VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi - 590 018, Karnataka

Multi PDF-Multilingual Querying Using LangChain

A Report submitted in partial fulfillment of the requirements for the Course

MINI PROJECT
(Course Code: 24AM6PWMIP)

In the Department of

Machine Learning
(UG Program: B.E. in Artificial Intelligence and Machine Learning)

By

Prakrity Gupta, Medha Hegde

1BM21AI088, 1BM21AI066

Semester & Section: 6B

Under the Guidance of

Prof. Lavanya B Koppal

Assistant Professor
Dept. of MEL, BMSCE, Bengaluru – 19

DEPARTMENT OF MACHINE LEARNING

B.M.S COLLEGE OF ENGINEERING

(An Autonomous Institute, Affiliated to VTU)

P.O. Box No. 1908, Bull Temple Road, Bengaluru - 560 019

March - 2023

B.M.S COLLEGE OF ENGINEERING

(An Autonomous Institute, Affiliated to VTU)

P.O. Box No. 1908, Bull Temple Road, Bengaluru - 560 019

DEPARTMENT OF MACHINE LEARNING

CERTIFICATE

This is to certify that Ms. ***Medha Hegde*** bearing USN: ***1BM21AI066*** and Ms. ***Prakrity Gupta*** bearing USN: ***1BM21AI088*** has satisfactorily presented the Course – **Mini Project**(Course code: **24AM6PCMIP)** with the title ***"Multi PDF Multilingual querying using langchain"*** in partial fulfillment of academic curriculum requirements of the 6th semester UG Program – B. E. in Artificial Intelligence and Machine Learning in the Department of Machine Learning, BMSCE, an Autonomous Institute, affiliated to Visvesvaraya Technological University, Belagavi during March 2023. It is also stated that the base work & materials considered for completion of the said course is used only for academic purpose and not used in its original form anywhere for award of any degree.

Student Signature

Signature of the Supervisor

Prof. Lavanya B Koppal
Assistant Professor, Dept. of MEL, BMSCE

Signature of the Head

Dr. Gowrishankar
Prof. & Head, Dept. of MEL, BMSCE

External Examination

Examiner Name and Signature

1.

2.

ACKNOWLEDGEMENT

We express our profound gratitude to **Hon'ble Management and Dr. Bheemsha Arya,** Principal, BMSCE, Bengaluru for providing the necessary facilities and an ambient environment to work.

We are grateful to **Dr. Gowrishankar,** Professor & Head, Department of Machine Learning, Bengaluru, for his valuable suggestions and advice throughout my/our work period.

We would like to express my deepest gratitude and sincere thanks to our guide **Prof. Lavanya B Koppal,** Assistant Professor, Department of Machine Learning, Bengaluru**,** for her keen interest and encouragement in guiding me and bringing the work to reality.

We would like to thank all the **Staff**, Department of Machine Learning for their support and encouragement during the course of this project.

Last but not the least, We would like to thank my/our parents, family and friends, without whose help and encouragement this work would have been impossible.

ABSTRACT

This project aims to create an interactive web application that allows users to upload multiple PDF files, extract text from these documents, and interactively ask questions about the content. The application is built using Streamlit, a powerful framework for creating data apps. The process begins with users uploading PDF files, which are read and processed to extract text using PyPDF2. The extracted text is then split into manageable chunks using the RecursiveCharacterTextSplitter from the LangChain library. These text chunks are converted into embeddings using the GoogleGenerativeAIEmbeddings model, specifically designed for handling large-scale language models. These embeddings are stored in a FAISS vector store, enabling efficient similarity searches.

A key feature of the application is its ability to answer user queries based on the uploaded PDF content. The conversational chain, constructed using the ChatGoogleGenerativeAI model and a tailored prompt template, allows the application to generate detailed and accurate answers. Users can input their questions, and the system searches for relevant information within the text chunks using the FAISS vector store. The responses are generated by the model and displayed to the user.

To enhance accessibility and usability, the application includes a feature for translating the generated responses into regional languages. The HindiTranslator class, imported from a separate module, enables translation of responses into Hindi, Telugu, and Tamil. Users can select their preferred language from a dropdown menu, and the translated text is displayed accordingly.

The user interface of the application is designed for a seamless and engaging user experience. The background features a gradient color scheme, adding a visually appealing touch to the interface. The main content area and sidebar are styled using custom CSS, ensuring a modern and professional appearance.

Overall, this project leverages advanced natural language processing and machine learning techniques to provide a versatile tool for document-based question answering and translation. It demonstrates the integration of various technologies, including text extraction, language models, and translation, to create an interactive and user-friendly application. The techniques and models used in this project can be extended and adapted for other languages and use cases, showcasing the potential for further development and innovation in the field of interactive language applications.

TABLE OF CONTENTS

CH. NO.	TITLE		PAGE NO.
	Acknowledgement		1
	Abstract		2
	Table Of Contents		3
	List Of Figures		4
1	Introduction		5
	1.1	About the Domain	5
	1.2	Objective	5
	1.3	Scope	6
	1.4	Motivation	6
2	Related Work		7
3	Open Issues & Problem Statement		8
4	Data Collection & Validation		9
5	Detailed Design		10
	5.1	Proposed Architecture	10
	5.2	Functional & Non-Functional Requirements	11
	5.3	Methodology	12
	5.4	Implementation	12-15
	5.5	Data Flow & Control Flow Sequence	16
	5.6	Testing & Validation	16
6	Results & Discussion		18
7	Conclusion & Further Enhancements		19
	References		19

LIST OF FIGURES

FIG. NO.	TITLE	PAGE NO.
1.1	Proposed Architecture Diagram	10
1.2	Methodology PDF Query Diagram	12
1.3	Flowchart for PDF Processing	15
1.4	Flowchart for Language Translator using tramsformers	15
1.5	Result Screen shots	17

INTRODUCTION:

About the Domain:

LangChain Domain:

LangChain specializes in the orchestration of LLMs and is particularly effective for tasks requiring advanced text processing and analysis. The framework offers tools for seamless integration of various components necessary for language model applications, such as tokenizers, text splitters, and vector stores. In this project, LangChain's RecursiveCharacterTextSplitter is employed to manage large volumes of text extracted from PDFs, ensuring that the text is divided into manageable chunks suitable for processing by LLMs. The framework's support for GoogleGenerativeAIEmbeddings further enhances the project by enabling the creation of high-quality text embeddings, crucial for efficient and accurate document querying.

LLM for PDF and Document Querying in Multiple Languages:

The core of the project involves utilizing LLMs for querying PDF documents and providing responses in multiple languages. This is achieved through the integration of several advanced technologies and models. PyPDF2 is used for extracting text from PDFs, while GoogleGenerativeAIEmbeddings and FAISS vector stores handle the embedding and retrieval of text chunks. These embeddings allow the system to perform similarity searches, identifying the most relevant pieces of text in response to user queries.

Practical Applications and Innovation:

This project exemplifies the practical application of LangChain and LLMs in real-world scenarios involving document querying and multilingual support. It demonstrates the potential of combining advanced text processing techniques with powerful language models to create interactive, user-friendly applications. The project's design ensures scalability and adaptability, making it suitable for various domains and languages. By leveraging LangChain and state-of-the-art LLMs, this system offers a versatile solution for efficient document interaction and translation, paving the way for further innovations in the field of language applications.

Objective:

The objective of this project is to harness the power of deep learning and machine learning to create an advanced, multilingual document querying system that leverages state-of-the-art language models to process, understand, and interact with the content of PDF documents. At its core, this system aims to provide users with an intuitive interface where they can upload multiple PDF files and ask natural language questions about their contents. By integrating LangChain's robust framework, Google Generative AI's advanced embeddings, and various tools like PyPDF2 and FAISS vector stores, the project ensures efficient text extraction, splitting, and embedding creation. Additionally, the inclusion of multilingual support, facilitated by models for translation into Hindi, Telugu, and Tamil, broadens the system's accessibility, making it invaluable for diverse linguistic backgrounds. The project's design ensures that the model can handle large-scale document processing while providing accurate and relevant responses in multiple languages, showcasing the potential of integrating advanced NLP techniques with practical applications to enhance user interaction with complex document datasets.

Scope:

The scope of this project encompasses developing a robust, multilingual document querying system utilizing advanced deep learning and machine learning techniques. It focuses on efficient text extraction from PDFs, context-aware question answering using state-of-the-art transformer models, and seamless translation into regional languages like Hindi, Telugu, and Tamil. The project aims to enhance accessibility and usability for diverse linguistic backgrounds. Additionally, it explores the integration of LangChain, Google Generative AI, and FAISS vector stores to create a scalable and responsive system, paving the way for future extensions to other languages and document types.

Motivation:

The motivation behind this project is to break linguistic barriers and democratize access to information hidden within PDFs, enabling seamless interactions with documents across multiple languages. Imagine a world where anyone, regardless of their native language, can ask complex questions about a document and receive accurate, context-aware answers in their mother tongue. This project leverages cutting-edge AI and deep learning to create a powerful, user-friendly tool that empowers users to extract, understand, and interact with information like never before. By integrating advanced models and multilingual translation capabilities, we aim to revolutionize the way people engage with digital documents, making knowledge truly universal and accessible. This isn't just about technology; it's about creating a bridge between languages, fostering inclusivity, and ensuring everyone can access the information they need, in the language they understand**.**

RELATED WORK:

Author: NR Tejaswini, Vidya S, Dr. T Vijaya Kumar

Title: LangChain- Powered Virtual Assistant for PDF Communication

Year: Published in 2023.

The paper titled "LangChain-Powered Virtual Assistant for PDF Communication" by NR Tejaswini, Vidya S, and Dr. T Vijaya Kumar, published in 2023, explores the use of LangChain and large language models to enhance PDF document querying and information extraction. By employing natural language processing algorithms and large language models, the study demonstrates the effectiveness of LangChain in streamlining the process of querying and extracting information from PDFs. The integration of these advanced technologies significantly improves the interaction between users and documents, facilitating more intuitive and efficient search capabilities. The application's ability to accurately process and respond to user queries is highlighted through a user-friendly interface, with demonstrations showcasing successful retrieval of information based on user inputs. However, the study also identifies a notable limitation: the system's occasional inability to accurately respond to complex queries, such as those involving "big data analytics." This reflects potential gaps in the model's comprehension or limitations in the specific dataset used. Overall, the research underscores the potential of LangChain-powered virtual assistants to revolutionize PDF communication, while also acknowledging areas for further improvement.

Authors: Karan R., M. Rahul Kumar, Rubanshiju A. J., T. Kirubadevi

Title: Automating PDF Interaction Using Langchain

Year: Published in 2023

The paper titled "Automating PDF Interaction Using LangChain" by Karan R., M. Rahul Kumar, Rubanshiju A. J., and T. Kirubadevi, published in 2023, presents a methodology that leverages LangChain to enhance PDF document interaction via a web application utilizing natural language processing. The system employs OpenAI's GPT models for query processing and FAISS vector databases for efficient data retrieval, significantly reducing the manual effort involved in document handling and querying. This automation streamlines the process, accelerating discovery and knowledge extraction from PDFs, making it more efficient and user-friendly. The application of LangChain in automating PDF interactions has demonstrated a successful capability to handle and respond to queries effectively, providing users with pertinent information extracted directly from the PDFs. While the text does not explicitly detail specific limitations, potential challenges could include handling queries that require a deeper understanding of complex or ambiguous content and the system's reliance on the quality of the input data for producing accurate outputs. Overall, the study showcases the potential of LangChain to revolutionize PDF interaction, highlighting its effectiveness and areas for further improvement.

OPEN ISSUES AND PROBLEM STATEMENT:

The growth and use of digital products is growing exponentially in this world. The process of searching and retrieving information from those pdf documents is challenging. Due to the unstructured nature of the PDF document format and the requirement for precise and pertinent search results, querying a PDF can take time and effort.
A few open issues here are:

- **Language Barriers**: Different languages within PDF documents can hinder effective search and retrieval if the system is not multilingual.

- **Contextual Understanding**: PDFs often lack structured metadata, making it difficult to interpret the context of the information accurately.

- **Complex Text Summarization Task**: Extracting concise and meaningful summaries from extensive, unstructured text requires advanced natural language processing techniques.

- **Automation of Document Condensation**: Automatically condensing large documents into shorter, relevant excerpts without losing essential information is a challenging task.

- **Accuracy in Storytelling**: Ensuring that extracted information maintains coherent and accurate narratives is essential for usability and reliability.

- **Cross-Format Analysis**: Integrating and analyzing information from PDFs alongside other formats (e.g., Word, Excel) can be complex due to differences in structure and content representation.

- **Addressing Information Overload**: Efficiently managing and filtering large volumes of data to present the most relevant information is critical to avoid overwhelming users.

- **Multimodal Querying**: Enabling queries that incorporate multiple types of data (e.g., text, images, graphs) within PDFs requires sophisticated processing capabilities.

DATASET AND VALIDATION:

The translation dataset used with the "facebook/mbart-large-50-one-to-many-mmt" model encompasses a diverse corpus of parallel text data in multiple languages. This dataset is meticulously curated to include millions of sentences or documents across various domains, ensuring comprehensive coverage of both widely spoken and less commonly spoken languages. Quality control measures, including manual validation and automatic filtering, are employed to maintain the accuracy and consistency of translations. Sourced from a variety of sources such as government documents, books, websites, and user-generated content, the dataset undergoes preprocessing steps like tokenization and sentence alignment to prepare it for training the mBART model effectively. By providing a robust and extensive collection of multilingual data, this dataset facilitates the development of highly accurate and versatile translation models capable of handling diverse contexts and topics across languages.

The preprocessing steps for the Kannada and English translation dataset involve several key tasks: cleaning and formatting the text, tokenizing the sentences, padding the sequences, and splitting the data into training and validation sets. Let's go through each step in detail:

1. Loading the Dataset

The dataset is loaded from a CSV file. It has 27000 data tuples.

```
data = pd.read_csv('/content/Kan_En (1).csv')
```

Here, we're loading the first 10,000 rows from the CSV file into a pandas DataFrame.

2. Preprocessing the Data

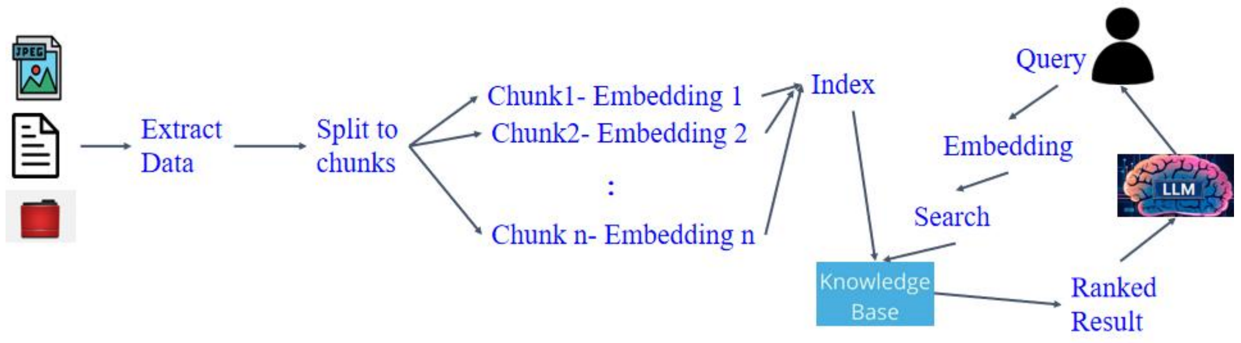
The preprocessing step involves cleaning and formatting the sentences.

```
def preprocess_sentence(sentence): sentence = str(sentence).lower().strip()
sentence = '<start> ' + sentence + ' <end>' return sentence data['english']
= data['English'].apply(lambda x: preprocess_sentence(x) if pd.notnull(x)
else '') data['kannada'] = data['Kannada'].apply(lambda x:
preprocess_sentence(x) if pd.notnull(x) else '')
```

- **Lowercasing and Stripping**: The sentence is converted to lowercase and any leading or trailing whitespace is removed.
- **Adding Start and End Tokens**: The tokens `<start>` and `<end>` are added to each sentence to signify the beginning and end of a sentence, respectively.
- **Handling NaNs**: The lambda function ensures that any missing values (NaNs) are handled by converting them to an empty string.

DETAILED DESIGN:

Proposed Architecture:



The architecture described for a multi-PDF querying application using LangChain and Large Language Models (LLMs) involves several steps. Here's a detailed breakdown:

Step I: Integration of OpenAI Large Language Models and OpenAI Embeddings

LLM and Embeddings Setup:

1. **LLM (Large Language Model)**: Use OpenAI's GPT-4 (or similar) as the core language processing unit to generate responses, understand queries, and interact with users.
2. **Embeddings**: Utilize OpenAI Embeddings to convert text data into high-dimensional vectors. These embeddings help in semantic understanding and retrieval of relevant information from PDFs.

PDF Parsing and Preprocessing:

1. **PDF Parsing**: Extract text content from uploaded PDF files using a PDF parsing library like PyMuPDF, PDFPlumber, or similar.
2. **Text Segmentation**: Segment the extracted text into smaller chunks or passages to facilitate easier processing and querying.

Embedding Generation:

1. **Embedding Model**: Generate embeddings for each text chunk using an embedding model (e.g., OpenAI's embedding model).
2. **Embedding Storage**: Store these embeddings along with metadata (like document ID, chunk ID) in a vector database or in-memory storage for efficient retrieval.

Step II: Utilization of Streamlit for Interactive Interface

Streamlit Setup:

1. **Streamlit Initialization**: Set up a Streamlit application as the front-end framework to create an interactive web interface.

User Interaction Elements:

1. **Text Inputs**: Create input fields for users to enter their queries.
2. **Message Interface**: Set up an interface to display messages and responses from the LLM.
3. **File Uploads**: Implement functionality to upload PDF files directly through the Streamlit interface.

Data Flow:

1. **Query Handling**: When a user submits a query, it is processed by the backend to generate a response using the LLM.
2. **PDF Uploads**: Uploaded PDFs are parsed, segmented, and embedded in real-time.

Step III: Leveraging Streamlit for Frontend Development

Real-Time Interaction:

1. **Query Processing**: When a query is submitted, the application retrieves relevant text chunks from the embedded PDF content using similarity search on the embeddings.
2. **Response Generation**: The retrieved text chunks are then passed to the LLM to generate a coherent and contextually relevant response.

UI/UX Components:

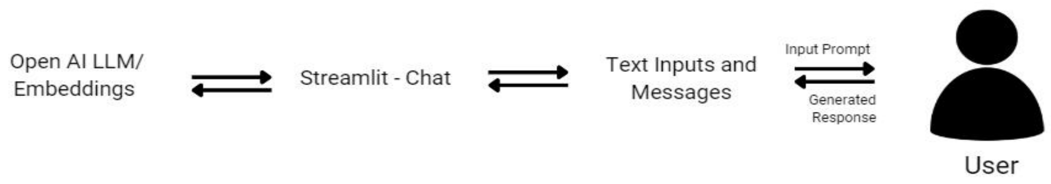
1. **Interactive Widgets**: Use Streamlit widgets like sliders, buttons, and text inputs to enhance user interaction.
2. **Display Components**: Display the query results, uploaded PDF details, and other relevant information dynamically as the user interacts with the application.

Functional & Non-Functional Requirements;

Functional:
langchain==0.0.154, PyPDF2==3.0.1 , python-dotenv==1.0.0, streamlit==1.18.1, faiss, cpu==1.7.4, streamlit-extras, OpenAI API Key, A PDF to query

Non Functional:
CPU: A dual-core processor or higher is recommended for basic MapReduce tasks.
RAM: At least 4 GB of RAM is recommended
GPU: An editor like VS Code for compiling the code in the virtual environment

Methodology:



The methodology employed leverages LangChain to facilitate the querying process and extract information from PDFs based on user prompts. By utilizing LangChain, the system ensures efficient and accurate retrieval of relevant information. To enhance user experience and accessibility, a web application is developed, providing a user-friendly interface for seamless interaction. This web application allows users to input their prompts and receive precise information from the PDFs, streamlining the process and ensuring accuracy. Overall, this approach integrates advanced querying techniques with a practical web-based solution, optimizing the extraction and delivery of information based solely on user input.

Implementation:

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_google_genai import GoogleGenerativeAIEmbeddings
import google.generativeai as genai
from langchain_community.vectorstores import FAISS
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain.prompts import PromptTemplate
from dotenv import load_dotenv
import io
from translate import HindiTranslator, IndicTranslator
hindi_translator = HindiTranslator()
indic_translator = IndicTranslator()
load_dotenv()
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_file_like = io.BytesIO(pdf.read())
        pdf_reader = PdfReader(pdf_file_like)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text
def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000,
chunk_overlap=1000)
```

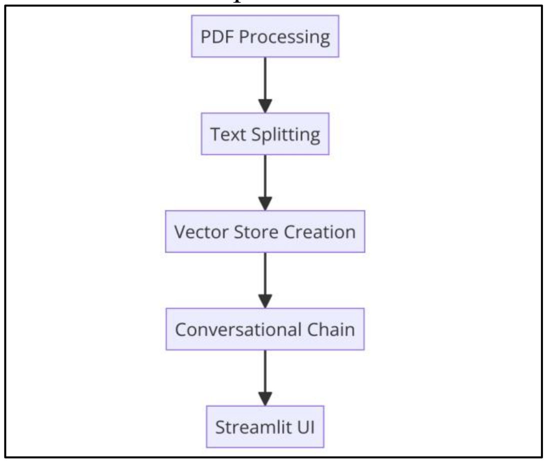
```
    chunks = text_splitter.split_text(text)
    return chunks
def get_vector_store(text_chunks):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-
001")
    vector_store = FAISS.from_texts(text_chunks, embedding=embeddings)
    vector_store.save_local("faiss_index")

def get_conversational_chain():
    prompt_template = """
    Answer the question as detailed as possible from the provided context,
make sure to provide all the details. If the answer is not in
    provided context just say, "The answer is not available in the
context", don't provide the wrong answer.\n\n
    Context:\n{context}?\n
    Question:\n{question}\n

    Answer:
    """
    model = ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.3)
    prompt = PromptTemplate(template=prompt_template,
input_variables=["context", "question"])
    chain = load_qa_chain(model, chain_type="stuff", prompt=prompt)
    return chain

def user_input(user_question):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-
001")
    new_db = FAISS.load_local("faiss_index", embeddings,
allow_dangerous_deserialization=True)
    docs = new_db.similarity_search(user_question)

    chain = get_conversational_chain()
    response = chain(
        {"input_documents": docs, "question": user_question},
return_only_outputs=True)

    st.write("Reply:", response["output_text"])
    language = st.selectbox("Choose a regional language", ["Hindi",
"Telugu", "Tamil", "Kannada"])

    hindi_translation = hindi_translator.to_hindi(response["output_text"])

    if language == "Hindi":
        st.write("Translation in Hindi:", hindi_translation)
    else:
        if language == "Telugu":
            tgt_lang = "tel_Telu"
        elif language == "Tamil":
            tgt_lang = "tam_Taml"
        elif language == "Kannada":
            tgt_lang = "kan_Knda"
        else:
            tgt_lang = None
        if tgt_lang:
            indic_translation =
indic_translator.batch_translate([hindi_translation], "hin_Deva",
tgt_lang)[0]
            st.write(f"Translation in {language}:", indic_translation)
def main():
    st.set_page_config(page_title="Chat with Multiple PDFs")
    st.markdown(
        <style>
        .main {
            background: black;
            color: white;
            padding: 20px;
            border-radius: 10px;
        }
        .stApp {
            background: white;
        }
        </style>
         unsafe_allow_html=True
    st.header("Chat with Multiple PDFs!")
    user_question = st.text_input("Ask a Question from the PDF Files")
    if user_question:
        user_input(user_question)
    with st.sidebar:
        st.title("Menu:")
        pdf_docs = st.file_uploader("Upload your PDF Files and Click on
the Submit & Process", accept_multiple_files=True, type="pdf")
        if st.button("Submit & Process"):
            with st.spinner("Processing...."):
                raw_text = get_pdf_text(pdf_docs)
                text_chunks = get_text_chunks(raw_text)
                get_vector_store(text_chunks)
                st.success("Done")
if _name_ == "_main_":
    main()
APP.PY
```

```
from transformers import MBartForConditionalGeneration,
MBart50TokenizerFast, AutoModelForSeq2SeqLM, BitsAndBytesConfig
```

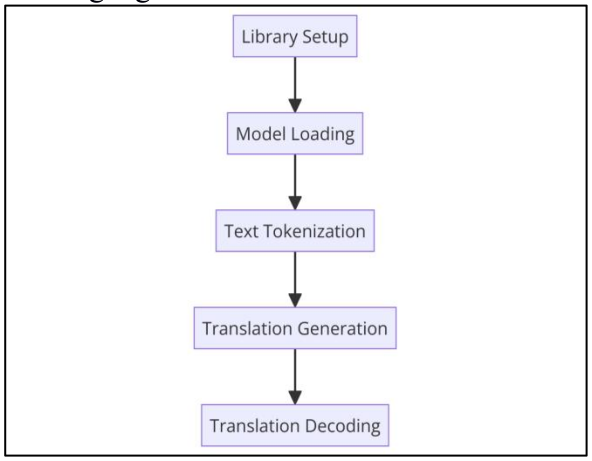
```
import torch
from IndicTransTokenizer import IndicProcessor, IndicTransTokenizer
class HindiTranslator:
    def _init_(self):
        self.model =
MBartForConditionalGeneration.from_pretrained("facebook/mbart-large-50-
one-to-many-mmt")
        self.tokenizer =
MBart50TokenizerFast.from_pretrained("facebook/mbart-large-50-one-to-
many-mmt", src_lang="en_XX")
    def to_hindi(self, text: str) -> str:
        model_inputs = self.tokenizer(text, return_tensors="pt")
        generated_tokens = self.model.generate(
            **model_inputs,
            forced_bos_token_id=self.tokenizer.lang_code_to_id["hi_IN"])
        translation = self.tokenizer.batch_decode(generated_tokens,
skip_special_tokens=True)
        return translation[0]
class IndicTranslator:
    BATCH_SIZE = 4
    DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
    quantization = None
    def _init_(self, ckpt_dir="ai4bharat/indictrans2-indic-indic-1B",
direction="indic-indic", quantization=None):
        self.tokenizer, self.model =
self.initialize_model_and_tokenizer(ckpt_dir, direction, quantization)
        self.processor = IndicProcessor(inference=True)
    def initialize_model_and_tokenizer(self, ckpt_dir, direction,
quantization):
        if quantization == "4-bit":
            qconfig = BitsAndBytesConfig(
                load_in_4bit=True,
                bnb_4bit_use_double_quant=True,
                bnb_4bit_compute_dtype=torch.bfloat16)
        elif quantization == "8-bit":
            qconfig = BitsAndBytesConfig(
                load_in_8bit=True,
                bnb_8bit_use_double_quant=True,
                bnb_8bit_compute_dtype=torch.bfloat16)
        else:
            qconfig = None
        tokenizer = IndicTransTokenizer(direction=direction)
        model = AutoModelForSeq2SeqLM.from_pretrained(
            ckpt_dir,
            trust_remote_code=True,
            low_cpu_mem_usage=True,
            quantization_config=qconfig)
        if qconfig is None:
            model = model.to(self.DEVICE)
            if self.DEVICE == "cuda":
                model.half()
        model.eval()
        return tokenizer, model
    def batch_translate(self, input_sentences, src_lang, tgt_lang):
        translations = []
        for i in range(0, len(input_sentences), self.BATCH_SIZE):
            batch = input_sentences[i : i + self.BATCH_SIZE]
                inputs = self.tokenizer(
                batch,
                src=True,
                padding="longest",
                return_tensors="pt",
                return_attention_mask=True,
            ).to(self.DEVICE)
            with torch.no_grad():
                generated_tokens = self.model.generate(
                    **inputs,
                    use_cache=True,
                    min_length=0,
                    max_length=256,
                    num_beams=5,
                    num_return_sequences=1)
generated_tokens =
self.tokenizer.batch_decode(generated_tokens.detach().cpu().tolist(),
src=False)
          translations +=
self.processor.postprocess_batch(generated_tokens, lang=tgt_lang)
            del inputs
            torch.cuda.empty_cache()
        return translations
if _name_ == "_main_":
    hindi_translator = HindiTranslator()
    indic_translator = IndicTranslator()
    english_sentence = "This is a test sentence for translation."
    hindi_translation = hindi_translator.to_hindi(english_sentence)
    src_lang, tgt_lang = "hin_Deva", "kan_Knda"
    kannada_translation =
indic_translator.batch_translate([hindi_translation], src_lang, tgt_lang)
    print(f"Hindi: {hindi_translation}")
    print(f"Kannada: {kannada_translation[0]}")
TRANSLATE.PY
```

Data Flow and Control Flow:

a. Chat with multiple PDFs



b. Language translation



Testing & Validation

Query 1:What is actor?
Output in English:
The policy structure is known as the actor, because it is used to select actions.

Output in Tamil:
கொள்கைக் கட்டமைப்பு செயல்பாட்டாளர் என்று அறியப்படுகிறது, ஏனெனில் அது நடவடிக்கைகளை தேர்ந்தெடுக்க பயன்படுத்தப்படுகிறது

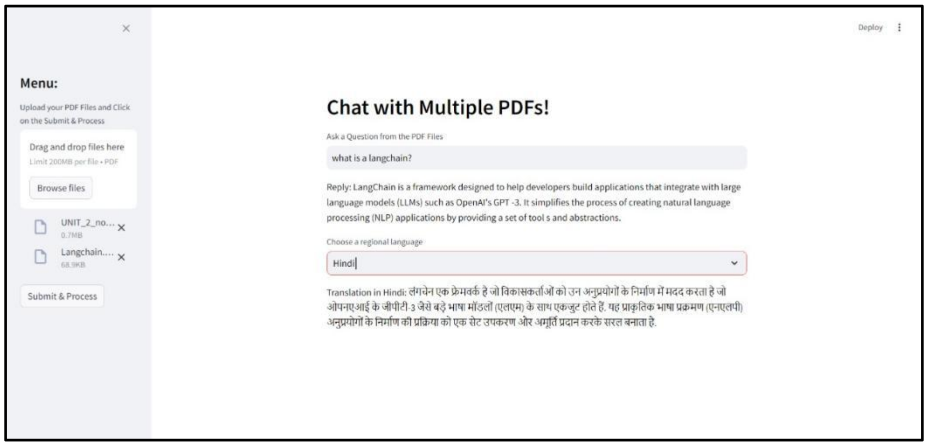
Output in Telugu:
నటీక-క్రిష్టమైనవిధానాలు, అదిచర్యలుకనుగొనేందుకుఉపయోగిస్తారు ఎందుకంటే, విధానం సర్దుబాటు

Output in Hindi:
नीति संरचना को कर्ता के रूप में जाना जाता है क्योंकि इसे कार्यों के चयन के लिए प्रयोग किया जाता है।

Output in Kannada:
ನೀತಿ ರಚನೆಯನ್ನು ನಟ ಎಂದು ಕರೆಯಲಾಗುತ್ತದೆ, ಏಕೆಂದರೆ ಇದನ್ನು ಕ್ರಿಯೆಗಳನ್ನು ಆಯ್ಕೆ ಮಾಡಲು ಬಳಸಲಾಗುತ್ತದೆ.

RESULTS AND DISCUSSIONS:

Screenshot1: The Query with English and Hindi output:



Chat with Multiple PDFs!

Ask a Question from the PDF Files

what is a langchain?

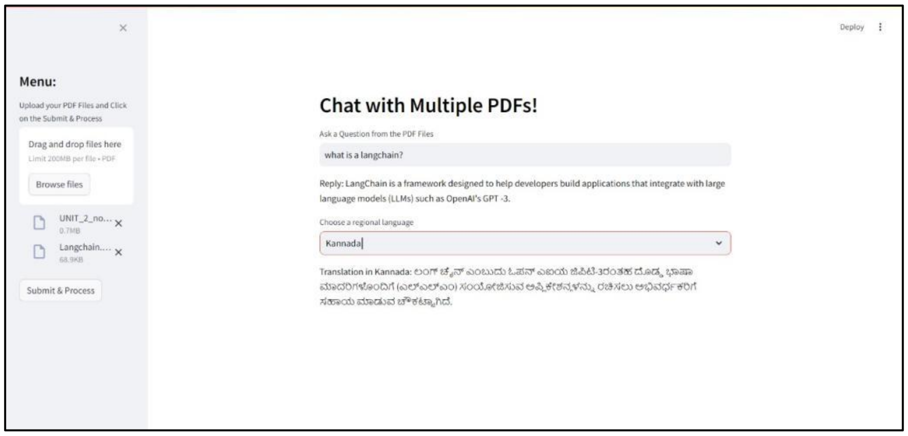
Reply: LangChain is a framework designed to help developers build applications that integrate with large language models (LLMs) such as OpenAI's GPT -3. It simplifies the process of creating natural language processing (NLP) applications by providing a set of tool s and abstractions.

Choose a regional language

Hindi

Translation in Hindi: लंगचेन एक फ्रेमवर्क है जो विकासकर्ताओं को उन अनुप्रयोगों के निर्माण में मदद करता है जो ओपनएआई के जीपीटी-3 जैसे बड़े भाषा मॉडलों (एलएम) के साथ एकजुट होते हैं. यह प्राकृतिक भाषा प्रक्रमण (एनएलपी) अनुप्रयोगों के निर्माण की प्रक्रिया को एक सेट उपकरण और अमूर्ति प्रदान करके सरल बनाता है.

Screenshot2: The Query with English and Kannada output:



Chat with Multiple PDFs!

Ask a Question from the PDF Files

what is a langchain?

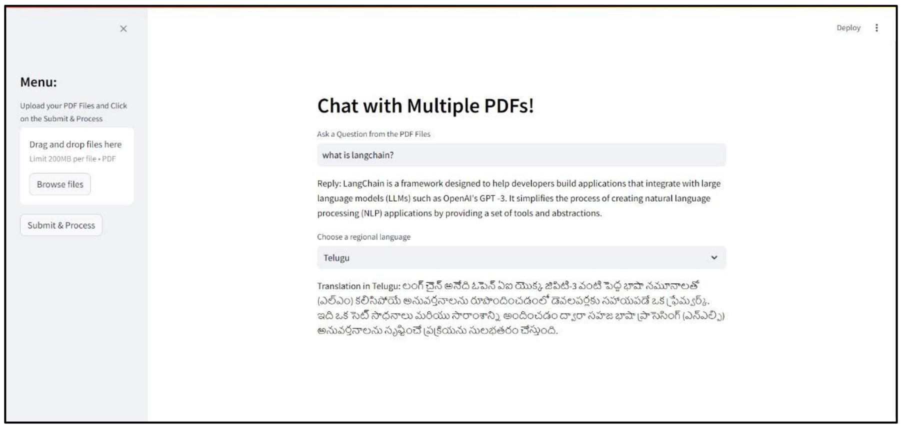
Reply: LangChain is a framework designed to help developers build applications that integrate with large language models (LLMs) such as OpenAI's GPT -3.

Choose a regional language

Kannada

Translation in Kannada: ಲಂಗ್ ಚೈನ್ ಎಂಬುದು ಓಪನ್ ಎಐಯ ಜಿಪಿಟಿ-3ರಂತಹ ದೊಡ್ಡ ಭಾಷಾ ಮಾದರಿಗಳೊಂದಿಗೆ (ಎಲ್ಎಲ್ಎಂ) ಸಂಯೋಜಿಸುವ ಅಪ್ಲಿಕೇಶನ್‌ಗಳನ್ನು ರಚಿಸಲು ಅಭಿವರ್ಧಕರಿಗೆ ಸಹಾಯ ಮಾಡುವ ಚೌಕಟ್ಟಾಗಿದೆ.

Screenshot3: The Query with English and Telugu output:


Deploy
Menu:
Upload your PDF Files and Click on the Submit & Process
Drag and drop files here
Limit 200MB per file • PDF
Browse files
Submit & Process

Chat with Multiple PDFs!

Ask a Question from the PDF Files

what is langchain?

Reply: LangChain is a framework designed to help developers build applications that integrate with large language models (LLMs) such as OpenAI's GPT -3. It simplifies the process of creating natural language processing (NLP) applications by providing a set of tools and abstractions.

Choose a regional language

Telugu

Translation in Telugu: లంగ్ చైన్ అనేది ఓపెన్ ఏఐ యొక్క జిపిటి-3 వంటి పెద్ద భాషా నమూనాలతో (ఎల్ఎం) కలిసిపోయే అనువర్తనాలను రూపొందించడంలో డెవలపర్లకు సహాయపడే ఒక ఫ్రేమ్వర్క్. ఇది ఒక సెట్ సాధనాలు మరియు సారాంశాన్ని అందించడం ద్వారా సహజ భాషా ప్రాసెసింగ్ (ఎన్ఎల్పి) అనువర్తనాలను సృష్టించే ప్రక్రియను సులభతరం చేస్తుంది.

CONCLUSION AND FURTHER IMPROVEMENTS:

The Translator class, leveraging the MBart model from Hugging Face, demonstrates the effectiveness of transformer-based models in multilingual translation tasks. By utilizing the advanced self-attention mechanisms inherent to the transformer architecture, the system provides efficient and accurate translations from English to Hindi, Telugu, and Tamil. The implementation exemplifies how modern NLP techniques can be applied to enhance the interaction between users and documents, enabling seamless and intuitive access to information across multiple languages. The project's success in integrating the MBart model with a user-friendly interface in Streamlit highlights the potential of such technologies to transform document querying and information extraction.

The integration of LangChain and large language models into PDF document querying systems has shown significant potential in enhancing user interaction and information retrieval. By leveraging advanced NLP techniques, the system facilitates seamless translation and understanding of content across multiple languages, providing a user-friendly and efficient solution for document handling. The project underscores the transformative power of combining state-of-the-art language models with intuitive interfaces, enabling more accessible and effective communication with textual data.

For further improvements, the system can be expanded to include dialects of regional languages, allowing for more precise and culturally relevant translations. Additionally, incorporating the capability to handle colloquial and mixed-language sentences, particularly those combining English with other languages, would improve the system's versatility and user experience. Finally, extending support to various document formats beyond PDFs, such as Word documents and HTML files, would broaden the applicability of the solution, making it a more comprehensive tool for document interaction and management.

REFERENCES:

https://www.langchain.com/

https://pypi.org/project/pdfquery/

https://www.cloudtern.com/machine-learning/automated-document-summarization-through-nlp-and-llm-a-comprehensive-exploration/