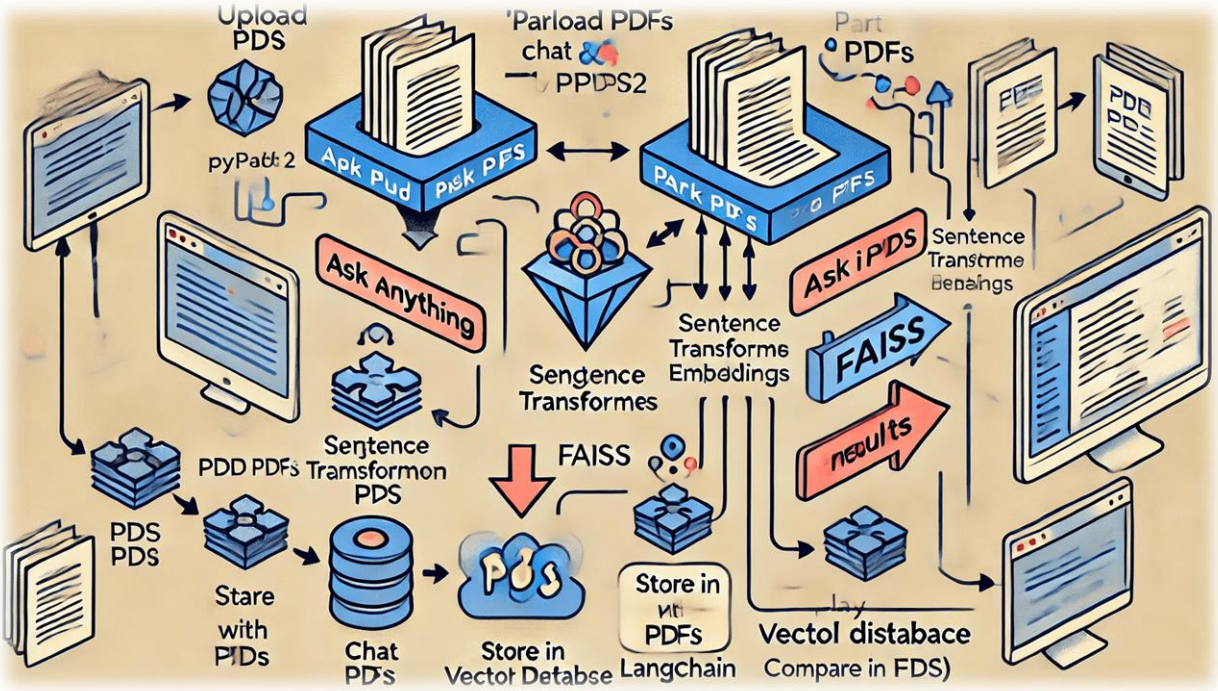


Smart PDF Explorer: A Local AI-Powered Chatbot for Document Insights

TABLE OF CONTENTS

1. Introduction

This project, **Smart PDF Explorer**, is designed to provide users with a powerful, interactive tool for exploring and analyzing PDF documents through natural language queries. It's built with a user-friendly interface that allows users to upload PDFs, ask questions about the content, and even compare multiple documents for insights—all powered by advanced AI capabilities working behind the scenes.

The core purpose of this project is to make document analysis straightforward and accessible. Whether a user needs specific information from one PDF or wants to compare details across several files, this tool simplifies the process by responding to questions in a conversational way.

In short, **Smart PDF Explorer** combines local embeddings, efficient storage, and a responsive language model to provide a robust, privacy-friendly document analysis experience.

2. Setup Requirements

To build this functionality, I used several key components and frameworks:

1. **Streamlit Interface**: This serves as the front end of the project. Streamlit's interactive elements allow users to easily upload PDFs, choose actions (such as "Ask Anything" or "Compare PDFs"), and view the chatbot's responses in a clear, conversational layout.
2. **LangChain Framework**: LangChain helps us manage the flow of information and handle complex document queries by enabling efficient retrieval techniques. It works as the main pipeline for how user queries are processed and answered by the language model.
3. **Local Embeddings**: We use a local embedding model, **SentenceTransformerEmbeddings** from the sentence-transformers library, which transforms the PDF content into vector representations. These embeddings are essential for comparing text content by capturing the meaning of the words and phrases within the documents.
4. **FAISS Vector Store**: For quick and efficient querying, we store the embeddings in a FAISS vector database, which helps retrieve similar documents or sections based on user queries. This allows the system to provide relevant responses quickly, even with large files.

5. **Local Language Model (LLM)**: We integrated a local LLaMA model for natural language processing to generate coherent, insightful responses. Running this LLM locally ensures data privacy and avoids dependency on external APIs, while still delivering reliable and context-aware answers.

Together, these components create a seamless workflow: PDFs are uploaded, parsed, and embedded into vectors, which are then stored for quick access and querying. The LLM handles user queries, searches for the most relevant document sections, and provides answers in a conversational format through the Streamlit interface.

In short, **Smart PDF Explorer** combines local embeddings, efficient storage, and a responsive language model to provide a robust, privacy-friendly document analysis experience.

2. Requirements

To set up and run **Smart PDF Explorer**, ensure you have the following tools and libraries installed. This list covers all the necessary software, libraries, and resources needed to implement the project effectively.

2.1 Software and Libraries

1. **Python 3.8+**: This project requires Python for scripting, running the server, and managing dependencies.
2. **Libraries**:
    - **Streamlit** (streamlit): Used to create the interactive web interface.
    - **Streamlit Option Menu** (streamlit_option_menu): For sidebar menu options within the Streamlit UI.
    - **PyPDF2** (pypdf2): For reading and parsing PDF documents.
    - **LangChain** (langchain): Manages document query and retrieval tasks, crucial for the chatbot functionality.
    - **ctransformers** (ctransformers): Allows the loading of local language models such as LLaMA for response generation.
    - **SentenceTransformers** (sentence-transformers): Provides SentenceTransformerEmbeddings for generating embeddings.
    - **FAISS** (faiss-cpu): A vector storage solution to store and retrieve document embeddings efficiently.
3. **Local Model Files**:

- **LLaMA Model**: A version of the LLaMA model (stored locally), to provide natural language processing and responses without external APIs.
- **SentenceTransformer Embedding Model**: all-MiniLM-L6-v2 for embedding PDF content locally.

2.2 Hardware Requirements

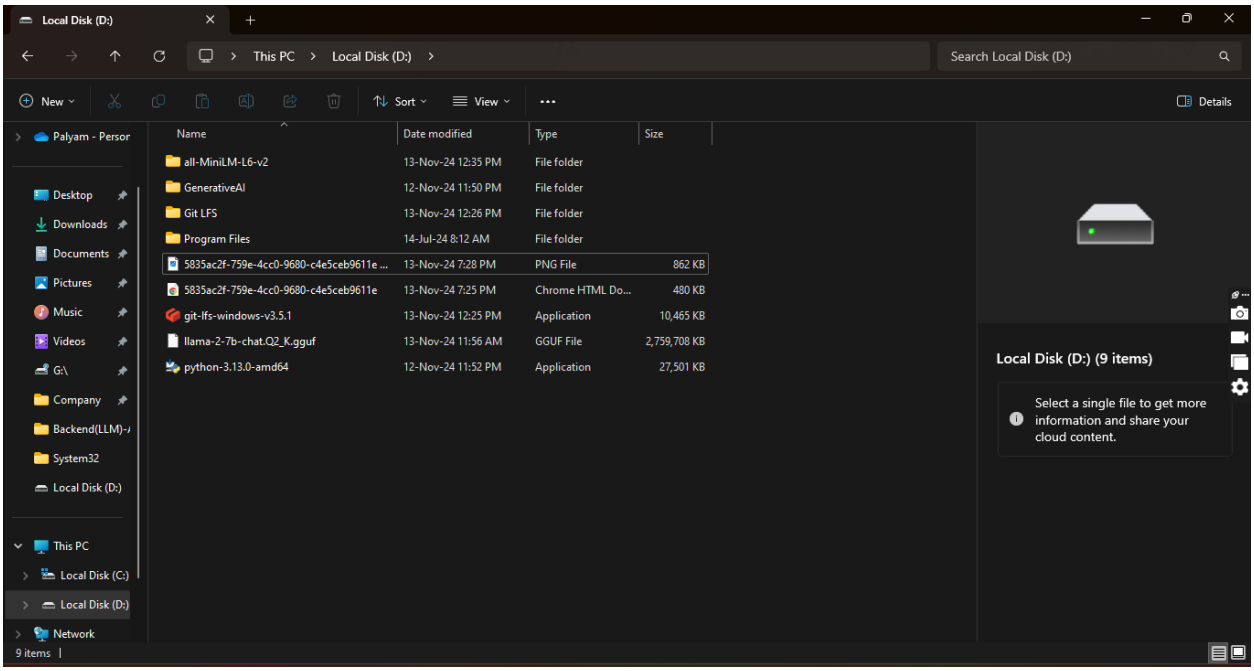
- **Storage**: Ensure enough storage space (approximately 10+ GB) for local model files and embeddings.
- **Memory (RAM)**: At least 8 GB of RAM is recommended for smooth operation, as working with local LLMs and embeddings can be memory-intensive.

2.3 Model Paths

Specify the paths to the local models within the code:

- **LLaMA Model Path**: Update the path to the locally stored LLaMA model file (llama-2-7b-chat.Q2_K.gguf).
- **SentenceTransformer Model Path**: Update the path to the SentenceTransformer model file (all-MiniLM-L6-v2).
- **I stored it in D**: since my c: have no space.

With these requirements met, the system can operate efficiently offline, providing privacy and speed without relying on external servers or APIs.



Screen shot of storing the files locally

3. Application Structure

The **Smart PDF Explorer** application is built with a modular structure, making it easy to maintain and extend. This section provides an overview of the key components, explaining how each module functions and interacts within the system to provide seamless PDF analysis and chatbot interactions.

3.1 Backend Framework

The backend is designed to process PDF documents, generate embeddings, and respond to user queries. The main components of the backend include:

- **LangChain**: This framework manages the pipeline for processing queries and retrieving the most relevant content from the PDFs. By using LangChain's retrieval capabilities, we can efficiently search through large text embeddings and return answers based on document content.
- **Local Embeddings**: The **SentenceTransformerEmbeddings** model from the sentence-transformers library is used to generate vector embeddings of the PDF content. Each document is transformed into an embedding that captures its semantic meaning, allowing for precise content retrieval during queries.
- **FAISS Vector Store**: We use FAISS (Facebook AI Similarity Search) as the vector storage solution. Embeddings generated from the PDFs are stored in this vector database, allowing fast, similarity-based searches. FAISS enables quick lookups, even with large document sizes, ensuring efficient performance.

```
import streamlit as st
from streamlit_option_menu import option_menu
from PyPDF2 import PdfReader
from langchain.embeddings import SentenceTransformerEmbeddings
from langchain.vectorstores import FAISS
from langchain.schema import Document
from ctransformers import AutoModelForCausalLM
from io import BytesIO
```

3.2 Frontend Framework

The frontend is implemented using **Streamlit**, which provides an interactive web-based interface for user interaction. The main features of the frontend include:

- **Sidebar Menu**: The sidebar in Streamlit allows users to upload PDFs and choose actions. Using the streamlit_option_menu library, we offer three main actions:
    - **Ask Anything**: Users can ask open-ended questions, answered by the language model without focusing on specific PDFs.
    - **Chat With PDFs**: Allows users to query specific content within the uploaded PDFs.
    - **Compare PDFs**: Enables comparisons of information across multiple PDFs.

```
# Sidebar menu for options
with st.sidebar:
    option = option_menu("Choose an Action", ["Ask Anything", "Chat With PDFs", "Compare PDFs"], default_index=1)
if option == "Chat With PDFs":
    st.title("Ask Questions About Your PDFs")
    chat_input = st.text_input("Enter your query")

    if chat_input:
        with st.spinner('Fetching insights...'):
            response = get_insights_from_query(chat_input)
```

```
            st.session_state["chat_history"].append({
                "query": chat_input,
                "response": response
            })
        display_chat()

# Option: Compare PDFs - Highlight differences and compare content
elif option == "Compare PDFs":
    st.title("Compare Your PDFs")
    compare_query = st.text_input("Enter a query to compare documents (e.g., revenue differences)")

    if compare_query:
        with st.spinner('Comparing documents...'):
            response = get_insights_from_query(compare_query)
            st.session_state["chat_history"].append({
                "query": compare_query,
                "response": response
            })
        display_chat()

# Option: Ask Anything - General LLM-based Chatbot
elif option == "Ask Anything":
    st.title("Ask Anything")
    essay_input = st.text_area("Enter a topic or question")

    if essay_input:
        with st.spinner('Generating response...'):
            response = llama_model(essay_input, max_new_tokens=200, temperature=0.7)
            st.session_state["chat_history"].append({
                "query": essay_input,
                "response": response
            })
        display_chat()

# Footer for the application
st.markdown("<div class='footer'>Made with 🤍 by Rohith</div>", unsafe_allow_html=True)
```

- **Custom UI Styling**: We use custom CSS for an enhanced user experience. Styling elements such as chat bubbles and a

responsive layout create a conversational feel and improve readability.

```
# Custom CSS for advanced UI styling
st.markdown("""
    <style>
        body {
            background: linear-gradient(to right, #f0f4f8, #d9e2ec);
            font-family: Arial, sans-serif;
        }
        .chat-container {
            padding: 15px;
            background-color: #f7fafc;
            border-radius: 20px;
            border: 1px solid #e0e0e0;
            margin-bottom: 15px;
            box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.05);
            font-size: 16px;
        }
        .user-bubble, .bot-bubble {
            display: flex;
            align-items: center;
            padding: 10px 15px;
            margin-bottom: 10px;
            font-size: 16px;
            border-radius: 20px;
        }
        .user-bubble {
            background-color: #0078D4;
            color: white;
            border: 2px solid #0078D4;
            border-radius: 20px 20px 0px 20px;
            justify-content: flex-start;
        }
        .bot-bubble {
            background-color: #f1f1f1;
            color: black;
            border: 2px solid #ffc107;
            border-radius: 20px 20px 20px 0px;
            justify-content: flex-start;
        }
        .avatar {
            width: 36px;
            height: 36px;
```

```
            border-radius: 50%;
            margin-right: 10px;
        }
        .footer {
            text-align: center;
            font-size: 14px;
            color: gray;
            padding-top: 10px;
            padding-bottom: 10px;
        }
        .stTextInput > div > div > input, .stTextArea > div > textarea {
            border-radius: 15px;
            padding: 10px;
            font-size: 16px;
        }
    </style>
""", unsafe_allow_html=True)

# Initialize session state for chat interaction and vector database
if "chat_history" not in st.session_state:
    st.session_state["chat_history"] = []
if "vector_database" not in st.session_state:
    st.session_state["vector_database"] = None
```

3.3 Key Modules and Functions

The primary functions and modules that enable the PDF Explorer's capabilities are:

- **PDF Parsing**: The parse_pdf function uses PyPDF2 to read and convert PDF pages into text, creating individual document objects for each page. These objects are then embedded and stored for retrieval.

```
# Function to parse PDF files and convert them into document objects
def parse_pdf(file):
    try:
        reader = PdfReader(BytesIO(file.read()))
        documents = [
            Document(page_content=page.extract_text(), metadata={"page": i + 1})
            for i, page in enumerate(reader.pages) if page.extract_text()
```

```
            ]
            return documents
        except Exception as e:
            st.error(f"Error parsing PDF: {e}")
            return []

```

- **Embedding Creation**: The SentenceTransformer model transforms each parsed document into vector embeddings, which are stored in the FAISS vector store. These embeddings capture the core meanings within the PDFs, allowing for relevant results during queries.

```
# Path to embeddings and Llama model files
model_path = "D:/"
llama_model_path = model_path + "llama-2-7b-chat.Q2_K.gguf"
embeddings_model_path = model_path + "all-MiniLM-L6-v2"

# Load LLaMA 2 model locally for LLM insights with quality-enhancing parameters
llama_model = AutoModelForCausalLM.from_pretrained(
    llama_model_path, model_type="llama", temperature=0.7, top_k=50
)
# Load SentenceTransformer locally for PDF vectorization
embeddings_model = SentenceTransformerEmbeddings(model_name=embeddings_model_path)

```

- **Vector Store and Retrieval**: Using the FAISS vector store, embeddings are organized and stored. During a user query, the get_insights_from_query function retrieves the most similar document sections by comparing the query's embedding with stored document embeddings, delivering relevant insights.

```
# Retrieve insights from PDF query with the top relevant result
def get_insights_from_query(query_text, top_k=3):  # Increase top_k for better context in comparison
    query_embedding = embeddings_model.embed_query(query_text)
```

```
    results = st.session_state["vector_database"].similarity_search_by_vector(query_embedding, k=top_k)
    insights = [result.page_content for result in results] if results else ["No relevant content found."]
    return "\n\n".join(insights) if insights else "No relevant response available."

```

- **LLM Integration**: The LLaMA model is loaded locally to handle both structured queries about the PDFs and open-ended questions in the "Ask Anything" section. This model enables nuanced, contextual responses by generating coherent, human-like answers.

```
# Path to embeddings and Llama model files
model_path = "D:/"
llama_model_path = model_path + "llama-2-7b-chat.Q2_K.gguf"
embeddings_model_path = model_path + "all-MiniLM-L6-v2"

# Load LLaMA 2 model locally for LLM insights with quality-enhancing parameters
llama_model = AutoModelForCausalLM.from_pretrained(
    llama_model_path, model_type="llama", temperature=0.7, top_k=50
)
# Load SentenceTransformer locally for PDF vectorization
embeddings_model = SentenceTransformerEmbeddings(model_name=embeddings_model_path)

```

- **Chat Display Function**: The display_chat function is responsible for rendering user queries and bot responses within a chat-like interface in Streamlit. The custom-designed chat bubbles enhance readability and simulate a conversational environment.

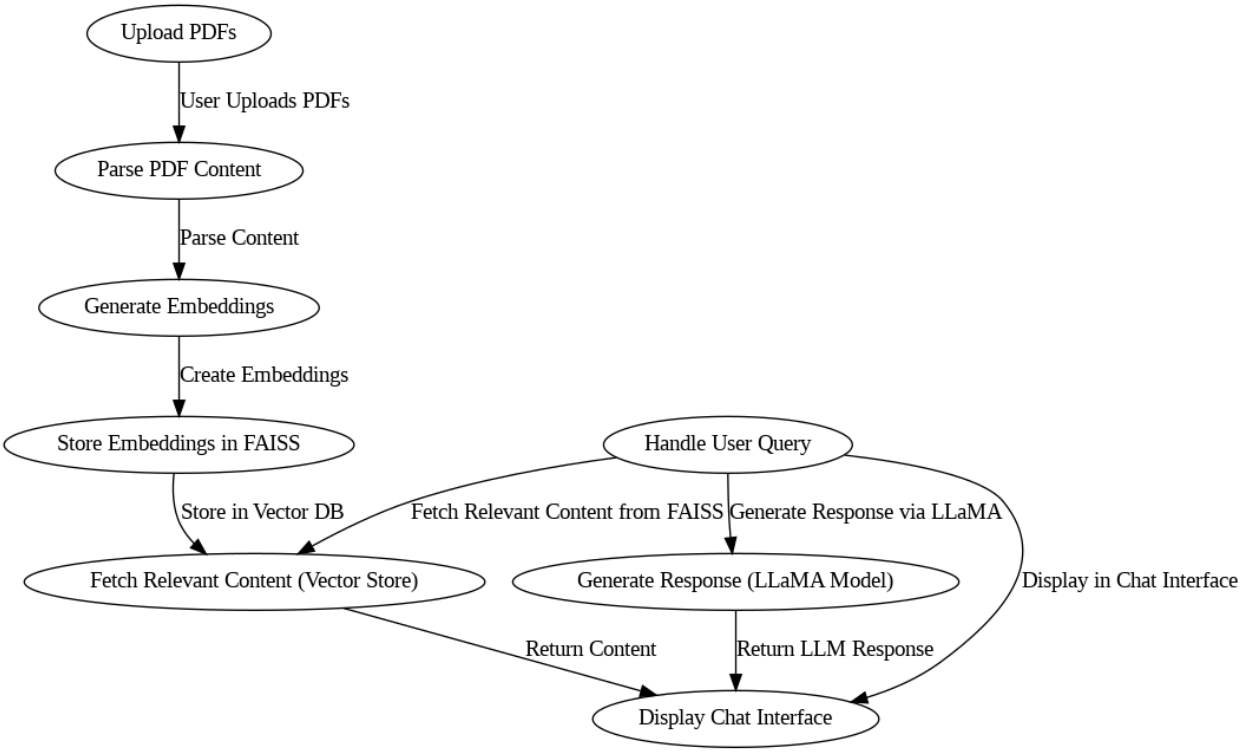
```
def display_chat():
    for i, entry in enumerate(st.session_state["chat_history"]):
        user_query = entry.get("query", "")
        bot_response = entry.get("response", "No response available.")
```

```
•
•           # User message styling
•           st.markdown(
•               f'<div class="user-bubble chat-container">'
•               f'<img src="https://img.icons8.com/color/48/000000/user.png"
   class="avatar"/>'
•               f'<b>User:</b> {user_query}</div>', unsafe_allow_html=True
•           )
•
•           # Bot response without icon, in a clean, simple format
•           st.markdown(
•               f'<div class="bot-bubble chat-container">'
•               f'{bot_response}</div>', unsafe_allow_html=True
•           )
•
```

3.4 Workflow Summary

1. **Upload PDFs**: Users upload one or more PDFs through the sidebar, which are then parsed and embedded.
2. **Store Embeddings**: Parsed document content is embedded and stored in the FAISS vector database.
3. **Query Handling**: Based on the selected action, the application fetches relevant content from the vector store or generates a response through the LLaMA model.
4. **Display Response**: The chat interface displays the user query and the chatbot's response, allowing for natural, back-and-forth interactions.

5.



In summary, the **Smart PDF Explorer** integrates various modules to create a seamless application for analyzing and comparing PDF content. The combination of Streamlit for the frontend, LangChain for query management, and FAISS for fast embedding retrieval enables efficient, offline operation with robust query capabilities.

5. Custom UI Styling

To create a user-friendly and visually appealing interface, we applied custom CSS styling in **Smart PDF Explorer**. These design elements enhance readability and add a conversational feel to the app, helping users to intuitively navigate and interact with the chatbot.

Here's an overview of the key UI styling elements:

5.1 Background and Font

- **Background Gradient**: The app's background has a smooth gradient, transitioning from a light gray-blue (#f0f4f8) to a slightly darker shade (#d9e2ec). This subtle gradient creates a modern, soft look that's easy on the eyes.
- **Font**: The chosen font, **Arial**, ensures readability with a clean, professional look. It pairs well with the simple interface elements.

5.2 Chat Bubbles

The chat interface uses distinct styles for user queries and bot responses, adding a conversational touch:

- **User Bubble**:
    - **Color**: A vibrant blue (#0078D4) background with white text.
    - **Border**: Solid blue border for contrast.
    - **Rounded Corners**: Border radius adjusted for a chat bubble effect (20px 20px 0px 20px), with a slightly pointed corner to distinguish it as a user message.
    - **Layout**: Left-aligned with a small icon, giving a personal touch to each user message.
- **Bot Bubble**:

    - **Color**: Light gray (#f1f1f1) with black text, which contrasts with the blue user bubbles.
    - **Border**: Soft yellow border (#ffc107) for subtle emphasis.
    - **Rounded Corners**: Opposite bubble effect (20px 20px 20px 0px) to visually separate bot messages from user messages.

5.3 Avatar and Icons

- **User Icon**: An avatar image (from icons8) is displayed next to each user message, providing a clear visual indication of who initiated the message.

5.4 Input Fields

- **Text Input and Text Area Styling**: For a polished look, input fields (used for entering queries) are rounded with a border radius of 15px and padded for a comfortable typing experience. The font size is set to 16px, making text easy to read.

5.5 Footer

- **Footer Text**: A small footer includes a custom message ("Made with ❤️ by Rohith") in a soft gray color, providing a personal touch without overwhelming the main content.

By integrating these styles, **Smart PDF Explorer** achieves a clean, professional look, making document analysis more engaging and approachable for users. This custom styling reflects a balance of functionality and aesthetics, helping users to interact with the app comfortably.

6. Sample Queries and Outputs

This section demonstrates sample queries and their respective outputs from each of the three main functionalities: **Chat With PDFs**, **Compare PDFs**, and **Ask Anything**. These examples highlight how **Smart PDF Explorer** can respond with contextually relevant and insightful answers based on user input.

6.1 Chat With PDFs

In the **Chat With PDFs** section, users can ask questions specific to the content of the uploaded PDFs. Here are some sample queries and outputs:

- **Query**: "What are the main risk factors mentioned in the Tesla report?"
    - **Output**: "The Tesla report identifies risk factors such as supply chain challenges, changes in government policies, and market competition. The company also highlights potential risks from rapid technological advancements and fluctuating raw material costs."
- **Query**: "Provide an overview of Alphabet Inc.'s revenue sources."
    - **Output**: "Alphabet Inc.’s revenue is primarily driven by Google Search and advertising, with additional contributions from YouTube, Google Cloud, and Other Bets, which include ventures in autonomous driving and life sciences."

6.2 Compare PDFs

In the **Compare PDFs** section, users can analyze differences between multiple documents, such as contrasting financial figures or company strategies. Here are some example queries:

- **Query**: "What are the differences in the business strategies of Tesla and Uber?"
    - **Output**: "Tesla's strategy focuses on sustainable energy solutions, with a heavy emphasis on electric vehicles, solar products, and battery technology. Uber, on the other hand, emphasizes service-based models, expanding in areas like ride-sharing, food delivery, and freight, with a goal of creating a global network for on-demand transportation."
- **Query**: "How do the revenue figures for 2020 compare between Alphabet and Tesla?"
    - **Output**: "For 2020, Alphabet reported a total revenue of $182 billion, largely driven by advertising and cloud services. Tesla reported $31.5 billion, primarily from vehicle sales and energy generation products."

6.3 Ask Anything

In the **Ask Anything** section, users can pose open-ended questions to the language model for broader insights or information not strictly related to the PDFs. Here are some sample questions:

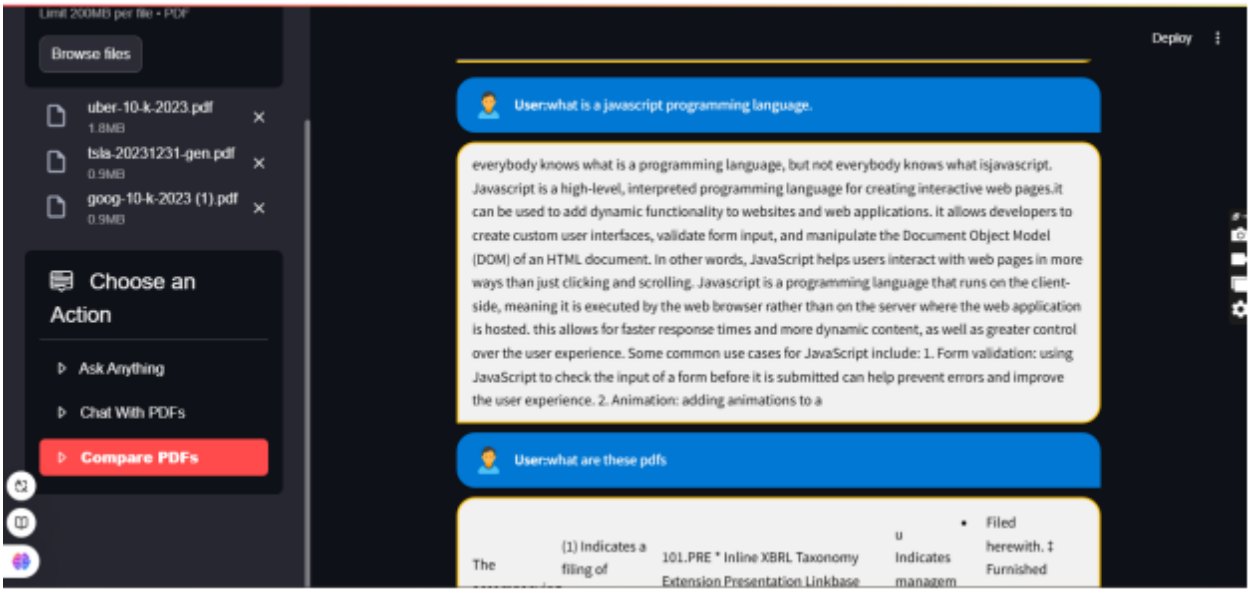
- **Query**: "Explain the concept of renewable energy in simple terms."
    - **Output**: "Renewable energy is energy generated from natural sources that can be replenished over time, like sunlight, wind, and water. Unlike fossil fuels, these sources don’t run out and have a smaller impact on the environment."
- **Query**: "What is artificial intelligence, and how is it used in business?"

- **Output**: "Artificial intelligence (AI) is a field of computer science that focuses on creating systems capable of performing tasks that typically require human intelligence, such as understanding language, recognizing images, and making decisions. In business, AI is used for automating processes, analyzing large sets of data, improving customer service through chatbots, and making data-driven predictions."

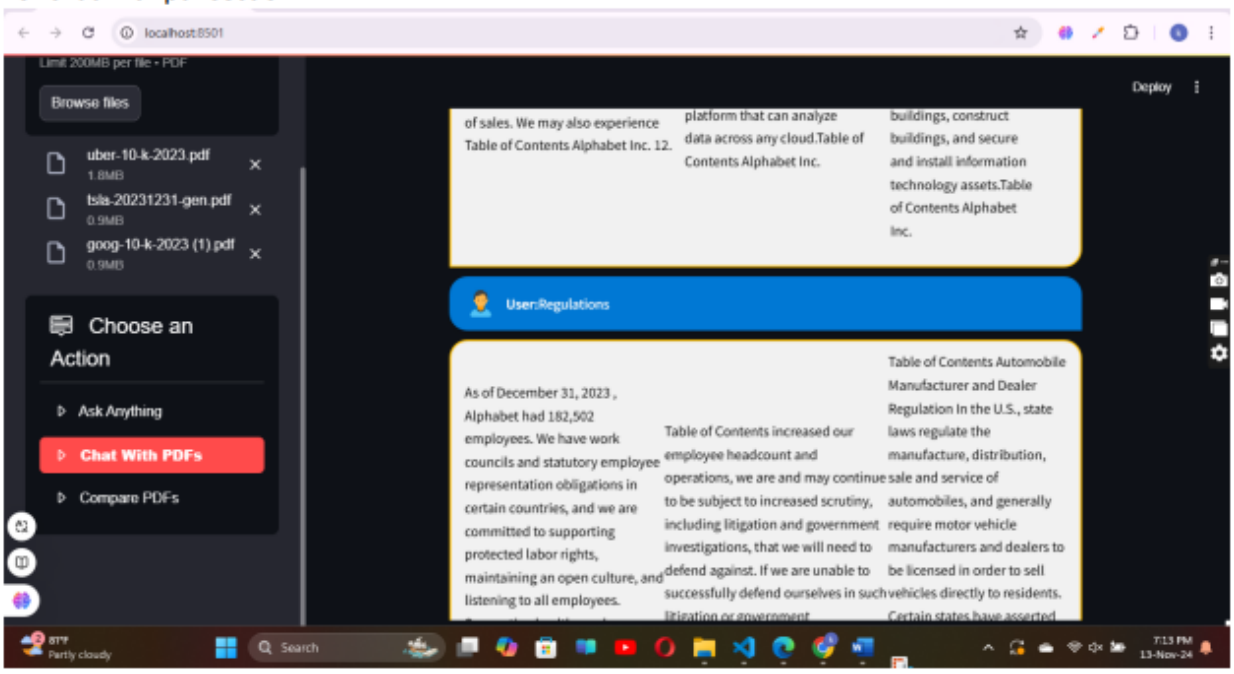
6.4 Output Display Example

For each section, responses are displayed in the chat-like interface in Streamlit. Users' queries are presented in blue bubbles (user side), while the bot's responses are shown in light gray bubbles, creating a conversational and intuitive experience.

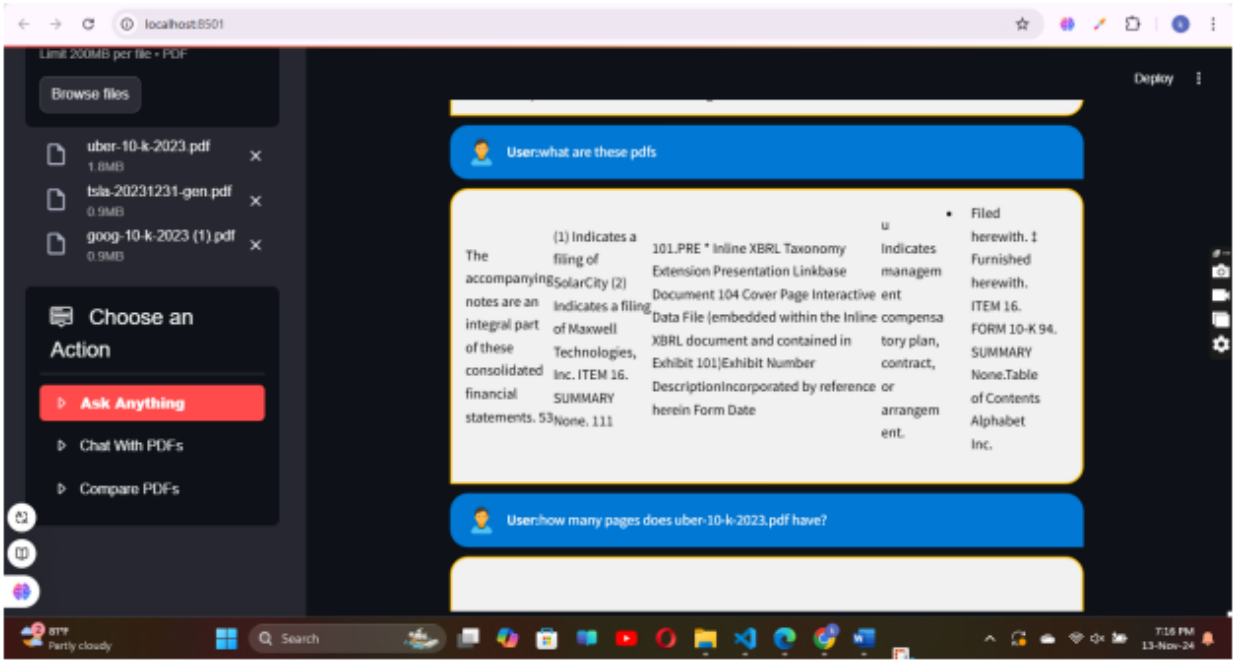
In compare pdf section for query : what are these pdfs got the following output



For chat with pdf section



For ask any thng section



These examples demonstrate the versatility of **Smart PDF Explorer** in handling detailed document-specific queries, comparative analysis, and general knowledge inquiries.

7. Footer and Credits

The footer in **Smart PDF Explorer** adds a friendly, professional touch to the application by including a message of acknowledgment and credit to the developer. This footer serves as a signature, reinforcing the personalized and crafted feel of the app.

7.1 Footer Design

- **Message**: The footer displays a simple message: "Made with ❤️ by Rohith." This note not only acknowledges the developer's work but also adds a warm, approachable tone to the application.
- **Styling**: The footer text is set in a soft gray color with a small font size to avoid distracting from the main content while still being visible. Padding around the footer area provides spacing to separate it from the main interface.

7.2 Credits and Acknowledgments

- **Developer**: Rohith, the creator of **Smart PDF Explorer**, implemented

8. Appendix

The appendix includes additional details such as code listings, resources, and further reading.

8.1 Code Listing

This section contains key pieces of code used throughout the project.

```
import streamlit as st
from streamlit_option_menu import option_menu
from PyPDF2 import PdfReader
from langchain.embeddings import SentenceTransformerEmbeddings
from langchain.vectorstores import FAISS
from langchain.schema import Document
from ctransformers import AutoModelForCausalLM
from io import BytesIO

# Custom CSS for advanced UI styling
st.markdown("""
   <style>
      body {
         background: linear-gradient(to right, #f0f4f8, #d9e2ec);
         font-family: Arial, sans-serif;
      }
      .chat-container {
```

```
    padding: 15px;
    background-color: #f7fafc;
    border-radius: 20px;
    border: 1px solid #e0e0e0;
    margin-bottom: 15px;
    box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.05);
    font-size: 16px;
}
.user-bubble, .bot-bubble {
    display: flex;
    align-items: center;
    padding: 10px 15px;
    margin-bottom: 10px;
    font-size: 16px;
    border-radius: 20px;
}
.user-bubble {
    background-color: #0078D4;
    color: white;
    border: 2px solid #0078D4;
    border-radius: 20px 20px 0px 20px;
    justify-content: flex-start;
```

```
}
.bot-bubble {
    background-color: #f1f1f1;
    color: black;
    border: 2px solid #ffc107;
    border-radius: 20px 20px 20px 0px;
    justify-content: flex-start;
}
.avatar {
    width: 36px;
    height: 36px;
    border-radius: 50%;
    margin-right: 10px;
}
.footer {
    text-align: center;
    font-size: 14px;
    color: gray;
    padding-top: 10px;
    padding-bottom: 10px;
}
.stTextInput > div > div > input, .stTextArea > div > textarea {
```

```
        border-radius: 15px;
        padding: 10px;
        font-size: 16px;
    }
  </style>
""", unsafe_allow_html=True)

# Initialize session state for chat interaction and vector database
if "chat_history" not in st.session_state:
  st.session_state["chat_history"] = []
if "vector_database" not in st.session_state:
  st.session_state["vector_database"] = None

# Path to embeddings and Llama model files
model_path = "D:/"
llama_model_path = model_path + "llama-2-7b-chat.Q2_K.gguf"
embeddings_model_path = model_path + "all-MiniLM-L6-v2"

# Load LLaMA 2 model locally for LLM insights with quality-enhancing parameters
llama_model = AutoModelForCausalLM.from_pretrained(
  llama_model_path, model_type="llama", temperature=0.7, top_k=50
```

```
)
# Load SentenceTransformer locally for PDF vectorization
embeddings_model = SentenceTransformerEmbeddings(model_name=embeddings_model_path)


# Function to parse PDF files and convert them into document objects
def parse_pdf(file):
    try:
        reader = PdfReader(BytesIO(file.read()))
        documents = [
            Document(page_content=page.extract_text(), metadata={"page": i + 1})
            for i, page in enumerate(reader.pages) if page.extract_text()
        ]
        return documents
    except Exception as e:
        st.error(f"Error parsing PDF: {e}")
        return []


# Retrieve insights from PDF query with the top relevant result
def get_insights_from_query(query_text, top_k=3):  # Increase top_k for better context in comparison
```

```
    query_embedding = embeddings_model.embed_query(query_text)
    results = st.session_state["vector_database"].similarity_search_by_vector(query_embedding, k=top_k)
    insights = [result.page_content for result in results] if results else ["No relevant content found."]
    return "\n\n".join(insights) if insights else "No relevant response available."

# Sidebar for uploading PDFs
pdf_files = st.sidebar.file_uploader("Upload PDFs", type=["pdf"], accept_multiple_files=True)

# Sidebar menu for options
with st.sidebar:
    option = option_menu("Choose an Action", ["Ask Anything", "Chat With PDFs", "Compare PDFs"], default_index=1)

# Process PDFs and create vector database if not initialized
if pdf_files and st.session_state["vector_database"] is None:
    with st.spinner('Processing your PDFs...'):
        documents = [doc for pdf_file in pdf_files for doc in parse_pdf(pdf_file)]
        if documents:
```

```
        st.session_state["vector_database"] =
FAISS.from_documents(documents, embeddings_model)
        st.success("PDFs are successfully processed and ready for
querying!")


# Chat-like interface display
# Chat-like interface display without bot symbol
def display_chat():
   for i, entry in enumerate(st.session_state["chat_history"]):
      user_query = entry.get("query", "")
      bot_response = entry.get("response", "No response available.")

      # User message styling
      st.markdown(
         f'<div class="user-bubble chat-container">'
         f'<img src="https://img.icons8.com/color/48/000000/user.png"
class="avatar"/>'
         f'<b>User:</b> {user_query}</div>', unsafe_allow_html=True
      )

      # Bot response without icon, in a clean, simple format
      st.markdown(
         f'<div class="bot-bubble chat-container">'
```

```
            f'{bot_response}</div>', unsafe_allow_html=True
        )


# Header for the application
st.markdown("<h2 style='text-align: center; color: #333;'> PDF Chat and Analysis Bot</h2>", unsafe_allow_html=True)


# Option: Chat With PDFs - Allows querying the content in uploaded PDFs
if option == "Chat With PDFs":
    st.title("Ask Questions About Your PDFs")
    chat_input = st.text_input("Enter your query")


    if chat_input:
        with st.spinner('Fetching insights...'):
            response = get_insights_from_query(chat_input)
            st.session_state["chat_history"].append({
                "query": chat_input,
                "response": response
            })
        display_chat()


# Option: Compare PDFs - Highlight differences and compare content
```

```
elif option == "Compare PDFs":
    st.title("Compare Your PDFs")
    compare_query = st.text_input("Enter a query to compare documents (e.g., revenue differences)")

    if compare_query:
        with st.spinner('Comparing documents...'):
            response = get_insights_from_query(compare_query)
            st.session_state["chat_history"].append({
                "query": compare_query,
                "response": response
            })
        display_chat()

# Option: Ask Anything - General LLM-based Chatbot
elif option == "Ask Anything":
    st.title("Ask Anything")
    essay_input = st.text_area("Enter a topic or question")

    if essay_input:
        with st.spinner('Generating response...'):
            response = llama_model(essay_input, max_new_tokens=200, temperature=0.7)
```

```
            st.session_state["chat_history"].append({
                "query": essay_input,
                "response": response
            })
        display_chat()


# Footer for the application
st.markdown("<div class='footer'>Made with ❤️ by Rohith</div>", unsafe_allow_html=True)
```