HAAG NLP Summarization Week 7

Michael Bock

October 2024

1 Slack Questions

What did you accomplish this week?

- Got PACE training working

What are you planning on working on next?

- Add parameters to the model by using the BERT weights on huggingface
- Training on UPenn data, we only just got the pdfs today.

What is blocking you from progressing?

- None

2 Abstract

Position encoding recently has shown effective in the transformer architecture. It enables valuable supervision for dependency modeling between elements at different positions of the sequence. In this paper, we first investigate various methods to integrate positional information into the learning process of transformer-based language models. Then, we propose a novel method named Rotary Position Embedding(RoPE) to effectively leverage the positional information. Specifically, the proposed RoPE encodes the absolute position with a rotation matrix and meanwhile incorporates the explicit relative position dependency in self-attention formulation. Notably, RoPE enables valuable properties, including the flexibility of sequence length, decaying inter-token dependency with increasing relative distances, and the capability of equipping the linear self-attention with relative position encoding. Finally, we evaluate the enhanced transformer with rotary position embedding, also called RoFormer, on various long text classification benchmark datasets. Our experiments show that it consistently overcomes its alternatives. Furthermore, we provide a theoretical analysis to explain some experimental results. RoFormer is already integrated into Huggingface: https://huggingface.co/docs/transformers/model_doc/roformer

Link: https://arxiv.org/abs/2104.09864v5

2.1 Brief Analysis

I like to think of RoPE embeddings kind of like a clock. Basically, the old position embedding from Attention is all you Need added the position of a token to the input. The authors of this paper point out that you can also multiply. Additionally, you can use polar coordinates and then rotate a token around a polar coordinate system according to its absolute position. The same way that 1:00 and 1:01 are close together, two tokens are close together in the embedding. Similarly, 11:00 is very different from 1:00. The main property that the authors talk about is the ability for this embedding to use absolute positions instead of relative positions and that it can take sequences of arbitrary length. But in the limitations section they say they fail to provide a faithful explanation of why this works well on long texts. Additionally, the improvements over BERT and the original transformers proposed in Attention is all you Need is small on the best tasks and the new embedding fails to improve over BERT on other tasks. I'm skeptical of the results in my short reading of this paper, but a lot of LLMs(which I eventually want to find a way to incorporate into a classifier) use this embedding. Even online blogs like this: https://github.com/adalkiran/llama-nuts-and-bolts/blob/main/docs/10-ROPE-ROTARY-POSITIONAL-EMBEDDINGS.md don't provide a satisfactory explanation of RoPE embeddings in my opinion. People say that its more "mathematically meaningful" compared to the old position embeddings. I don't see why rotation or multiplication is more meaningful than adding embeddings to an input; the same information is represented. I also don't see why we need something to be mathematically meaningful in order for it to work. Are neural networks necessarily more "mathematically meaningful" than other ML algorithms? Sure we have neurons and there's the analogy to neurons in a neural network, but really a neural network doesn't have neurons at all, instead it had matrices. You can't "disconnect" neurons in a neural network, you can only set them to zero but you can definetly lose neural connections in your brain. Ensemble methods like random forests have a claim to being more mathematical than a neural network because they are explainable and neural networks are black boxes. Does this mean that random forests are preferable to neural networks because they are more mathematically meaningful?

3 Scripts and Code Blocks

MLM Pipeline:

model.py

```
from torchtext.data.utils import get_tokenizer
from torch.utils.data import DataLoader
from torchtext.vocab import build_vocab_from_iterator
import spacy
import string
import sys
import torch
from torch import nn
from tqdm import tqdm
import time
from torch.utils.data.dataset import random_split
from torch.utils.tensorboard import SummaryWriter
from torchtext.data.functional import to_map_style_dataset
import datetime
import os
from matplotlib import pyplot as plt
```

```python
import seaborn as sns
from torchmetrics import ConfusionMatrix
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from transformers import AutoTokenizer, BertModel

sys.path.append('../')
from mistral.mistral_datasets import DocumentClassificationDataset, ISSUE_IDS

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = get_tokenizer("basic_english")

# Load SpaCy's English model
nlp = spacy.load("en_core_web_sm")


class TextClassificationModel(nn.Module):
    def __init__(self, num_class):
        super(TextClassificationModel, self).__init__()
        #self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.embedding = BertModel.from_pretrained("bert-base-uncased", torch_dtype
    = torch.float32, attn_implementation="sdpa")
        self.fc = nn.Linear(self.embedding.config.hidden_size, num_class)
        #self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, attn, token_text_id):
        embedded = self.embedding(input_ids = text, attention_mask = attn,
    token_type_ids = token_text_id)
        return self.fc(embedded.pooler_output)

def train(model, dataloader, optimizer, criterion):

    model.train()
    total_acc, total_count = 0, 0
    total_loss = 0
    log_interval = 1
    start_time = time.time()

    for idx, (label, text, attn, type_id) in enumerate(dataloader):
        optimizer.zero_grad()
        predicted_label = model(text, attn, type_id)
        loss = criterion(predicted_label, label)
        loss.backward()
        optimizer.step()
        total_acc += (predicted_label.argmax(1) == label.argmax(1)).sum().item()
        total_count += label.size(0)
        total_loss += loss.item()
        #if idx % log_interval == 0:
        #    elapsed = time.time() - start_time
        #    print(
        #        "| epoch {:3d} | {:5d}/{:5d} batches "
        #        "| accuracy {:8.3f} loss {:8.3f}".format(
```

```python
        #            epoch, idx, len(dataloader), total_acc / total_count, loss.item()
        #        )
        #    )
        #    total_acc, total_count = 0, 0
        #    start_time = time.time()

    return total_acc/total_count, total_loss/total_count

def evaluate(model, dataloader, criterion):
    model.eval()
    total_acc, total_loss, total_count = 0, 0, 0
    preds = []
    trues = []
    with torch.no_grad():
        for idx, (label, text, attn, type_id) in enumerate(dataloader):
            predicted_label = model(text, attn, type_id)
            loss = criterion(predicted_label, label)
            total_acc += (predicted_label.argmax(1) == label.argmax(1)).sum().item()
            total_count += label.size(0)
            total_loss += loss.item()

            preds.append(predicted_label.argmax(1))
            trues.append(label.argmax(1))
    return total_acc / total_count, total_loss / total_count, (torch.cat(preds), torch.cat(trues))

def normalize_text(text):
    # Process the text using SpaCy
    doc = nlp(text)

    # Define a list to hold normalized tokens
    normalized_tokens = []

    for token in doc:
        # Convert to lowercase, remove punctuation and stop words, and lemmatize the tokens
        if not token.is_punct and not token.is_stop:
            lemma = token.lemma_.lower()  # Lowercase and lemmatize
            normalized_tokens.append(lemma)

    # Join the tokens back into a normalized string
    normalized_text = ' '.join(normalized_tokens)

    return normalized_text

def yield_token(data_iter):
    for text, lbl in data_iter:
        yield tokenizer(normalize_text(text))

def pad(text_processed, text_len):
    text = text_processed[len(text_processed)//2 - text_len//2 : len(text_processed)//2 + text_len//2]
    while len(text) < text_len:
        text.append(0)
    return text

if __name__ == '__main__':
```

```
    ds = DocumentClassificationDataset(None, cases_path = './all_cases_clearinghouse.pkl', n = -1)
    print('DS made, building vocabulary')
    #vocab = build_vocab_from_iterator(yield_token(ds), specials = ["<unk>"])
    #vocab.set_default_index(vocab["<unk>"])
    text_len = 256
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

    print('Text pipeline')
    text_preprocessing_pipeline = lambda x: tokenizer(normalize_text(x), padding="max_padding", truncation=True, max_length=128)
    print(normalize_text(ds[0][0]))

    #ds.prepare_corpus(vocab, normalize_text, tokenizer, pad, text_len)
    ds.prepare_corpus_for_bert(normalize_text, tokenizer, text_len)

    def collate_fn(batch):
        text_batch = []
        attention_batch = []
        token_type_ids_batch = []
        label_batch = []
        for text, label in batch:
            text_batch.append(text['input_ids'])
            attention_batch.append(text['attention_mask'])
            token_type_ids_batch.append(text['token_type_ids'])
            label_batch.append(label)

        label_batch = torch.tensor(label_batch).double()
        text_batch = torch.tensor(text_batch).long()
        attention_batch = torch.tensor(attention_batch).double()
        token_type_ids_batch = torch.tensor(token_type_ids_batch).long()

        return label_batch.to(device), text_batch.to(device), attention_batch.to(device), token_type_ids_batch.to(device)

    print(len(ds[0]))
    train_ds, val_ds = ds.train_test_split()

    train_dataloader = DataLoader(train_ds, batch_size = 2, shuffle = True, collate_fn = collate_fn)
    val_dataloader = DataLoader(val_ds, batch_size = 2, shuffle = True, collate_fn = collate_fn)
    #dataloader = DataLoader(ds, batch_size = 8, shuffle = False, collate_fn = collate_fn)

    print('Data Loaded, total length = ', len(train_dataloader) + len(val_dataloader))
    num_class = 26#len(set([label for (label, text, offset) in dataloader]))
    #emsize = 64
    model = TextClassificationModel(num_class).to(device)

    # Hyperparameters
    EPOCHS = 100  # epoch

    #total_accu = None
    #print('Num Train: ', num_train)
    #print(train_dataloader, len(train_dataloader))
    LR = 1e-3  # learning rate
```

```python
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = LR)

    now = datetime.datetime.now()
    logdir  = now.strftime('/home/hice1/mbock9/scratch/runs/tensorboard/%Y%m%d_%H%M%S')
    savedir = now.strftime('/home/hice1/mbock9/scratch/runs/checkpoints/%Y%m%d_%H%M%S')
    writer = SummaryWriter(logdir, flush_secs = 1)
    os.makedirs(savedir)
    confmat = ConfusionMatrix(task = 'multilabel', num_labels = num_class)

    for epoch in range(1, EPOCHS + 1):
        accu_train, loss_train = train(model, train_dataloader, optimizer, criterion)
        accu_val, loss_val, (preds, trues) = evaluate(model, val_dataloader, criterion)
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss_train,
            }, os.path.join(savedir, f'{epoch}_{loss_val}.pt'))
        writer.add_scalar("Accuracy/train", accu_train, epoch)
        writer.add_scalar("Accuracy/val", accu_val, epoch)
        writer.add_scalar("Loss/train", loss_train, epoch)
        writer.add_scalar("Loss/val", loss_val, epoch)

        fig, ax1 = plt.subplots()
        cm = confusion_matrix(trues.cpu().numpy(), preds.cpu().numpy(), labels = np.arange(num_class))
        ConfusionMatrixDisplay(confusion_matrix=cm, display_labels = list(ISSUE_IDS.keys())).plot(ax = ax1)
        # Log confusion matrix to TensorBoard
        writer.add_figure("Confusion Matrix", fig, epoch)
        plt.close(fig)
```

model.py

4 Documentation

The main modification that was made here wasn't in the code, rather it was in the infrastructure I ran on. I switched from running on a local machine with a 1660 GPU to PACE, which uses a much larger H-100 gpu. The important difference here is that PACE has more GPU RAM, I only have 4 GB on my local machine which limits how large a model I can make. I basically was testing with no parameters because of my low GPU ram, but now we can start training models like BERT and Llama.

5 Scription Validation(Optional)

My script validation doubles as a results visualization. The important thing to notice is that I was able to overfit the data, indicated by the training loss reaching 0. Figure 1

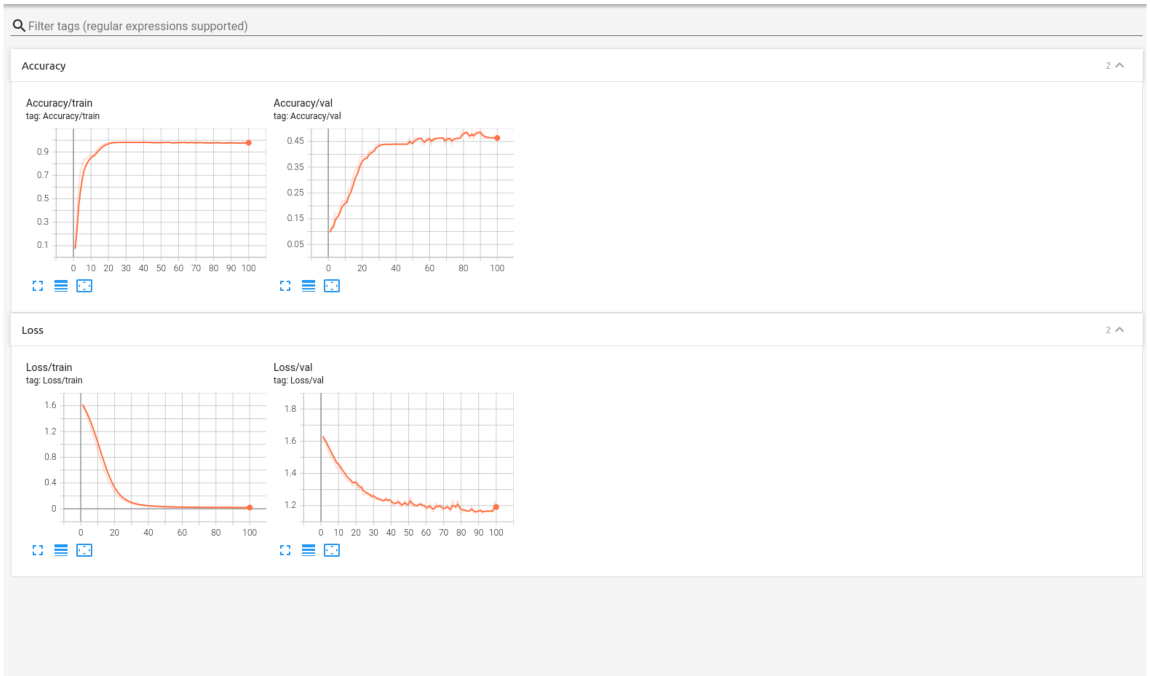


Figure 1: My training pipeline with no parameters, but this time its running on PACE

6 Results Visualization

Now, all errors have been resolved. I was able to achieve 45% accuracy, which is low. However, my model currently has no hidden layers, its only parameters are in the classification head so its basically making a prediction given no features. 45% is actually much higher than I would have predicted. Before these results, I would predict just under 4% accuracy, which is $\frac{1}{26}$%, where there are 26 classes. We can see the diagonal on the confusion matrix Figure 2 having more examples than the off-diagonal elements, indicating the model is learining something.

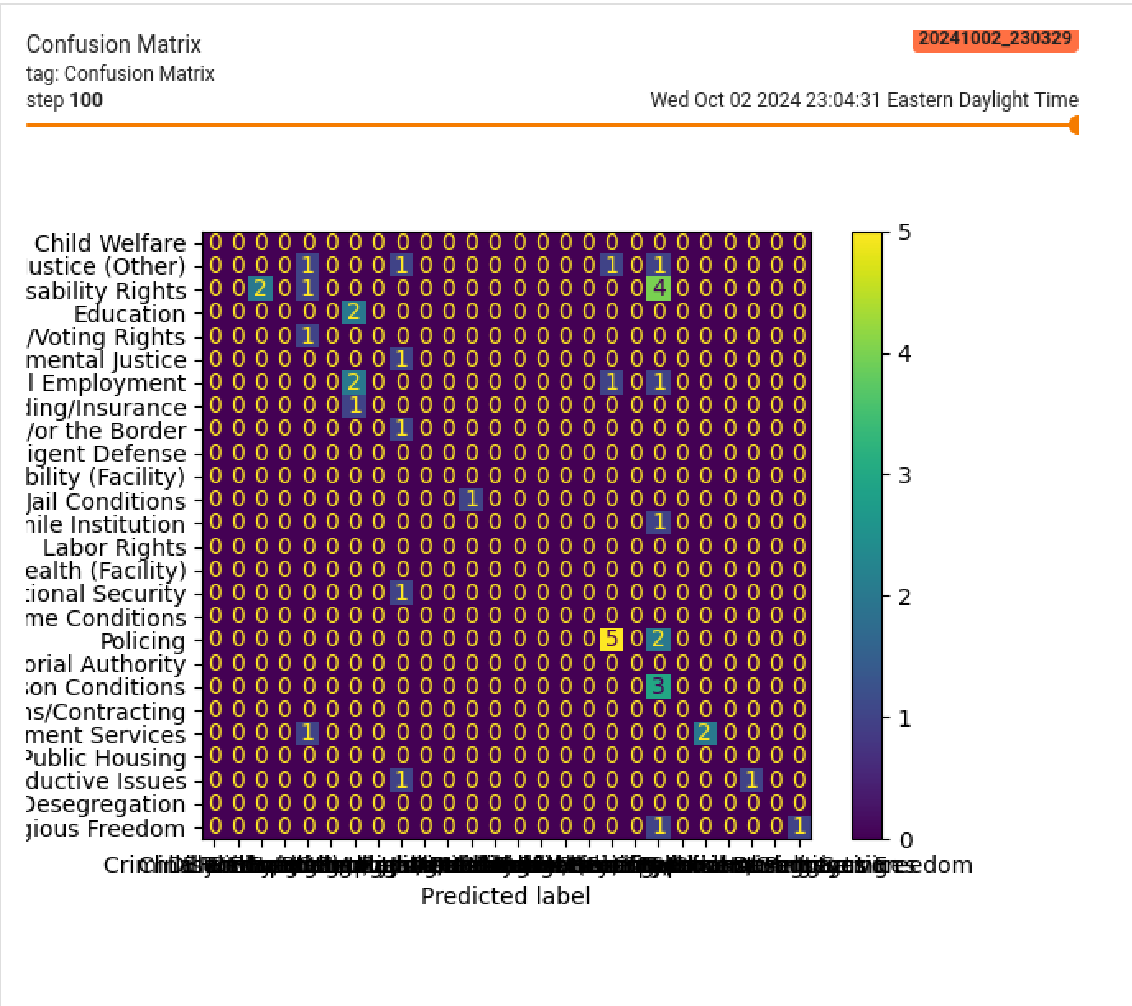


Figure 2: Enter Caption

7 Proof of work

Figure 2 and Figure 1 serve as proof of results.

8 Next Week's proposal

- Add parameters
- Use UPenn Data(we only just got the UPenn data today, October 4th 2024.

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 7

Víctor C. Fernández

October 2024

CONTENTS

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Created code for new pipeline where we're focusing on date retrieval.
- Generated new input template for date context retrieval.
- Generated new output template for date context retrieval.
- Met with the NLP-Sentencias team on Saturday 28th to align on our goals and distribute our tasks more efficiently.
- Research how to create a common email address and mailing list for HAAG.
- Meeting with the NLP team on October 4th for our weekly meeting.
- Meeting with Dr. Alexander and Nathan Dahlberg on October 4th to get further insights on NLP research.

What progress are you making next?

- Generate additional templates for input and output of date context retrieval.
- Connect code to output from prior trained model returning extracted dates.
- Generate outputs with data retrieved from prior model on multiple models to compare outputs.
- Meet with the NLP team on October 11th for our weekly meeting.
- Meet with Dr. Alexander and Nathan Dahlberg on October 11th to get further insights on NLP research.

Is there anything blocking you from making progress?

No significant blockers at this time.

2 ABSTRACTS

1. **Title:** Timeline Extraction from Decision Letters Using ChatGPT
   - **URL:** https://aclanthology.org/2024.case-1.3.pdf

   - **Abstract:** Freedom of Information Act (FOIA) legislation grants citizens the right to request information from various levels of the government, and aims to promote the transparency of governmental agencies. However, the processing of these requests is often met with delays, due to the inherent complexity of gathering the required documents. To obtain accurate estimates of the processing times of requests, and to identify bottlenecks in the process, this research proposes a pipeline to automatically extract these timelines from decision letters of Dutch FOIA requests. These decision letters are responses to requests, and contain an overview of the process, including when the request was received, and possible communication between the requester and the relevant agency. The proposed pipeline can extract dates with an accuracy of .94, extract event phrases with a mean ROUGE- L F1 score of .80 and can classify events with a macro F1 score of .79.Out of the 50 decision letters used for testing (each letter containing one timeline), the model correctly classified 10 of the timelines completely correct, with an average of 3.1 mistakes per decision letter.

   - **Summary:** This paper presents a pipeline for automatically extracting timelines from decision letters related to Dutch Freedom of Information Act (FOIA) requests. The pipeline uses SpaCy for date extraction and ChatGPT for event phrase extraction and classification. The authors created a dataset of 100 annotated Dutch decision letters and evaluated their approach on 50 test documents. The pipeline achieved high accuracy for date extraction (94%), good performance on event phrase extraction (80% ROUGE-L F1 score), and reasonable event classification (79% macro F1 score). Overall, 76% of date-event-class triples were extracted correctly, with an average of 3.1 mistakes per decision letter timeline.

   - **Relevance:** Given the new direction our project has taken to focus on extracting dates from legal documents, the pipeline approach mentioned in this paper combining SpaCy and ChatGPT could be adapted to our documents.

Additionally, it focuses on date extraction and context of the date extraction which is what we are also focusing on. It also provides evaluation metrics we could use to address our research and indications that potential research could be carried out in the same line we are already working towards.

3 SCRIPTS AND CODE BLOCKS

All scripts have been uploaded to the HAAG NLP Repo. Outputs files, processed sentencias and any other document that may contain sensitive information is located in the private NLP-Sentencias Repo.

The following code contains the logic and functions I have been working on this week.

1. New input template for querying the LLM block in charge of retrieving the context of the identified date here.

```
Analiza el siguiente texto:
{{DOCUMENT_CONTENT}}

Por favor, según la información en el texto, sustituye
↪   "TO_BE_FILLED_IN" con el contexto adecuado y devuelve solo un
↪   JSON:
{{MODEL_OUTPUT_FORMAT}}

Utilizando solo las siguientes opciones para la respuesta:
{{OPTIONS}}

Importante: Incluye solo el JSON en la respuesta.
```

Code 1—Input template for querying the model, containing place-holders to be replaced

2. New output template for the data to be retrieved by the LLM model with the context of the date here.

```
{
        "context": {
                "fecha": "{{DATE}}",
                "objeto de la fecha": "TO_BE_FILLED_IN"
        },
        "options": [
                "fecha de presentacion de demanda",
                "fecha de notificacion de demanda",
                "fecha de audiencias",
                "fecha de fallo reservado",
                "fecha de lectura de sentencia",
                "Otra: "
        ]
}
```

Code 2—Output template for the model's output, containing options for classifying the identified dates

3. Adapted code for replacing placeholders in input template and querying a model using Ollama to retrieve date context here.

```python
processor = OllamaModelProcessor("llama3.1", temperature=0.01, top_k=10, top_p=0.5, seed=42)

input_template = "./input_text_dates_v1.txt"
with open(input_template, 'r', encoding='utf-8') as f:
    input_template = f.read()
sample_file = "./test_file.txt"
with open(sample_file, 'r', encoding='utf-8') as f:
    sample_file = f.read()
# Now we inject the file into the input template
input_text = input_template.replace("{{DOCUMENT_CONTENT}}", sample_file)

output_template = "./output_json_dates_v1.json"
with open(output_template, 'r', encoding='utf-8') as f:
    output_template = json.load(f)
date = "veinticinco (25) días del mes de enero del año dos mil veintitrés (2023); 25/01/2023"
# Now we extract the date options from the output template:
dates_options = json.dumps(output_template["options"])
# We now extract the expected output converting the JSON under the context key:
expected_output = json.dumps(output_template["context"]).replace("{{DATE}}", date)
# We now replace the output placeholder in the input template with the expected output:
query_text = input_text.replace("{{MODEL_OUTPUT_FORMAT}}", expected_output)
# We now replace the options placeholder in the input template with the date options:
query_text = query_text.replace("{{OPTIONS}}", dates_options)
# We then generate 10 output files to verify output value is stable.
for i in range(10):
    processor.query_model(input_text=query_text, output_path="./output_json_dates_v1.txt", save_output=True)
```


Code 3—String comparison function

4 DOCUMENTATION

Based on the new direction we are taking, the new pipeline/flow we will be following is the one below, where we'll maintain the initial processes we already have in place to extract and clean the documents. Afterwards, a new process will take care of diving the clean documents into smaller pieces that can be then passed as input to a new layer where a Bert based model in Spanish, that has been fine tuned to better identify dates over legal documents for the Dominican Republic, is used to retrieve the dates from the corpus. Once these dates have been identified, they will be passed on to an additional model that will then retrieve the context of the date to identify what it is representing. Finally, all dates will be grouped and included in one file, representing the output of all the pieces of the original document being put together.

The following diagram represents the new intended flow:

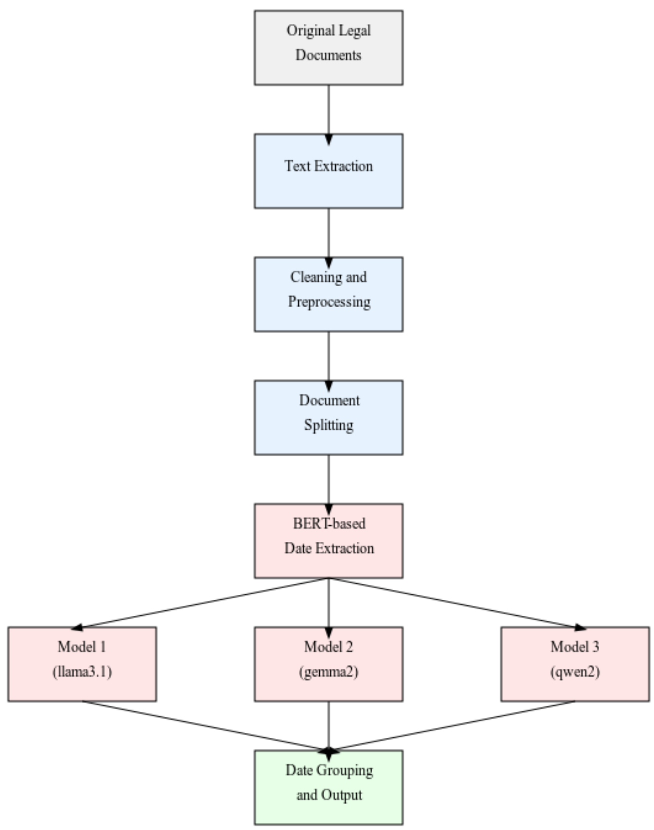


Figure 1—Full date extraction process

This week my focus has been on the second to last step, using a model to retrieve the context of the date.

Date context extraction

- Input template generated in txt format to feed the model and retrieve the date context. This template contains placeholders to fill in:
  - Content of the piece of text extracted from the original file where a date is contained.
  - Output template containing the output format expected from the model.
  - Fix set of options to classify the date, with one extra option to include cases not contained within the previous ones.
- Output template containing the expected model's output as presented in the code section.
- Model implementation to be fed either a single query or a bulk set of pieces. The code accepts the following input:
  - Output from previous model extracting the date within the text both in its original format and in DD/MM/YYYY format.
  - Input template containing query to be used for the model.
  - Output template containing the output format expected from the model.
  - Fix set of options to classify the date, with one extra option to include cases not contained within the previous ones.

  The output of the model will be a single text file containing a JSON with the input date and a field that should be updated by the model containing the identified context. Additionally, there will be a second JSON object containing configuration details for the executed model such as hyperparameters used, model's name and execution time.

5 SCRIPT VALIDATION

The model was queried over a sample file, given the full pipeline is not yet in place. For this, a sample of the original "Sentencias" documents containing a date was extracted and the date was identified manually, passing it as an input to the model.

The model was triggered 10 consecutive times, informing a seed and with the following hyperparameters:

- Temperature = 0.01,
- Top_k = 10,
- Top_p = 0.5
- Seed = 42

The seed and the low temperature should guarantee stable results over multiple executions. Unfortunately, this wasn't the actual case and although results were very similar with a Llama 3.1 model, they weren't exactly the same for the sample text used. Below are some examples retrieved within those 10 executions for the exact same input:

- fecha de notificacion de demanda
- fecha de sentencia
- Otra: Fecha de expedición de ordenanza civil
- fecha de presentacion de demanda

All generated files and content may be found here.

6 RESULTS VISUALIZATION

The following file content were generated upon the models results, retrieving the context for the date given as an input to the model.

```
{"fecha": "veinticinco (25) días del mes de enero del año dos mil
↪   veintitrés (2023); 25/01/2023", "objeto de la fecha": "fecha
↪   de notificacion de demanda"}

{
  "execution_details": {
    "model_name": "llama3.1",
    "hyperparameters": {
      "temperature": 0.01,
      "top_k": 10,
      "top_p": 0.5,
      "seed": 42
    },
    "processing_time": 1.9965579509735107,
    "timestamp": "2024-10-03 22:50:55"
  }
}
```

Code 4—Example output retrieved from the model

This output is based on the provided output template where the model is informs the field for the date context returning a response that includes both the input date and the identified context for such date.

7 PROOF OF WORK

The implemented system returns in general terms stable results, although these heavily depend on the hyperparameters used and the model itself.

In this case, a Llama 3.1 model was triggered 10 consecutive times, with the following hyperparameters:

- Temperature = 0.01,
- Top_k = 10,
- Top_p = 0.5
- Seed = 42

The seed and the low temperature should guarantee stable results over multiple executions. This wasn't the actual case and although results were very similar in content with a Llama 3.1 model, they weren't exactly the same for the sample text used. Below are some examples retrieved within those 10 executions for the exact same input:

- fecha de notificacion de demanda
- fecha de sentencia
- Otra: Fecha de expedición de ordenanza civil
- fecha de presentacion de demanda

All generated files and content may be found here. All documents were generated correctly without any issues in the output generation process. Only matter to highlight is the difference between multiple responses.

After thorough review of the text, it was identified there is a certain level of ambiguity as to what that date represents. A multiple model execution layer could be added, retrieving the most frequent output from all executions, which in this case would have been "fecha de notificacion de demanda". This would be working similarly to a quantum approach where the correct result isn't the first output from the model but the most statistically frequent one.

8 NEXT WEEK'S PROPOSAL

1. Generate additional templates for input and output of date context retrieval.
2. Connect code to output from prior trained model returning extracted dates.
3. Generate outputs with data retrieved from prior model on multiple models to compare outputs.

Week 7 | HAAG - NLP | Fall 2024

Alejandro Gomez

October 4th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- This week, I focused my efforts on finetuning. This is a topic I had previously sandboxed with spaCy, but this approach was far more refined given my improved understanding of ML and the project scope. During our team meeting last week, we had a large pivot on our project focus - originally we were focused on extracting all possible metadata, but we have honed in on solving 1 problem: extracting dates and their correspoding events with our spanish data set. To fullfill this, the team split up some of the efforts before a larger convergence next week and this meant I did not have a proper datas set to work with for training. However, I ended up hosting a deploying a Label Studio instance on a vm and manually annotated a small portion of our current data set. I then exported this to JSON and leveraged this for the training with the hugging face model "MMG/xlm-roberta-large-ner-spanish". This required some preprocessing for preparing/cleaning the data set to be able to work with the huggingface libraries e.g. Datasets. Ultimately I was able to load in the dataset and parse it, where i split it into test + training and then was able to run the code to fine tune the model. I was using PACE with an H100 Nvidia GPU to speed this workload but my final test did not produce my expected NER's. I expect that by having the intended dataset and a preprocessing script as was provided to me by our NLP advisor, I will be able to properly fine tune this model and focus on this effort since the manual annotation should be abstracted away by having a dataset prepared by the team.

1.2 What are you planning on working on next?

- I'll be meeting with the NLP DR team over the weekend to discuss a plan of action for the upcoming as we get closer to converging our efforts. At this time, my tentative plan is to hook into the dataset that one of my team members has been preparing as well as leverage a preprocessing script that was shared with our team by an advisor. These two components should assist in the finetuning efforts for the Spanish NER model.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

Freedom of Information Act (FOIA) legisla- tion grants citizens the right to request infor- mation from various levels of the government, and aims to promote the transparency of gov- ernmental agencies. However, the processing of these requests is often met with delays, due to the inherent complexity of gathering the re- quired documents. To obtain accurate estimates of the processing times of requests, and to iden- tify bottlenecks in the process, this research pro- poses a pipeline to automatically extract these timelines from decision letters of Dutch FOIA requests. These decision letters are responses to requests, and contain an overview of the pro- cess, including when the request was received, and possible communication between the re- quester and the relevant agency. The proposed pipeline can extract dates with an accuracy of .94, extract event phrases with a mean ROUGE- L F1 score of .80 and can classify events with a macro F1 score of .79. Out of the 50 decision letters used for testing (each letter containing one timeline), the model correctly classified 10 of the timelines com- pletely correct, with an average of 3.1 mistakes per decision letter. doi[BVHM24]

2.2 Summary

This paper is the most aligned with our project goals. It focuses on extracting date triplets and classifies them using chatGPT. This serves as an excellent model for us to consider for publication because it validates that our current proposed project is novel for a publication, but we have an edge that we are using Spanish data and exploring open source models with finetuning for ease of availability and cost effectiveness. Our NLP DR group will be extracting dates in Spanish format and DD/MM/YYYY format and then using generative AI to for the date content in similar fashion to thius article.

3 Scripts and Code Blocks

3.1 Code

```
json = [

    {
      "text": "Demanda principal notificada mediante acto n m. 292\/2022, de fecha veintid s (22) de noviembre del a o dos mil veintid s (2022), instrumentado por el ministerial  ngel  Dar o Castillo Mej a, de estrados de la Cuarta Sala de la C mara Penal del Juzgado de Primera Instancia del Distrito Nacional. ",
      "id": 2100,
      "label": [
        {
          "start": 67,
          "end": 127,
          "text": "veintid s (22) de noviembre del a o dos mil veintid s (2022)",
          "labels": [
            "DATE"
          ]
        }
      ],
      "annotator": 1,
      "annotation_id": 15,
      "created_at": "2024-10-04T04:34:32.249694Z",
      "updated_at": "2024-10-04T04:34:32.249722Z",
      "lead_time": 13.351
    },
    {
      "text": " ",
      "id": 2101,
      "annotator": 1,
      "annotation_id": 64,
      "created_at": "2024-10-04T04:42:12.335105Z",
      "updated_at": "2024-10-04T04:42:12.335134Z",
      "lead_time": 0.984
    },
    {
      "text": "Demandas en intervenci n voluntarias notificadas por: a) se ora Milagros Pineda mediante acto n mero 966\/2022, de fecha dos (2) de diciembre del a o dos mil veintid s (2022), instrumentado por el ministerial Alejandro Antonio Rodr guez, ordinario de la Primera Sala de la Suprema Corte de Justicia; b) se ora Clara Elena Guzm n Mart nez mediante acto n m. 1020\/2022, de fecha catorce (14) de diciembre del a o dos mil veintid s (2022), instrumentado por el ministerial Alejandro Antonio Rodr guez, ordinario de la Primera Sala de la Suprema Corte de Justicia. ",
      "id": 2102,
      "label": [
        {
          "start": 113,
          "end": 173,
```

```
            "text": " fecha dos (2) de diciembre del a o dos mil veintid s (2022)",
            "labels": [
              "DATE"
            ]
          }
        ],
        "annotator": 1,
        "annotation_id": 16,
        "created_at": "2024-10-04T04:34:46.538994Z",
        "updated_at": "2024-10-04T04:34:46.539023Z",
        "lead_time": 9.454
      },

]
```

3.2 Documentation

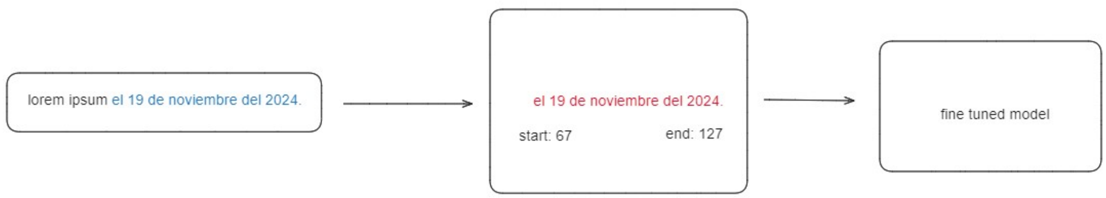


Figure 1: pipeline step visualization

3.3 Script Validation (optional)

3.4 Results Visualization

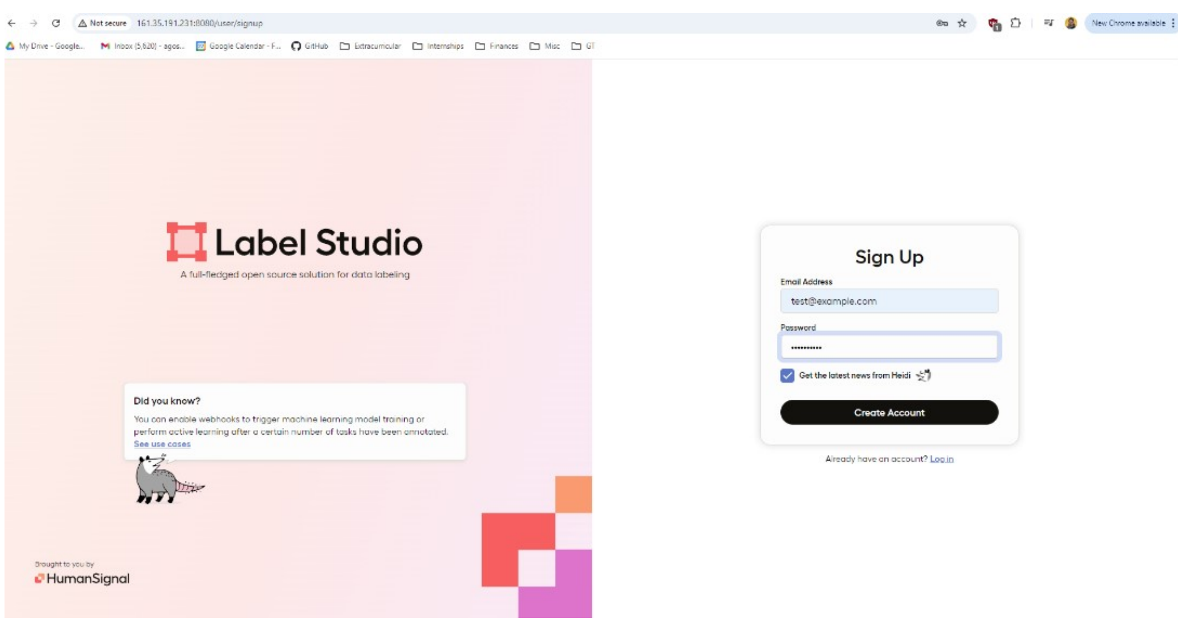


Figure 2: cloud hosted label studio

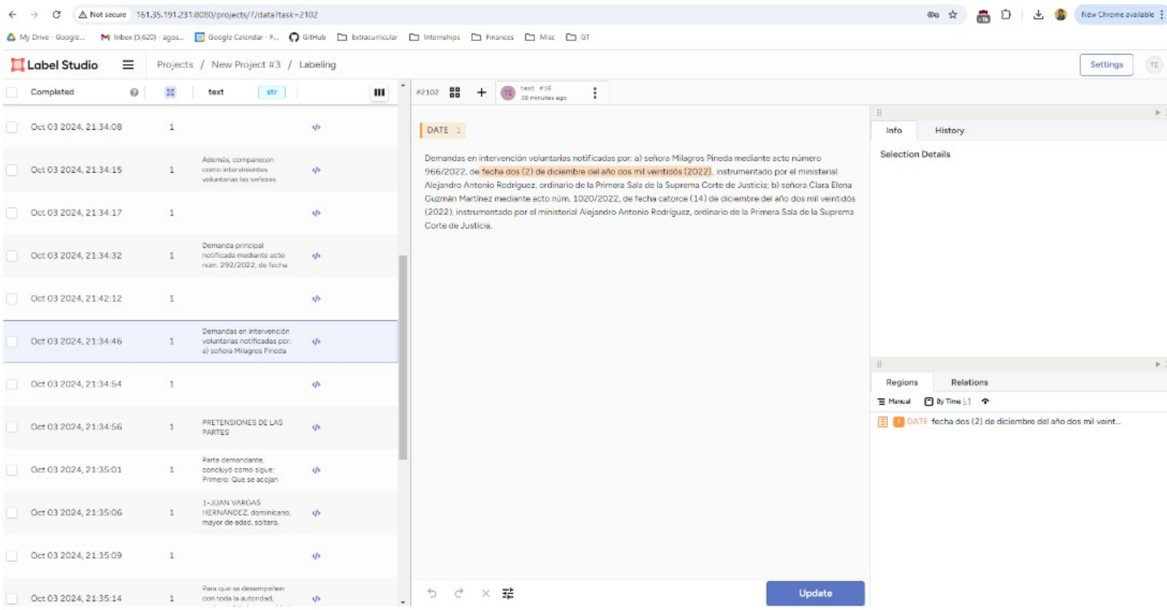


Figure 3: manual annotation of dates

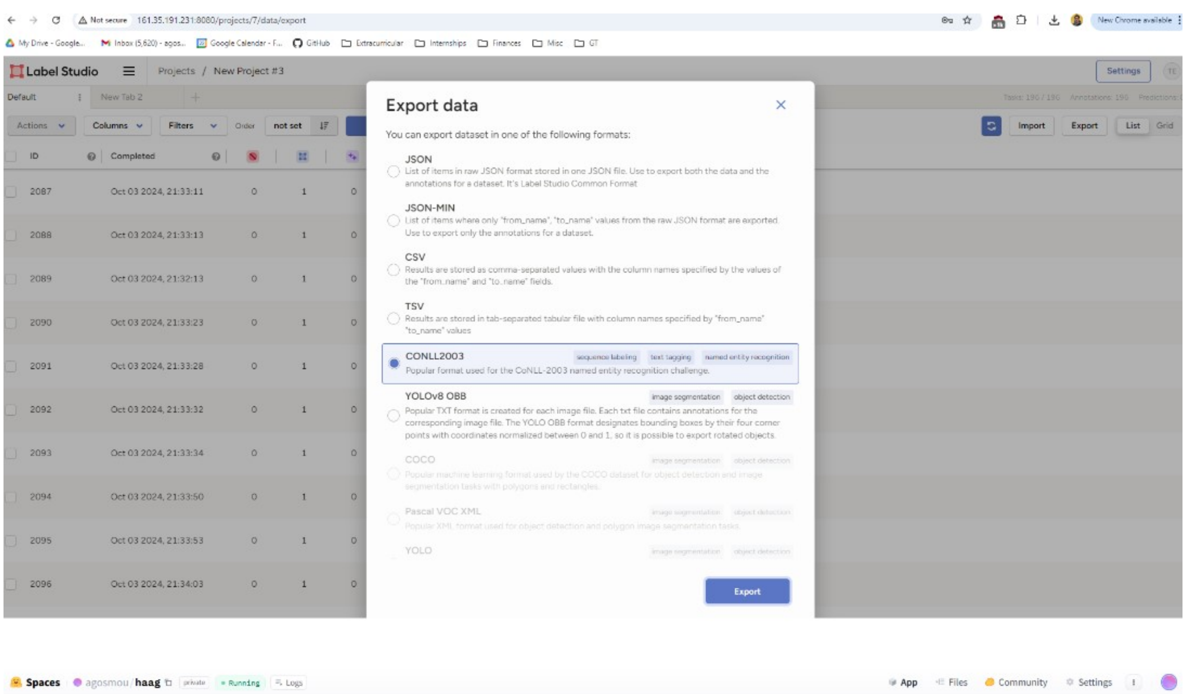


Figure 4: exporting custom dataset

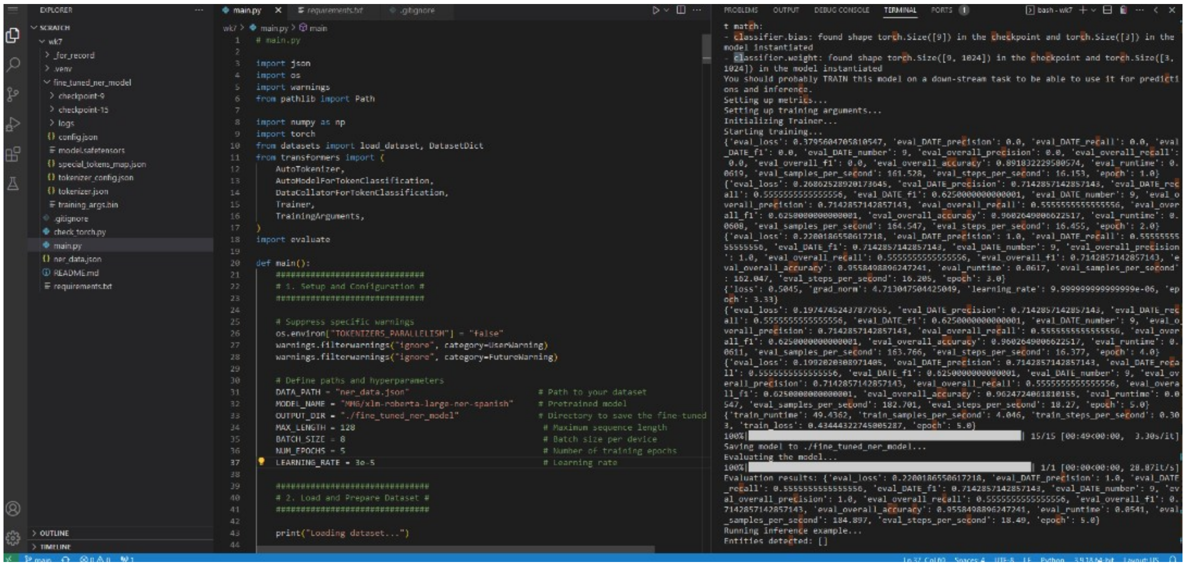


Figure 5: finetuning script on PACE

3.5 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- I'll be meeting with the NLP DR team this weekend to discuss actions steps but I have a tentative plan based on the general NLP meeting
- As mentioned in the weekly log above, the goal for this week is to use the script shared by the NLP mentor to faciliate the data prep with regard to labelling: label prepping script. Additionally I will be using a formal data set prepared by the team to finetune the model which I expect will reduce the overhead of data cleanse/prep.
- Update current documentation, e.g. NLP website

References

[BVHM24] Femke Bakker, Ruben Van Heusden, and Maarten Marx. Timeline extraction from decision letters using ChatGPT. In Ali Hürriyetoğlu, Hristo Tanev, Surendrabikram Thapa, and Gökçe Uludoğan, editors, *Proceedings of the 7th Workshop on Challenges and Applications of Automated Extraction of Socio-political Events from Text (CASE 2024)*, pages 24–31, St. Julians, Malta, March 2024. Association for Computational Linguistics.

HAAG NLP Sentencias — Week 7 Report
NLP-Gen Team

Karol Gutierrez

October 4, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Setup of Azure OpenAI environment to use ChatGPT LLMs from Python code.
- Scripts for processing of large PDF files (Dominican Republic Supreme Court sentencias) and extraction of dates and their context.
- Scripts to use ChatGPT4 to generate JSON files for individual sentencias, including dates, ranges within the document and context. This will be cleaned and used as training data to improve models.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Meetings with Dr. Alexander, Nathan and team to discuss progress on project and publication options, as well as internal meetings with team to sync on next steps.

1.2 What are you planning on working on next?

- Generate more training data using the monthly releases of Dominican Republic Supreme Court.
- Add scripts to clean generated JSON files and ensure the information is accurate.
- Use SpaCy and the generated data to train model.
- Compare results with and without training.
- Continue fulfilling my role as Meet Manager/Documentor by working on the tasks expected for my position (gather notes from meetings and prepare recordings).

1.3 Is anything blocking you from getting work done?

No.

2 Literature Review

Paper: Pre-trained Language Models for the Legal Domain: A Case Study on Indian Law [PMGG23].

2.1 Abstract

NLP in the legal domain has seen increasing success with the emergence of Transformer-based Pre-trained Language Models (PLMs) pre-trained on legal text. PLMs trained over European and US legal text are available publicly; however, legal text from other domains (countries), such as India, have a lot of distinguishing characteristics. With the rapidly increasing volume of Legal NLP applications in various countries, it has become necessary to pre-train such LMs over legal text of other countries as well. In this work, we attempt to investigate pre-training in the Indian legal domain. We re-train (continue pre-training) two popular legal PLMs, LegalBERT and CaseLawBERT, on Indian legal data, as well as train a model from scratch with a vocabulary based on Indian legal text. We apply these PLMs over three benchmark legal NLP tasks – Legal Statute Identification from facts, Semantic Segmentation of Court Judgment Documents, and Court Appeal Judgment Prediction – over both Indian and non-Indian (EU, UK) datasets. We observe that our approach not only enhances performance on the new domain (Indian texts) but also over the original domain (European and UK texts). We also conduct explainability experiments for a qualitative comparison of all these different PLMs.

2.2 Summary

The paper titled "Pre-trained Language Models for the Legal Domain: A Case Study on Indian Law" presents a case study that investigates the development and fine-tuning of language models specifically for Indian legal texts. It focuses on adapting existing models like LegalBERT and CaseLawBERT by retraining them on a large corpus of Indian legal documents. The contributions of this work are:

Key Results: The model InLegalBERT (based on LegalBERT) showed significant improvements in performance for Indian legal texts over its original version. Additionally, CustomInLawBERT demonstrated strong performance even though it was trained on fewer steps, showcasing the importance of custom vocabularies for legal-specific NLP tasks.

Explainability: The paper also explored model explainability by comparing attention scores from the fine-tuned models with expert annotations to ensure the model was making decisions based on relevant portions of legal texts.

- Pre-training with Indian Legal Texts: The study retrained two popular models—LegalBERT and CaseLawBERT—on Indian legal data and introduced a custom model, CustomInLawBERT, trained from scratch using a specialized vocabulary tailored to Indian legal documents.
- End-Task Applications: The models were evaluated on three specific tasks relevant to the legal domain:
  - Legal Statute Identification (LSI): Automatically identifying relevant legal statutes given the facts of a case.
  - Semantic Segmentation of Court Judgements: Classifying different sections in legal judgements (e.g., facts, ruling, arguments).
  - Court Judgement Prediction (CJP): Predicting the final decision of a court based on the case's facts and arguments.
- Key Results: The model InLegalBERT (based on LegalBERT) showed significant improvements in performance for Indian legal texts over its original version. Additionally, CustomInLawBERT demonstrated strong performance even though it was trained on fewer steps, showcasing the importance of custom vocabularies for legal-specific NLP tasks.
- Explainability: The paper also explored model explainability by comparing attention scores from the fine-tuned models with expert annotations to ensure the model was making decisions based on relevant portions of legal texts.

2.3 Relevance

This paper is highly relevant to our project on NLP for extracting procedural history from legal documents (sentencias). Like our work, it emphasizes the need to fine-tune models to domain-specific legal texts. The method of pre-training models such as InLegalBERT and CustomInLawBERT using specialized legal vocabularies is particularly applicable to our need for customizing models to extract key procedural information from sentencias.

3 Scripts and code blocks

The code is in the private repository repository. The progress for this week is in `./karol/week7/` .

3.1 Code developed

The following items were developed this week. The full workflow of the code is shown in Figure 1.

- I created a script to split PDF file into specific sentences, shown in Figure 2
- Cleaning of data and convert the documents into txt files, shown in Figure 3.
- Use ChatGPT 4o to send the sentencias text alongside a prompt to generate an output JSON file for each sentence, such JSON files contain an array of dates in their original format and a standardize one, as well as the context of the date, this is shown in Figure 4.

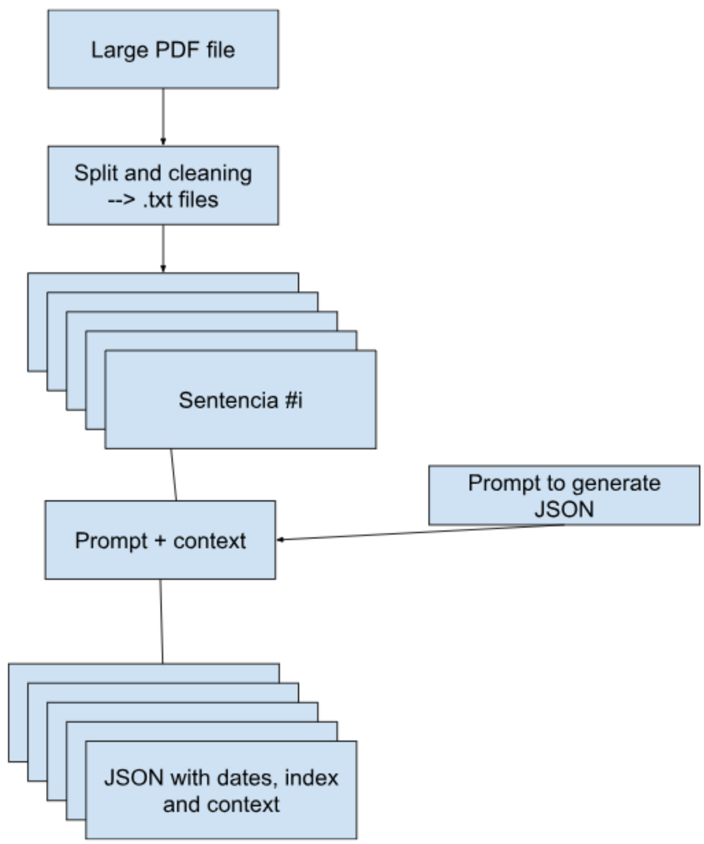


Figure 1: Code logic workflow to process file.

4 Documentation

The documentation is present in the README.md file in the repository. Refer to the repository to get the most updated instructions on how to run the code. For this week, the useful readme is in `./karol/week7/readme.md`

Different to previous weeks, to run the GPT code it is required to setup and environment in Azure Open AI and set the API KEY as an environmental variable. This code also uses the following libraries.

```
pip install python-docx
pip install PyMuPDF
pip install openai
```

```python
def split_pdf_sections(input_pdf_path, output_folder, list_page_numbers, offset=48):
    """
    Splits a large PDF into multiple sections based on provided page numbers.

    :param input_pdf_path: Path to the input PDF file.
    :param output_folder: Directory where split PDFs will be saved.
    :param list_page_numbers: List of section start pages as per the list.
    :param offset: Number to adjust list page numbers to actual PDF pages.
    """
    # Adjust list page numbers to actual PDF pages
    pdf_page_numbers = [page + offset for page in list_page_numbers]
    pdf_page_numbers = sorted(pdf_page_numbers)

    # Append None to represent the end of the document
    pdf_page_numbers.append(None)

    # Read the PDF
    reader = PdfReader(input_pdf_path)
    total_pages = len(reader.pages)

    # Create output directory if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Iterate through each section
    for i in range(len(list_page_numbers)):
        start = pdf_page_numbers[i]
        end = pdf_page_numbers[i+1] - 1 if pdf_page_numbers[i+1] else total_pages
        if end > total_pages:
            end = total_pages

        print(f"Processing Section {i+1}: Pages {start} to {end}")

        # Initialize PdfWriter
        writer = PdfWriter()

        # Add pages to the writer
        for page_num in range(start - 1, end):
            writer.add_page(reader.pages[page_num])

        # Define output PDF name
        output_pdf_name = f"section_{start}-{end}.pdf"
        output_pdf_path = os.path.join(output_folder, output_pdf_name)

        # Write the split PDF
        with open(output_pdf_path, "wb") as output_pdf_file:
            writer.write(output_pdf_file)
```

Figure 2: Code to split PDF document into sentencias.

5 Script Validation

The scripts are validated by analyzing the final JSON results. The running of the scripts is shown in Figure 5. This script add all the resulting documents into a folder, as shown in Figure 6.

6 Results Visualization

Figure 7 shows one example of an original sentencia PDF file after the splitting process. Figure 8 shows the process after cleaning the documents. Once the txt files are processed by ChatGPT 4o and the resulting response is parsed to extract the JSON component, this component is saved in a local folder to be used in a later stage as training data. An example of a final generated JSON file is shown in Figure 9 .

```python
# Main function to process multiple files and save date information as JSON
def process_all_files():
    # Input and output folder paths
    input_folder = 'cleaned'  # Folder containing the text files
    output_folder = 'dates'  # Folder to save the JSON files

    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Iterate through each file in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith('.txt'):
            file_path = os.path.join(input_folder, filename)

            print(f"Processing file: {filename}")

            # Read the content from the file
            text = read_text_file(file_path)

            # Extract date information
            extracted_data = extract_dates_from_text(text)

            # Get the extracted JSON content or raw string
            dates = extract_json_content(extracted_data)

            # Save the extracted data to a JSON file
            output_file = os.path.join(output_folder, f'{os.path.splitext(filename)[0]}.json')
            save_json(dates, output_file)

            print(f"Extracted data saved to {output_file}")

if __name__ == '__main__':
    process_all_files()
```

Figure 3: Code to generate clean txt files.

7 Proof of Work

Figure 9 shows Azure OpenAI Studio, where the deployments of the models were done. All the scripts work end to end from the starting PDF file, as shown in the Figure 9, the final results correspond to real dates. Further manual inspection and scripting can be used to ensure quality of the generated JSON files, so they can be used as training data for our models.

8 Next Week's Proposal

Refer to section 1.2 for details (avoid repetition).

References

[PMGG23] Shounak Paul, Arpan Mandal, Pawan Goyal, and Saptarshi Ghosh. Pre-trained language models for the legal domain: A case study on indian law, 2023.

```python
# Function to extract dates and related context using Azure OpenAI
def extract_dates_from_text(text):
    client = AzureOpenAI(
        api_version="2023-03-15-preview",
        azure_endpoint="https://openai-haag.openai.azure.com/"
    )

    # Prepare a prompt to ask GPT-4 to extract date information
    prompt = f"Extract all dates from the following text. For each date, return: \n\
    1. The date in standard format (YYYY/MM/DD)\n\
    2. The date in its original format from the document\n\
    3. The index of where the date starts and ends in the text\n\
    4. Only reply with a JSON with the array of dates elements, nothing else\n\
    5. The context of what happened during that date\n\n\
    Text: {text}"

    # Send the request to Azure OpenAI
    completion = client.chat.completions.create(
        model="gpt-4o",
        messages=[
            {
                "role": "user",
                "content": prompt,
            },
        ],
    )

    return completion.to_dict()  # Return as a dictionary for easy saving as JSON
```

Figure 4: Code to call ChatGPT using custom prompt.

```
(haag-nlp) C:\Users\karol\haag\sentencias\karol\week7>python extract_dates.py
Extracted data saved to dates_json_example\extracted_dates.json

(haag-nlp) C:\Users\karol\haag\sentencias\karol\week7>python extract_dates.py
Extracted data saved to dates_json_example\extracted_dates.json

(haag-nlp) C:\Users\karol\haag\sentencias\karol\week7>python extract_dates.py
Processing file: section_1008-1015_cleaned.txt
Extracted data saved to dates\section_1008-1015_cleaned.json
Processing file: section_1016-1027_cleaned.txt
Extracted data saved to dates\section_1016-1027_cleaned.json
Processing file: section_102-106_cleaned.txt
```

Figure 5: Execution of code processing sentencias texts

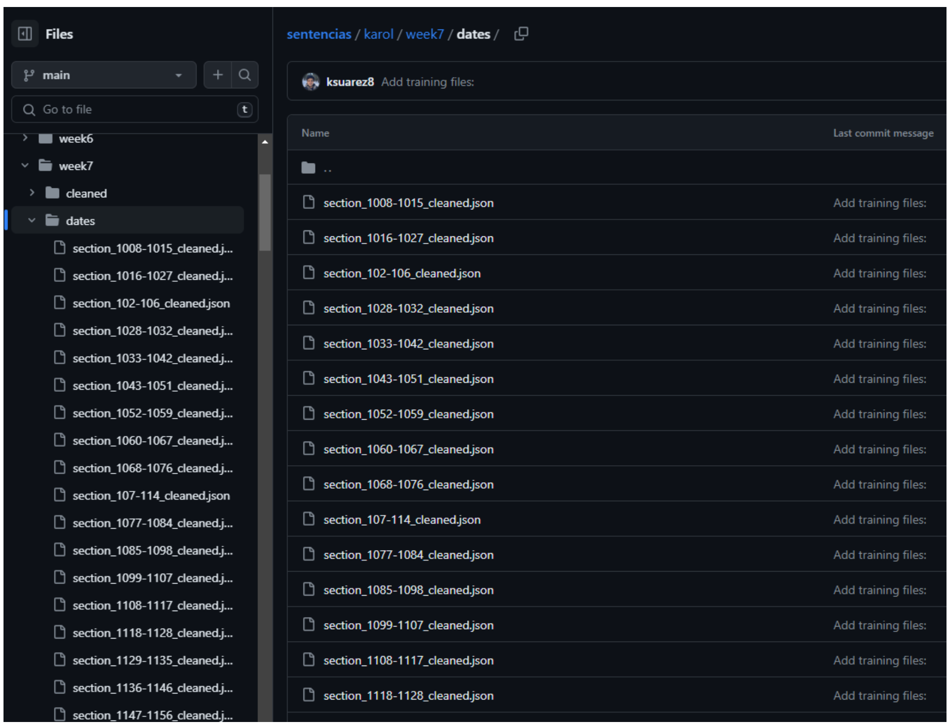
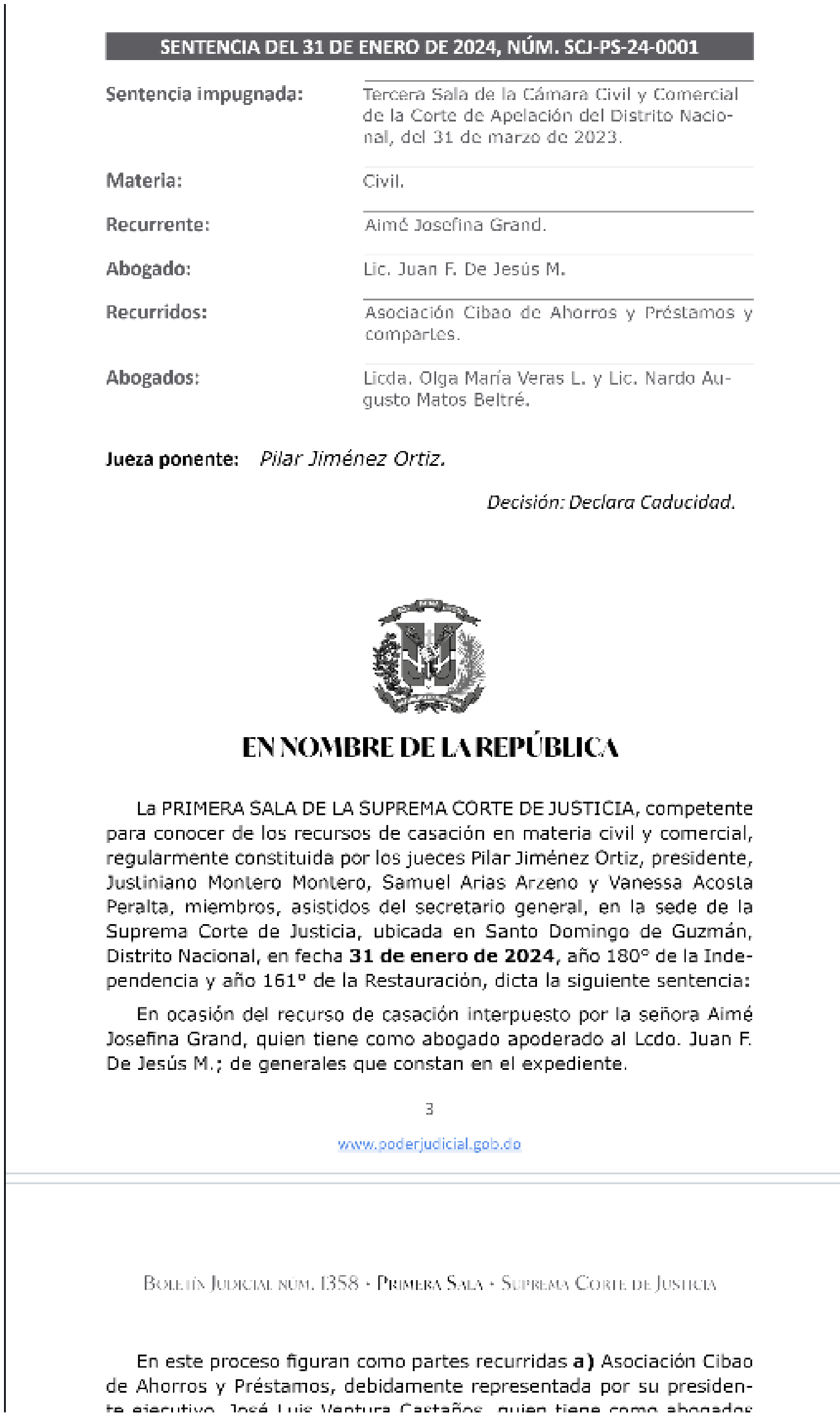


Figure 6: Resulting folder with JSON files



SENTENCIA DEL 31 DE ENERO DE 2024, NÚM. SCJ-PS-24-0001

Sentencia impugnada:	Tercera Sala de la Cámara Civil y Comercial de la Corte de Apelación del Distrito Nacional, del 31 de marzo de 2023.
Materia:	Civil.
Recurrente:	Aimé Josefina Grand.
Abogado:	Lic. Juan F. De Jesús M.
Recurridos:	Asociación Cibao de Ahorros y Préstamos y compartes.
Abogados:	Licda. Olga María Veras L. y Lic. Nardo Augusto Matos Beltré.

Jueza ponente: *Pilar Jiménez Ortiz.*

Decisión: Declara Caducidad.

EN NOMBRE DE LA REPÚBLICA

La PRIMERA SALA DE LA SUPREMA CORTE DE JUSTICIA, competente para conocer de los recursos de casación en materia civil y comercial, regularmente constituida por los jueces Pilar Jiménez Ortiz, presidente, Justiniano Montero Montero, Samuel Arias Arzeno y Vanessa Acosta Peralta, miembros, asistidos del secretario general, en la sede de la Suprema Corte de Justicia, ubicada en Santo Domingo de Guzmán, Distrito Nacional, en fecha **31 de enero de 2024**, año 180° de la Independencia y año 161° de la Restauración, dicta la siguiente sentencia:

En ocasión del recurso de casación interpuesto por la señora Aimé Josefina Grand, quien tiene como abogado apoderado al Lcdo. Juan F. De Jesús M.; de generales que constan en el expediente.

3

www.poderjudicial.gob.do

Boletín Judicial núm. 1358 • Primera Sala • Suprema Corte de Justicia

En este proceso figuran como partes recurridas **a)** Asociación Cibao de Ahorros y Préstamos, debidamente representada por su presidente ejecutivo, José Luis Ventura Castaños, quien tiene como abogados

Figure 7: Original Sentencia sample file

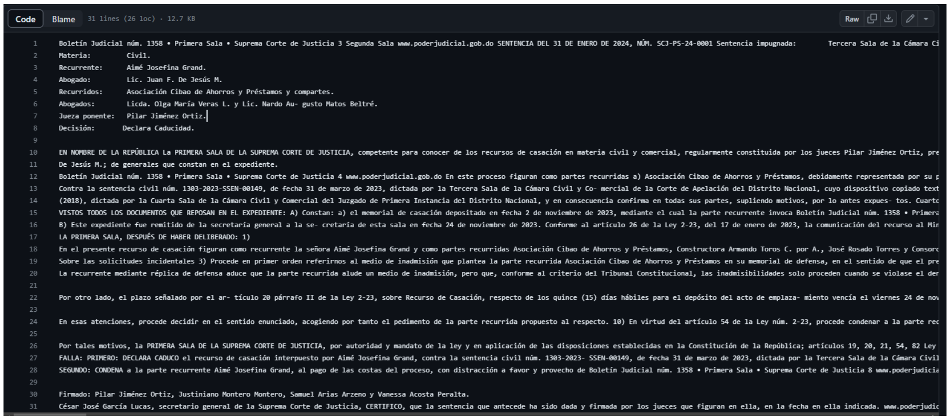


Figure 8: Processed Sentencia text file

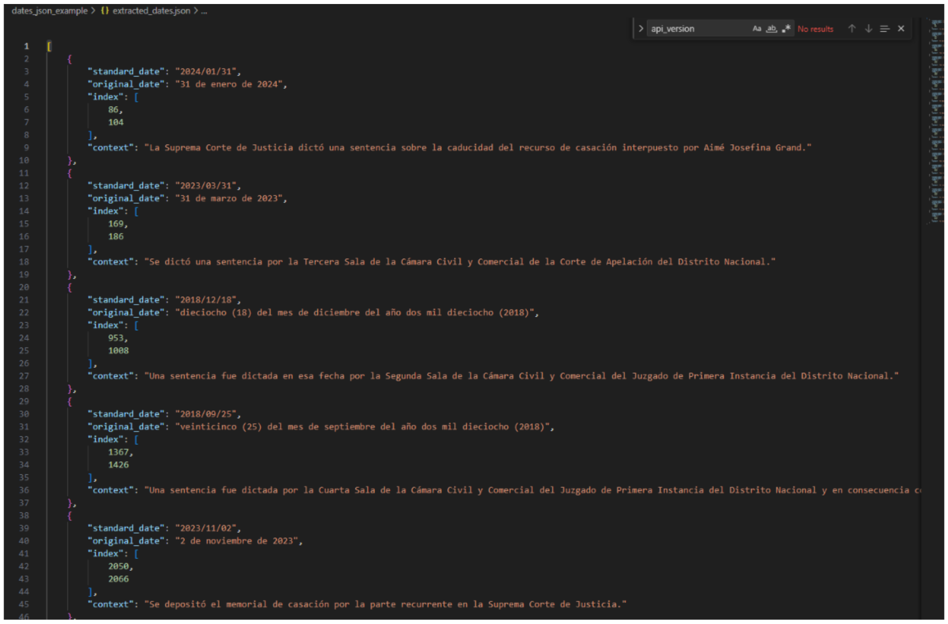


Figure 9: Final JSON files showing dates and context

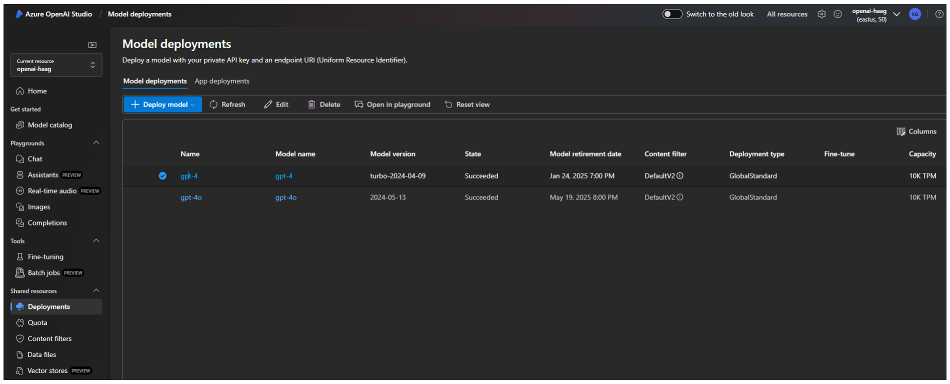


Figure 10: Azure OpenAI Studio

Week 7 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

October 2024

0.1 What did you work on this week?

1. Adjust dataset based on discussions with Dr. Alexander
2. Generated Summaries using an adjusted form of Summary Chain of Thought
3. Wrote prompt for entity extraction to attempt to follow clearinghouse guidelines concretely
4. Explored Mistral
5. Read up on AdalFlow

0.2 What are you planning on working on next?

1. Generate more summaries for validation by interview team
2. Scale Summary CoT work with chunking
3. Continue experiment with entity extraction work to create summaries

0.3 Is anything blocking you from getting work done?

1. None

1 Abstracts

- Title: Reasoning with Language Model Prompting: A Survey. Conference: ACL 2023. Link: https://aclanthology.org/2023.acl-long.294.pdf
- Abstract: Reasoning, as an essential ability for complex problem-solving, can provide back-end support for various real-world applications, such as medical diagnosis, negotiation, etc. This paper provides a comprehensive survey of cutting-edge research on reasoning with language model prompting. We introduce research works with comparisons and summaries and provide systematic resources to help beginners. We also discuss the potential reasons for emerging such reasoning abilities and highlight future research directions.

- Summary: This paper is a comprehensive review of different LLM prompting techniques, the challenge and limitations, and the need for robust evaluation.
- Relevance: There could be new techniques here that we should investigate.

2 Relevant Info

- Summary Chain of Thought (CoT) is a technique in my last report to create element driven summaries with LLMs
- Llama 3.2 is a popular LLM given its performance
- Ollama is a way to serve LLMs locally
- Langchain is a popular library for interacting with LLMs‘

3 Scripts

1. All scripts uploaded to https://github.com/Human-Augment-Analytics/NLP-Gen
2. Scripts were run with the following file for testing: `https://gatech.box.com/s/bb2ay159jlwhow6epsq0u80xn6u3u88u`
3. Thomas-Orth/summary_chain_of_thought.py
    - Brief Description: Run a modified version of Summary Chain of thought on the data
    - Status: Tested by running the pipeline to completion without issue
    - Important Code Blocks:
        (a) First block: Read in CSV file, choose document
        (b) Second block: Run through prompts
        (c) Third Block: Evaluate via manual inspection
    - Screenshot of code:

```python
import pandas as pd
from langchain_ollama import ChatOllama
from langchain_core.messages import AIMessage
from tqdm import tqdm

model = "llama3.2"
temperature = 0.4
llm = ChatOllama(model=model, temperature=temperature)
df = pd.read_csv("/Users/thomasorth/law-clearinghouse-ocr/parsed_documents.csv", sep="|").dropna()

def generate_sum_cot_prompt(document):
    prompt = f"""
    CASE: ```{document}```
    For the above court case:
    What are the important entities in this document?
    What are the important dates in this document?
    What events are happening in this document?
    What is the result of these events?
    Please answer the above questions:
    """
    return prompt
def generate_reduce_cot_prompt(document, context):
    prompt = f"""
    CASE: ```{document}```
    CONTEXT: ```{context}```
    By using the above context, summarize the case in past tense and in paragraph format into a concise by informative summary. If a date is mentioned, mention that in the sum
    If no date is found, do not mention it. Do not make note of any content you decided to exclude. Only use the information provided in the case and the content to create the
    Only provide the summary, do not include any text outside the summary:
    """
    return prompt

predicted_summaries = []
cot_info_extracted = []
pbar = tqdm(range(1))
for i in pbar:
    doc = df["Document"].iloc[0]
    pbar.set_description(f"Size of Document: {len(doc)}")
    prompt = generate_sum_cot_prompt(doc)

    ai_msg = llm.invoke(prompt)
    cot_info = ai_msg.content
    sum_prompt = generate_reduce_cot_prompt(doc, cot_info)
    ai_msg_final = llm.invoke(sum_prompt)
    predicted_summaries.append(ai_msg_final.content)
    cot_info_extracted.append(cot_info)
    print(ai_msg_final.content)
```

Figure 1: Summary CoT

4. Thomas-Orth/extract_relevant_info.py
   - Brief Description: Run entity Extraction Prompts on complaint docs
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Load CSV and choose document from dataframe
     (b) Second block: Run prompts
     (c) Third Block: Evaluate results based on manual inspection
   - Screenshot of code:

```python
import pandas as pd
from langchain_ollama import ChatOllama
from langchain_core.messages import AIMessage

df = pd.read_csv("/Users/thomasorth/law-clearinghouse-ocr/parsed_documents.csv", sep="|")
doc = df["Document"].iloc[0]

def generate_prompt(document):
    """Generates a prompt for the model to summarize a legal document with emphasis on detailed legal claims and chronological storytelling."""
    prompt = f"""
      CASE: ```{document}```
      You are a law student tasked with identifying major entities in the above complaint case. You are to find the listed entities below:
      1. The filing date. If no explicit date is given but a year is, provide the year only.
      2. Full name of the court where the case was filed e.g. “U.S. District Court for the District of New York”. This should include the state district that this course is ta
      3. The name and title of the Judge. Example: District Judge J. Paul Oetken.
      4. Type of counsel. Such as: private, legal services, state protection & advocacy system, ACLU, etc. Do not list organizations or names.
      5. Indiciate if this is a class action lawsuit or if it involves individual plantiffs. Do not name any attorneys as plaintiffs.
      6. Who are the defendents?
      7. Who are the plaintiffs? If plaintiffs are not an organization, just describe them.
      7. The plaintiffs legal claims which includes: The Statutory or constitutional basis for claim If there is a state claim, note the state.
      8. As part of the remedies for the case, was injuctive relief sought? If so, describe the injuctive relief sought in relation to any judgement.
      9. As part of the remedies for the case, was Declaratory relief sought?
      10. As part of the remedies for the case, was Attorney fees sought and how much?
      11. As part of the remedies for the case, was money damages sought? If so, what kind?
      Provide only the requested entities in this list above.
    """
    return prompt

prompt = generate_prompt(doc)

llm = ChatOllama(model="llama3.2")

ai_msg = llm.invoke(prompt)
print(ai_msg.content)
```

Figure 2: Entity extraction code

5. Flow Diagram:

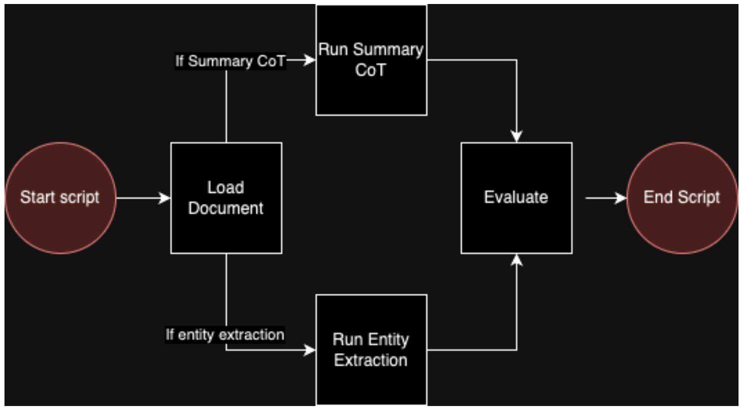


Figure 3: Flow diagram

6. Running scripts:
   (a) Download the scripts, the csv from the box link and llm.requirements.txt
   (b) Install ollama: `https://ollama.com/download`
   (c) To pull and run llama 3.2, run: ollama run llama3.2
   (d) Run: python -m pip install -r llm.requirements.txt
   (e) Run: python chosen python script

4 Documentation

1. Download CSV file, with two columns: Document and Summary
2. Update scripts to point to CSV file
3. Run scripts to output generated summaries
4. Manually evaluate summary

5 Results

5.1 Summary CoT example

Below is an example summary made by the Summary Chain of Thought technique:

A class-action lawsuit was filed on November 18, 1987, against the Defendants responsible for the conditions at the TCDC (Tennessee Correctional Complex), alleging violations of various constitutional rights. The Plaintiffs, unnamed individuals likely imprisoned at the TCDC, claim that they and their relatives and visitors are subjected to poor living conditions, inadequate food

```
1. Year: 1987
2. Court: U.S. District Court for the District of Maryland
3. Judge: Not specified
4. Counsel: Private (ACLU, ACLU National Prison Project)
5. Class Action/Legal Claims: Class action lawsuit involving individual plaintiffs
6. Defendants: The Tennessee Correctional Complex and its officials
7. Plaintiffs: Inmates at the Tennessee Correctional Complex and their relatives and visitors
8. Legal Claims:
       * Violation of First Amendment rights (free speech, association)
       * Violation of Sixth Amendment rights (right to counsel, access to courts)
       * Violation of Eighth Amendment rights (cruel and unusual punishment)
       * Violation of Fourteenth Amendment rights (due process, equal protection)
9. Injunctive Relief: Permanently enjoin Defendants from engaging in unlawful practices
10. Declaratory Relief: Issued a declaratory judgment stating Defendants' policies and actions violate Plaintiffs' constitutional rights
11. Attorney Fees: Awarded reasonable costs and attorneys' fees pursuant to 42 U.S.C. §1988
12. Money Damages: Not specified
```

Figure 4: Entity extraction results

and medical care, and denial of access to courts, violating their First, Sixth, Eighth, and Fourteenth Amendment rights. The ACLU Maryland and ACLU National Prison Project represent the Plaintiffs in this lawsuit, seeking a declaratory judgment, injunctive relief, and damages for the alleged mistreatment and poor conditions at the TCDC.

The summary contains key information and relevancy. While still missing some entities the clearing house would like such as the court that this took place in. But its a good first pass.

5.2 Entity extraction

Here is an example of entity extraction using LLMs:

The LLM is able to extract entities pretty well on complaints. Leveraging this context will be great to create better summarizations

6 Proof of Results

Llama 3.2 is one of the best performing LLMs currently so its a good OSS model choice.

Summary CoT is a paper that was published at ACL (`https://arxiv.org/abs/2305.13412`) with results showing the fact asking questions before going right to summarization helped alot.

6.1 Known Limitations

Currently, I am feeding the document directly to the LLM. For scalability, chunking of the document will be needed.