-248M model and tokenizer.
    - Tunes text generation with:
        - max_length=500 to limit output length.
        - do_sample=True for diverse responses.
        - temperature=0.3 for controlled randomness.
        - top_p=0.95 for token selection diversity.
3. Wraps in LangChain: Integrates the pipeline with LangChain using HuggingFacePipeline.
4. Returns the pipeline: Makes it available for use in the application.

Key Points:

- Optimizes performance through caching.
- Provides a versatile text generation toolkit based on LaMini-Flan-T5-248M.
- Allows control over response length, creativity, and diversity.
- Integrates seamlessly with LangChain for conversational use.

Function-3: Extracts text content from multiple PDF files.

Steps:

1. Takes a list of PDF files as input.
2. Iterates through each PDF:
    - Opens the PDF using PyPDF2.
    - Extracts text from each page.
    - Appends extracted text to a string.
3. Returns the combined text from all PDFs.

Key Points:

- Enables text processing and understanding of uploaded PDFs within the DocuChat application.
- Utilizes PyPDF2 for PDF handling.
- Handles multiple PDFs efficiently.
- Returns a single string of extracted text for further use.

Function-4: Divides a large text into smaller, manageable chunks for efficient processing and retrieval.

Steps:

1. Creates a text splitter:
    - Instantiates a CharacterTextSplitter object with:
        - separator="\n": Splits text at newline characters.
        - chunk_size=1000: Aims for chunks of 1000 characters.
        - chunk_overlap=200: Allows 200 characters of overlap between chunks.
        - length_function=len: Uses the len function to measure text length.
2. Splits the text:
    - Calls the split_text method on the splitter to divide the input text into chunks.
3. Returns the chunks:
    - Returns a list of smaller text chunks for further processing or storage.

Key Points:

- Chunking facilitates efficient retrieval and analysis of relevant text segments.
- Overlap ensures smoother context transitions and avoids information loss at chunk boundaries.
- Configurable parameters allow adjustments for different text types and processing needs.

Function-5: Creates a vector store for fast and efficient text similarity search.

Steps:

1. Instantiates embedding model:
    - Creates a SentenceTransformerEmbeddings object using the "all-MiniLM-L6-v2" model, known for efficient sentence-level embeddings.
2. Alternative model (commented out):
    - Notes the option of using HuggingFaceInstructEmbeddings with the "hkunlp/instructor-xl" model for tasks requiring explicit instruction following.
3. Builds FAISS vector store:
    - Constructs a FAISS vector store using:
        - texts=text_chunks: The text chunks to be indexed.
        - embedding=embeddings: The model to generate numerical representations of the text.
4. Returns the vector store:
    - Returns the FAISS vector store, ready for similarity search queries.

Key Points:

- Stores text chunks in a searchable format for rapid retrieval.
- Leverages SentenceTransformers for efficient sentence embeddings.
- Offers flexibility with alternative embedding models for specific tasks.
- Utilizes FAISS for fast and accurate similarity search among text chunks.

Function-6: Sets up a conversational chain to manage conversation flow and retrieve relevant information from the vector store.

Steps:

1. Fetches the language model pipeline:
    - Retrieves the cached llm_pipeline for text generation and understanding.
2. Alternative model (commented out):
    - Notes the option of using a different model from Hugging Face Hub if needed.
3. Creates conversation memory:
    - Initializes a ConversationBufferMemory to store conversation history for context

awareness.

4. Builds conversation chain:
    - Constructs a ConversationalRetrievalChain using:
        - llm: The language model pipeline for text processing and generation.
        - retriever: The vector store for finding relevant text chunks.
        - memory: The conversation memory to track past dialogues.
5. Returns the chain:
    - Returns the conversation chain object, ready to handle conversational interactions.

Key Points:

- Manages conversation flow and context.
- Integrates the language model with the vector store for retrieval and generation.
- Maintains conversation history for context-aware responses.
- Serves as the core framework for DocuChat's conversational interactions

Function-7: Processes user input, generates responses, and updates the chat interface.

Steps:

1. Forwards user question:
    - Passes the user_question to the active conversation chain (st.session_state.conversation) for processing.
2. Receives response and history:
    - Stores the response, including updated chat history, in st.session_state.chat_history.
3. Displays messages:
    - Iterates through the chat history:
        - For even-indexed messages (user's):
            - Uses st.write with user_template to display them.
        - For odd-indexed messages (bot's):
            - Uses st.write with bot_template to display them.
        -

        - Employs unsafe_allow_html=True to permit HTML formatting in templates.

Key Points:

- Handles user input within the conversation chain.
- Updates the chat history for context.
- Manages chat display using Streamlit and templates.
- Separates user and bot messages for visual clarity.
- Allows basic HTML formatting in message content.

Function-8: Orchestrates the main flow of the DocuChat application, handling user interactions and PDF processing.

Steps:

1. Sets up page:
    - Configures page title and icon using Streamlit.
    - Applies CSS styling for visual presentation.
2. Initializes session state:
    - Ensures variables for conversation and chat history exist in session state.
3. Displays chat interface:
    - Presents a header and prompts the user for a question.
    - Call handle_userinput if a question is entered.
4. Provides sidebar for PDF uploads:
    - Offers a file uploader for multiple PDFs.
    - Upon clicking "Process":
        - Displays a spinner during processing.
        - Extracts text from uploaded PDFs using get_pdf_text.
        - Splits text into chunks with get_text_chunks.
        - Builds a vector store for efficient retrieval using get_vectorstore.
        - Creates the conversation chain using get_conversation_chain.
5. Runs the application:

    ○ Executes the main function when the script is run directly.

Key Points:

- Serves as the central hub for user interactions and PDF processing.
- Leverages Streamlit for building a user-friendly web interface.
- Manages conversation state and chat history.
- Integrates PDF processing, text chunking, vector store creation, and conversation chain setup.
- Enables users to chat with the application based on the content of uploaded PDFs.

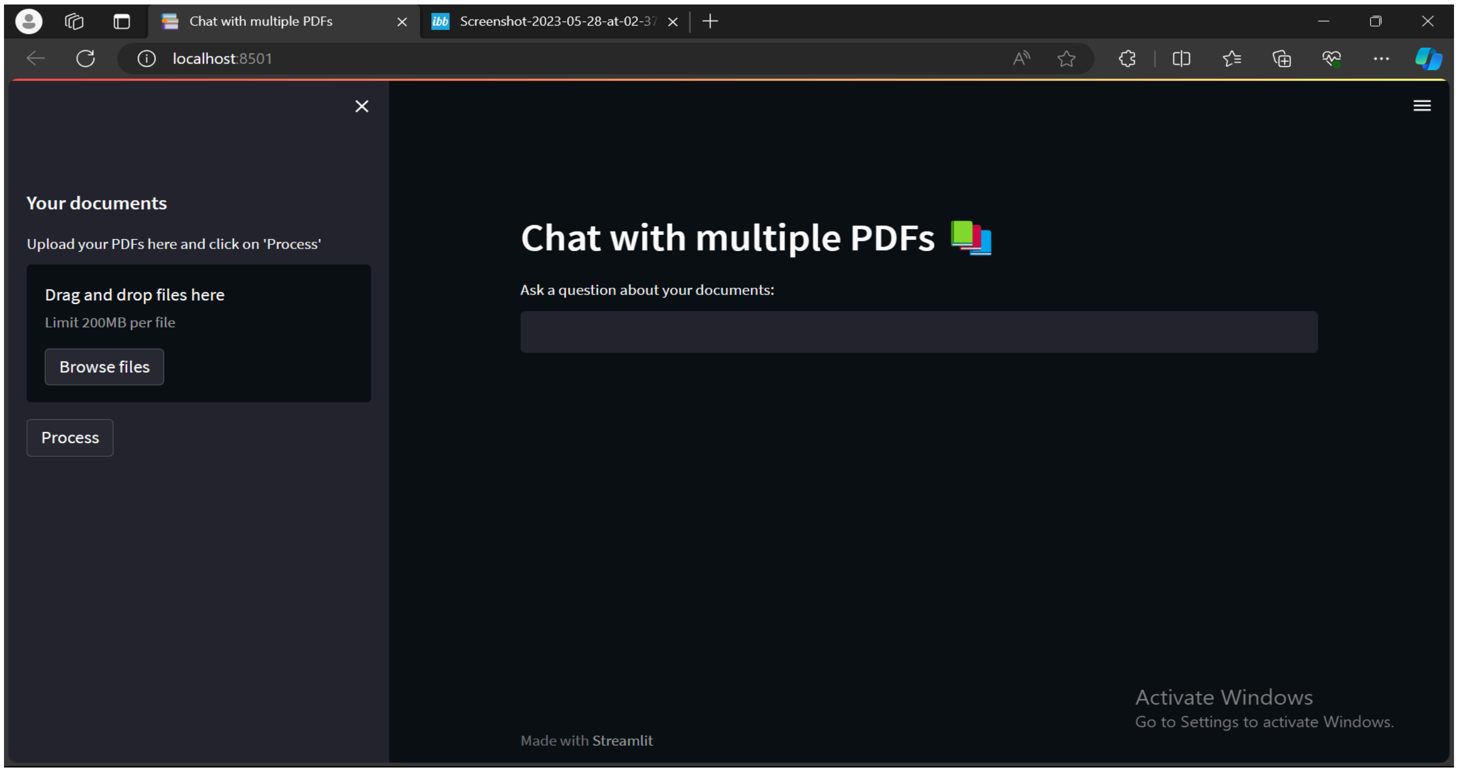
Screen-2:

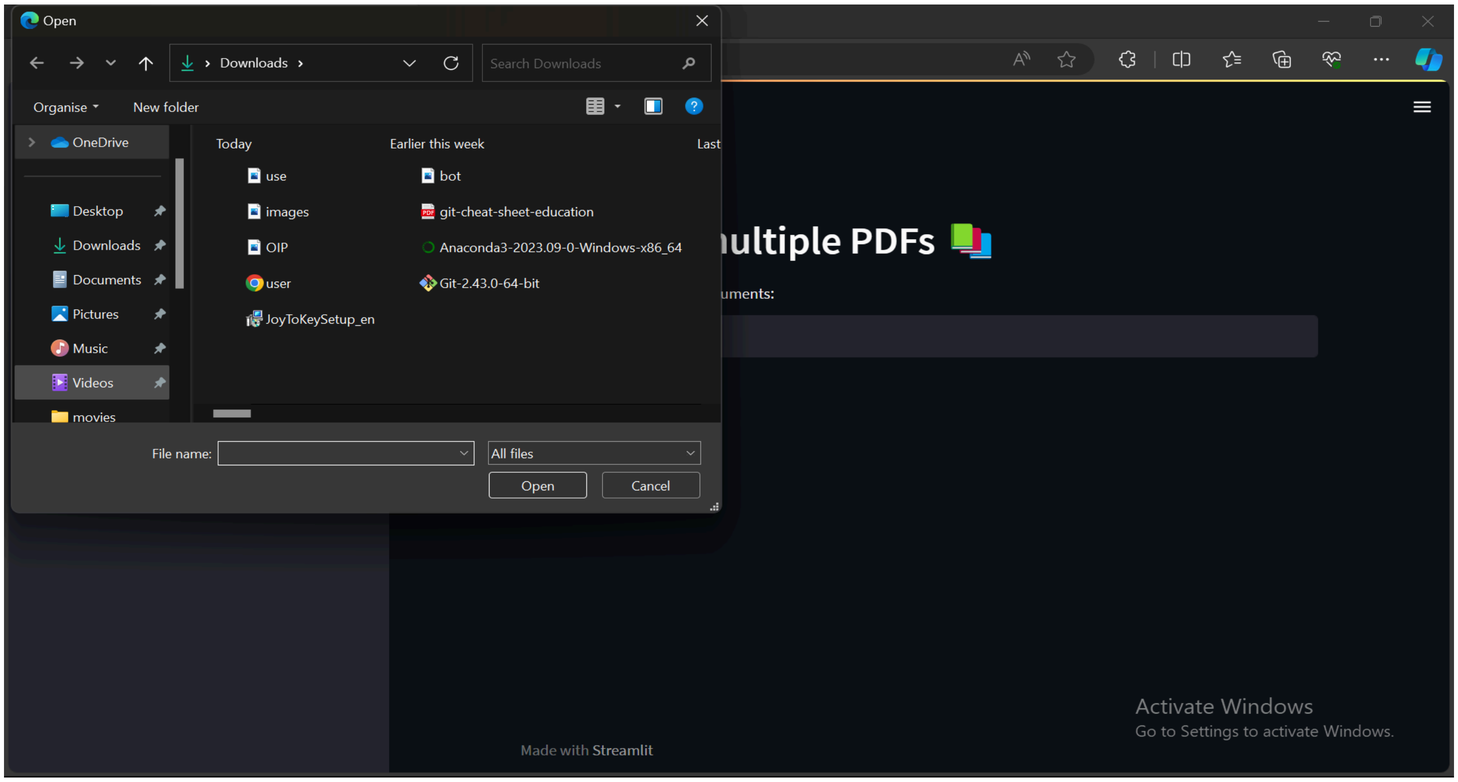
The Language Translation app project involves identifying the source language from user input, preprocessing the text for accuracy, translating it into the desired target language using an API or library, and presenting the translated text to the user through the app's interface. With support for 184 languages, users can convert text or speech into any of these languages effortlessly, enhancing communication across diverse linguistic contexts.

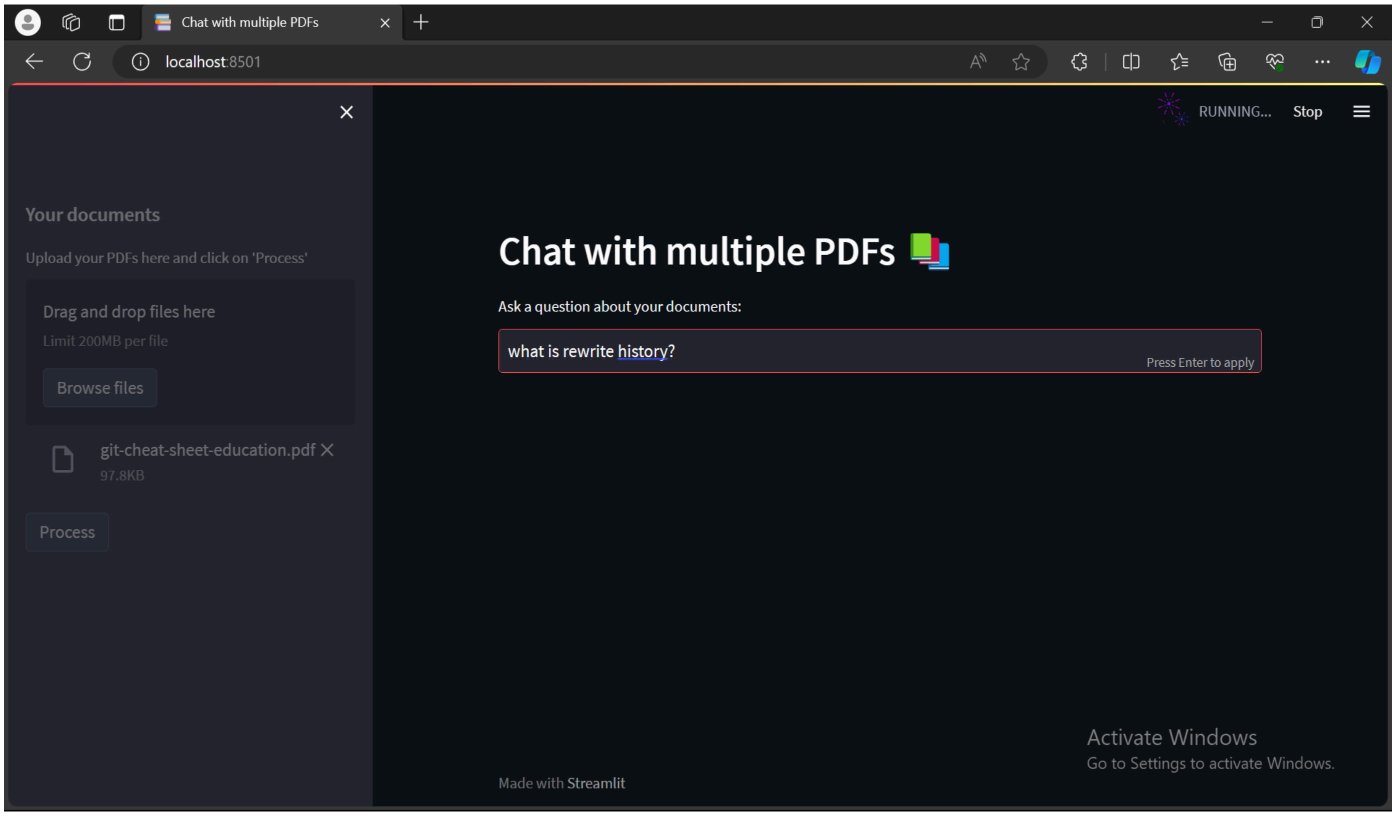
Screen-3:

Extracting Physics and Mathematical equations from .tex LaTeX files entails parsing the document content with TexSoup, a Python library, to identify equations based on their structural patterns. TexSoup's querying capabilities facilitate automated extraction of equations, enabling streamlined tasks for academic research, educational material creation, or technical documentation.

5.2 User Interface Screenshots






Chat with multiple PDFs

localhost:8501

RUNNING... Stop

Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here

Limit 200MB per file

Browse files

git-cheat-sheet-education.pdf

97.8KB

Process

Chat with multiple PDFs 📚

Ask a question about your documents:

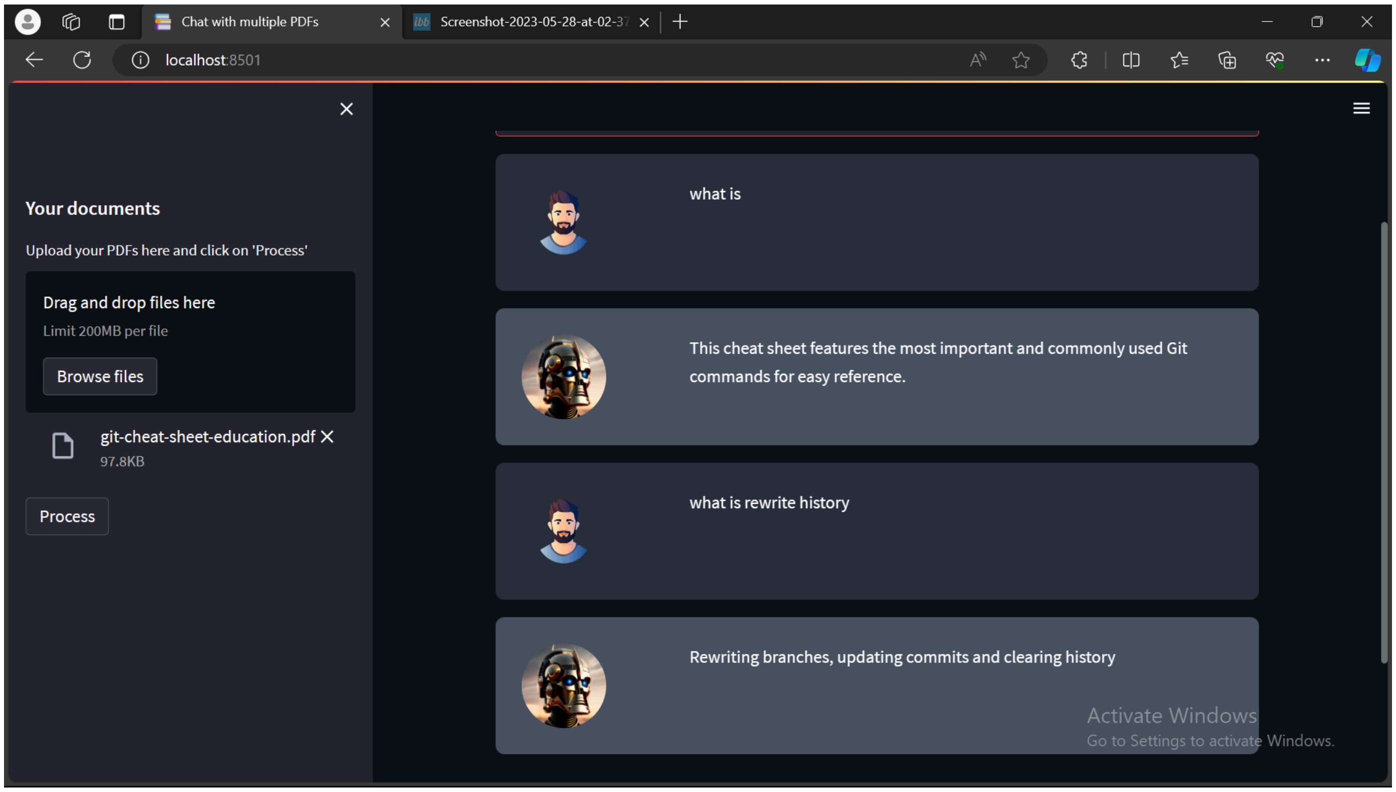
what is rewrite history?

Press Enter to apply

Activate Windows

Go to Settings to activate Windows.

Made with Streamlit


Chat with multiple PDFs

Screenshot-2023-05-28-at-02-37

localhost:8501

Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here

Limit 200MB per file

Browse files

git-cheat-sheet-education.pdf

97.8KB

Process

what is

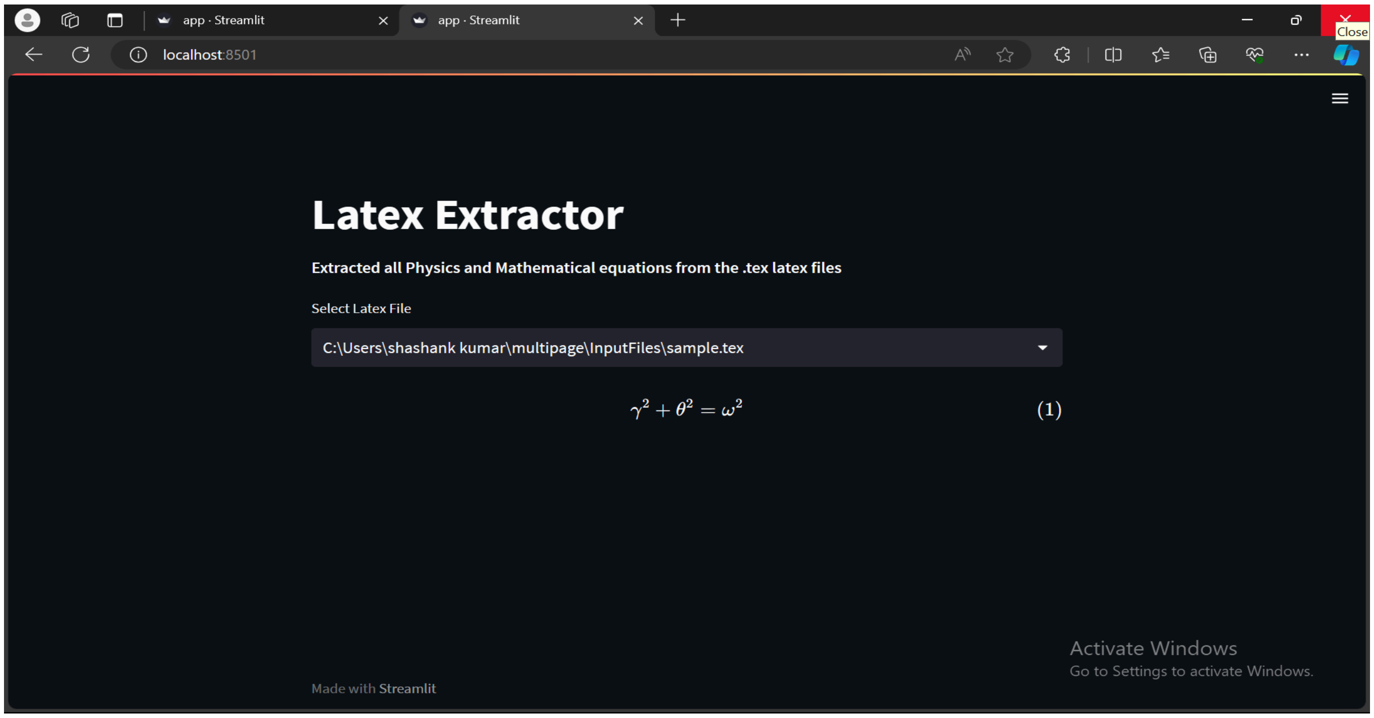
This cheat sheet features the most important and commonly used Git commands for easy reference.

what is rewrite history

Rewriting branches, updating commits and clearing history

Activate Windows

Go to Settings to activate Windows.



Latex Extractor

Extracted all Physics and Mathematical equations from the .tex latex files

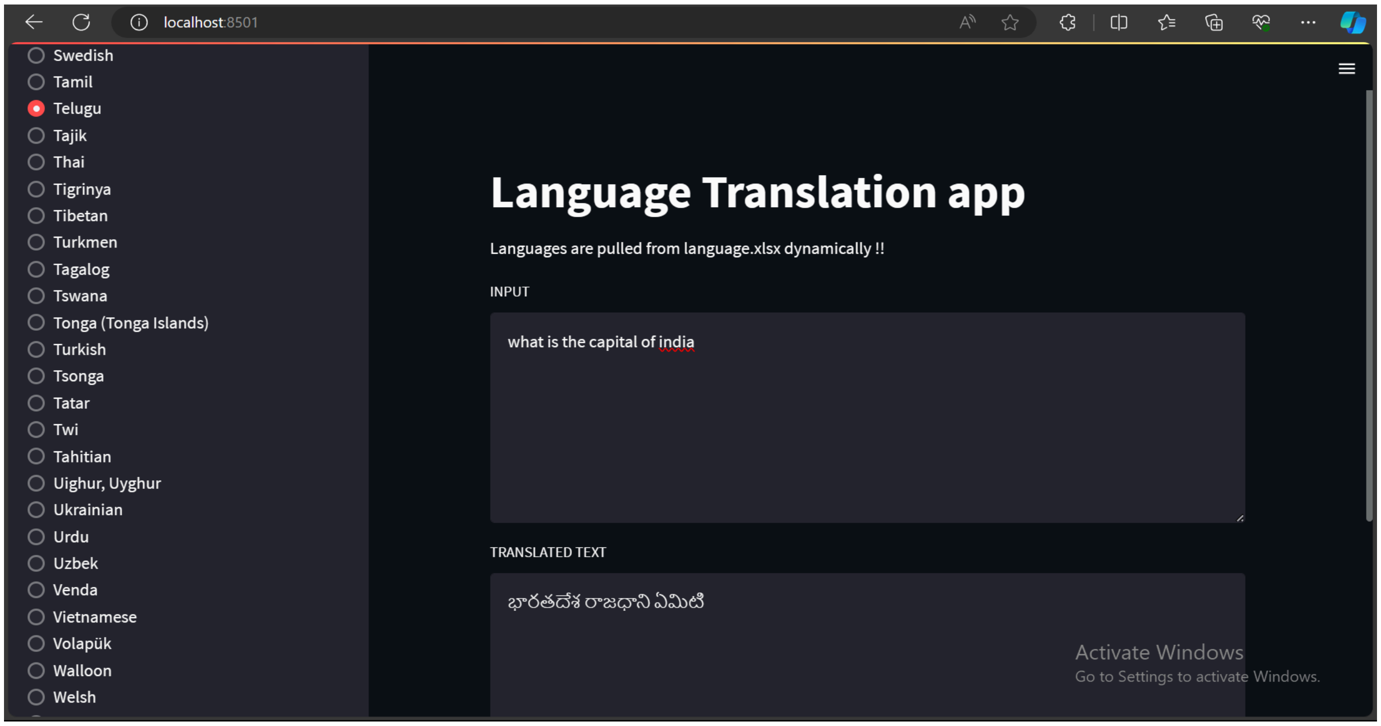
Select Latex File

C:\Users\shashank kumar\multipage\InputFiles\sample.tex

$$\gamma^2 + \theta^2 = \omega^2 \quad (1)$$

Made with Streamlit

Activate Windows
Go to Settings to activate Windows.



Swedish
Tamil
● Telugu
Tajik
Thai
Tigrinya
Tibetan
Turkmen
Tagalog
Tswana
Tonga (Tonga Islands)
Turkish
Tsonga
Tatar
Twi
Tahitian
Uighur, Uyghur
Ukrainian
Urdu
Uzbek
Venda
Vietnamese
Volapük
Walloon
Welsh

Language Translation app

Languages are pulled from language.xlsx dynamically !!

INPUT

what is the capital of india

TRANSLATED TEXT

భారతదేశ రాజధాని ఏమిటి

Activate Windows
Go to Settings to activate Windows.

CHAPTER 6: SYSTEM TESTING

6.1 System Testing

Comprehensive testing was conducted to ensure the functionality, performance, security, usability, and maintainability of the Chat with Multiple PDFs system. The following testing phases were undertaken:

1. Functional Testing

Objective: Verify that all system functionalities operate as intended.

Key Tests:

- User Input: Successfully processed various natural language queries, including simple, complex, and multi-part questions.
- Document Processing: Accurately loaded and extracted text from different PDF formats and sizes.
- Conversation Chain: Generated relevant, coherent responses, maintaining context across multiple interactions.

6.2 Performance Testing

Objective: Assess the system's responsiveness, scalability, and resource utilization.

Key Tests:

- Load Testing: Maintained stability and acceptable response times under varying load conditions, including peak usage scenarios.
- Scalability Testing: Effectively handled increasing numbers of PDFs and users, demonstrating performance resilience.

6.3 Usability Testing

Objective: Evaluate the system's user-friendliness and accessibility.

Key Tests:

- Interface Evaluation: Confirmed the intuitiveness and clarity of the chat interface, promoting a positive user experience.
- Accessibility Testing: Ensured compatibility with assistive technologies and adherence to accessibility standards.

CHAPTER 7: CONCLUSION

Envision delving into multiple PDFs effortlessly, steering clear of tedious keyword searches, and instead, engaging in natural conversation. The "Chat with Multiple PDFs" initiative transforms this vision into reality by employing advanced language models and lightning-fast vector stores. The collaboration between LaMini-Flan-T5-248M, an intricate language model, and FAISS, a swift vector store, allows for intuitive interaction with uploaded documents through plain English queries and adding the functionalities like exacting formulas from the LateX file and translation of the text.

This project surpasses mere information retrieval, as LaMini-Flan-T5-248M comprehends questions and extracts pertinent passages from PDFs, facilitated by FAISS for rapid and precise document navigation. The system not only recalls past queries but also establishes a context-aware conversation, facilitating follow-up questions and seamless exploration of specific topics.

The implications are vast—students can delve deeper into research papers, professionals can dissect complex reports, and businesses can create AI assistants adept at addressing customer inquiries. However, this marks just the project's inception. Subsequent iterations promise even more nuanced interactions, tailored to specific domains and enriched with visualization tools. As this technology evolves, it foretells a shift in how we interact with information—an era where document exploration is intuitive and effortless, driven by conversation. The prospect of facing a stack of PDFs now comes with the assurance of an engaging AI companion, poised to unveil their secrets, one question at a time.

CHAPTER 8: FUTURE SCOPE

Enhanced Response Generation:

Domain-Specific Fine-Tuning: Explore fine-tuning the language model on domain-specific data for more accurate and informative responses.

Diverse Text Generation Techniques: Experiment with summarization and question-answering techniques to tailor responses to specific user needs.

Multilingual Conversations: Introduce text translation capabilities for conversations in multiple languages, expanding accessibility and overcoming linguistic barriers.

Multi-Modal Integration: Incorporate multi-modal capabilities for handling diverse document formats.

Continuous Model Fine-Tuning: Ensure continuous adaptation to evolving language patterns for sustained performance improvements.

Improved Information Retrieval:

Advanced Embedding Models: Investigate alternative embedding models or vector store configurations for more precise retrieval of relevant text segments.

Semantic Search Capabilities: Explore incorporating semantic search to better understand user intent and context.

Text Summarization Features: Add text summarization features for concise overviews, enhancing comprehension and efficiency.

Conversational Context Awareness:

History Utilization: Strengthen the conversation chain's ability to track and utilize conversation history for more consistent and context-aware responses.

External Knowledge Integration: Explore ways to incorporate external knowledge sources or databases for enriched conversations.

User Experience Refinements:

Chat Interface Enhancements: Improve the chat interface with features like conversation summaries, follow-up question suggestions, and interactive visualizations.
Transparency and Export: Provide options for saving and exporting chat transcripts for reference or analysis.
Language Switching: Seamlessly integrate text translation into the chat interface for effortless language switching.
Summarization Options: Offer summarization options for both user questions and retrieved text for flexible information consumption.

Evaluation and Testing:

User Studies: Conduct user studies to assess effectiveness in various use cases and gather feedback for further improvements.
Performance Metrics: Establish metrics for accuracy, relevance, and user satisfaction, including translation and summarization quality.

Large-Scale Deployment:

Optimized Performance: Optimize performance for handling large document collections and multiple concurrent users.
Cloud-Based Deployment: Explore cloud-based deployment options for broader accessibility and scalability.

CHAPTER 9: REFERENCES

Langchain:	https://python.langchain.com/docs/get_started/
Streamlit:	https://docs.streamlit.io
Hugging Face	https://huggingface.co/models
LLM & Langchain	https://www.youtube.com/@AIAnytime
	https://www.youtube.com/@engineerprompt
Text Soup and Translate Packages	https://pypi.org/project/

[1] Coavoux, Maximin, Hady Elsahar, and Matthias Gallé. "Unsupervised aspect-based multi-document abstractive summarization." Proceedings of the 2nd Workshop on New Frontiers in Summarization. 2019.

[2] Huang, Kung-Hsiang, et al. "Embrace divergence for richer insights: A multi-document summarization benchmark and a case study on summarizing diverse information from news articles." arXiv preprint arXiv:2309.09369 (2023).

[3] Kurisinkel, Litton J., and Nancy F. Chen. "LLM Based Multi-Document Summarization Exploiting Main-Event Biased Monotone Submodular Content Extraction." arXiv preprint arXiv:2310.03414 (2023).

[4] Laban, Philippe, et al. "SummEdits: Measuring LLM Ability at Factual Reasoning Through The Lens of Summarization." Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. 2023.