Ps-17 Business Contract Validation -To Classify Content within the Contract Clauses and Determine Deviations from Templates and highlight them.

Problem:

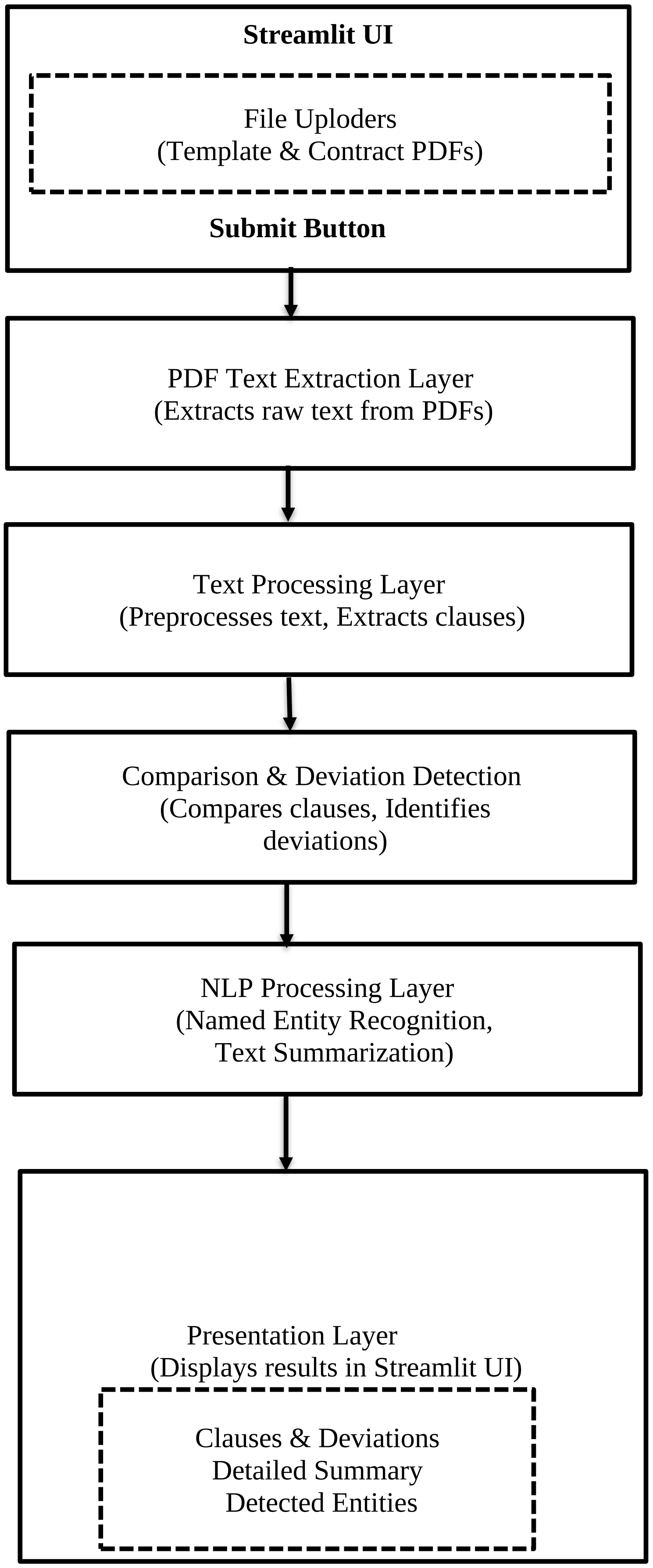
Business contracts are legal documents. The first task is to parse these documents so that have a structure to them. Determine the key details within the contract document. Every contract has clauses and sub-clauses. The next step is to classify the contents of the parsed documents to these clauses. Typically, a contract has an associated template to it, and it is important to determine the deviations from that template and highlight them.

Solution:

The project aims to develop a web application that validates business contracts by:

1. **Upload and Processing:** Allowing users to upload their contracts in PDF format, which are then converted to text using PyPDF2.
2. **Preprocessing:** Cleaning and normalizing the text using regular expressions and NLTK.
3. **Named Entity Recognition (NER):** Identifying entities in the contract text using a pre-trained NER model.
4. **Clause Identification:** Detecting clauses and subclauses in the contract text using regular expressions.
5. **Template Comparison:** Comparing the contract text with a selected template to identify deviations and calculate a similarity score.
6. **Highlighting and Summarization:** Highlighting keywords and generating a summary of the contract text.
7. **Visualization:** Displaying the original contract text, highlighted contract text, summary, entities, clauses, and subclauses in a user-friendly interface using Streamlit.

Architecture:


Streamlit UI
File Uploders
(Template & Contract PDFs)
Submit Button
PDF Text Extraction Layer
(Extracts raw text from PDFs)
Text Processing Layer
(Preprocesses text, Extracts clauses)
Comparison & Deviation Detection
(Compares clauses, Identifies deviations)
NLP Processing Layer
(Named Entity Recognition, Text Summarization)
Presentation Layer
(Displays results in Streamlit UI)
Clauses & Deviations
Detailed Summary
Detected Entities

Technologies Used:

1. Python: The primary programming language used for developing the application, providing flexibility and a rich ecosystem of libraries.

2. Streamlit: A powerful framework for building interactive web applications, enabling quick deployment and a user-friendly interface.

3. Transformers: A library from Hugging Face for implementing state-of-the-art NLP models, used for Named Entity Recognition (NER) and text summarization.

4. PyMuPDF (fitz): A library for extracting text from PDF documents, facilitating the parsing of contract files.

5. Regular Expressions (re): Used for pattern matching to extract clauses and titles from the contract text effectively.

6. pdfplumber: A utility for working with PDFs, ensuring accurate text extraction for complex document layouts.

7. HTML/CSS: Basic web technologies for customizing the appearance of the application, including styling and layout.

8. Docker: For containerizing the application, ensuring consistency across different environments and simplifying deployment.

9. Git: For version control and collaboration among team members throughout the development process.

Source Code :

```python
import re
import fitz  # PyMuPDF
import pdfplumber
from io import BytesIO
from transformers import pipeline
import streamlit as st
from threading import Thread

st.set_page_config(
    page_title="Business Contract Validation",
    page_icon="📃",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Define custom CSS for highlighting and centering the button
highlight_css = """
    <style>
        .highlight {
            background-color: #D3D3D3; /* Light Grey background for highlighting
*/
            font-weight: bold;
        }
        .center-button {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100px;
        }
    </style>
"""
# Add the CSS to the Streamlit app
st.markdown(highlight_css, unsafe_allow_html=True)

# Load pre-trained models
@st.cache_resource
def load_ner_pipeline():
    return pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-
english", aggregation_strategy="simple")

@st.cache_resource
def load_summarizer():
    return pipeline("summarization", model="facebook/bart-large-cnn")

ner_pipeline = load_ner_pipeline()
summarizer = load_summarizer()

# Function to preprocess text
def preprocess_text(text):
    text = re.sub(r'\s+', ' ', text)  # Normalize whitespace
    return text

# Function to extract text from a PDF file
def extract_text_from_pdf(pdf_file):
    document = fitz.open(stream=pdf_file.read(), filetype="pdf")
    text = ""
    for page_num in range(len(document)):
        page = document.load_page(page_num)
        text += page.get_text("text")
    return text

# Function to extract clauses and titles from the text
def extract_clauses_and_titles(text):
    # Regular expression to detect clauses and sub-clauses with their titles
    clause_pattern = re.compile(r"(\d+(\.\d+)*)\.\s+([^\n]+)")

    matches = clause_pattern.findall(text)
    clauses_and_titles = [(match[0], match[2].strip()) for match in matches]

    return clauses_and_titles

# Function to compare clauses and determine deviations
def compare_clauses(template_clauses, contract_clauses):
    deviations = []
    template_clause_dict = {clause: title for clause, title in template_clauses}
    contract_clause_dict = {clause: title for clause, title in contract_clauses}

    for clause, title in template_clause_dict.items():
        if clause not in contract_clause_dict:
            deviations.append((clause, title, "Missing in Contract"))
        elif contract_clause_dict[clause] != title:
            deviations.append((clause, title, f"Different in Contract:
{contract_clause_dict[clause]}"))

    for clause, title in contract_clause_dict.items():
        if clause not in template_clause_dict:
            deviations.append((clause, title, "Extra in Contract"))
```

```python
    return deviations

def extract_detailed_summary(text, entities):
    text = preprocess_text(text)

    if len(text) < 50:
        st.write("Input text is too short for summarization.")
        return "Summary cannot be generated due to insufficient text length."

    try:
        summary = summarizer(text, max_length=500, min_length=150,
do_sample=False)
        text_summary = summary[0]['summary_text']
    except Exception as e:
        st.write(f"Error during summarization: {e}")
        return "Summary generation failed."

    highlighted_summary = text_summary
    for entity in entities:
        entity_text = re.escape(entity['word'])
        highlighted_summary = re.sub(rf'\b{entity_text}\b', f'<span
class="highlight">{entity["word"]}</span>', highlighted_summary)

    # Highlight dates, years, and amounts
    dates = re.findall(r'\b\d{1,2} \w+ \d{4}\b', text)
    years = re.findall(r'\b\d{4}\b', text)
    money = re.findall(r'\$\d+(?:,\d{3})*(?:\.\d{2})?', text)

    for date in dates:
        highlighted_summary = re.sub(rf'\b{re.escape(date)}\b', f'<span
class="highlight">{date}</span>', highlighted_summary)
    for year in years:
        highlighted_summary = re.sub(rf'\b{year}\b', f'<span
class="highlight">{year}</span>', highlighted_summary)
    for amount in money:
        highlighted_summary = re.sub(rf'\b{re.escape(amount)}\b', f'<span
class="highlight">{amount}</span>', highlighted_summary)

    detailed_summary = f"Text Summary:\n{highlighted_summary}\n\n"
    return detailed_summary


# Streamlit app
st.title("Business Contract Validation 📃")
st.write("Upload your business contract for validation.")

# Upload template PDF file
uploaded_template_file = st.file_uploader("Choose a Template PDF file",
type="pdf", key="template")

# Upload contract PDF file
uploaded_contract_file = st.file_uploader("Choose a Contract PDF file",
type="pdf", key="contract")

# Add a submit button
st.markdown('<div class="center-button">', unsafe_allow_html=True)
submit_button = st.button("Submit")
st.markdown('</div>', unsafe_allow_html=True)

if submit_button:
    if uploaded_template_file is not None and uploaded_contract_file is not None:
        with st.spinner('Processing...'):
            # Extract text from template PDF
            template_text = extract_text_from_pdf(uploaded_template_file)

            # Extract text from contract PDF
            contract_text = extract_text_from_pdf(uploaded_contract_file)

            # Extract clauses and titles from template
            template_clauses_and_titles =
extract_clauses_and_titles(template_text)

            # Extract clauses and titles from contract
            contract_clauses_and_titles =
extract_clauses_and_titles(contract_text)

            # Display the clauses and titles from template
            st.subheader("Extracted Clauses and Titles from Template")
            for clause, title in template_clauses_and_titles:
                st.markdown(f"**{clause}. {title}**")

            # Display the clauses and titles from contract
            st.subheader("Extracted Clauses and Titles from Contract")
            for clause, title in contract_clauses_and_titles:
                st.markdown(f"**{clause}. {title}**")

            # Compare clauses and determine deviations
            deviations = compare_clauses(template_clauses_and_titles,
contract_clauses_and_titles)
```

```
            # Display deviations
            st.subheader("Deviations")
            if deviations:
                for clause, title, deviation in deviations:
                    st.markdown(f"**{clause}. {title}** - {deviation}")
            else:
                st.write("No deviations detected.")

            # Perform Named Entity Recognition (NER) on the contract
            entities = ner_pipeline(contract_text)

            # Display the summarized contract text
            st.subheader("Detailed Contract Summary")
            contract_summary = extract_detailed_summary(contract_text, entities)
            st.markdown(contract_summary, unsafe_allow_html=True)

            # Show unique entities detected
            st.subheader("Entities Detected")
            unique_entities = {entity['word']: entity['entity_group'] for entity
in entities}
            for entity, label in unique_entities.items():
                st.write(f"Entity: {entity}, Label: {label}")
    else:
        st.write("Please upload both template and contract PDF files.")
```

Team Members and Their Contribution:

- **Project Coordination and Management**:

  Kulashekar Inkollu is responsible for overseeing the project timeline, coordinating tasks among team members, and ensuring effective communication within the team.

- **Frontend Development**:

  - **Mohammad Asiya** focused on building the Streamlit app's user interface, implementing features like file uploads, displaying results, and adding custom styles (CSS) for better user experience.

- **NLP Model Integration**:
  - **Gunupuru Ravi Kumar** tasked with selecting, fine-tuning, and integrating NLP models (e.g., for Named Entity Recognition and summarization) into the application to analyze contract documents.
- **PDF Text Extraction and Processing**:
  - **Gorle Saikiran** responsible for developing and optimizing the text extraction logic from PDF files, including parsing clauses and titles using regex or other techniques.
- **Testing and Quality Assurance**:
  - **Inamanamelluru Venkatesh** dedicated to testing the application, identifying bugs, and ensuring that the output (e.g., clause comparisons, summaries) meets the expected standards and accuracy.

Conclusion:

The Business Contract Validation project successfully implements a comprehensive solution for analyzing and validating business contracts through an interactive web application. By leveraging advanced NLP techniques and machine learning models, the application efficiently extracts key clauses, identifies deviations from standard templates, and provides detailed summaries of contract content. This tool not only enhances the accuracy of contract review processes but also streamlines workflow for legal professionals and businesses alike. With user-friendly features and real-time processing capabilities, our application stands to significantly improve contract management practices, ensuring compliance and reducing potential legal risks. Future enhancements could include further model optimization, additional contract types, and expanded functionality to cater to evolving user needs.

Deployment:

To facilitate easy deployment and execution of the Business Contract Validation application, we have provided a Docker image.
https://hub.docker.com/r/asiyamohammad/business_contract_validation

This allows users to run the application in a consistent environment without the need for manual installation of dependencies. You can pull and run the Docker container.