Generative AI and Its Applications

NAME : KARTIK S MAVINKATTA

SRN : PES2UG23CS260

SECTION : D

PROBLEM STATEMENT: Legal Clause Finder

Abstract:

This project presents a Legal Clause Finder that enables users to upload contract documents (PDF or TXT) and query them using extractive question answering. The system extracts text from uploaded files, applies a transformer-based QA model to locate relevant clauses, and returns the most likely answer with a confidence indicator. A simple GUI built with notebook widgets allows file upload and question input without coding. The solution supports quick clause discovery for contracts and demonstrates how domain-oriented NLP can improve document understanding and legal information retrieval.

DOCUMENTATION:

1. Introduction

The Legal Clause Finder is a document-understanding system that helps users quickly locate answers inside legal contracts. Users can upload a PDF or TXT file through a simple notebook GUI, ask natural-language questions, and receive an extracted answer with a confidence indicator. The project uses a transformer-based extractive question-answering model to search the

document text and return the most relevant clause, making contract review faster and more accessible for non-technical users.

2. Objective:

To build a simple legal-document QA tool that lets users upload contracts (PDF/TXT) and retrieve relevant clauses by asking natural-language questions using a transformer-based model.

3. Technology Used:

- Programming Language: Python
- Library: Hugging Face Transformers
- Model: deepset/roberta-large-squad2
- PDF Handling: PyPDF2
- Environment: Jupyter Notebook / Google Colab

4. System Working

- User uploads a PDF/TXT file through the GUI.
- The system extracts text from the document.
- A transformer-based QA model searches the text using the user's question.
- The best matching answer and confidence score are displayed.

5. Model Description

The system uses deepset/roberta-large-squad2, a RoBERTa-based transformer fine-tuned on the SQuAD2 dataset for extractive question answering. It identifies the most relevant text span in the document for a

given question and provides a confidence score, enabling accurate clause extraction from contracts.

6. Limitations

- Context window restricted to model's max token limit (~512 tokens for extractive QA).
- Extractive only—cannot generate or paraphrase answers outside the document text.
- PDF extraction quality depends on document formatting and scannability.
- Single-document search—cannot compare or aggregate clauses across multiple contracts.
- Confidence scores may be unreliable for ambiguous or legal-specific terminology.
- Requires well-formed, specific questions for accurate results.

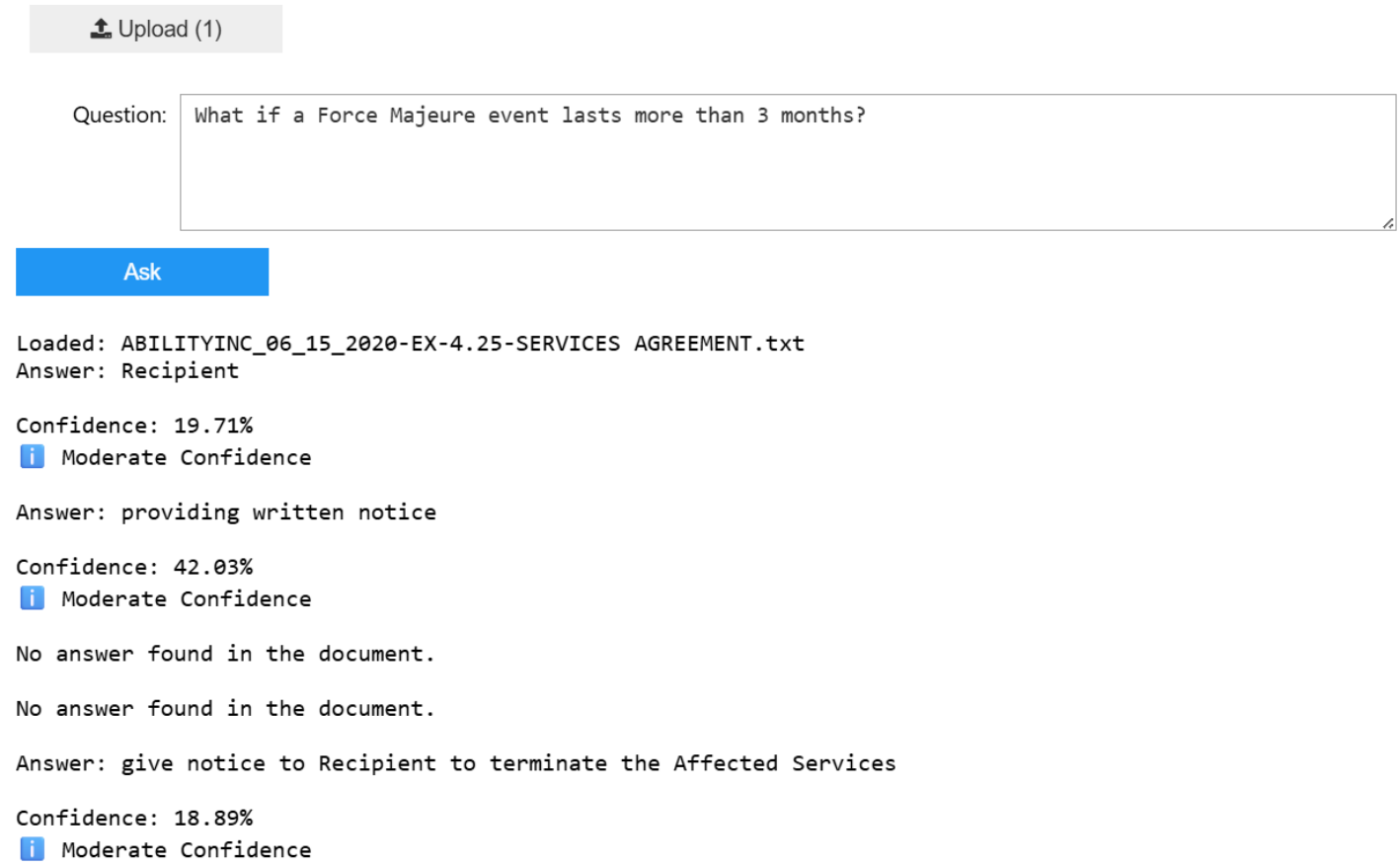
7. Applications

- Contract review and due diligence for legal teams.
- Quick clause lookup for non-lawyers (HR, procurement, business teams).
- Educational tool for law students studying contract structure.
- Compliance verification—checking for specific terms or obligations.
- Real estate and employment agreement analysis.
- Pre-signing contract summaries for individuals.

8. Conclusion

The Legal Clause Finder demonstrates how transformer based NLP can streamline contract analysis by enabling natural language queries over legal documents. The system provides an accessible GUI for uploading and querying PDF/TXT contracts, returning relevant clauses with confidence scores. While limited to extractive answers and single document searches, it serves as a practical tool for quick legal information retrieval and lays groundwork for more advanced legal AI applications.

SAMPLE OUTPUT:



CODE:

```
# ------------------------------------------------------------------
# LEGAL CLAUSE FINDER (Unit 1 Project) - PDF/TXT + GUI
# ------------------------------------------------------------------
# Prerequisites:
# You must install the required libraries before running this code.
# Run this in your terminal or Colab cell:
# !pip install transformers PyPDF2
# ------------------------------------------------------------------

from transformers import pipeline
from PyPDF2 import PdfReader
from pathlib import Path
from io import BytesIO
import ipywidgets as widgets
from IPython.display import display

# 1. Load the Pipeline (SQuAD2-style model handles unanswerable questions)
print("Loading model... please wait.")
qa_pipeline = pipeline(
    "question-answering",
    model="deepset/roberta-large-squad2",
    tokenizer="deepset/roberta-large-squad2"
)

# 2. File Reading
def read_pdf_bytes(data: bytes) -> str:
    """Extract text from all pages of a PDF byte stream."""
    reader = PdfReader(BytesIO(data))
    pages_text = []
    for page in reader.pages:
        text = page.extract_text() or ""
        pages_text.append(text)
    return "\n".join(pages_text).strip()

def read_txt_bytes(data: bytes) -> str:
    """Read text from TXT bytes."""
    return data.decode("utf-8", errors="ignore").strip()

def read_document_bytes(filename: str, data: bytes) -> str:
    """Route to PDF or TXT reader based on extension."""
    suffix = Path(filename).suffix.lower()
    if suffix == ".pdf":
        return read_pdf_bytes(data)
    if suffix == ".txt":
        return read_txt_bytes(data)
    return ""

# 3. QA Logic
```

```python
def find_legal_clause(contract_text: str, question: str) -> str:
    """
    Takes a contract snippet and a question, finds the answer,
    and filters for confidence.
    """
    if not contract_text or not question:
        return "Please provide both contract text and a question."

    result = qa_pipeline(question=question, context=contract_text)
    answer = result.get("answer", "")
    score = round(result.get("score", 0.0) * 100, 2)

    if answer.strip() == "":
        return "No answer found in the document."

    # Simple confidence message
    if score < 10:
        confidence_msg = "⚠ Low Confidence: The model is unsure. Please verify manually."
    elif score > 80:
        confidence_msg = "☑ High Confidence"
    else:
        confidence_msg = "ⓘ Moderate Confidence"

    return f"Answer: {answer}\n\nConfidence: {score}%\n{confidence_msg}"

# 4. GUI Flow
upload = widgets.FileUpload(accept=".pdf,.txt", multiple=False)
question_box = widgets.Textarea(
    placeholder="Type your question here...",
    description="Question:",
    layout=widgets.Layout(width="100%", height="80px")
)
ask_button = widgets.Button(description="Ask", button_style="primary")
output = widgets.Output()

contract_text = ""
loaded_filename = ""

def handle_upload(change):
    global contract_text, loaded_filename
    output.clear_output()
    if not upload.value:
        return
    file_info = next(iter(upload.value.values()))
    loaded_filename = file_info["metadata"]["name"]
    data = file_info["content"]
    contract_text = read_document_bytes(loaded_filename, data)
    with output:
        if contract_text:
            print(f"Loaded: {loaded_filename}")
```

```
        else:
            print("Unsupported file or no text extracted.")

def handle_ask(_):
    with output:
        if not contract_text:
            print("Please upload a PDF or TXT file first.")
            return
        question = question_box.value.strip()
        if not question:
            print("Please enter a question.")
            return
        print(find_legal_clause(contract_text, question))
        print()

upload.observe(handle_upload, names="value")
ask_button.on_click(handle_ask)

display(upload, question_box, ask_button, output)
```