TEAM AUTOLAW: Project Report

Previously, we incorporated Legal BERT into our legal document analysis pipeline to leverage its contextual understanding for domain-specific tasks. Using Legal BERT, we were able to tokenise and preprocess legal documents effectively, allowing for initial Named Entity Recognition (NER) to identify common entities like organisations, dates, and legal terms. While this approach provided a solid foundation, the accuracy of NER was limited when extracting highly specific legal entities such as clause types and nuanced legal relationships. Moreover, our previous workflow did not handle images of legal documents, leaving a gap in analysing scanned files or photographed content. This led us to explore improvements in NER performance and expand the pipeline's capabilities by adding Optical Character Recognition (OCR) and clause extraction features.

To address these limitations, we enhanced the pipeline by integrating pytesseract for OCR, enabling the extraction of text from image-based legal documents, such as scanned contracts, statutes, or judgments. Preprocessing steps, including grayscale conversion and noise reduction, were implemented to ensure clean input for OCR, thereby improving the accuracy of text recognition. This addition allowed the pipeline to handle various file formats like PNG, JPEG, and PDF, ensuring versatility in document analysis. Simultaneously, we improved the NER component by fine-tuning a dslim/bert-base-NER model on a legal-specific dataset. This fine-tuning significantly enhanced the model's ability to identify domain-specific entities, such as legal clauses (e.g., confidentiality, indemnity), parties involved, and jurisdictional references. These improvements addressed the gaps in our earlier workflow, making the entity extraction process more robust and reliable.

In addition to OCR and improved NER, we added a clause extraction module using the T5-small model for conditional generation tasks. By training the model on examples of legal clauses, such as "Extract clauses from: The agreement includes a confidentiality clause and an indemnity clause," the system can now summarize and isolate key clauses in a legal document. This capability enhances the interpretability of complex legal texts by breaking down intricate legal language into concise summaries, aiding professionals and non-experts alike. With these enhancements, the pipeline now offers a comprehensive solution for legal document analysis, supporting multiple file formats and providing high-quality entity recognition and clause extraction. Moving forward, we plan to further expand the training datasets for clause extraction, optimize OCR for low-quality scans, and deploy the pipeline as a scalable, interactive tool for legal professionals.

NCR + OCR:

```
import os
import json
from transformers import AutoTokenizer, pipeline
from pytesseract import image_to_string
from PIL import Image

# Load tokenizer and NER pipeline
tokenizer = AutoTokenizer.from_pretrained("nlpaueb/legal-bert-base-uncased")
ner_pipeline = pipeline("ner", model="dslim/bert-base-NER")

# Function to preprocess and tokenize legal text
def preprocess_text(text):
    tokens = tokenizer.tokenize(text.lower())
    return " ".join(tokens)

# Function to perform OCR on image files
def extract_text_from_image(image_path):
    try:
        image = Image.open(image_path)
        text = image_to_string(image)
        return text
    except Exception as e:
        print(f"Error extracting text from {image_path}: {e}")
        return ""

# Function to perform NER on text
def extract_entities(text):
    entities = ner_pipeline(text)
    return entities

# Process documents
data_dir = "legal_dataset/"
categories = ["contracts", "case_laws", "statutes"]
processed_data = []

for category in categories:
    category_path = os.path.join(data_dir, category)
    if not os.path.isdir(category_path):
        continue

    for file_name in os.listdir(category_path):
        file_path = os.path.join(category_path, file_name)
        if file_name.lower().endswith(('.txt', '.pdf')):  # Text or PDF files
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()
        elif file_name.lower().endswith(('.png', '.jpg', '.jpeg')):  # Image files
            content = extract_text_from_image(file_path)
        else:
            print(f"Unsupported file format: {file_name}")
            continue

        # Preprocess and tokenize text
        processed_content = preprocess_text(content)
        entities = extract_entities(content)

        processed_data.append({
```

```
            "category": category,
            "file_name": file_name,
            "content": processed_content,
            "entities": entities
        })

# Save processed data to a JSON file
output_file = "processed_legal_data.json"
with open(output_file, 'w', encoding='utf-8') as json_file:
    json.dump(processed_data, json_file, ensure_ascii=False, indent=4)

print(f"Processing complete. Data saved to {output_file}")
```

Clause Extraction Model:

```
from transformers import T5ForConditionalGeneration, T5Tokenizer

# Load pre-trained T5 model
model_name = "t5-small"
model = T5ForConditionalGeneration.from_pretrained(model_name)
tokenizer = T5Tokenizer.from_pretrained(model_name)

# Example: Prepare data for clause extraction
inputs = ["Extract clauses from: The agreement includes a confidentiality clause and an indemnity clause."]
targets = ["Confidentiality clause, Indemnity clause"]

# Tokenize input and target
input_encodings = tokenizer(inputs, truncation=True, padding=True, max_length=512, return_tensors="pt")
target_encodings = tokenizer(targets, truncation=True, padding=True, max_length=512, return_tensors="pt")

# Train model
outputs = model(**input_encodings, labels=target_encodings["input_ids"])
loss = outputs.loss
logits = outputs.logits
```