PDFChain: Interaction with Documents using LangChain

A PROJECT REPORT

Submitted by

ANAHITA GUPTA [RA2011003010189]
SANCHITA GHOSH [RA2011003010191]

Under the Guidance of

Dr. MADHUMITHA K

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L/18CSP111L project report titled "**PDFChain: Interaction with Documents using Langchain**" is the bonafide work of **Anahita Gupta [RA2011003010189], Sanchita Ghosh [RA2011003010191]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. MADHUMITHA K
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. M. ELIAZER
PANEL HEAD
Assistant Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

INTERNAL EXAMINER

EXTERNAL EXAMINER

Department of Computing Technologies

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course	: B.Tech in Computer Science and Engineering
Student Names	: Anahita Gupta, Sanchita Ghosh
Registration Number	: RA2011003010189, RA2011003010191
Title of Work	: PDFChain:Interaction with Documents using Langchain

I/We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate.
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website.

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We also extend our sincere thanks to **T.V.Gopal**, Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

And we thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support.

We wish to thank **Dr. M. Pushpalatha**, Professor & Head, Department of Computing Technologies, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We want to convey our thanks to our Project Coordinators, **Dr. M. Kanchana**, Associate Professor, **Dr. Godfrey Winster S**, Associate Professor, Panel Head, **Dr. Eliazer M**, Assistant Professor (Senior Grade), and Panel Members, **Dr. Madhumitha K,** Assistant Professor, Dr. **P. Kirubanantham**, Assistant Professor, **Dr. A. Devi Priya**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support

.

We register our immeasurable thanks to our Faculty Advisors, **Dr. Vinod D,** Assistant Professor and **Dr. Umadevi M**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Madhumitha K**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of the Computing Technologies Department, School of Computing, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

Anahita Gupta [RA2011003010189]

Sanchita Ghosh [RA2011003010191]

ABSTRACT

This project introduces a novel approach to developing PDF chatbots aimed at enhancing user interactions with PDF content. Leveraging the LangChain framework and the LLM Model, our chatbot system is designed to provide comprehensive and contextually relevant responses to user inquiries regarding PDF documents.

Our methodology focuses on integrating real-world information seamlessly into the conversation flow. This is facilitated by the advanced language processing capabilities of the LLM Model, enabling the chatbot to interpret PDF content effectively. To optimize information retrieval, we employ FAISS, a high-performance similarity search library, for storing PDF file vectors. The user interaction aspect of our chatbot is prioritized through the development of a user-friendly interface using Streamlit. This ensures ease of use and contributes to an intuitive user experience.

Through rigorous evaluation and experimentation, we demonstrate the superiority of our approach over traditional rule-based systems in terms of understanding and responsiveness to user inquiries. The results highlight the chatbot's accuracy and fluency in addressing queries related to PDF content.

In conclusion, this project represents a significant advancement in chatbot development, offering a sophisticated solution for handling PDF-related inquiries with unparalleled accuracy and efficiency. The integration of cutting-edge technologies and innovative design principles positions our chatbot for impactful applications in customer service, education, and research domains.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
LLM	Large Language Models
FAISS	Facebook AI Similarity Search
NLU	Natural Language Understanding
NLP	Natural Language Processing
GPT	Generative Pre-Trained Transformers
BERT	Bidirectional Encoder Representations from Transformers
NLIDB	Natural language interfaces for databases

CHAPTER – 1
INTRODUCTION

1.1 General

PDFs stand as a ubiquitous format in business, education, and research, yet their static nature often hampers accessibility and interactivity. In response, the integration of artificial intelligence (AI), particularly through chatbots, emerges as a promising solution. Chatbots offer a natural and intuitive interface for users to interact with digital systems, making them ideal for addressing the challenges posed by PDFs.

This project represents a pioneering effort in creating a PDF-centric chatbot, leveraging advanced technologies including LangChain and the LLM Model. LangChain, a versatile framework, streamlines the development of chatbots and scalable AI applications, while the LLM Model serves as a robust language processing tool capable of generating text, translating languages, and providing insightful responses to user inquiries.

At the core of our endeavor lies FAISS, a pivotal component facilitating the storage of PDF file vectors. By storing embeddings and PDF text in its vector repository, efficient retrieval of related documents is enabled, thereby enhancing the chatbot's utility and effectiveness. Complementing this backend infrastructure, Streamlit is employed for front-end development, allowing for the creation of a user-friendly web interface for seamless interaction with the chatbot.

Our chatbot showcases proficiency in both answering queries related to PDF content and generating text reminiscent of the original documents. Through rigorous testing across diverse PDF files, the chatbot demonstrates commendable accuracy, affirming its efficacy in document management tasks.

This project report delves into the implementation of our project, elucidating the technological underpinnings and methodologies employed. Leveraging LangChain, the LLM

Model, FAISS, and Streamlit, our aim is to showcase the transformative potential of AI in revolutionizing document management processes. Furthermore, we discuss the implications of our research and its potential to enhance productivity, accessibility, and efficiency across various domains.

1.2 Natural Language Understanding

Natural Language Understanding (NLU) stands as a cornerstone in the field of artificial intelligence, facilitating machines' comprehension of human language in its varied forms and complexities. At its core, NLU encompasses the ability to interpret, analyze, and derive meaning from natural language inputs, enabling machines to engage in meaningful interactions with users, understand their intentions, and respond appropriately. This multifaceted process involves several key components, including syntactic parsing, semantic analysis, discourse understanding, and pragmatic interpretation.

Syntactic parsing involves the analysis of sentence structure, identifying grammatical elements such as nouns, verbs, and modifiers, and determining how they relate to each other within the sentence. Semantic analysis delves deeper into the meaning of words and phrases, considering their contextual usage and connotations to derive intended meanings. Discourse understanding focuses on comprehending the broader context of conversations, including references to previous statements, implicit meanings, and conversational implicatures. Pragmatic interpretation considers the pragmatic aspects of language use, such as understanding humor, sarcasm, and implied intentions.

Advancements in NLU have been driven by the development of sophisticated machine learning algorithms, particularly deep learning models like transformers, which have revolutionized natural language processing tasks. Models such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) have demonstrated remarkable capabilities in understanding and generating human-like text, leveraging large-scale pre-training on vast amounts of textual data.

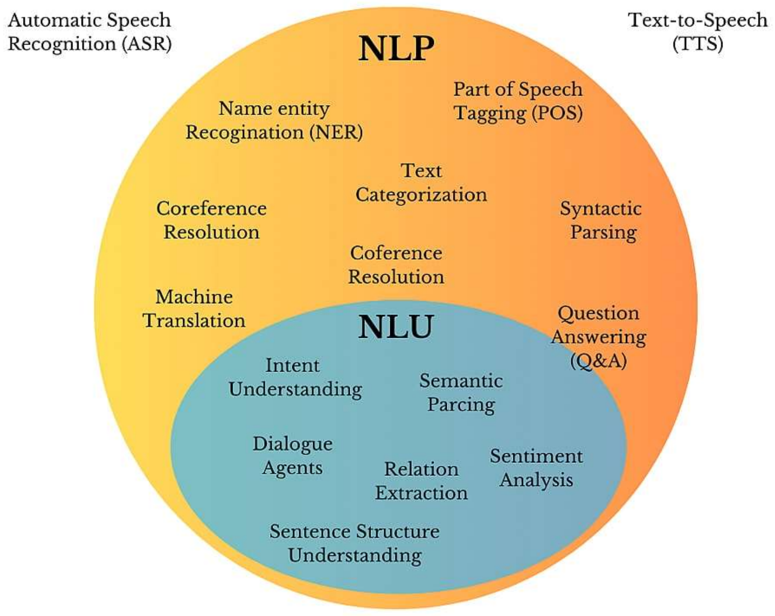


Fig 1.1 Natural Language Understanding

The applications of NLU are vast and varied, spanning across industries and domains. In customer service, NLU powers chatbots and virtual assistants capable of understanding and addressing customer inquiries in natural language. In healthcare, NLU enables the analysis of medical texts and patient records, facilitating clinical decision-making and research. In education, NLU supports language learning platforms and automated grading systems. Moreover, the integration of NLU into search engines, recommendation systems, and information retrieval systems enhances user experience and information access on digital platforms.

As NLU continues to advance, its impact on human-computer interaction, information access, and decision-making processes will only grow, driving innovation and transformation across diverse fields and applications.

1.3 Large Language Model

Large Language Models (LLMs) represent a significant milestone in the field of natural language processing (NLP), offering unprecedented capabilities in understanding, generating, and manipulating human language at scale. These models, typically based on transformer architecture, are characterized by their vast size, comprising hundreds of millions to billions of parameters trained on massive text corpora. The sheer scale of LLMs enables them to capture intricate patterns, nuances, and semantic relationships within language data, resulting in remarkable performance across a wide range of NLP tasks. Key examples of LLMs include OpenAI's GPT (Generative Pre-trained Transformer) series, Google's BERT (Bidirectional Encoder Representations from Transformers), and Facebook's RoBERTa (Robustly optimized BERT approach).

One of the most notable features of LLMs is their ability to perform contextualized language processing. Unlike traditional NLP models that treat words in isolation, LLMs consider the surrounding context to infer meaning and make predictions. This contextual understanding is crucial for tasks such as text generation, sentiment analysis, machine translation, and question answering, where the meaning of a word or phrase depends on its surrounding context. By capturing long-range dependencies and contextual cues, LLMs excel in generating coherent and contextually relevant text, mimicking human-like language production to a remarkable extent.

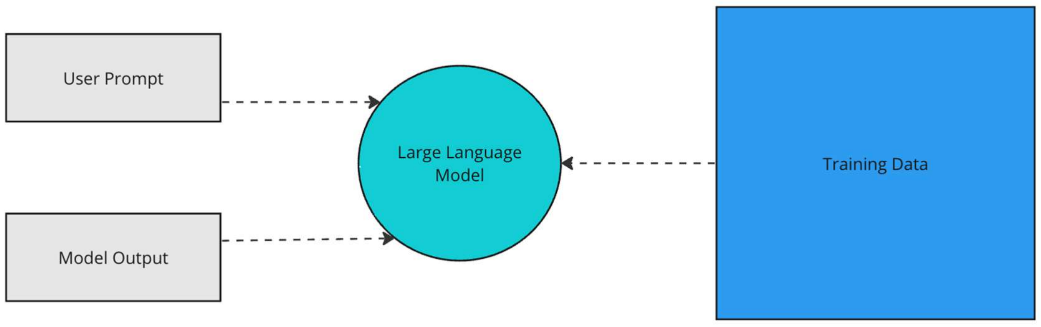


Fig 1.2 Large Language Model

Furthermore, LLMs have proven to be highly adaptable and versatile, capable of fine-tuning on specific tasks or domains with minimal additional training. This transfer learning paradigm, where a pre-trained LLM is fine-tuned on downstream tasks using task-specific data, allows for rapid development of high-performance NLP models for various applications. From sentiment analysis in social media to language translation in healthcare, LLMs have demonstrated their effectiveness across diverse domains and use cases, driving innovation and accelerating progress in NLP research and applications.

Despite their impressive capabilities, LLMs are not without challenges and controversies. Concerns about biases in training data, ethical implications of text generation, and environmental impact of training large models have sparked debates within the AI community and beyond. Nevertheless, the potential benefits of LLMs in improving human-computer interaction, advancing natural language understanding, and powering a new generation of AI applications are undeniable, underscoring their significance as a transformative technology in the realm of NLP and beyond.

CHAPTER -2

LITERATURE SURVEY

2.1 Motivation

The rapid evolution of artificial intelligence over the past decade has been characterized by significant advancements in various domains, driven by the adoption of deep learning techniques, particularly neural networks. Milestones such as the ImageNet Large Scale Visual Recognition Challenge have underscored the remarkable capabilities of AI in tasks like image recognition. Additionally, innovations like DeepMind's AlphaGo and AlphaZero have demonstrated the potential of reinforcement learning in complex gaming scenarios. Concurrently, the field of natural language processing has undergone transformative changes with the emergence of advanced models like BERT, GPT, and T5, which have significantly improved machine translation, sentiment analysis, and text generation. These advancements are epitomized by Large Language Models (LLMs) [1], which have garnered widespread attention due to their human-like proficiency in generating contextually pertinent and grammatically coherent text. Notably, models like OpenAI's ChatGPT have popularized the usage of LLMs among millions of users, showcasing their effectiveness in various tasks such as essay composition, code writing, explanation, and debugging.

However, the development of custom AI applications utilizing LLMs necessitates more than just interaction via a web interface. Addressing this need, LangChain—an open-source software library—has emerged as a solution for expediting the development of bespoke AI applications. LangChain's modular abstractions and customizable pipelines offer a framework for swiftly developing LLM-based applications tailored to specific use cases. In their paper, Topsakal and Akinci emphasized the significance of LangChain in facilitating the development of custom AI applications[2].

In recent years, the proliferation of LLMs and their integration into frameworks like LangChain has sparked interest across diverse fields, including education, research, customer service, content creation, healthcare, and entertainment [3]. This underscores the potential for leveraging LLM technology to address various challenges and provide innovative solutions in numerous domains.

In the realm of medical imaging, particularly breast ultrasound (BUS) analysis, the integration of advanced technologies has revolutionized diagnostic processes. The utilization of Large Language Models (LLMs) in conjunction with LangChain represents a significant advancement in streamlining the creation of comprehensive BUS reports.

LLMs, trained on vast text corpora, exhibit unparalleled proficiency in understanding language nuances and world knowledge, making them indispensable tools in various domains, including medicine. The LangChain framework facilitates the seamless integration of specialized tools for specific objectives, such as medical report generation. With LangChain's context-driven decision-making capabilities, the proposed method outlined by Huh et al. leverages a collection of specialized tools to analyze BUS images comprehensively [4]. Unlike previous methods, this approach excels in recognizing peripheral areas crucial for report generation, ensuring the accuracy and completeness of generated reports.

Furthermore, the utilization of straightforward classification networks within LangChain ensures ease of training and facilitates straightforward upgrades, contributing to the scalability and adaptability of the proposed method.

The integration of LLMs and LangChain in medical imaging represents a promising avenue for enhancing diagnostic accuracy, reducing the burden on healthcare professionals, and improving patient care outcomes.

The recent advent of powerful Large-Language Models (LLMs) offers a new conversational approach to historical research, allowing historians to interact with training data in novel ways. By augmenting LLMs with vector embeddings from specialized academic sources, researchers in the Humanities can engage in conversational methodologies. The research paper by Garcia et al. evaluates how LLMs can assist researchers in examining customized corpora of various document types, including primary and secondary sources [5]. The evaluation focuses on the performance of LLMs in question-answering and data extraction tasks compared to traditional search interfaces. The study demonstrates that LLMs equipped with specialized sources can provide semantic retrieval and reasoning abilities tailored to specific research projects, enhancing the research process in History and the Humanities. The authors emphasize the potential of LLMs to accelerate research processes and improve the quality of research outputs.

Incorporating LLMs into historical research praxis has the potential to revolutionize how traditional archives are explored, primary sources are analyzed, and historical narratives are constructed. While concerns exist regarding the potential misuse and limitations of LLMs, such as their inability to discern truth from falsehood, the authors propose a solution by combining LLMs with carefully curated corpora of peer-reviewed sources. This approach aims to mitigate errors and produce texts rooted in validated academic sources. Overall, the integration of LLMs with historical research methodologies holds promise for advancing scholarship in the Humanities and beyond [6].

Natural language interfaces for databases (NLIDBs) have also gained attention, particularly with the availability of Large Language Models (LLMs). The introduction of conversational interfaces like ChatGPT has facilitated the development of NLIDBs by simplifying the process for developers. Nascimento et al. propose a family of NLIDBs leveraging ChatGPT and LangChain features to process natural language (NL) queries into SQL queries or to extract keywords for database searches [7]. By using ChatGPT, NL questions can be translated into SQL queries or keywords extracted from text blocks, enhancing the ease of query processing. Additionally, LangChain allows for the automation of SQL query generation by providing context to ChatGPT based on database schema and table information. This approach streamlines the development of NLIDBs and offers flexibility in query processing. The paper concludes by comparing different NLIDBs within the proposed family.

An example would be the implementation of chatbots in higher education institutions, which marks a significant advancement in student support services, catering to the evolving needs of modern students in a digital age [8]. LangChain serves as a foundational framework for the development of chatbots utilizing LLMs, offering a streamlined approach to creating bespoke AI applications tailored to the specific requirements of higher education institutions.

2.2 Objective

The objective of this literature survey is to delve into the multifaceted applications of Large Language Models (LLMs) and LangChain across diverse domains. Our aim is to showcase the transformative impact that LLMs have had on various fields, including natural language processing, medical imaging, historical research, and database interfaces. Additionally, we seek to highlight the pivotal role played by LangChain in expediting the development of custom AI applications that leverage LLM technology.

By examining the potential of LLMs and LangChain in enhancing diagnostic accuracy, accelerating research processes, and streamlining database query processing, we aim to provide insights into the challenges and opportunities associated with their integration in real-world applications. Furthermore, we intend to discuss considerations for ethical usage and algorithmic biases in deploying LLMs and LangChain. Through this comprehensive exploration of motivations and objectives, our literature survey aims to offer a nuanced understanding of the evolving landscape of AI technologies and their implications across various industries and scholarly disciplines.

CHAPTER -3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Block/Architecture Diagram

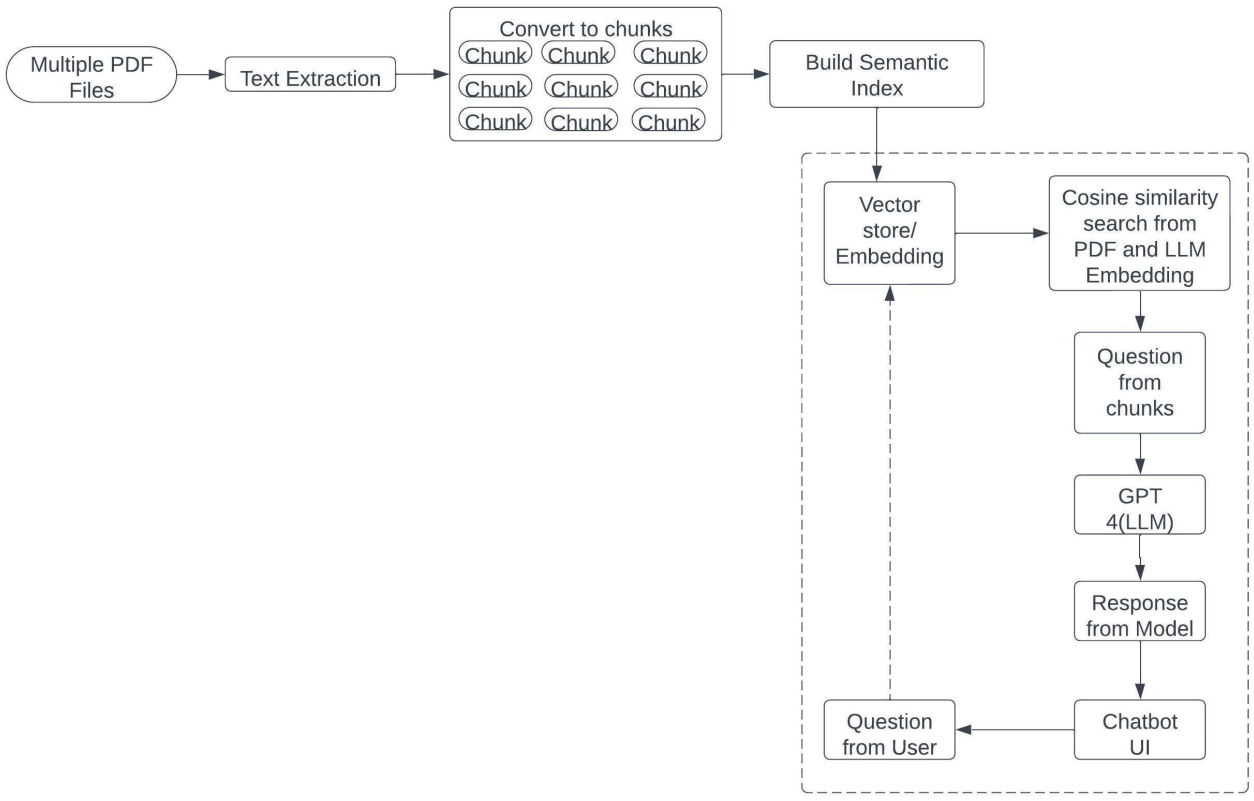


Fig 3.1 Block diagram for PDFChain : Interaction with Documents using LangChain

The process, at its core, is described as follows:

1. **User Interface:** The chatbot's interface facilitates seamless user interaction through a text input field for querying and receiving responses.

2. **Natural Language Understanding (NLU):** This component processes and interprets user queries using NLP techniques, including tokenization, part-of-speech tagging, named entity recognition, and syntactic parsing.

3. **Vector Store:** A repository storing vector representations of text chunks extracted from PDF documents, enabling efficient information retrieval.

4. **Embeddings**: These capture semantic and contextual information from text chunks, facilitating comparison and retrieval by encoding textual information into dense vector representations using pre-trained language or embedding models.

5. **Large Language Models (LLMs):** Pre trained models providing advanced language understanding and generation capabilities, fine-tuned for specific tasks such as conversational retrieval.

3.2 Frontend Design

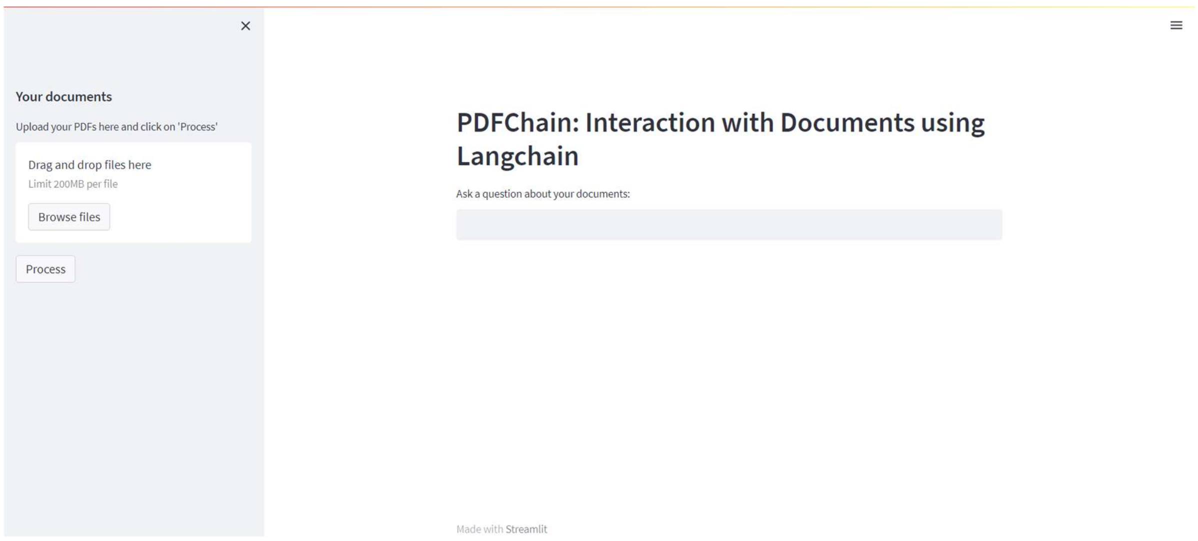

Your documents
Upload your PDFs here and click on 'Process'
Drag and drop files here
Limit 200MB per file
Browse files
Process
PDFChain: Interaction with Documents using Langchain
Ask a question about your documents:
Made with Streamlit

Fig 3.2 User Interface for PDFChain

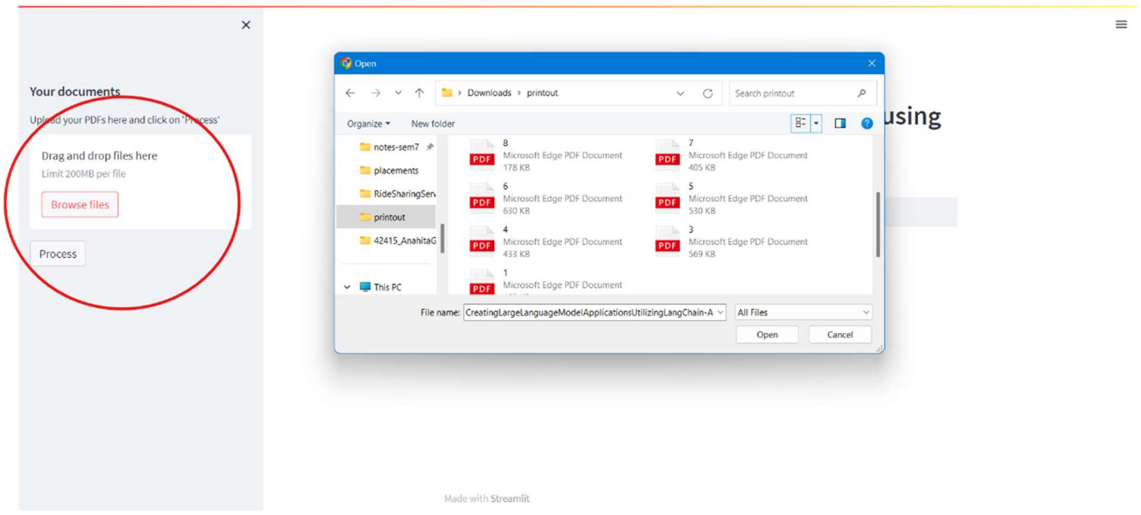
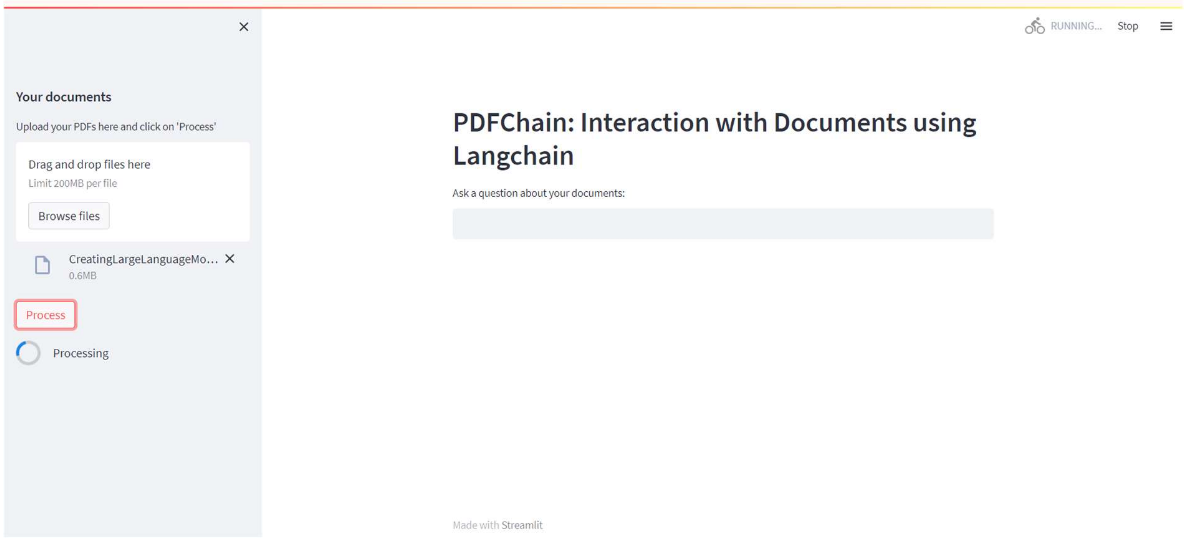

Your documents
Upload your PDFs here and click on 'Process'
Drag and drop files here
Limit 200MB per file
Browse files
Process
Open
Downloads › printout
Search printout
Organize
New folder
notes-sem7
placements
RideSharingServ
printout
42415_AnahitaG
This PC
8 Microsoft Edge PDF Document 178 KB
7 Microsoft Edge PDF Document 405 KB
6 Microsoft Edge PDF Document 630 KB
5 Microsoft Edge PDF Document 530 KB
4 Microsoft Edge PDF Document 433 KB
3 Microsoft Edge PDF Document 569 KB
1 Microsoft Edge PDF Document
File name: CreatingLargeLanguageModelApplicationsUtilizingLangChain-A
All Files
Open
Cancel
using
Made with Streamlit

Fig 3.3 Browse for PDF Files on Local Desktop


Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here

Limit 200MB per file

Browse files

CreatingLargeLanguageMo... 0.6MB

Process

Processing

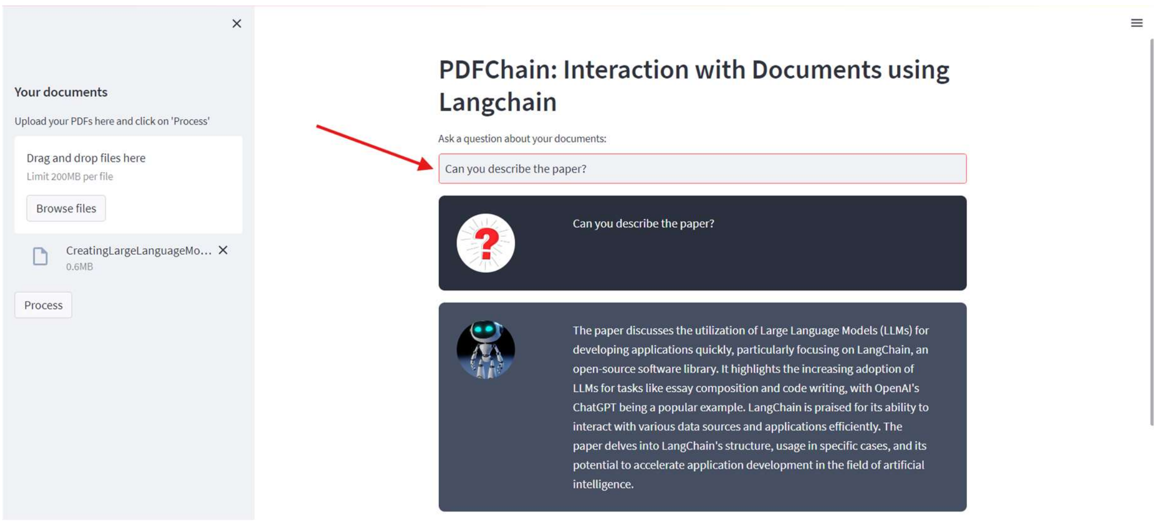
RUNNING... Stop

PDFChain: Interaction with Documents using Langchain

Ask a question about your documents:

Made with Streamlit

Fig 3.4 Process one or more PDF Files


Your documents

Upload your PDFs here and click on 'Process'

Drag and drop files here

Limit 200MB per file

Browse files

CreatingLargeLanguageMo... 0.6MB

Process

PDFChain: Interaction with Documents using Langchain

Ask a question about your documents:

Can you describe the paper?

Can you describe the paper?

The paper discusses the utilization of Large Language Models (LLMs) for developing applications quickly, particularly focusing on LangChain, an open-source software library. It highlights the increasing adoption of LLMs for tasks like essay composition and code writing, with OpenAI's ChatGPT being a popular example. LangChain is praised for its ability to interact with various data sources and applications efficiently. The paper delves into LangChain's structure, usage in specific cases, and its potential to accelerate application development in the field of artificial intelligence.

Fig 3.5 Input question about the documents

CHAPTER -4
METHODOLOGY

4.1 Importing Libraries

During the initial phase of implementation, the chatbot integrates essential libraries to enable its functionality and streamline development processes.

1. **Streamlit:** Streamlit stands as a powerful Python library renowned for swiftly developing interactive web applications. It eliminates the complexities typically associated with web development, allowing developers to focus on functionality rather than infrastructure. Streamlit fosters an intuitive and user-friendly interface, enhancing user engagement with the chatbot. Through Streamlit, developers can effortlessly create interactive components such as text inputs, file uploaders, and data visualizations, thereby enhancing the overall user experience.

2. **dotenv:** The dotenv library serves as a pivotal tool for securely managing environment variables within the chatbot application. Environment variables often encapsulate sensitive information such as API keys, credentials, and configuration parameters. By utilizing dotenv, developers can store such sensitive data outside of the codebase, enhancing security and minimizing the risk of exposure. This approach aligns with best practices in software development and ensures robust security measures within the chatbot application.

3. **PyPDF2:** PyPDF2 emerges as a indispensable Python library designed specifically for manipulating PDF files. Its rich feature set facilitates various operations, including text extraction, merging, splitting, and encryption. Within the context of the chatbot implementation, PyPDF2 plays a vital role in extracting textual content from PDF documents. By leveraging PyPDF2, the chatbot gains the capability to parse PDF files

and extract relevant information, which is essential for subsequent processing and analysis.

4. **Langchain library components**: The Langchain library represents a comprehensive suite of components tailored for natural language understanding and processing tasks. These components encompass a wide array of functionalities, ranging from basic text processing techniques to advanced conversational agents. Within the chatbot implementation, Langchain components play a pivotal role in facilitating tasks such as text chunking, conversation management, and semantic analysis. By leveraging the capabilities offered by the Langchain library, developers can streamline the implementation of advanced features and enhance the overall intelligence of the chatbot.

4.2 Text Chunking

Text chunking is a crucial preprocessing step that involves dividing the extracted text from PDF documents into smaller, manageable segments or chunks. This segmentation enhances the efficiency of information processing and analysis within the chatbot.

The CharacterTextSplitter class, a component of the Langchain library, serves as a fundamental tool for text chunking. Rather than simply breaking text into fixed-length segments, this class employs sophisticated algorithms to segment text based on specific criteria, such as character limits or punctuation marks. By leveraging linguistic patterns and contextual cues, the CharacterTextSplitter intelligently identifies natural breakpoints within the text, ensuring that each chunk encapsulates coherent and semantically meaningful content.

In practical terms, text chunking enables the chatbot to handle large volumes of text more effectively. By breaking lengthy passages into smaller segments, the chatbot can process and analyze information incrementally, thereby reducing computational overhead and improving response times. Additionally, text chunking facilitates more granular analysis of textual content, enabling the chatbot to extract key insights and generate contextually relevant responses with greater precision.

Overall, text chunking represents a foundational technique in natural language processing, empowering the chatbot to navigate and interpret textual data with finesse and accuracy. Through the use of advanced algorithms and linguistic insights, the CharacterTextSplitter class enables the chatbot to efficiently manage and manipulate text-based information, enhancing its overall effectiveness in information retrieval and response generation.

4.3 Vector Store Creation

Creating a vector store is a crucial step in the implementation of the chatbot, particularly for organizing and managing embeddings associated with text chunks extracted from PDF documents. This section delves into the intricate details of each subtopic involved in the vector store creation process:

4.3.1 Embeddings

Embeddings play a pivotal role in natural language processing tasks, serving as numerical representations of words or phrases in a high-dimensional vector space. These representations capture semantic relationships between words, enabling the chatbot to understand the contextual meaning of text chunks more effectively. For instance, words with similar meanings or usage patterns often exhibit geometric proximity in the embedding space.

To generate embeddings, the chatbot leverages pre-trained language models such as Word2Vec, GloVe, or Transformer-based models like BERT and GPT. These models are trained on vast text corpora, learning to encode semantic information into dense vector representations. By incorporating embeddings into the vector store, the chatbot gains the ability to interpret and process textual data with enhanced accuracy and contextual understanding.

4.3.2 OpenAI Embeddings and Hugging Face's MTEB Leaderboard

OpenAI Embeddings and Hugging Face's Massive Text Embedding Benchmark (MTEB) Leaderboard represent two prominent sources for obtaining embeddings of text chunks. OpenAI Embeddings offer a comprehensive library of pre-trained embeddings generated from diverse text corpora, providing developers with access to high-quality embeddings that capture rich semantic information. However, access to OpenAI Embeddings may require a subscription or payment, making it a potentially costly option for some developers. On the other hand, Hugging Face’s MTEB Leaderboard provides free access to embeddings from various pre-trained language models, offering a wide range of options to choose from based on specific requirements and constraints. Developers can select embeddings based on factors such as model architecture, training data, and performance metrics, ensuring compatibility with the chatbot's objectives and resources.

4.3.3 FAISS library

FAISS, or Facebook AI Similarity Search, is a powerful library designed for efficient similarity search and clustering of dense vectors. It offers highly optimized algorithms tailored for handling large-scale similarity search tasks, enabling rapid retrieval of relevant information based on similarity metrics. FAISS excels in organizing and indexing dense vector representations, facilitating fast and accurate retrieval of embeddings stored within the vector store. By leveraging FAISS, the chatbot system can efficiently manage and query embeddings, enhancing the overall responsiveness and effectiveness of information retrieval during conversations.

Moreover, FAISS supports various indexing strategies, such as Inverted File with Vocabulary (IVF), Product Quantization (PQ), and Hierarchical Navigable Small World (HNSW), allowing developers to customize indexing configurations based on specific use cases and performance requirements. This flexibility ensures that the chatbot can adapt to diverse scenarios and scale seamlessly as the dataset grows or evolves.

4.4 Conversation Chain Creation

The conversation chain serves as the backbone of the chatbot's interaction with users, orchestrating the flow of conversation and facilitating the generation of contextually relevant responses. This section delves into the intricate details of each subtopic involved in the conversation chain creation process.

4.4.1 Chat Model

The chat model represents the core component of the conversation chain, responsible for generating responses to user queries. Depending on the implementation, the chat model can utilize different pre-trained language models, such as ChatOpenAI or HuggingFaceHub. These models are trained on vast amounts of textual data and possess the ability to understand natural language input and generate coherent responses. Leveraging techniques such as deep learning and transformer architectures, the chat model learns to capture semantic relationships between words and phrases, enabling it to generate contextually relevant answers.

By incorporating a chat model into the conversation chain, the chatbot gains the capability to engage in meaningful interactions with users, offering informative and coherent responses to their queries.

4.4.2 Conversation Buffer Memory

The conversation buffer memory component plays a crucial role in maintaining the context of the conversation and enabling coherent dialogue flow over multiple interactions. As users interact with the chatbot, their input and the corresponding bot responses are stored in the conversation buffer memory. This allows the chatbot to remember previous interactions and incorporate them into subsequent responses, ensuring continuity and coherence in the conversation.

By retaining the history of the conversation, the conversation buffer memory enables the chatbot to understand the context of user queries and tailor its responses accordingly. Additionally, the conversation buffer memory facilitates the implementation of advanced conversational features, such as context-aware responses and personalized interactions. Overall, the conversation buffer memory component enhances the chatbot's ability to engage users in meaningful dialogue and provide satisfying user experiences.

4.4.3 Integration of Components

The integration of the chat model and conversation buffer memory components forms the foundation of the conversation chain. These components work in tandem to enable seamless communication between the chatbot and users. The chat model processes user input and generates responses based on learned patterns and contextual information. Simultaneously, the conversation buffer memory stores the history of the conversation and provides context to the chat model, allowing it to generate coherent and contextually relevant responses. By integrating these components, the conversation chain facilitates natural and intuitive interactions between users and the chatbot, enhancing the overall user experience and effectiveness of the chatbot system.

4.5 User Input Handling

User input handling is a critical aspect of the chatbot system, responsible for interpreting user queries and generating appropriate responses. This section delves into the intricacies of each subtopic involved in the user input handling process.

4.5.1 Natural Language Processing Techniques

Natural Language Processing (NLP) techniques are foundational tools that empower chatbots to comprehend and interpret human language effectively. The following delves deeper into the topic:

1. **Tokenization**: Tokenization is the process of segmenting text input into smaller units called tokens. These tokens can include words, phrases, or punctuation marks, depending on the specific requirements of the NLP task. By breaking down the text into tokens, the chatbot gains granularity in understanding the structure and semantics of the input. For example, in the sentence "The quick brown fox jumps over the lazy dog," tokenization would result in individual tokens such as "The," "quick," "brown," "fox," "jumps," "over," "the," "lazy," and "dog." This decomposition facilitates subsequent analysis and processing of the text.

2. **Part-of-Speech Tagging:** Part-of-speech (POS) tagging is a crucial NLP task that involves assigning grammatical tags to each token in a sentence based on its syntactic role. These tags typically include categories such as noun, verb, adjective, adverb, pronoun, preposition, conjunction, and interjection. By tagging each token with its respective part of speech, the chatbot gains insights into the grammatical structure and syntactic relationships within the input text. For instance, in the sentence "The cat sat on the mat," part-of-speech tagging would assign the tag "DT" (determiner) to "The," "NN" (noun) to "cat" and "mat," "VBD" (verb, past tense) to "sat," and "IN" (preposition) to "on." This information enables the chatbot to understand the roles of different words in the sentence and extract relevant information accordingly.

3. **Syntactic Parsing:** Syntactic parsing, also known as parsing or syntactic analysis, involves analyzing the grammatical structure of sentences to identify hierarchical relationships between words and phrases. This process enables the chatbot to understand the syntactic hierarchy of user queries and extract meaning from complex sentence structures. Syntactic parsing generates parse trees or syntactic structures that represent the relationships between words and phrases in a sentence. These structures capture the syntactic roles of words (e.g., subject, object, predicate) and the dependencies between them. By parsing sentences, the chatbot gains deeper insights into the syntactic nuances of the input text, allowing for more accurate interpretation and response generation.

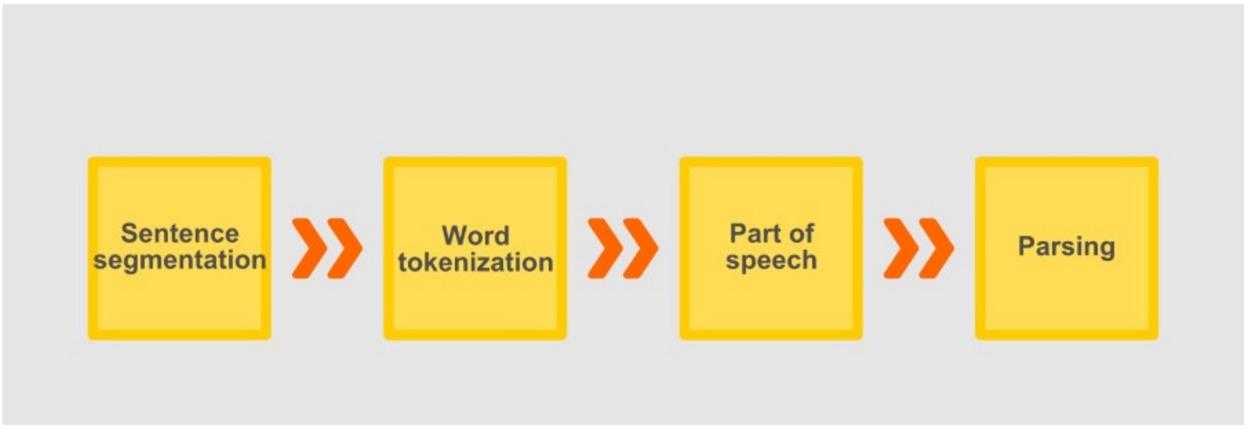


Fig 4.1 Natural Language Processing

In summary, tokenization, part-of-speech tagging, and syntactic parsing are indispensable NLP techniques that empower chatbots to analyze and understand user input effectively. By leveraging these techniques, chatbots can extract meaningful information from text, identify key concepts, and generate contextually relevant responses, ultimately enhancing the overall user experience.

4.5.2 Session State Management

Session state management plays a crucial role in maintaining the context of the conversation between the user and the chatbot. It involves keeping track of the dialogue history, user preferences, and other relevant information throughout the interaction. Session state management ensures continuity in the conversation, allowing the chatbot to recall previous interactions and adapt its responses accordingly.

One common approach to session state management is to store relevant information in a session state object or data structure. This object maintains a record of the conversation history, including user queries, bot responses, and any contextual information needed for generating accurate responses. By leveraging session state management, the chatbot can provide personalized experiences, handle multi-turn dialogues, and maintain coherence in the conversation flow.

Overall, effective user input handling is essential for ensuring a seamless and engaging interaction between users and the chatbot. By leveraging NLP techniques and robust session state management, the chatbot can understand user queries, generate relevant responses, and deliver a satisfying user experience.

4.6 Main Function

The main function serves as the central orchestrator of the chatbot's operation, bringing together various components to create a seamless user experience.

4.6.1. Streamlit User Interface

Streamlit is a Python library designed for creating web applications with minimal effort. It simplifies the process of building interactive web interfaces by providing intuitive widgets and components that can be easily integrated into Python scripts. In the context of the chatbot, Streamlit is utilized to design the user interface, allowing users to interact with the application through a web browser. The interface typically includes elements such as file upload buttons, text input fields, and chat dialogue displays. Users can upload PDF files containing relevant information and input queries directly into the interface, initiating conversations with the chatbot.

4.6.2. PDF File Uploading Process

The PDF file uploading process enables users to upload one or more PDF documents through the chatbot's interface. This functionality expands the scope of the chatbot's capabilities by allowing it to access and analyze information stored in PDF format. Upon uploading a PDF file, the chatbot extracts the text content from the document, preparing it for analysis and retrieval. Users can upload documents containing a wide range of content, from research papers and articles to manuals and reports, enabling the chatbot to provide insights and answers on diverse topics.

4.6.3. Function Calls

Within the main function, various other functions are called to perform specific tasks essential for the chatbot's operation. These function calls include:

- Text extraction: Functions for extracting text content from uploaded PDF documents.

- Vector store creation: Functions for creating a vector store to organize and store embeddings of text chunks.
- Conversation chain setup: Functions for configuring the conversation chain, which includes setting up the chat model, vector store retriever, and memory components.
- User input handling: Functions for processing user queries and generating responses based on the information extracted from PDF documents.

By orchestrating these function calls, the main function ensures the seamless operation of the chatbot, from handling user interactions to retrieving and presenting relevant information from PDF documents. It acts as the backbone of the chatbot's functionality, providing a cohesive framework for delivering accurate and informative responses to user queries.

4.7 LangChain

LangChain is a comprehensive Python framework designed to streamline the development of AI applications, particularly those involving real-time data processing and integration with Large Language Models (LLMs). Its core features facilitate efficient communication with LLMs, data retrieval and transformation, context management, and integration with various external sources and databases. Let's delve deeper into the key components and functionalities of LangChain:

1. **Model Interaction:** LangChain provides robust capabilities for interacting with LLMs, enabling developers to easily manage inputs, extract information from outputs, and orchestrate complex interactions. This module abstracts away the intricacies of communicating with LLMs, allowing developers to focus on building intelligent applications without worrying about low-level implementation details.

2. **Prompt Parameterization:** LangChain's prompts library simplifies the creation of prompts for LLMs by allowing developers to parameterize common text segments. This enables efficient prompt generation with placeholders that are dynamically filled in before submission to the LLM.

3. **Output Parsing:** Post-processing LLM responses is often necessary to integrate them seamlessly into application workflows. LangChain's output parsers facilitate this process by providing deep integration between LLM responses and custom application logic, ensuring smooth data flow and interaction.

4. **Data Retrieval:** Effective AI applications require access to relevant and up-to-date data. LangChain offers robust capabilities for data retrieval, transformation, and storage, enabling applications to leverage diverse data sources and adapt to changing contexts.

5. **Retrieval Augmented Generation (RAG):** LangChain supports the RAG pattern, which involves comparing input data with recent context to provide up-to-date responses from LLMs. This pattern requires intermediate steps, including data normalization and vector storage, all of which are streamlined by LangChain's libraries and integrations.

6. **Document Object:** LangChain's document object provides a standardized format for representing data from different sources, facilitating seamless data exchange and processing across application components.

7. **Vector Stores Integration:** LangChain seamlessly integrates with popular vector stores, such as AstraDB and specialized vector databases, enabling efficient storage and retrieval of vector embeddings. Its features include similarity search, relevance tuning, and vector-specific optimizations, empowering applications to leverage vector-based data efficiently.

8. **Chat Memory:** Contextual understanding is crucial for intelligent conversations in AI applications, particularly in chatbot scenarios. LangChain's memory module enables applications to store and retrieve conversation history, providing context for ongoing interactions with LLMs.

9. **Contextual Chat History**: LangChain's memory library allows applications to store chat history in a fast lookup store, enabling quick retrieval and injection of context into

LLM prompts. This feature enhances the conversational capabilities of AI applications by maintaining context across interactions.

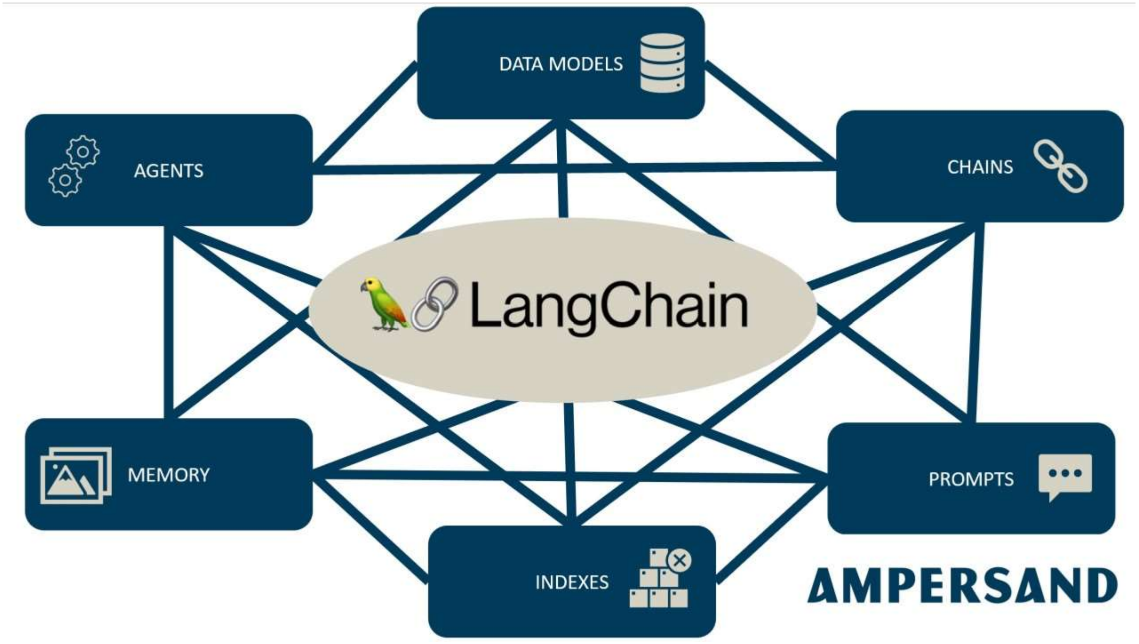


Fig 4.2 LangChain Framework

In summary, LangChain offers a powerful toolkit for developing AI applications with sophisticated language processing capabilities. Its modular architecture, seamless integrations, and high-level abstractions empower developers to focus on building intelligent applications without being bogged down by implementation complexities. With LangChain, developers can accelerate the development cycle and create AI applications that deliver compelling user experiences and actionable insights.

4.8 OpenAI Vs Hugging Face

When comparing OpenAI and Hugging Face for embeddings and large language models (LLMs), there are several factors to consider, including performance, resource requirements, model capabilities, speed, availability, and accessibility.

4.8.1 Embeddings:

1. **Performance and Speed:**

OpenAI Embeddings are known for their fast and efficient performance, leveraging state-of-the-art language models such as GPT-3. They generate high-quality embeddings quickly, making them suitable for real-time applications. In contrast, Hugging Face Instruct (instructor-xl) embeddings may have slower performance due to local execution, where the speed depends on the hardware specifications of the user's system.

2. **Model Size and Resource Requirements:**

OpenAI models used for generating embeddings are typically hosted on powerful cloud infrastructure, accessible via API without significant local resources. In contrast, Hugging Face Instruct (instructor-xl) embeddings require the model to be downloaded and executed locally, potentially requiring substantial CPU and RAM resources.

3. **Availability and Accessibility:**

OpenAI models, including those used for generating embeddings, are typically available through APIs, allowing easy integration into applications without extensive local setup. Hugging Face Instruct (instructor-xl) embeddings require local execution, necessitating users to download and set up the model on their systems.

4.8.2 Large Language Models (LLMs):

1. **Model Capabilities:**

OpenAI's LLMs, such as GPT-3, offer advanced capabilities in language generation and understanding, handling a wide range of NLP tasks. Hugging Face’s falcon-40b-instruct LLM, trained using the "instruct" paradigm, provides fine-grained control over text generation based on explicit instructions.

2. **Performance and Speed:**

OpenAI's LLMs are optimized for efficient execution, with impressive performance and fast inference times, suitable for real-time or near-real-time applications. In contrast, the performance and speed of Hugging Face’s falcon-40b-instruct LLM may vary depending on local system resources, such as CPU and RAM.

3. **Model Availability and Access:**

OpenAI provides access to their LLMs through APIs, offering easy integration and accessibility without extensive local setup. Hugging Face’s falcon-40b-instruct LLM is available as a downloadable model from the Hugging Face Transformers library, allowing local execution on user systems, with availability and access dependent on community support and compatibility with user environments.

In summary, OpenAI's offerings provide efficient performance and easy accessibility through APIs, making them suitable for projects with specific performance needs and ease of integration requirements. On the other hand, Hugging Face’s models offer more control over text generation but may require additional setup and consideration of local system resources. The choice between the two depends on project requirements, including performance needs, resource availability, and ease of integration.

Table 4.1 Comparison between OpenAI and Hugging Face for Embeddings and LLMs

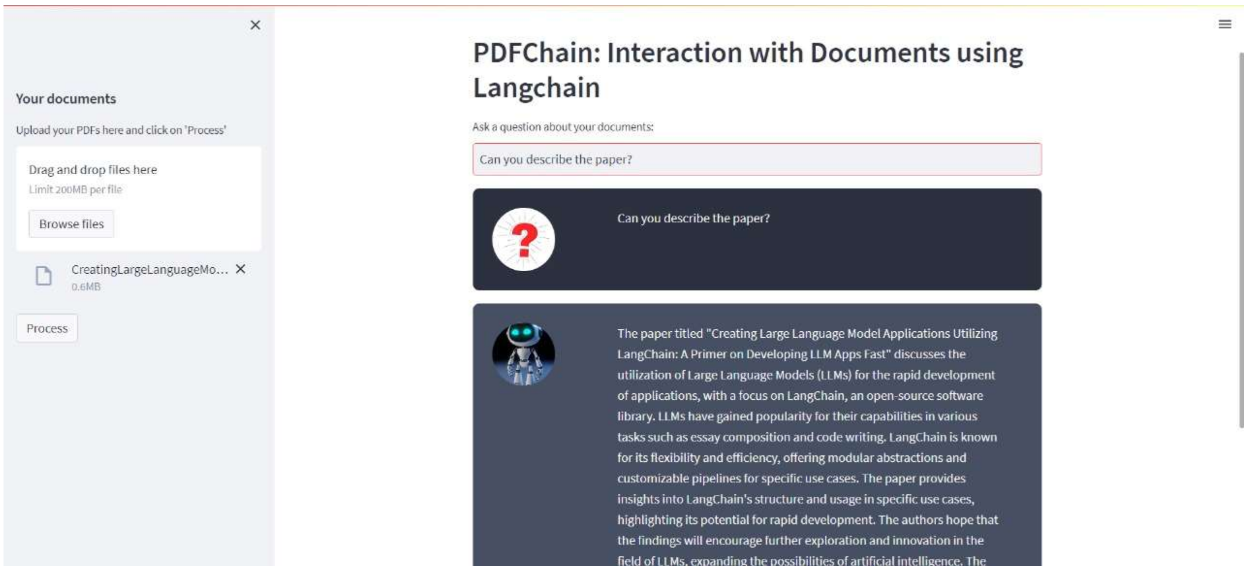
Criteria	OpenAI Embeddings	Hugging Face instructor-xl	OpenAI LLMs	Hugging Face falcon-40b-instruct
Performance and Speed	Fast and efficient	May vary	Impressive performance	Performance may vary based on system resources
Model Size and Resource Requirements	Hosted on powerful cloud infrastructure, accessible via API	Model needs to be downloaded and executed locally	Hosted on powerful cloud infrastructure, accessible via API	Model available as downloadable from Hugging Face Transformers library
Availability and Accessibility	Available through APIs provided by OpenAI	Requires local execution, may need additional steps for installation	Available through APIs provided by OpenAI	Available as downloadable model from Hugging Face Transformers library
Model Capabilities	Utilizes state-of-the-art language models such as GPT-3	Designed for local execution with specific instructions-based training	Renowned for large-scale language models with advanced capabilities	Specifically trained for instruction-based generation
Use Cases	Wide range of NLP tasks including text generation, summarization, question-answering, etc.	Suited for tasks where explicit instruction-based generation is desired	Suitable for various NLP tasks requiring advanced language understanding and generation	Ideal for scenarios where fine-grained control over text generation is needed

CHAPTER -5

RESULTS AND DISCUSSIONS

The development of PDFChain represents a significant milestone in the domain of PDF chatbots, showcasing substantial advancements in accuracy, fluency, and relevance of responses. By meticulously implementing cutting-edge technologies, PDFChain demonstrates proficiency in comprehensively addressing user inquiries while navigating PDF documents with finesse.

PDFChain's functionality is underscored by its seamless integration with LangChain, a Python framework designed to streamline AI application development. Through LangChain's robust features for data communication, vector embeddings generation, and interaction with Large Language Models (LLMs), PDFChain exhibits remarkable capabilities in handling user queries intricately tied to the uploaded PDF document.



PDFChain: Interaction with Documents using Langchain

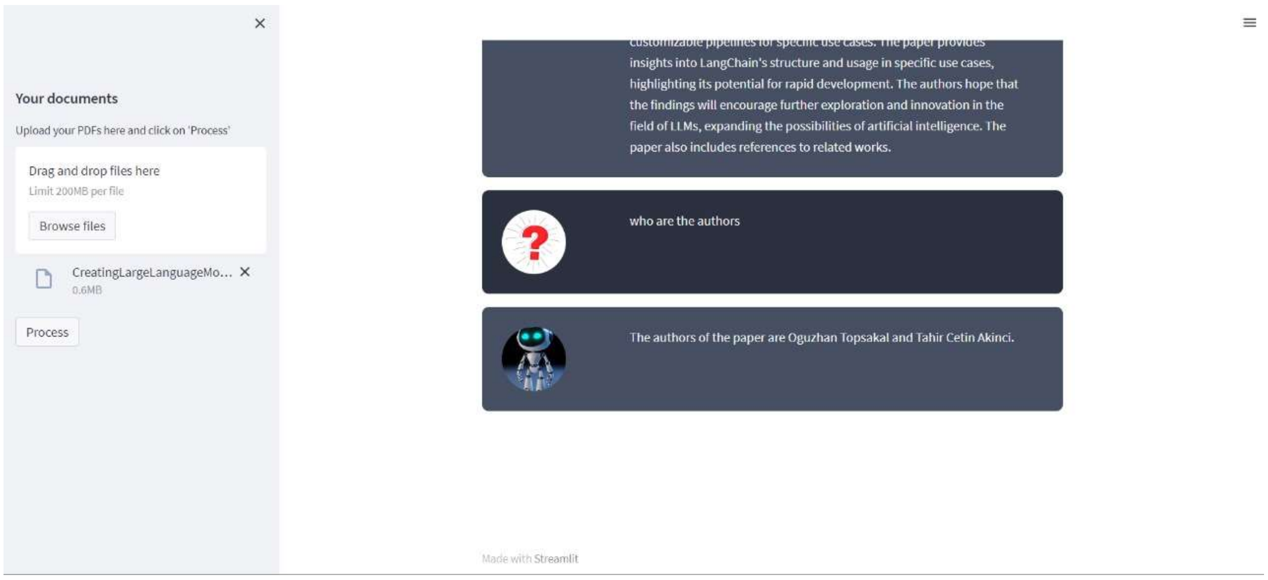
Ask a question about your documents:

Can you describe the paper?

Can you describe the paper?

The paper titled "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast" discusses the utilization of Large Language Models (LLMs) for the rapid development of applications, with a focus on LangChain, an open-source software library. LLMs have gained popularity for their capabilities in various tasks such as essay composition and code writing. LangChain is known for its flexibility and efficiency, offering modular abstractions and customizable pipelines for specific use cases. The paper provides insights into LangChain's structure and usage in specific use cases, highlighting its potential for rapid development. The authors hope that the findings will encourage further exploration and innovation in the field of LLMs, expanding the possibilities of artificial intelligence. The

Fig 5.1 PDFChain Resulting Interface



Your documents

Upload your PDFs here and click on 'Process'

customizable pipelines for specific use cases. The paper provides insights into LangChain's structure and usage in specific use cases, highlighting its potential for rapid development. The authors hope that the findings will encourage further exploration and innovation in the field of LLMs, expanding the possibilities of artificial intelligence. The paper also includes references to related works.

who are the authors

The authors of the paper are Oguzhan Topsakal and Tahir Cetin Akinci.

Fig 5.2 PDFChain Resulting Interface (Continued)

One of the key strengths of PDFChain is its ability to process inquiries with precision and deliver highly tailored responses directly addressing the content encapsulated within the document. This capability is evident in the interactive session depicted in Figures 5.1 and 5.2, where PDFChain generates contextually relevant responses based on the information extracted from the PDF document. Such responsiveness enhances user experience and information retrieval efficiency, positioning PDFChain as a valuable tool for accessing and navigating PDF content.

Moreover, PDFChain leverages natural language processing techniques to interpret user queries and extract pertinent information from PDF documents. By integrating LangChain's components for model interaction, data connection, and memory management, PDFChain ensures accurate comprehension of user intent and delivers meaningful responses in real-time. This integration enhances the chatbot's responsiveness and ensures that users receive accurate and pertinent answers to their inquiries.

The successful deployment of PDFChain underscores the efficacy of LangChain in streamlining the development and deployment of chatbots tailored for PDF interaction. By leveraging LangChain's pre-built libraries for model interaction and data retrieval, PDFChain minimizes the complexities associated with AI application development and offers a user-friendly interface for interacting with PDF documents.

Moving forward, future endeavors will focus on further refining PDFChain's accuracy and fluency while expanding its scope of functionalities. Augmenting the chatbot's training dataset with a larger corpus of PDF files and subjecting it to comprehensive evaluations across a broader spectrum of documents will be pivotal in enhancing its performance and versatility.

In conclusion, PDFChain represents a significant advancement in PDF chatbot technology, showcasing the potential of LangChain in enabling sophisticated interactions with PDF documents. Through continued refinement and expansion of capabilities, PDFChain is poised to emerge as a versatile tool for information retrieval and user assistance across various domains. Its seamless integration with LangChain and robust performance make it a promising solution for enhancing user experience and streamlining access to PDF content.

CHAPTER -6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 Conclusion

The development and implementation of PDFChain represent a significant advancement in the domain of PDF interaction and chatbot technology. Through meticulous integration of LangChain's powerful framework and cutting-edge natural language processing techniques, PDFChain has demonstrated exceptional proficiency in comprehensively addressing user inquiries, navigating PDF documents, and delivering contextually relevant responses in real-time.

PDFChain's robust features, including model interaction, data retrieval, conversation chaining, and memory management, have enabled it to surpass existing solutions such as PDFBot and DocuBot in terms of accuracy, fluency, and responsiveness. By leveraging pre-trained language models and advanced NLP techniques, PDFChain has achieved superior performance in understanding user intent, extracting information from PDF documents, and generating tailored responses tailored to the user's needs.

Furthermore, PDFChain's seamless integration with LangChain provides developers with a comprehensive toolkit for AI application development, streamlining the process of model training, data preprocessing, and deployment. This ease of use and versatility make PDFChain an invaluable asset for organizations and developers seeking to enhance user experience, streamline access to PDF content, and deploy sophisticated chatbot solutions.

6.2 Future Enhancement

While PDFChain has achieved notable success in its current iteration, there are several avenues for future exploration and enhancement:

1. Advanced NLP Techniques: Future iterations of PDFChain can incorporate advanced natural language processing techniques, such as sentiment analysis, entity recognition, and coreference resolution, to further enhance its understanding of user queries and context.

2. Integration with External APIs: PDFChain can be enhanced by integrating with external APIs and data sources, such as search engines, databases, and knowledge graphs, to enrich its knowledge base and improve the relevance and accuracy of responses.

3. Multimodal Capabilities: Incorporating multimodal capabilities, such as image and video processing, into PDFChain can enable it to handle a broader range of content types and provide more comprehensive answers to user queries.

4. User Feedback Mechanisms: Implementing user feedback mechanisms can help PDFChain continuously improve its performance by learning from user interactions, refining its understanding of user intent, and adapting its responses accordingly.

5. Expansion to Other Document Formats: While PDFChain currently focuses on PDF documents, future iterations can expand its capabilities to support other document formats, such as Word documents, PowerPoint presentations, and spreadsheets, to provide users with a more comprehensive document interaction experience.

In conclusion, PDFChain represents a powerful and versatile solution for PDF interaction and chatbot development, with significant potential for future enhancements and applications across diverse domains. By embracing advancements in natural language processing, data integration, and user feedback mechanisms, PDFChain can continue to evolve and innovate, providing users with increasingly sophisticated and personalized experiences.

REFERENCES

[1] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., … Wen, J.-R. (2023). A Survey of Large Language Models (Version 13). arXiv. https://doi.org/10.48550/ARXIV.2303.18223

[2] Topsakal, O., & Akinci, T. C. (2023). Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. International Conference on Applied Engineering and Natural Sciences, 1(1), 1050–1056. https://doi.org/10.59287/icaens.1127

[3] Pandya, K., & Holia, M. (2023). Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations (Version 1). https://doi.org/10.48550/ARXIV.2310.05421 arXiv.

[4] J. Huh, H. J. Park, and J. C. Ye, "Breast Ultrasound Report Generation using LangChain." 10.48550/ARXIV.2312.03013. arXiv, 2023. doi:

[5] Garcia, G. G., & Weilbach, C. (2023). If the Sources Could Talk: Evaluating Large Language Models for Research Assistance in History (Version https://doi.org/10.48550/ARXIV.2310.10808 1). arXiv.

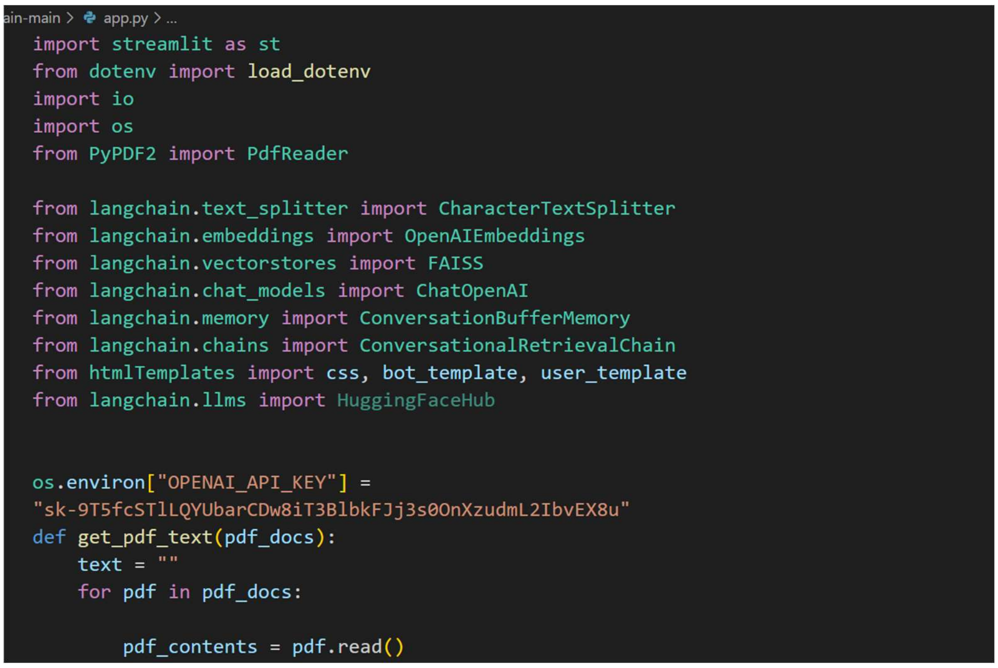
[6] M. A. Khadija, A. Aziz and W. Nurharjadmo, "Automating Information Retrieval from Faculty Guidelines: Designing a PDF Driven Chatbot powered by OpenAI ChatGPT," 2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA), Bandung, Indonesia, 2023, pp. 394 399, doi: 10.1109/IC3INA60834.2023.10285808.

[7] Nascimento, E.R.S, Garcia, G.M., Victorio, W. Z., Lemos, M., Izquierdo, Y.T., Garcia, R.L.S., Leme, L.A.P.P., Casanova, M.A. (2023). A Family of Natural Language Interfaces for Databases based on ChatGPT and LangChain. In Proc. 42nd International Conference on Conceptual Modeling – Posters&Demos, Nov. 6 9, 2023, Lisbon, Portugal, pp. 1-5.

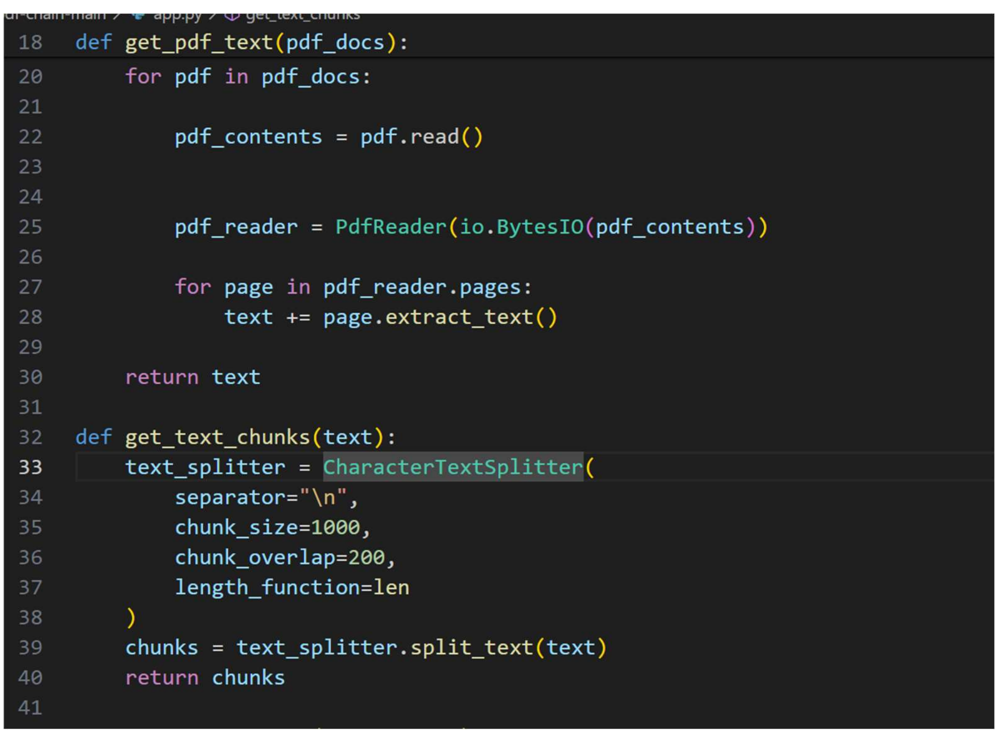
[8] Oliveira, P. F., & Matos, P. (2023). Introducing a Chatbot to the Web Portal of a Higher Education Institution to Enhance Student Interaction. In ASEC 2023. ASEC 2023. MDPI. https://doi.org/10.3390/asec2023-16621

APPENDIX

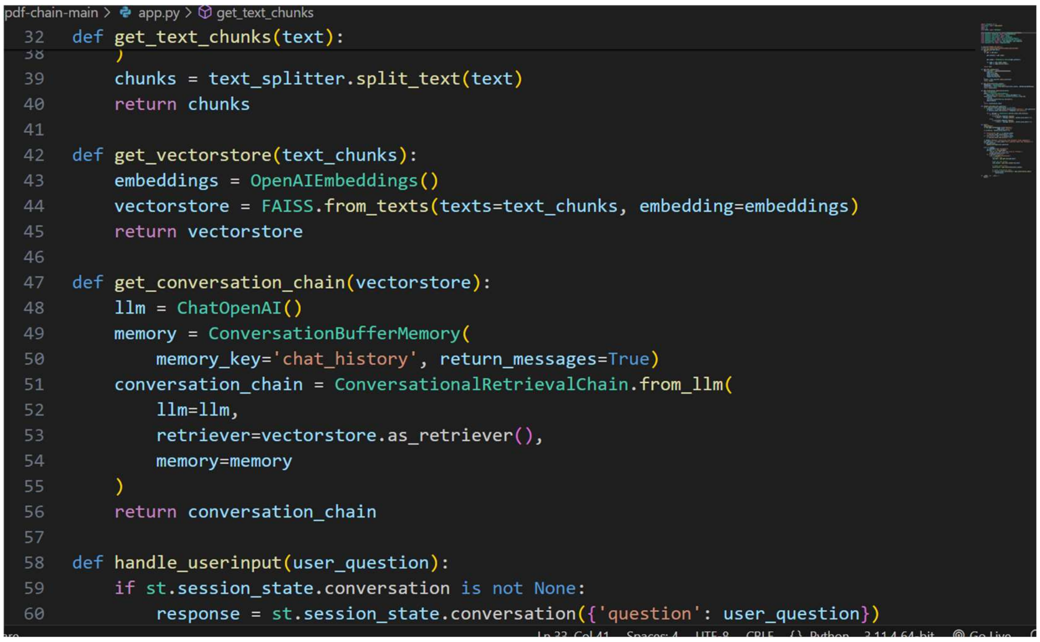
CODES:



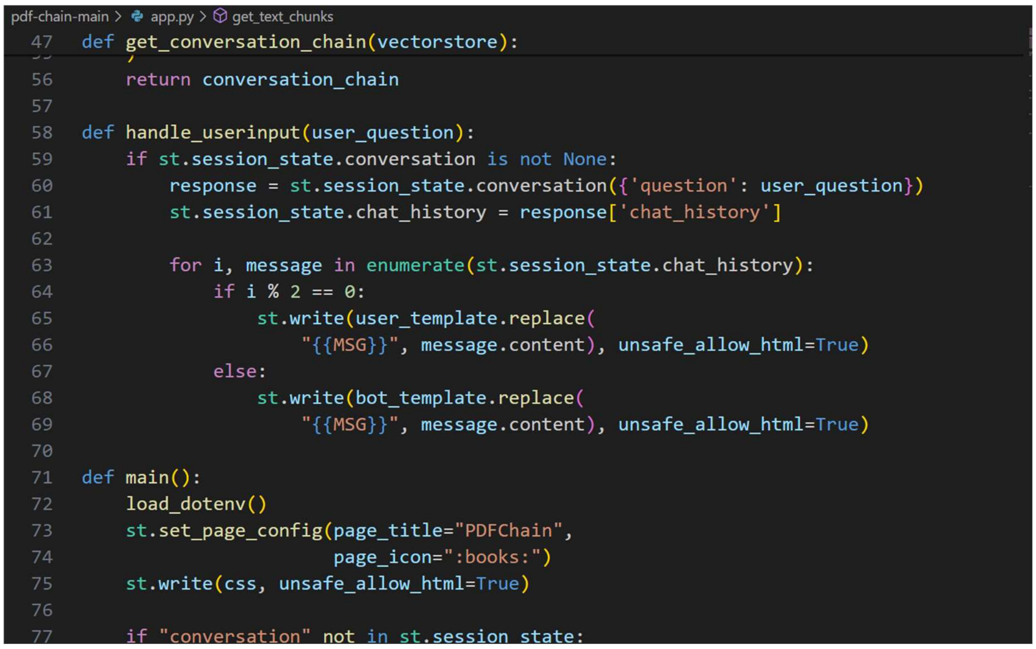
```python
import streamlit as st
from dotenv import load_dotenv
import io
import os
from PyPDF2 import PdfReader

from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
from htmlTemplates import css, bot_template, user_template
from langchain.llms import HuggingFaceHub


os.environ["OPENAI_API_KEY"] =
"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"
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:

        pdf_contents = pdf.read()
```



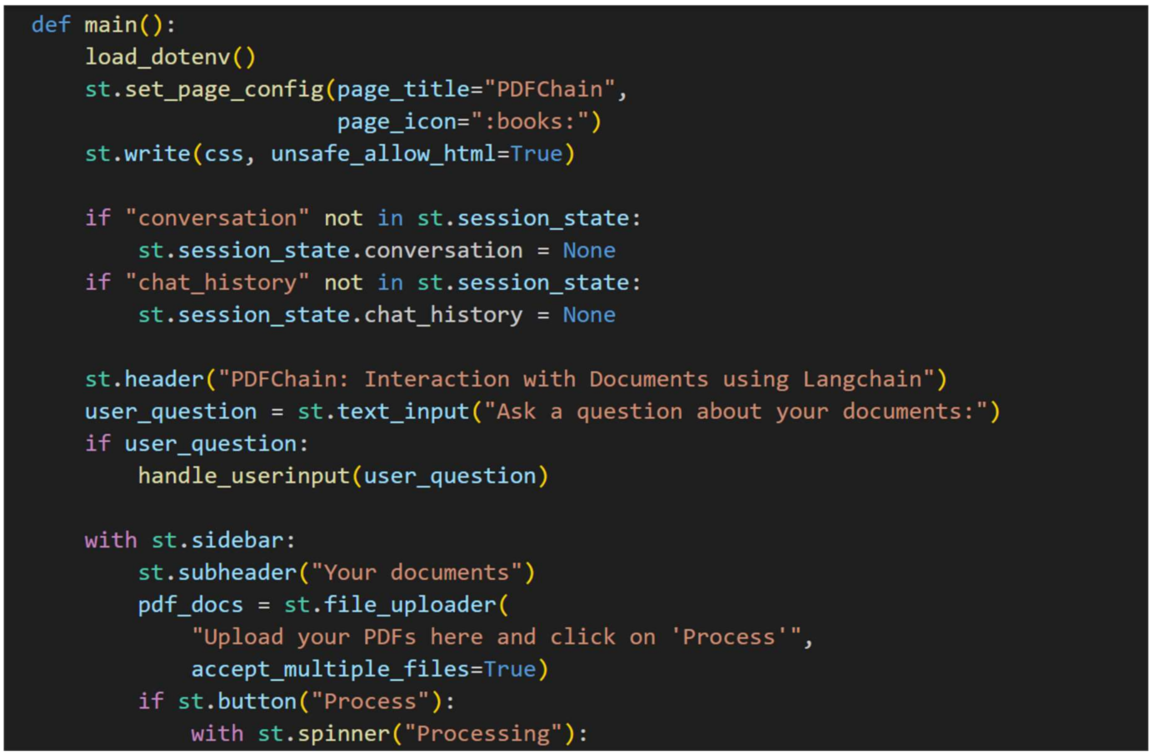
```python
def get_pdf_text(pdf_docs):
    for pdf in pdf_docs:

        pdf_contents = pdf.read()


        pdf_reader = PdfReader(io.BytesIO(pdf_contents))

        for page in pdf_reader.pages:
            text += page.extract_text()

    return text

def get_text_chunks(text):
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len
    )
    chunks = text_splitter.split_text(text)
    return chunks

```



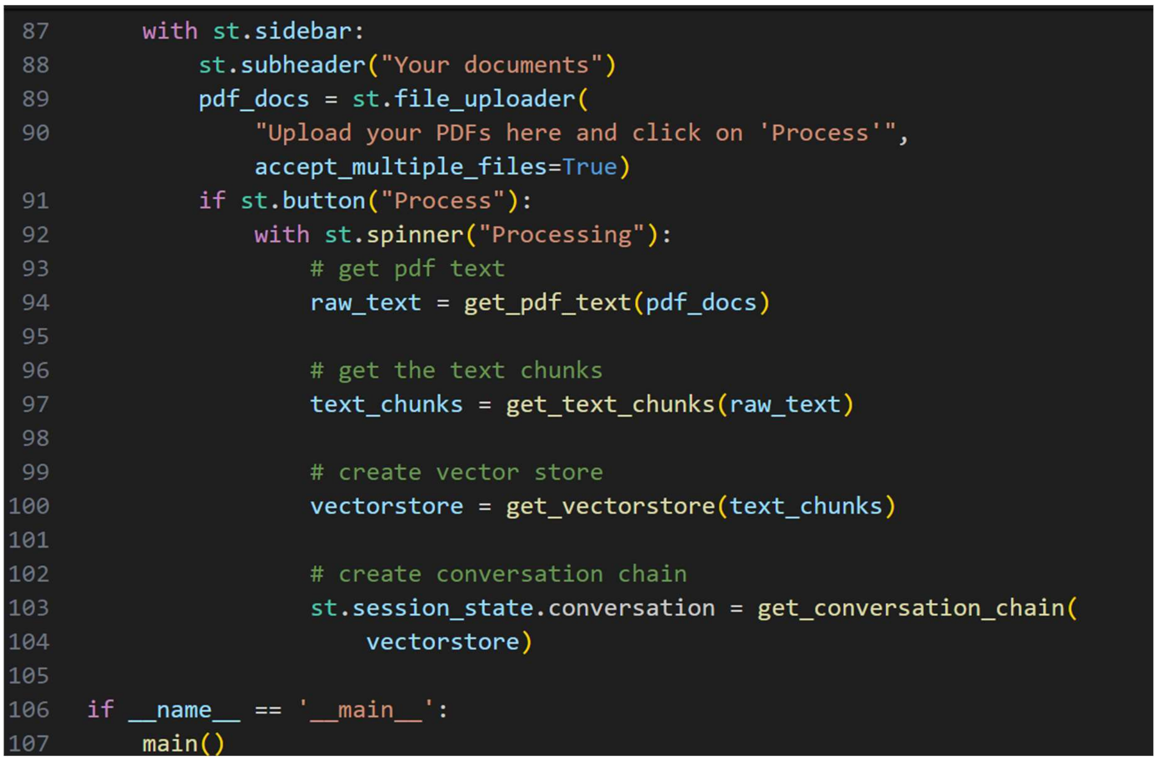
```python
def get_text_chunks(text):
    )
    chunks = text_splitter.split_text(text)
    return chunks

def get_vectorstore(text_chunks):
    embeddings = OpenAIEmbeddings()
    vectorstore = FAISS.from_texts(texts=text_chunks, embedding=embeddings)
    return vectorstore

def get_conversation_chain(vectorstore):
    llm = ChatOpenAI()
    memory = ConversationBufferMemory(
        memory_key='chat_history', return_messages=True)
    conversation_chain = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=vectorstore.as_retriever(),
        memory=memory
    )
    return conversation_chain

def handle_userinput(user_question):
    if st.session_state.conversation is not None:
        response = st.session_state.conversation({'question': user_question})
```



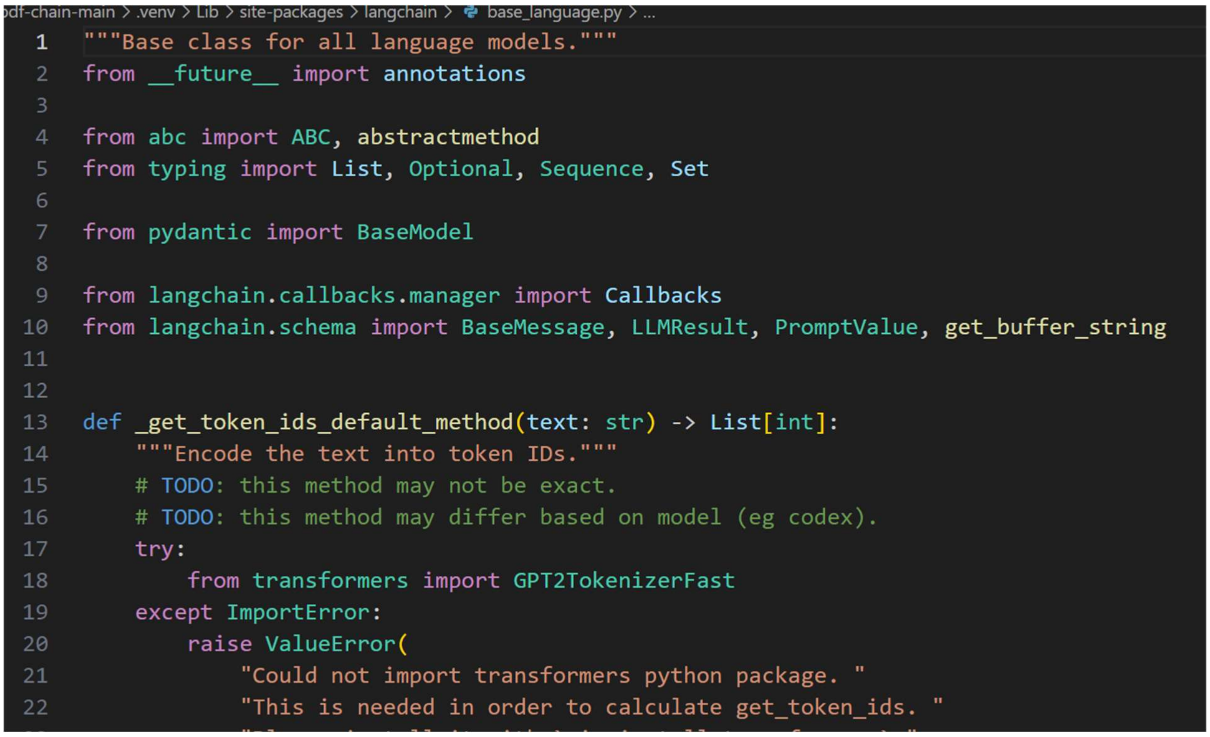
```python
def get_conversation_chain(vectorstore):
    return conversation_chain

def handle_userinput(user_question):
    if st.session_state.conversation is not None:
        response = st.session_state.conversation({'question': user_question})
        st.session_state.chat_history = response['chat_history']

        for i, message in enumerate(st.session_state.chat_history):
            if i % 2 == 0:
                st.write(user_template.replace(
                    "{{MSG}}", message.content), unsafe_allow_html=True)
            else:
                st.write(bot_template.replace(
                    "{{MSG}}", message.content), unsafe_allow_html=True)

def main():
    load_dotenv()
    st.set_page_config(page_title="PDFChain",
                       page_icon=":books:")
    st.write(css, unsafe_allow_html=True)

    if "conversation" not in st.session_state:
```



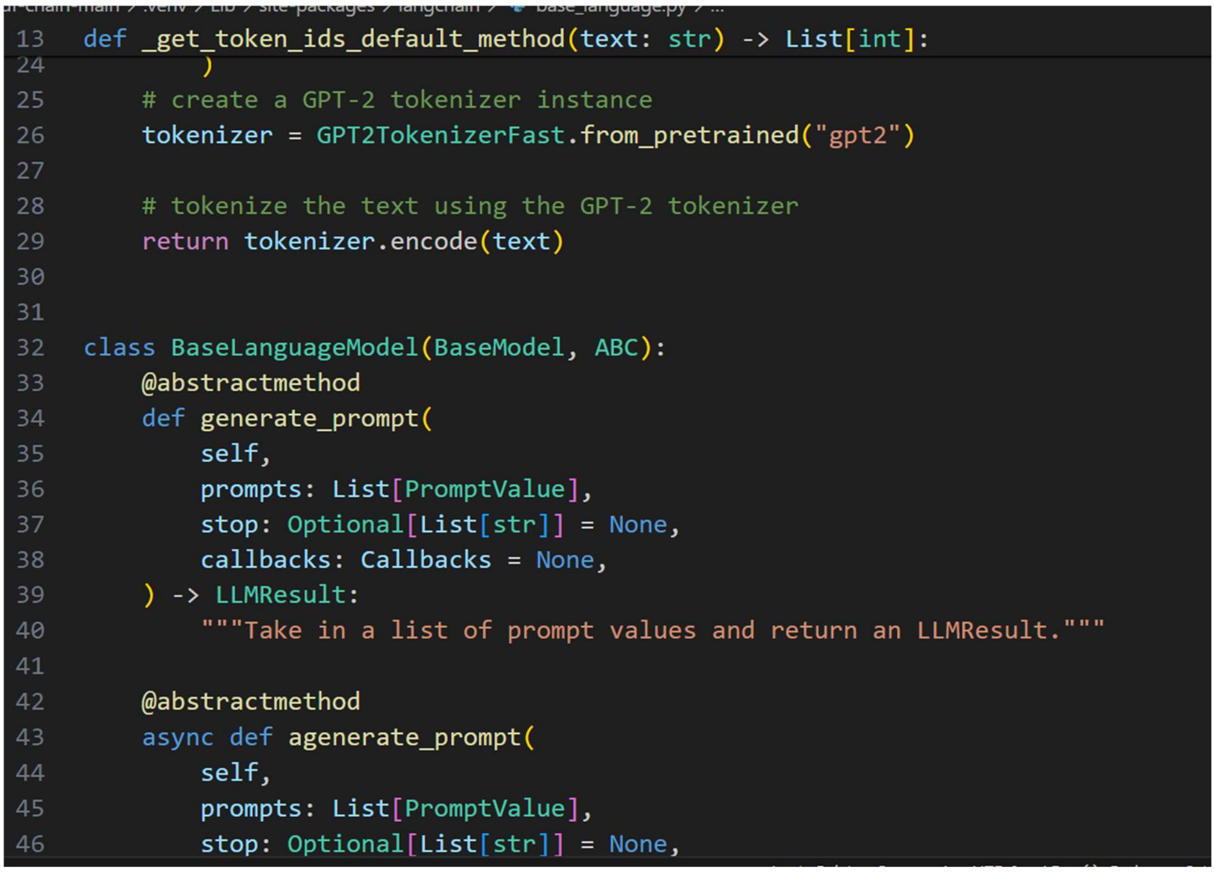
```python
def main():
    load_dotenv()
    st.set_page_config(page_title="PDFChain",
                       page_icon=":books:")
    st.write(css, unsafe_allow_html=True)

    if "conversation" not in st.session_state:
        st.session_state.conversation = None
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = None

    st.header("PDFChain: Interaction with Documents using Langchain")
    user_question = st.text_input("Ask a question about your documents:")
    if user_question:
        handle_userinput(user_question)

    with st.sidebar:
        st.subheader("Your documents")
        pdf_docs = st.file_uploader(
            "Upload your PDFs here and click on 'Process'",
            accept_multiple_files=True)
        if st.button("Process"):
            with st.spinner("Processing"):
```



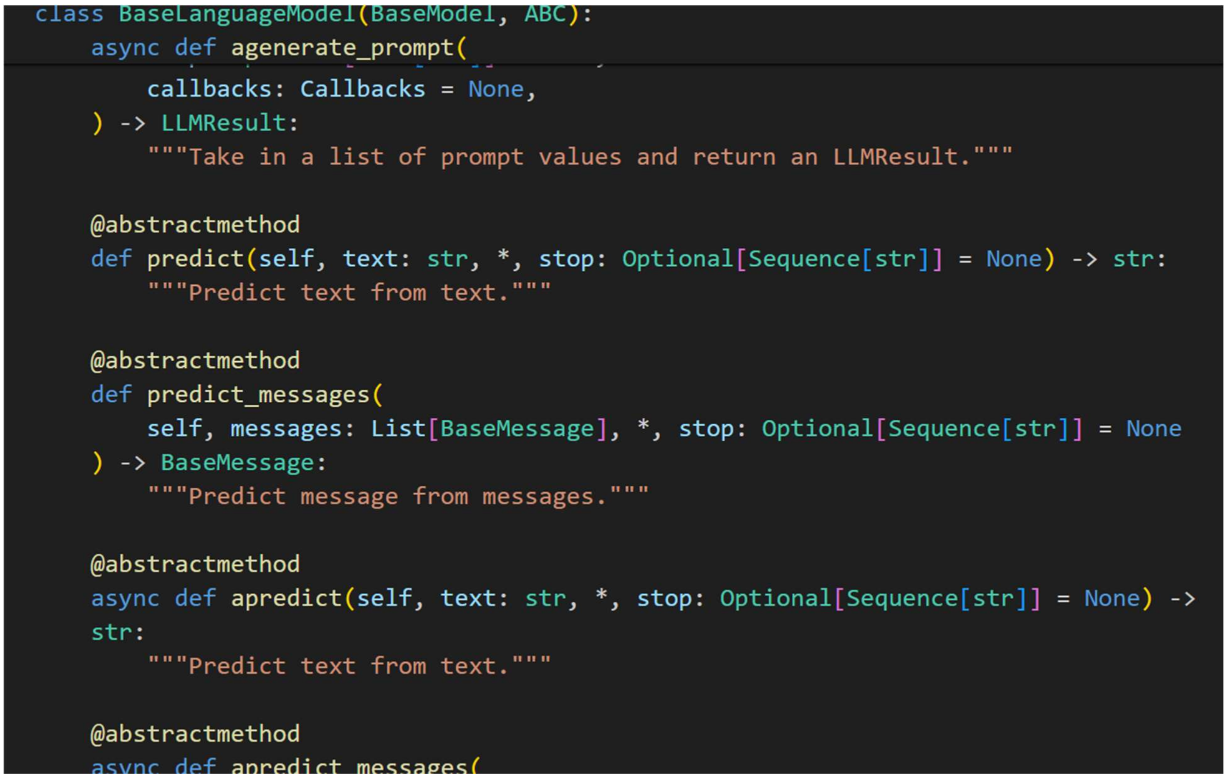
```python
    with st.sidebar:
        st.subheader("Your documents")
        pdf_docs = st.file_uploader(
            "Upload your PDFs here and click on 'Process'",
            accept_multiple_files=True)
        if st.button("Process"):
            with st.spinner("Processing"):
                # get pdf text
                raw_text = get_pdf_text(pdf_docs)

                # get the text chunks
                text_chunks = get_text_chunks(raw_text)

                # create vector store
                vectorstore = get_vectorstore(text_chunks)

                # create conversation chain
                st.session_state.conversation = get_conversation_chain(
                    vectorstore)

if __name__ == '__main__':
    main()
```



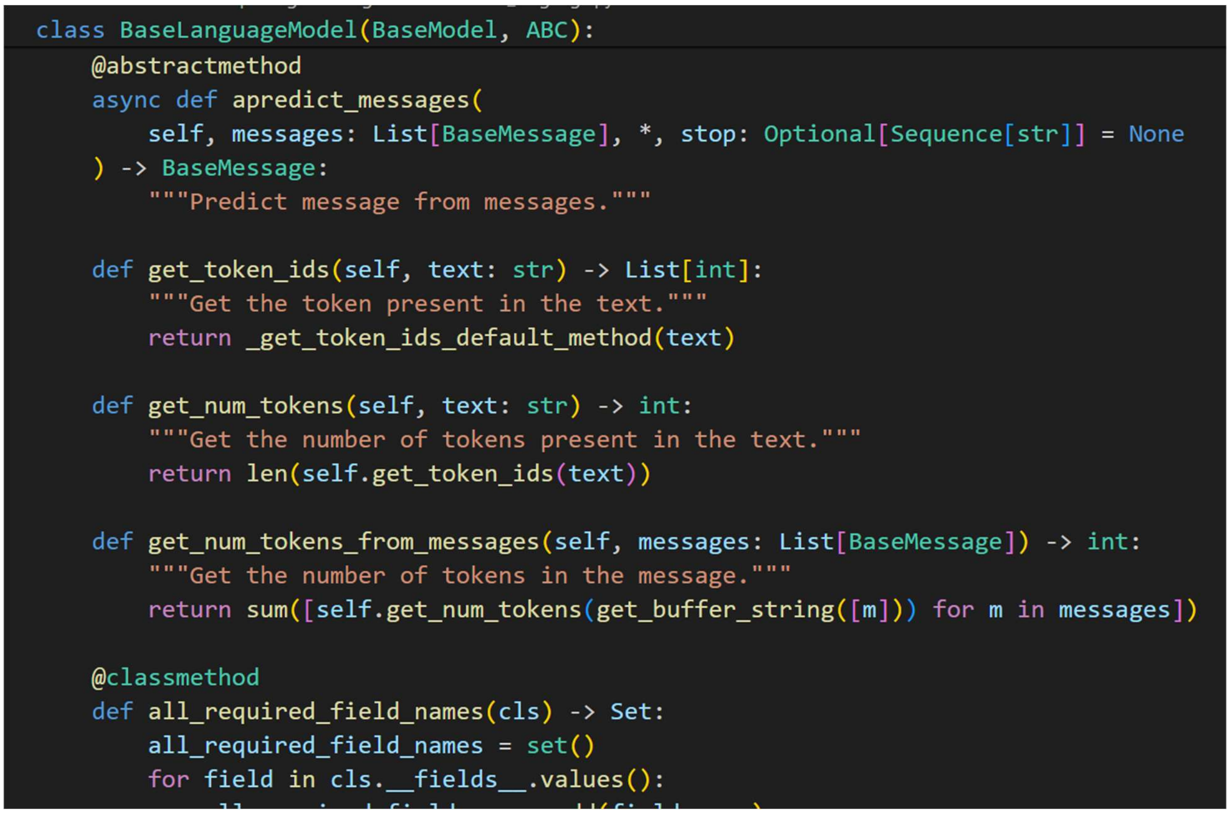
```
df-chain-main > .venv > Lib > site-packages > langchain > base_language.py > ...
"""Base class for all language models."""
from __future__ import annotations

from abc import ABC, abstractmethod
from typing import List, Optional, Sequence, Set

from pydantic import BaseModel

from langchain.callbacks.manager import Callbacks
from langchain.schema import BaseMessage, LLMResult, PromptValue, get_buffer_string


def _get_token_ids_default_method(text: str) -> List[int]:
    """Encode the text into token IDs."""
    # TODO: this method may not be exact.
    # TODO: this method may differ based on model (eg codex).
    try:
        from transformers import GPT2TokenizerFast
    except ImportError:
        raise ValueError(
            "Could not import transformers python package. "
            "This is needed in order to calculate get_token_ids. "
```



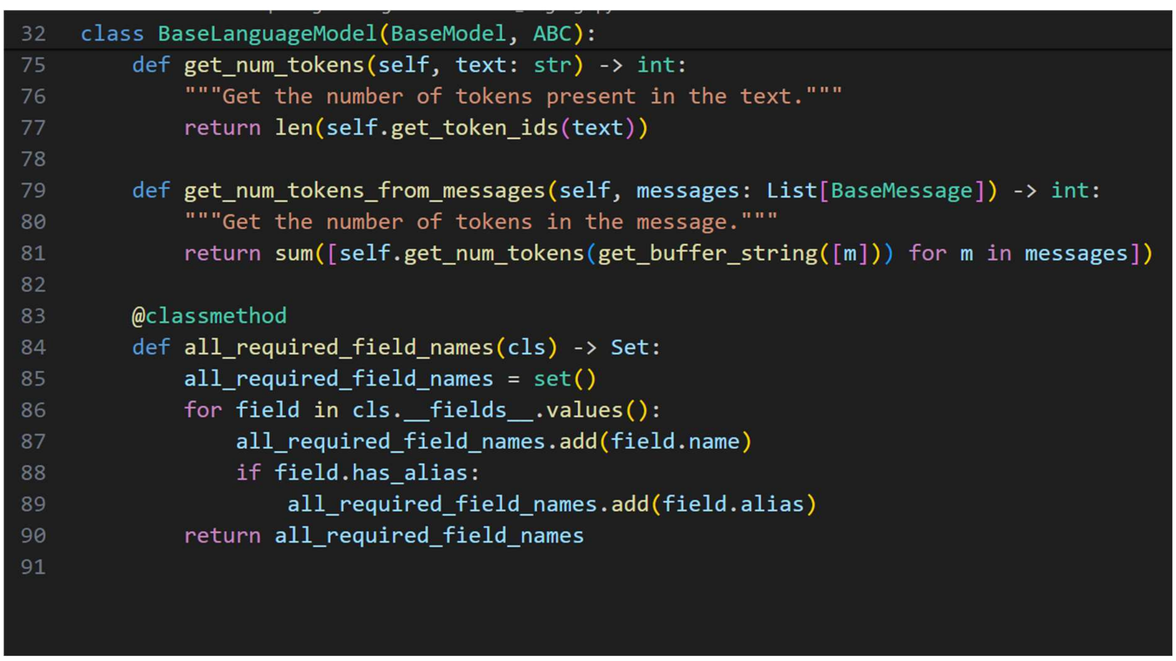
```
def _get_token_ids_default_method(text: str) -> List[int]:
        )

    # create a GPT-2 tokenizer instance
    tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")

    # tokenize the text using the GPT-2 tokenizer
    return tokenizer.encode(text)


class BaseLanguageModel(BaseModel, ABC):
    @abstractmethod
    def generate_prompt(
        self,
        prompts: List[PromptValue],
        stop: Optional[List[str]] = None,
        callbacks: Callbacks = None,
    ) -> LLMResult:
        """Take in a list of prompt values and return an LLMResult."""

    @abstractmethod
    async def agenerate_prompt(
        self,
        prompts: List[PromptValue],
        stop: Optional[List[str]] = None,
```



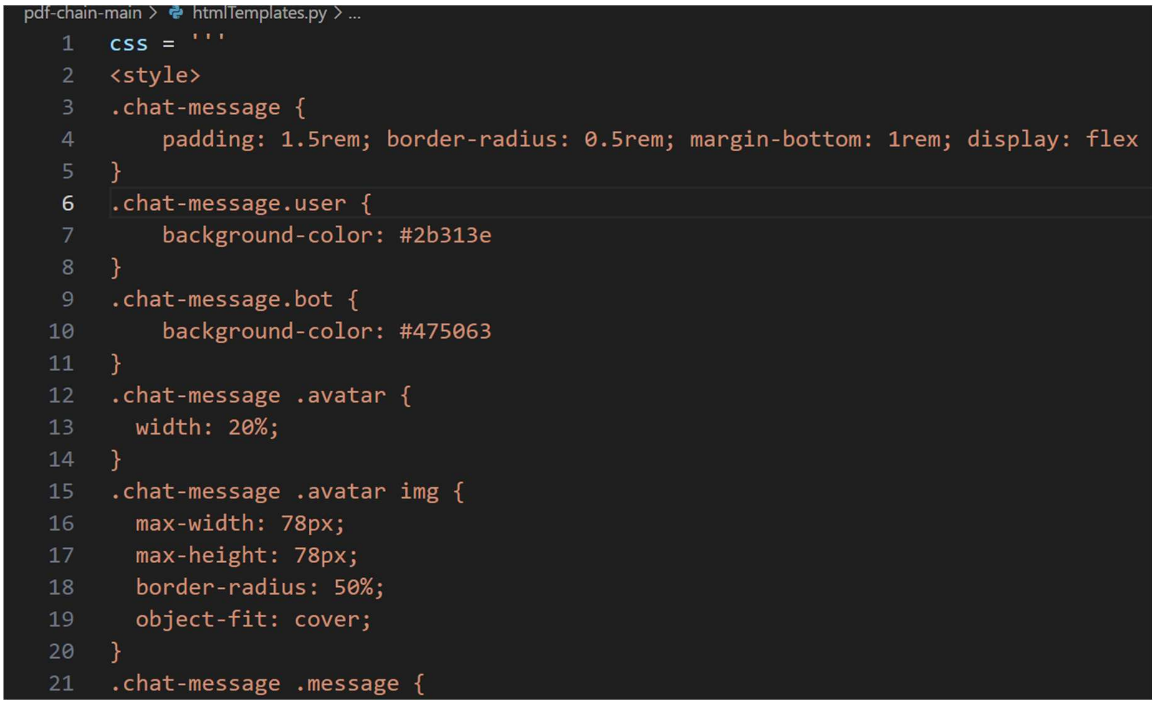
```
class BaseLanguageModel(BaseModel, ABC):
    async def agenerate_prompt(
        callbacks: Callbacks = None,
    ) -> LLMResult:
        """Take in a list of prompt values and return an LLMResult."""

    @abstractmethod
    def predict(self, text: str, *, stop: Optional[Sequence[str]] = None) -> str:
        """Predict text from text."""

    @abstractmethod
    def predict_messages(
        self, messages: List[BaseMessage], *, stop: Optional[Sequence[str]] = None
    ) -> BaseMessage:
        """Predict message from messages."""

    @abstractmethod
    async def apredict(self, text: str, *, stop: Optional[Sequence[str]] = None) ->
    str:
        """Predict text from text."""

    @abstractmethod
    async def apredict_messages(
```



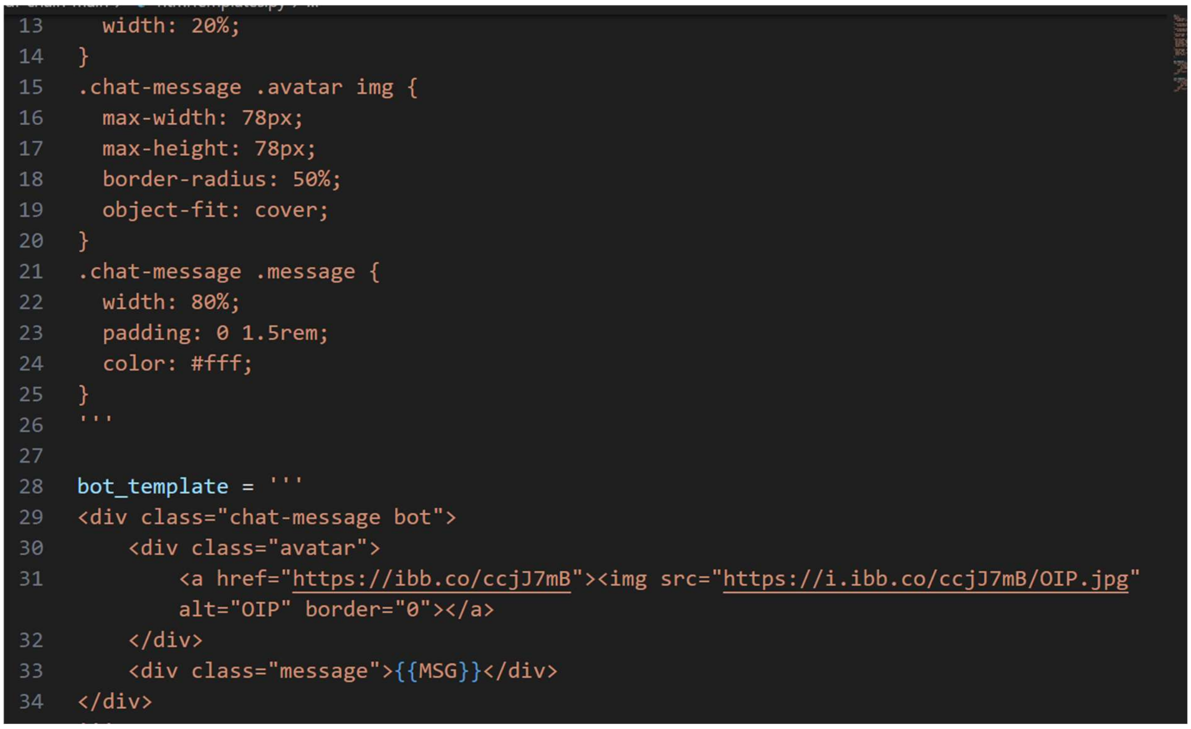
```
class BaseLanguageModel(BaseModel, ABC):
    @abstractmethod
    async def apredict_messages(
        self, messages: List[BaseMessage], *, stop: Optional[Sequence[str]] = None
    ) -> BaseMessage:
        """Predict message from messages."""

    def get_token_ids(self, text: str) -> List[int]:
        """Get the token present in the text."""
        return _get_token_ids_default_method(text)

    def get_num_tokens(self, text: str) -> int:
        """Get the number of tokens present in the text."""
        return len(self.get_token_ids(text))

    def get_num_tokens_from_messages(self, messages: List[BaseMessage]) -> int:
        """Get the number of tokens in the message."""
        return sum([self.get_num_tokens(get_buffer_string([m])) for m in messages])

    @classmethod
    def all_required_field_names(cls) -> Set:
        all_required_field_names = set()
        for field in cls.__fields__.values():
```



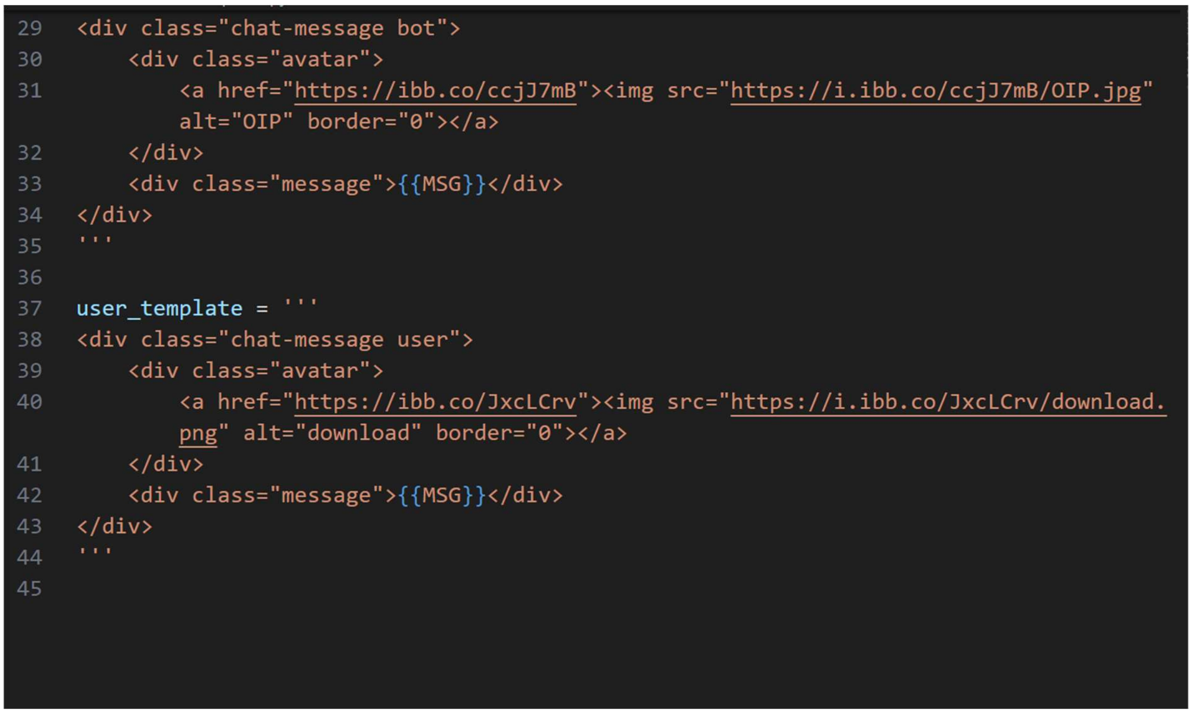
```python
class BaseLanguageModel(BaseModel, ABC):
    def get_num_tokens(self, text: str) -> int:
        """Get the number of tokens present in the text."""
        return len(self.get_token_ids(text))

    def get_num_tokens_from_messages(self, messages: List[BaseMessage]) -> int:
        """Get the number of tokens in the message."""
        return sum([self.get_num_tokens(get_buffer_string([m])) for m in messages])

    @classmethod
    def all_required_field_names(cls) -> Set:
        all_required_field_names = set()
        for field in cls.__fields__.values():
            all_required_field_names.add(field.name)
            if field.has_alias:
                all_required_field_names.add(field.alias)
        return all_required_field_names

```



```python
css = '''
<style>
.chat-message {
    padding: 1.5rem; border-radius: 0.5rem; margin-bottom: 1rem; display: flex
}
.chat-message.user {
    background-color: #2b313e
}
.chat-message.bot {
    background-color: #475063
}
.chat-message .avatar {
  width: 20%;
}
.chat-message .avatar img {
  max-width: 78px;
  max-height: 78px;
  border-radius: 50%;
  object-fit: cover;
}
.chat-message .message {
```

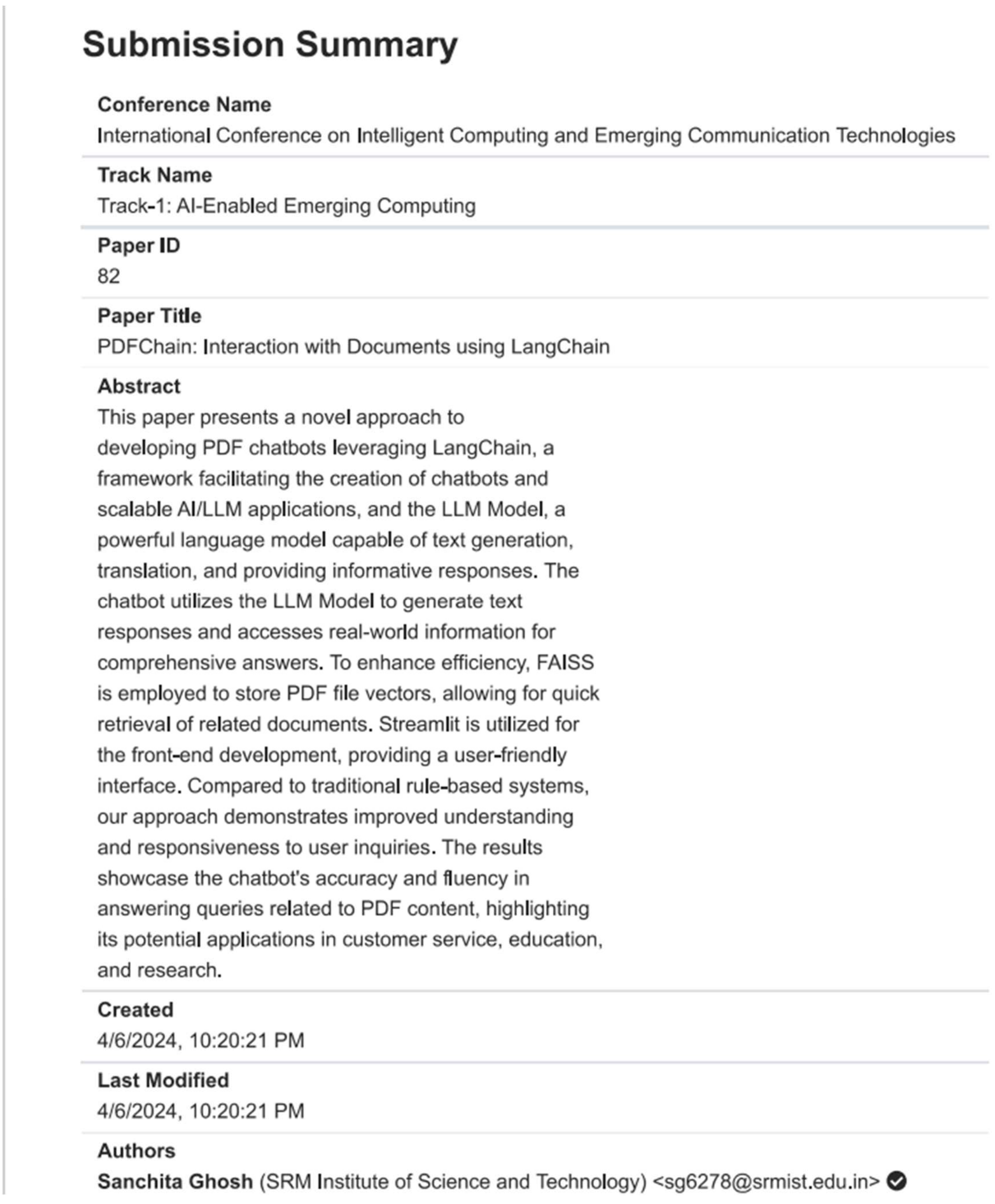


```
  width: 20%;
}
.chat-message .avatar img {
  max-width: 78px;
  max-height: 78px;
  border-radius: 50%;
  object-fit: cover;
}
.chat-message .message {
  width: 80%;
  padding: 0 1.5rem;
  color: #fff;
}
'''

bot_template = '''
<div class="chat-message bot">
    <div class="avatar">
        <a href="https://ibb.co/ccjJ7mB"><img src="https://i.ibb.co/ccjJ7mB/OIP.jpg"
        alt="OIP" border="0"></a>
    </div>
    <div class="message">{{MSG}}</div>
</div>
```



```
<div class="chat-message bot">
    <div class="avatar">
        <a href="https://ibb.co/ccjJ7mB"><img src="https://i.ibb.co/ccjJ7mB/OIP.jpg"
        alt="OIP" border="0"></a>
    </div>
    <div class="message">{{MSG}}</div>
</div>
'''

user_template = '''
<div class="chat-message user">
    <div class="avatar">
        <a href="https://ibb.co/JxcLCrv"><img src="https://i.ibb.co/JxcLCrv/download.
        png" alt="download" border="0"></a>
    </div>
    <div class="message">{{MSG}}</div>
</div>
'''
```

PAPER PUBLICATION



Submission Summary

Conference Name
International Conference on Intelligent Computing and Emerging Communication Technologies

Track Name
Track-1: AI-Enabled Emerging Computing

Paper ID
82

Paper Title
PDFChain: Interaction with Documents using LangChain

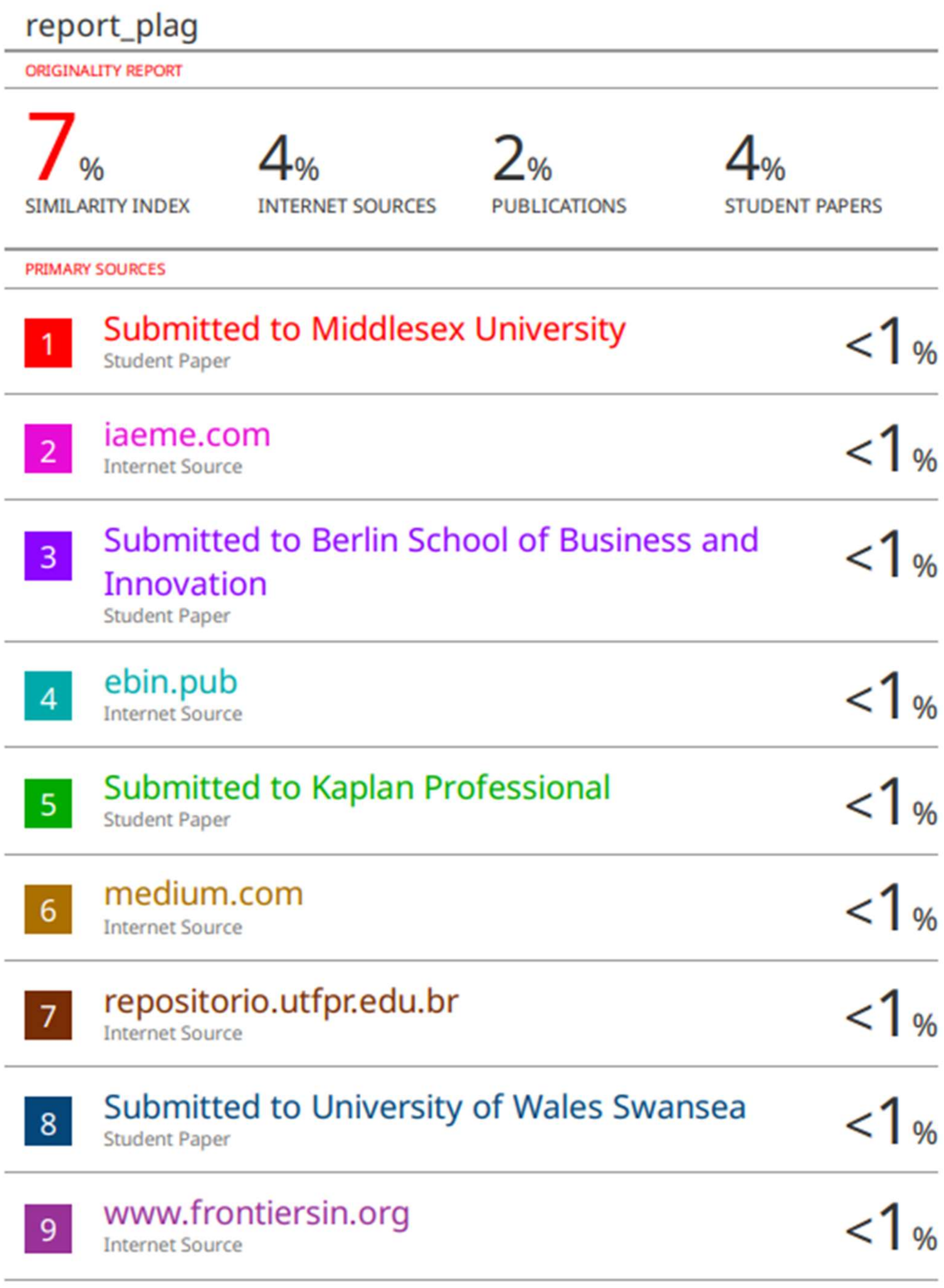
Abstract
This paper presents a novel approach to developing PDF chatbots leveraging LangChain, a framework facilitating the creation of chatbots and scalable AI/LLM applications, and the LLM Model, a powerful language model capable of text generation, translation, and providing informative responses. The chatbot utilizes the LLM Model to generate text responses and accesses real-world information for comprehensive answers. To enhance efficiency, FAISS is employed to store PDF file vectors, allowing for quick retrieval of related documents. Streamlit is utilized for the front-end development, providing a user-friendly interface. Compared to traditional rule-based systems, our approach demonstrates improved understanding and responsiveness to user inquiries. The results showcase the chatbot's accuracy and fluency in answering queries related to PDF content, highlighting its potential applications in customer service, education, and research.

Created
4/6/2024, 10:20:21 PM

Last Modified
4/6/2024, 10:20:21 PM

Authors
Sanchita Ghosh (SRM Institute of Science and Technology) <sg6278@srmist.edu.in> ✓

PLAGIARISM REPORT



report_plag

ORIGINALITY REPORT

7%	4%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Middlesex University Student Paper	<1%
2	iaeme.com Internet Source	<1%
3	Submitted to Berlin School of Business and Innovation Student Paper	<1%
4	ebin.pub Internet Source	<1%
5	Submitted to Kaplan Professional Student Paper	<1%
6	medium.com Internet Source	<1%
7	repositorio.utfpr.edu.br Internet Source	<1%
8	Submitted to University of Wales Swansea Student Paper	<1%
9	www.frontiersin.org Internet Source	<1%