HAAG NLP Summarization Week 12

Michael Bock

November 2024

1 Slack Questions

What did you accomplish this week?

- Used Nathan's code, its good for a basic model but does not provide enough flexibility to do things like K folding easily, so I think I'll keep the parts I can and replace the rest with the code I had 2 weeks ago
- Downloaded the entire UPenn database so that I wouldn't have to return there to get more data next time.

What are you planning on working on next?

- Adding K Folding and class weighting to the training in the hopes that this will produce a model that works well given the class imbalance.

What is blocking you from progressing?

- None

2 Abstract

Foundation models, now powering most of the exciting applications in deep learning, are almost universally based on the Transformer architecture and its core attention module. Many subquadratic-time architectures such as linear attention, gated convolution and recurrent models, and structured state space models (SSMs) have been developed to address Transformers' computational inefficiency on long sequences, but they have not performed as well as attention on important modalities such as language. We identify that a key weakness of such models is their inability to perform content-based reasoning, and make several improvements. First, simply letting the SSM parameters be functions of the input addresses their weakness with discrete modalities, allowing the model to selectively propagate or forget information along the sequence length dimension depending on the current token. Second, even though this change prevents the use of efficient convolutions, we design a hardware-aware parallel algorithm in recurrent mode. We integrate these selective SSMs into a simplified end-to-end neural network architecture without attention or even MLP blocks (Mamba). Mamba enjoys fast inference (5× higher throughput than Transformers) and linear scaling in sequence length, and its performance improves on real data up to million-length sequences.

As a general sequence model backbone, Mamba achieves state-of-the-art performance across several modalities such as language, audio, and genomics. On language modeling, our Mamba-3B model outperforms Transformers of the same size and matches Transformers twice its size, both in pretraining and downstream evaluation.

2.1 Brief Analysis

Last I remember, Mamba fell out fashion at the beginning of this year (at least where I work) as people learned about new architectures like KANs. Mamba proposes switching out transformers for a structured state space model. Structured state sapce models are a model that combines CNNs and RNNs. Mamba introduces the ability to select data similar to an attention mechanism, but the Mamba attention mechanism scales linearly with input size, while attention scales quadratically, so in theory structured state space models like Mamba could be larger given the same hardware. They also improve the CNN and RNN layers to be faster by computing the model recurrently. This avoids increased IO accesses for a GPU, which are very slow.

3 Scripts and Code Blocks

hf_training.py

```
import os
import ast
import pandas as pd
from labels import DNO_ISSUES
from datasets import Dataset
from transformers import AutoTokenizer
from peft import prepare_model_for_kbit_training, AutoPeftModel
from peft import LoraConfig, get_peft_model
import datetime
import os
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification, AutoModel
from transformers import TrainingArguments, Trainer, BitsAndBytesConfig
import numpy as np
import evaluate
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold

model_name = "meta-llama/Llama-3.2-1B"
batch_size = 8
df = pd.read_csv('dno_labels.csv').dropna()
df['labels'] = [[x[issue_name] for issue_name in DNO_ISSUES] for _, x in df.iterrows
    ()]

dataset = Dataset.from_pandas(df)#.train_test_split(test_size=0.2)

tokenizer = AutoTokenizer.from_pretrained(model_name, token = os.environ['HF_TOKEN'
    ])
#tokenizer = AutoTokenizer.from_pretrained("FacebookAI/xlm-roberta-base")
#tokenizer.add_special_tokens({'pad_token': '[PAD]'})
tokenizer.pad_token = tokenizer.eos_token

```

```
max_seq_length = tokenizer.model_max_length  # or model.config.
    max_position_embeddings
print(f"Model's maximum sequence length: {max_seq_length}")

def tokenize_function(examples):
    examples['Text'] = [e.replace('\n', '') for e in examples["Text"]]
    return tokenizer(examples['Text'], padding="max_length", max_length=
    max_seq_length//10, truncation=True, return_tensors='pt')

tokenized_dataset = dataset.map(tokenize_function, batched=True)
tokenized_dataset = tokenized_dataset.remove_columns(["Text"])
tokenized_dataset = tokenized_dataset.remove_columns(["Name"])
tokenized_dataset.set_format("torch")

quantization_config = dict(load_in_4bit=True, bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype="bfloat16")

base_model = AutoModelForSequenceClassification.from_pretrained(model_name,
    num_labels=len(DNO_ISSUES), problem_type="multi_label_classification",
    quantization_config=quantization_config, device_map={"": 0})

base_model = prepare_model_for_kbit_training(base_model)

lora_config = LoraConfig(r=8, lora_alpha=32, target_modules="all-linear",
    lora_dropout=0.01, task_type="SEQ_CLS",)

model = get_peft_model(base_model, lora_config)
model.config.pad_token_id = tokenizer.pad_token_id

def numpy_sigmoid(x):
    return 1 / (1 + np.exp(-x))

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = (numpy_sigmoid(logits) >= 0.5).astype(int)
    x = precision_recall_fscore_support(preds, labels, average='macro')
    metrics = dict(precision=x[0], recall=x[1], f1=x[2])
    x = precision_recall_fscore_support(preds, labels)
    for i, label_name in enumerate(DNO_ISSUES):
        metrics[f'{label_name}__precision'] = x[0][i]
        metrics[f'{label_name}__recall'] = x[1][i]
        metrics[f'{label_name}__f1'] = x[2][i]
    return metrics

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')
skf = StratifiedKFold(n_splits=k)

#training_args = TrainingArguments(
#    output_dir=savedir,
#    num_train_epochs=1,
#    per_device_train_batch_size=batch_size,
#    per_device_eval_batch_size=batch_size,
#    gradient_accumulation_steps=1,
#    learning_rate=3e-4,
#    warmup_ratio=0.03,
#    save_steps=0.1,
```

```
#    eval_steps=0.1,
#    eval_strategy='steps',
#    save_total_limit = 2,
#    load_best_model_at_end = True,
#    report_to='tensorboard',
#    logging_dir=logdir,
#    logging_steps=2,
#    overwrite_output_dir=True,
#)

#labels = tokenized_dataset['labels']
#for fold, (train_idx, val_idx) in enumerate(skf.split(np.zeros(len(labels)), labels)):
#    print(f"\nFold {fold + 1}/{k}")
#
#    # Create train and validation datasets for this fold
#    train_dataset = dataset.select(train_idx)
#    val_dataset = dataset.select(val_idx)
#    trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, eval_dataset=val_dataset, compute_metrics=compute_metrics,)
#
#    trainer.train()
```

get_dataset.py

```
import mysql.connector
import pandas as pd
import json
import numpy as np
from tqdm import tqdm
import sys
sys.path.append('../../')
from summarizers.ocr import extract_text_from_pdf
import os
import pypdf

dno = mysql.connector.connect(
        host="127.0.0.1",
        user="report",
        password=os.environ['UPENN_PASSWORD'],
        port=3307,
        database='sla2_prod'
       )

cursor = dno.cursor()

cursor.execute("select * from taxonomy_term_data where vid=43;")
terms = cursor.fetchall()
term_dict = {}
for x in terms:
    term_dict[x[0]] = x[2]
json.dump(term_dict, open("terms.json", 'w'))

cursor.execute("select etp.entity_id event_nid, fc.field_case_target_id case_nid, (
    select group_concat(substr(fm.uri,11) order by d.delta separator '\n') from
    field_data_field_documents d, file_managed fm where d.entity_type='node' and d.
    entity_id=etp.entity_id and fm.fid=d.field_documents_fid) docs from
    field_data_field_event_type etp inner join field_data_field_case fc on fc.
    entity_type='node' and fc.entity_id=etp.entity_id;")
```

```
30
31 myresult = cursor.fetchall()
32 issues_columns = np.zeros((len([x for x in myresult if x[-1] is not None and os.path
       .exists(os.path.join('pdfs_answers', os.path.split(x[-1])[-1]))]) - 1, len(terms
       )))
33 case_names = []
34 case_text = []
35 case_num = 0
36 for i, x in enumerate(tqdm(myresult)):
37     try:
38         if x[-1] is not None and os.path.exists(os.path.join('pdfs_answers', os.path
       .split(x[-1])[-1])):
39             cursor.execute(f"select field_d_o_issues_tid from
       field_data_field_d_o_issues where entity_id={x[0]};")
40             issues_in_case = cursor.fetchall()
41             if len(issues_in_case) > 0:
42                 case_text.append(extract_text_from_pdf(os.path.join('pdfs_answers',
       os.path.split(x[-1])[-1])))
43                 print(x)
44                 print(issues_in_case)
45                 for issue in issues_in_case:
46                     issues_columns[case_num, list(term_dict.keys()).index(issue[0])]
        = int(1)
47                 case_num += 1
48                 case_names.append(x[-1])
49         else:
50             if x[-1] is not None:
51                 print(os.path.join('pdfs_answer', os.path.split(x[-1])[-1]), ' dne')
52     except:# pypdf.errors.PdfStreamError:
53         pass
54
55 issues_columns = issues_columns[:case_num, :]
56 df_dict = {}
57 for tid in term_dict.keys():
58     df_dict[term_dict[tid]] = issues_columns[:, list(term_dict.keys()).index(tid)]
59     print(len(df_dict[term_dict[tid]]))
60 df_dict['Name'] = case_names
61 df_dict['Text'] = case_text
62 print(len(case_text))
63 print(len(case_names))
64
65 pd.DataFrame.from_dict(df_dict).to_csv('dno_labels.csv', index = False)
```

4 Documentation

This does the exact same thing as the old model, except it uses huggingface now. Huggingface's documentation on K Folding takes you to a 404 page, but there are many forum posts about K folding with huggingface. But all of those either prevent me from collecting metrics during training or prevent me from getting the metrics to aggregate across multiple folds. So the best way to proceed is to get rid of the huggingface trainer and use a normal training loop. I have a training loop from 2 weeks ago that I know works, and collects a confusion matrix that I can probably adapt to work for K Folding and class weighting.

The dataset script is also the same as before, except instead of fetching only complaints I got data for every type of legal document. The labels can be different for every legal document in a

case so I have an example for every document in the UPenn database; the model will inference separately on each document as opposed to inferencing on all of them concanated.

5 Script Validation(Optional)

CSV in Figure 1.

6 Results Visualization

Figure 1: Shows my csv with the new labels in it

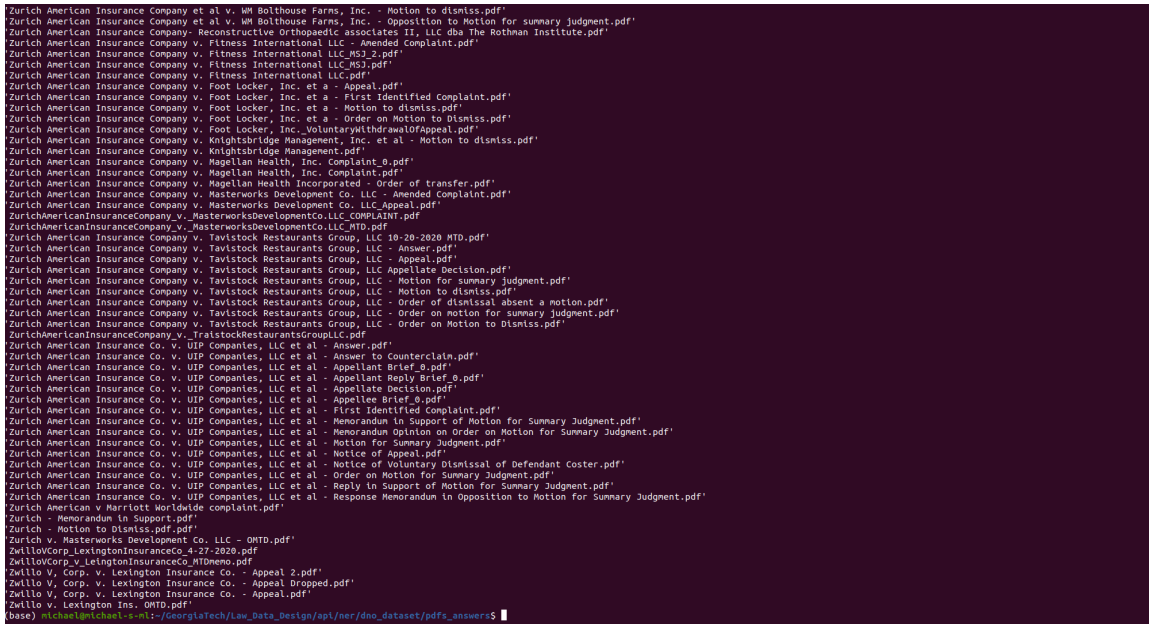


Figure 2: Downloaded PDFs

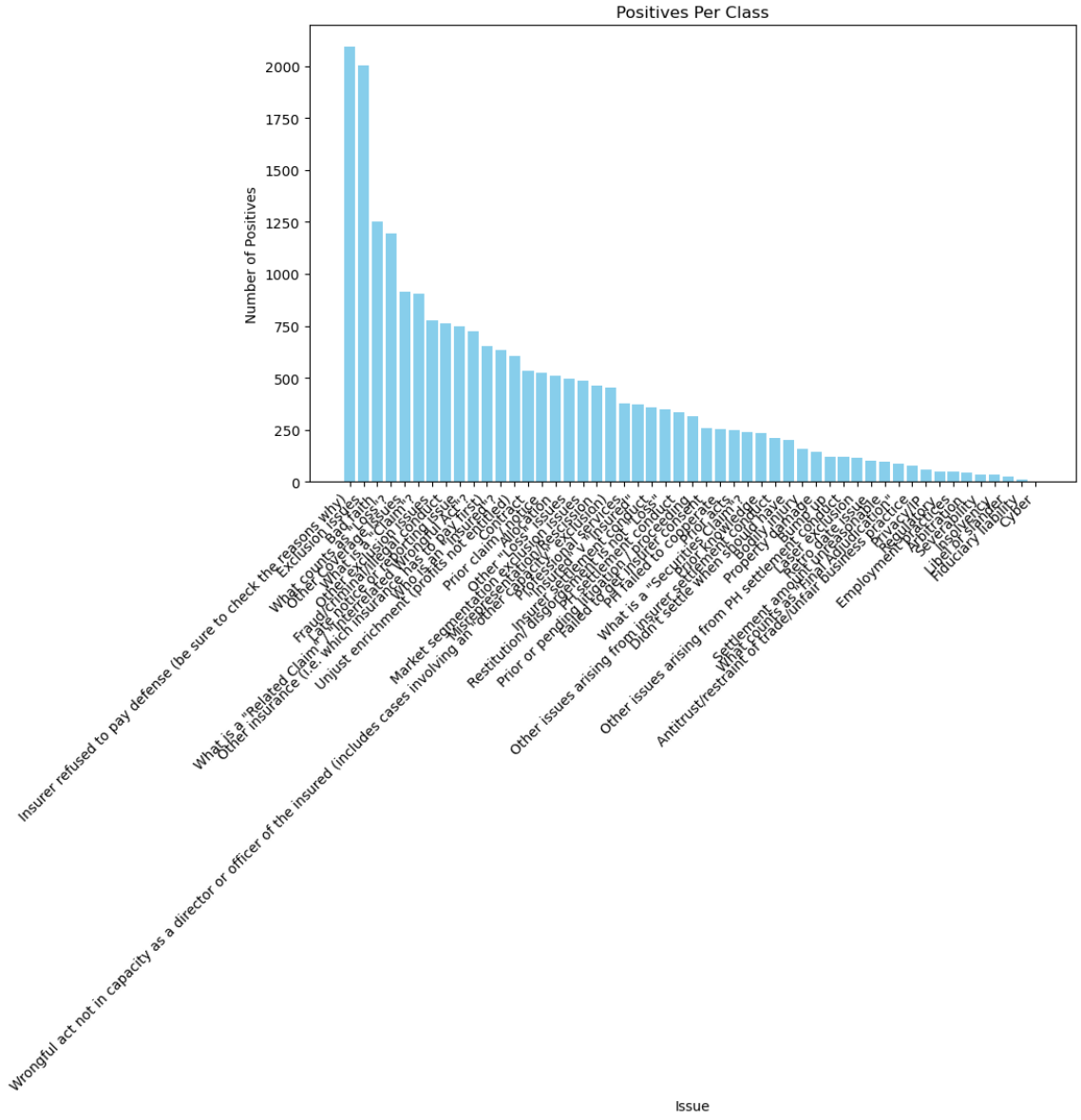


Figure 3: New Class Counts

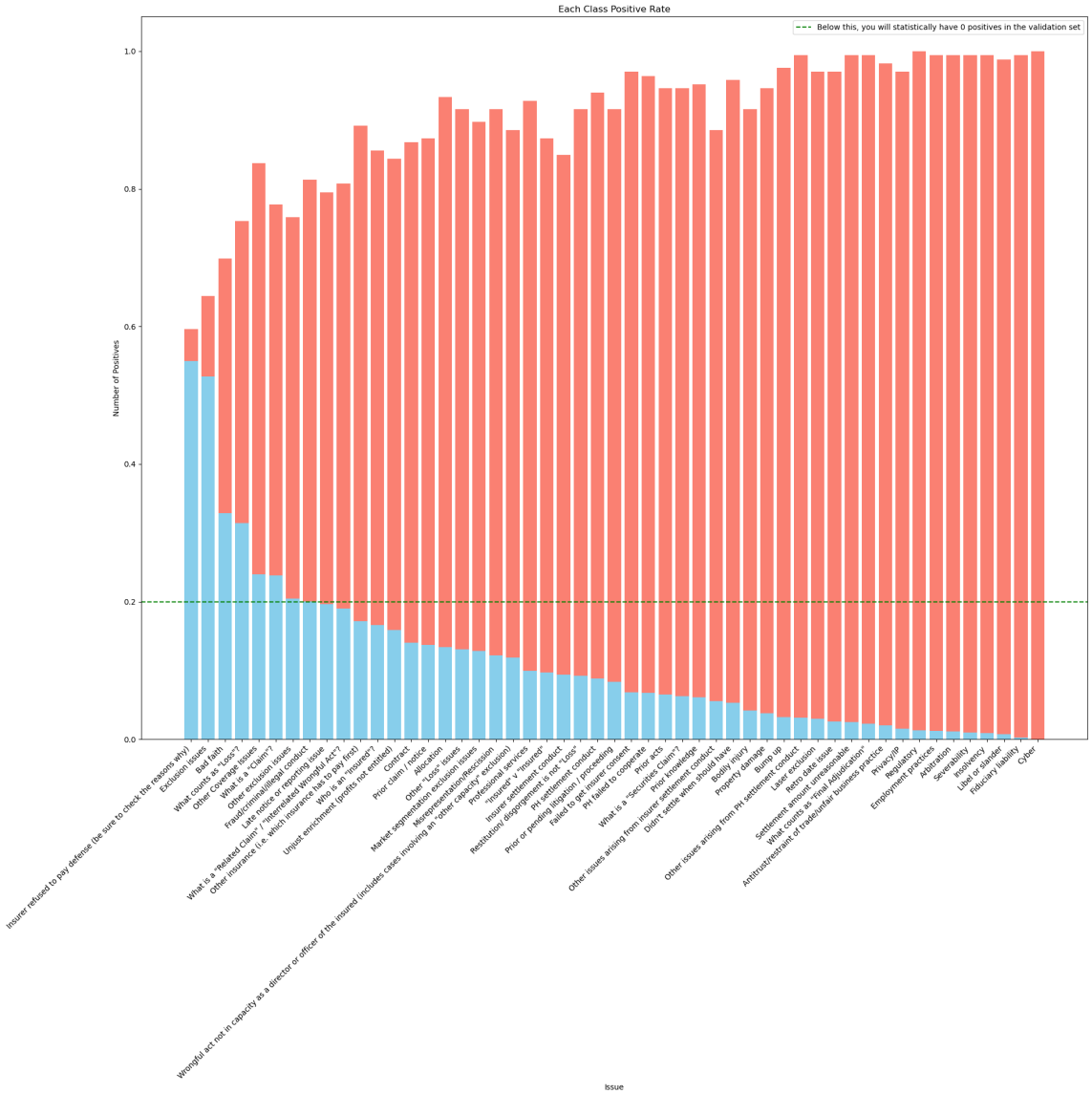


Figure 4: Positive Rate Per Class, better class balance but class weighting is still needed.

7 Proof of work

Figure 1 and Figure 2 show the new dataset. Figure 3 and Figure 4 show the balances of the new dataset.

8 Next Week's proposal

- Add K Folding and class weighting to the training in the hopes that this will produce a model that works well given the class imbalance.

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 12

Víctor C. Fernández
November 2024

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Adapted model querying by reducing the input text to a window of 2000 characters around the identified date.
- Fixed issue with access to PACE for running the adapted implementation for the models.
- Worked on the presentation/PPT for our future call with the judge Miguel.
- Worked on a pipeline to orchestrate the whole process using prefect.

What progress are you making next?

- Keep working on improving context retrieval for the dates.
- Have a presentation with Judge Miguel on Friday November 15th.
- Finalize implementation of a simple pipeline to run all processes end to end.
- Work on a simple UI to interact with the pipeline and upload and process a single file.
- Keep working on getting more content in for our paper.

Is there anything blocking you from making progress?

No, no blockers right now.

2 ABSTRACTS

1. **Title:** Structural text segmentation of legal documents
   - **URL:** https://dl.acm.org/doi/10.1145/3462757.3466085 (may require accessing with GaTech credentials to view actual paper)

   - **Abstract:** The growing complexity of legal cases has lead to an increasing interest in legal information retrieval systems that can effectively satisfy user-specific information needs. However, such downstream systems typically require documents to be properly formatted and segmented, which is often done with relatively simple pre-processing steps, disregarding topical coherence of segments. Systems generally rely on representations of individual sentences or paragraphs, which may lack crucial context, or document-level representations, which are too long for meaningful search results. To address this issue, we propose a segmentation system that can predict topical coherence of sequential text segments spanning several paragraphs, effectively segmenting a document and providing a more balanced representation for downstream applications. We build our model on top of popular transformer networks and formulate structural text segmentation as topical change detection, by performing a series of independent classifications that allow for efficient fine-tuning on task-specific data. We crawl a novel dataset consisting of roughly 74,000 online Terms-of-Service documents, including hierarchical topic annotations, which we use for training. Results show that our proposed system significantly outperforms baselines, and adapts well to structural peculiarities of legal documents. We release both data and trained models to the research community for future work.

   - **Summary:** The paper presents a novel approach for predicting argument structures in legal decisions from the Court of Justice of the European Union (CJEU) regarding fiscal state aid. The authors focus on identifying relationships between propositions—such as support, rebuttal, and undercut relations—in judicial documents. They introduce a unique annotation scheme and apply it to an existing dataset, adding layers that classify types of argumentative links between premises and conclusions. Using this enriched dataset, they conduct experiments to assess different NLP models' performance in predicting argumentative links, ultimately finding that an ensem-

ble of residual networks yields the best results. The study contributes to computational legal argumentation, a field that seeks to enhance access to legal reasoning by automating the extraction and understanding of complex argument structures.

- **Relevance:**This paper demonstrates methods for parsing and identifying complex relationships in legal text. The annotation approach for linking premises and conclusions in CJEU cases provides a structured method that could inform our project's extraction framework, especially in identifying the sequence of dates and events relevant to judicial delays. The study's focus on overcoming the challenges posed by complex legal arguments aligns with our need to parse the often intricate and implicit timelines in Dominican civil cases.

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.

For this week I'm maintaining the same changes from last week. Additionally was able to execute this code in PACE after fixing the issue that was preventing me from progressing in this area.

1. Function to combine all results from all dates into a single file here.

```
    def extract_context_around_date(text, date,
    ↪  window_size=1000):
# text_lower = text.lower()
date_lower = date.strip()
start = 0
start_position = text.find(date_lower, start)
if start_position == -1:
    print(f"Date '{date}' not found in the text.")
else:
    start_index = max(start_position - window_size, 0)
    end_index = min(start_position + len(date_lower) +
    ↪  window_size, len(text))
    context = text[start_index:end_index]
return context
```

Code 1—Function to extract smaller date window of 2000 characters

```
    def generate_output(ollama_models: list, query_template:
    ↪  str, input_folder: str, dates_folder: str,
    ↪  clusters_file: str, output_folder: str, repetitions:
    ↪  int = 1, delete_models_after_query: bool = False,
    ↪  model_hyperparameters: dict = {}):
# Instantiate the OllamaModelProcessor
for model in ollama_models:
    # Log the model being processed:
    log_in_color(f"Processing model: {model}", "green")
    # Step 1: Instantiate the OllamaModelProcessor
    processor = OllamaModelProcessor(model,
    ↪  **model_hyperparameters)

    # Step 2: Get the output options from the clusters.json
    ↪  file contained in the options key
    options_file = clusters_file
    with open(options_file, 'r', encoding='utf-8') as f:
        options_content = json.load(f)
    # Now set the options to be the value of the options key
    options = json.dumps(options_content["options"],
    ↪  ensure_ascii=False)

    for filename in os.listdir(input_folder):
        if filename.endswith(".txt"): # Process only txt files
            # Log the file being processed:
            log_in_color(f"Processing file: {filename}",
            ↪  "blue")
            # Remove the extension from the filename
            filename_name = os.path.splitext(filename)[0]
            # We append locate the output folder under a
            ↪  folder with the file name first and then a
            ↪  folder with the model name
            file_output_folder = os.path.join(output_folder,
            ↪  filename_name, model)
            # Create the output folder if it doesn't exist
            os.makedirs(file_output_folder, exist_ok=True)
```

```
                # Read the content of the document
                document_path = os.path.join(input_folder,
                ↪  filename)
                with open(document_path, 'r', encoding='utf-8')
                ↪  as f:
                    document_content = f.read()
                # Now we query the model replacing the last place
                ↪  holder with each date independently
                # Read the dates JSON file with same name as the
                ↪  document
                dates_file = os.path.join(dates_folder,
                ↪  f"{filename_name}.json")

                with open(dates_file, 'r', encoding='utf-8') as
                ↪  f:
                    date_objects = json.load(f)

                for i, date_object in enumerate(date_objects):
                    date_for_context = date_object["date"]
                    date_context = extract_context_around_date(do
                    ↪  cument_content,
                    ↪  date_for_context)
                    query_without_date =
                    ↪  generate_query(query_template,
                    ↪  date_context, options)

                    # Log the date position being processed:
                    log_in_color(f"Processing date: {i}",
                    ↪  "magenta")
                    expected_output = json.dumps(date_object,
                    ↪  ensure_ascii=False)
                    query = query_without_date.replace("{{MODEL_O
                    ↪  UTPUT_FORMAT}}",
                    ↪  expected_output)
```

```
                # For each query, we generate <repetitions>
                ↪ outputs to ensure output consistency
                for repetition in range(repetitions):
                    # Log the repetition being processed:
                    log_in_color(f"Processing repetition:
                    ↪ {repetition + 1}", "yellow")
                    output = processor.query_model(query)
                    output_path =
                    ↪ os.path.join(file_output_folder,
                    ↪ f"{filename_name}_{i}_{repetition +
                    ↪ 1}.txt")
                    with open(output_path, 'w',
                    ↪ encoding='utf-8') as f:
                        # First write a line with the date
                        ↪ object passed to the model
                        # f.write(expected_output + "\n\n")
                        f.write(output)
    # Log the model being deleted:
    log_in_color(f"Deleting model: {model}", "red")
    # Delete the ollama model to free up space
    if delete_models_after_query:
        processor._delete_model()
```

Code 2—Updated function to use smaller date text window for retrieving the context

4 DOCUMENTATION

Similar to what was indicated in past reports, the pipeline/flow we're currently following is the one below, where we first extract and clean the documents. Afterwards, a process takes 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 this flow:

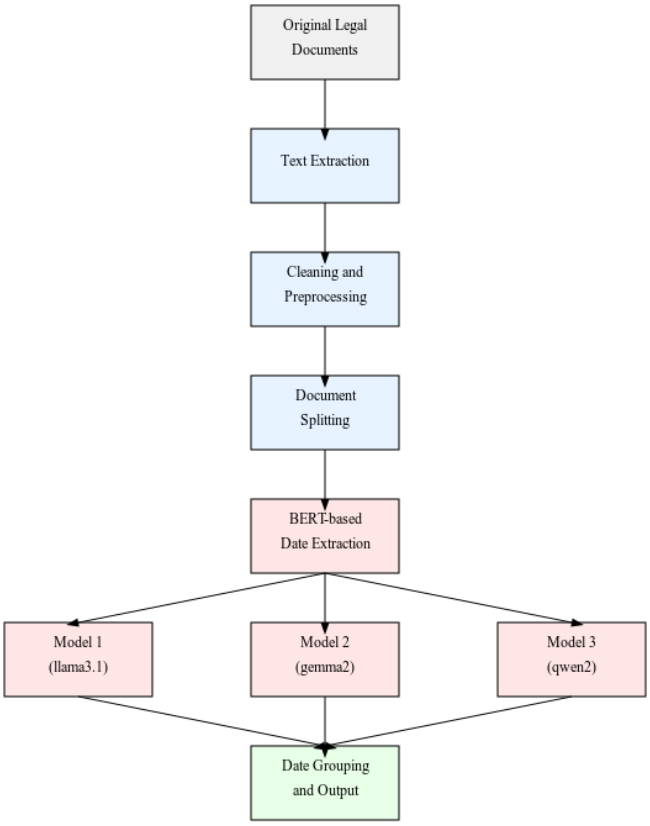


Figure 1—Full date extraction process

This week, my focus has been on the second to last step, with the goal of improving the models results given the low accuracy obtained when performing the benchmark. For the updates, I focused on the Llama models, running the changes mainly with Llama 3.1, Llama 3.2 and Llama 3.1 70b.

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:
  - Date retrieved by model in previous steps, copied exactly from the original

text document.

- 2000 characters window (1000 before and 1000 after the date) from the original text where the date is contained.
- Options/clusters template containing the categories by which to classify the different dates retrieved.

The output of the model will be a single text file containing a JSON object with the input date, a JSON object with the model output and a JSON object containing configuration details for the executed model such as hyperparameters used, model's name and execution time.

5 SCRIPT VALIDATION

The intention is to query the model over a set of 5 files generating 10 outputs for each of the dates contained in the files. Obtaining additionally, performance metrics from the execution.

Results when benchmarking the updated logic with the 3 mentioned models were the following:


```
    [
{
  "model_name": "llama3.2",
  "hyperparameters": {
    "temperature": 1e-13,
    "top_k": 5,
    "top_p": 0.5,
    "seed": 42
  },
```

```
    "documents_evaluated": [
      "Demanda en designacion de administrador
      ↪  judicial-0164-2023",
      "Demanda en entrega de documentos-0013-2023",
      "Demanda en entrega de documentos 1130-2023",
      "Demanda en entrega de documentos-0891-2023",
      "Demanda en entrega de documentos-0441-2022"
    ],
    "total_correct": 3,
    "total_incorrect": 43,
    "total_false_positives": 417,
    "total_validation_dates": 46,
    "total_model_dates": 450,
    "average_precision": 0.007076305220883534,
    "average_recall": 0.0654040404040404,
    "average_f1_score": 0.01277070679899386,
    "average_accuracy": 0.0654040404040404,
    "total_processing_time": 185.08245015144348,
    "average_processing_time_per_execution": 0.4112943336698744
  },
  {
    "model_name": "llama3.1",
    "hyperparameters": {
      "temperature": 1e-13,
      "top_k": 5,
      "top_p": 0.5,
      "seed": 42
    },
    "documents_evaluated": [
      "Demanda en designacion de administrador
      ↪  judicial-0164-2023",
      "Demanda en entrega de documentos-0013-2023",
      "Demanda en entrega de documentos 1130-2023",
      "Demanda en entrega de documentos-0891-2023",
      "Demanda en entrega de documentos-0441-2022"
    ],
```

```
    "total_correct": 6,
    "total_incorrect": 40,
    "total_false_positives": 412,
    "total_validation_dates": 46,
    "total_model_dates": 457,
    "average_precision": 0.014300206946056233,
    "average_recall": 0.1304040404040404,
    "average_f1_score": 0.025773626855414684,
    "average_accuracy": 0.1304040404040404,
    "total_processing_time": 248.18897771835327,
    "average_processing_time_per_execution": 0.5430831022283441
  },
  {
    "model_name": "llama3.1:70b",
    "hyperparameters": {
      "temperature": 1e-13,
      "top_k": 5,
      "top_p": 0.5,
      "seed": 42
    },
    "documents_evaluated": [
      "Demanda en designacion de administrador
      ↪  judicial-0164-2023",
      "Demanda en entrega de documentos-0013-2023",
      "Demanda en entrega de documentos 1130-2023",
      "Demanda en entrega de documentos-0891-2023",
      "Demanda en entrega de documentos-0441-2022"
    ],
    "total_correct": 5,
    "total_incorrect": 41,
    "total_false_positives": 405,
    "total_validation_dates": 46,
    "total_model_dates": 449,
    "average_precision": 0.012924926099808729,
    "average_recall": 0.1154040404040404,
    "average_f1_score": 0.023246053340694295,
    "average_accuracy": 0.1154040404040404,
    "total_processing_time": 1253.9444787502289,
    "average_processing_time_per_execution": 2.7927493958802425
  }
]
```

Execution would be carried out with the following hyperparameters:

- Temperature = 0.0000000000001,
- Top_k = 5,
- Top_p = 0.5
- Seed = 42

Here is a brief explanation of these hyperparameters:

- **Temperature**: A very low temperature (0.0000001) ensures that the outputs will be highly predictable. This is useful when we are looking for consistency and want results to be stable over time.
- **Top-k**: This limits the choices to only the top 5 probable words. This ensures that the model generates meaningful outputs without straying into highly unlikely predictions. It balances between randomness and relevance.
- **Top-p**: Combined with top-k, this gives fine control over the diversity of model output. A top_p value of 0.5 means the model will only consider words that make up 50% of the total probability distribution, ensuring more relevant results.
- **Seed**: Setting the seed makes the experiments reproducible, helpful for research purposes. With the same inputs and hyperparameters, in theory, we should get the same outputs every time (but in practice this doesn't always happen).

6 RESULTS VISUALIZATION

The following images provide the results compared between the different models when retrieving the context for the date given as an input to the model.

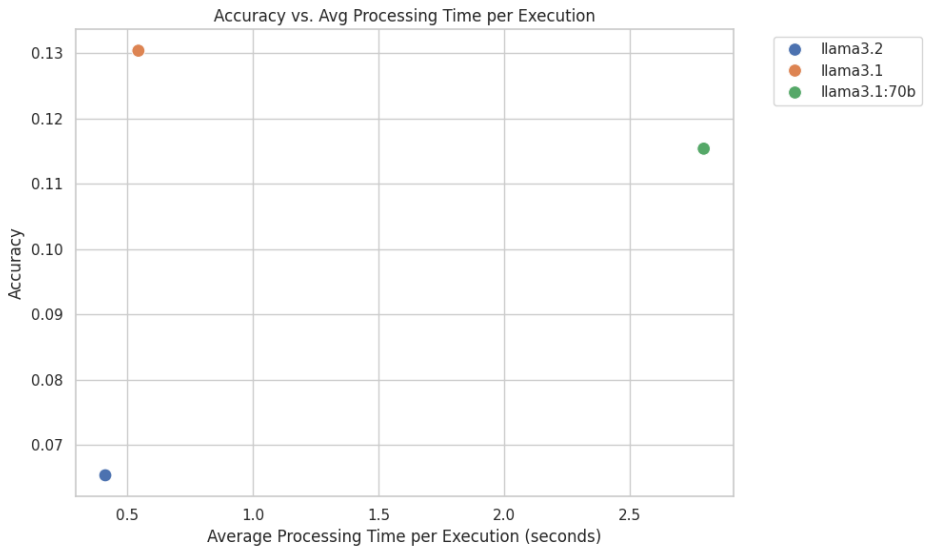


Figure 2—Accuracy vs. Processing time for each model

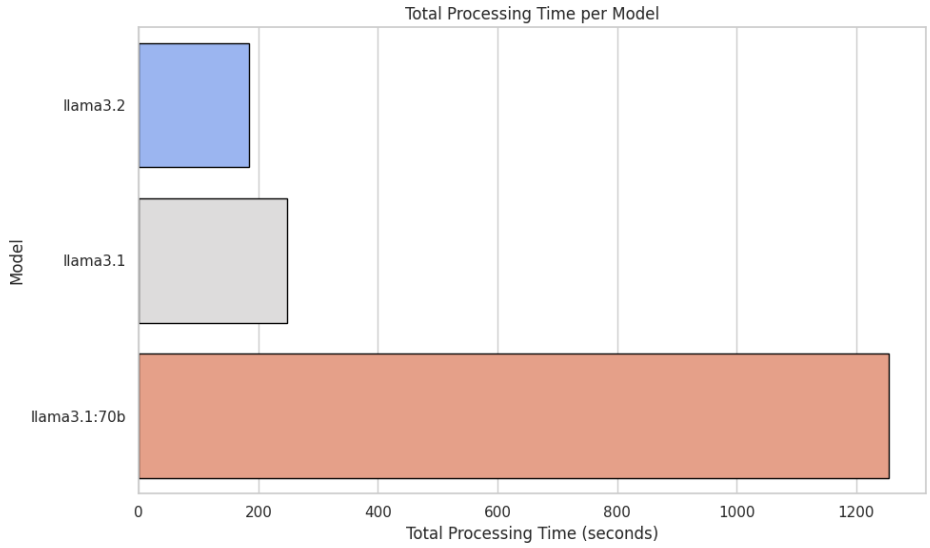


Figure 3—Total processing time for each model

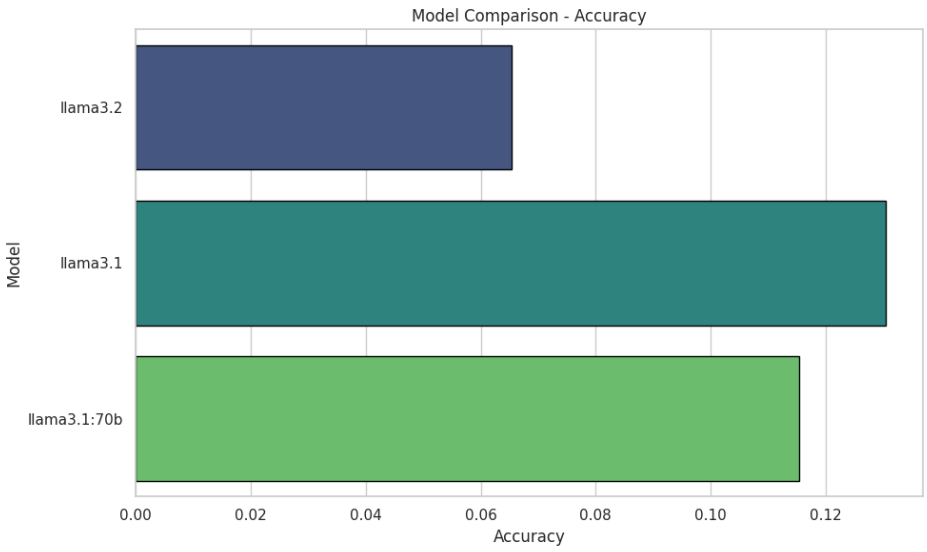


Figure 4—Accuracy comparison between models

7 PROOF OF WORK

The implemented function now uses a smaller piece of context to retrieve the class related to the date. This has improved the model's accuracy, although performance is still similar.

Below is an example of a date and context retrieved by the updated code to be used for verifying the model's new accuracy. Similarly to what was performed on the previous week, to ensure stability in the results, each input will be used 10 times to generate an output.

```
'seis (06) días del mes de enero del año dos mil
↪  veintitrés (2023)'
```

Code 4—Date to be classified

```
\nPRESIDENCIA DE LA CÁMARA CIVIL Y COMERCIAL DEL JUZGADO
↪ DE PRIMERA INSTANCIA DEL DISTRITO NACIONAL \n
↪ Ordenanza civil núm. 123-4567-ABCD-8901 Número único
↪ de caso (NUC) 1234-0158080 EN NOMBRE DE LA REPÚBLICA
↪ \n Ordenanza civil núm. 504-2023-SORD-0013 Número
↪ único de caso (NUC) 1234-0158080 En la ciudad de Santo
↪ Domingo de Guzmán, Distrito Nacional, capital de la
↪ República Dominicana, a los seis (06) días del mes de
↪ enero del año dos mil veintitrés (2023); años ciento
↪ setenta y nueve (179) de la Independencia y ciento
↪ sesenta (160) de la Restauración. \n \nPresidencia de
↪ la Cámara Civil y Comercial del Juzgado de Primera
↪ Instancia del Distrito Nacional, localizada en el
↪ primer piso del Palacio de Justicia del Centro de los
↪ Héroes de Constanza, Maimón y Estero Hondo, en el
↪ Distrito Nacional, República Dominicana, presidida por
↪ XXXXXXXXXX, quien dicta esta ordenanza en sus
↪ atribuciones de juez presidente de los referimientos y
↪ en audiencia pública constituida por la secretaria
↪ XXXXXXXX. \nXXXXXXXX, y el alguacil de estrados de
↪ turno. \n \nCon motivo de la demanda en referimiento
↪ sobre producción forzosa y entrega inmediata de
↪ certificado de matrícula interpuesta por la señora
↪ XXXXXXXX, dominicana, mayor de edad, titular de la
↪ cédula de identidad y electoral núm. 123-45678-1, con
↪ su domicilio en la calle XXXXXX, núm. 1, torre XXXXXX,
↪ apartamento núm. 1, urbanización XXXX, XXXX
```

Code 5—Window for retrieving date class

8 NEXT WEEK'S PROPOSAL

1. Keep working on improving context retrieval for the dates.
2. Have a presentation with Judge Miguel on Friday November 15th.
3. Finalize implementation of a simple pipeline to run all processes end to end.

4. Work on a simple UI to interact with the pipeline and upload and process a single file.
5. Keep working on getting more content in for our paper.

Week 12 | HAAG - NLP | Fall 2024

Alejandro Gomez

November 8th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- This week's focus was mostly on identifying and mitigating a bug, which was ultimately a successful endeavor with the help of the group's consultant, Nathan Dahlberg. There was a tokenization limit that I had overseen and then Nathan helped with a chunking function that chunks the training and validation sets, but also drops chunks if a date is cut off. It will also not train on blank data without chunks.

  We also began some work on a presentation for a judge in the Dominican republic judicial system where we will share our progress and gather further intel on helpful methods we can implement.

1.2 What are you planning on working on next?

- Given this new revelation, I'll be adding this to the large refactor I'm working on for the full ML pipeline (i.e. the chunking). This effort is part of an epic to prepare our code for conference submission but also to modularize the code so that when we create proof of concept this upcoming week, then we can tweak hyperparamters easily using sliders on gradio or streamlit.
- It would also be good to get the remaining data prepared by the team. More data is always nice.
- The refactoring also includes preparing documentation for the public deployment of the NER model, so this should feature full documentation with examples on how to use the model.
- Continuing: The labelling for the model on HugginFace also shows "1,2,3" in the JSON file instead of "DATE" in the BIO format so I need to look into this next.
- We will be presenting to the judge on the 15th so we should be wrapped up with coding and only writing the paper through the end of the term.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

Legal syllogism is a form of deductive reasoning commonly used by legal professionals to analyze cases. In this paper, we propose legal syllogism prompting (LoT), a simple prompting method to teach large language models (LLMs) for legal judgment prediction. LoT teaches only that in the legal syllogism the major premise is law, the minor premise is the fact, and the conclusion is judgment. Then the models can produce a syllogism reasoning of the case and give the judgment without any learning, fine-tuning, or examples. On CAIL2018, a Chinese criminal case dataset, we performed zero-shot judgment prediction experiments with GPT-3 models. Our results show that LLMs with LoT achieve better performance than the baseline and chain of thought prompting, the state-of-art prompting

method on diverse reasoning tasks. LoT enables the model to concentrate on the key information relevant to the judgment and to correctly understand the legal meaning of acts, as compared to other methods. Our method enables LLMs to predict judgment along with law articles and justification, which significantly enhances the explainability of models. doi[JY23]

2.2 Summary

This is an interesting paper and serves as a model for our writing. The topic is a bit different but it's adjacent to what we are moving toward in the future with regards to predictions in the legal space. Currently we want to extract dates and then we want to grab the context from the dates. Thereafter we can look into making predictions. The only caution I would have is prediction in judgement since machine learning models are essentially bias predictors, but nevertheless it's helpful to understand where these efforts are in the industry.

3 Scripts and Code Blocks

3.1 Code

```
import pandas as pd


def make_it_chonky(examples, max_length=128, max_empty_example_ratio=0.2):
  examples = [{'text': examples['text'][i], 'entities': examples['entities'][i]} for i
     in range(len(examples['text']))]

  all_chunks = []
  for example in examples:
    inputs = tokenizer(example['text'], return_offsets_mapping=True,
    add_special_tokens=False)

    input_ids, offset_mapping = inputs['input_ids'], inputs['offset_mapping']
    entities = example['entities']

    # Important that we use these offsets moving forward instead of trying to decode
    chunk input_ids,
    # since leading spaces will be removed by the tokenizer
    chunk_offset_mapping = [offset_mapping[i:i+max_length] for i in range(0, len(
    offset_mapping), max_length)]
    chunk_offset_mapping = [(x[0][0], x[-1][-1]) for x in chunk_offset_mapping]
    chunk_texts = [example['text'][x[0]:x[1]] for x in chunk_offset_mapping]

    chunks_spans = []
    for chunk_offset in chunk_offset_mapping:
      chunks_spans.append([])
      while len(entities) and chunk_offset[1] > entities[0]['end']:
        entity = entities.pop(0)
        entity['start'] -= chunk_offset[0]
        entity['end'] -= chunk_offset[0]
        chunks_spans[-1].append(entity)
    chunks = [{
        'text': chunk_texts[i],
        'entities': chunks_spans[i],
    } for i in range(len(chunk_texts))]
    chunks = pd.DataFrame(chunks)

    # drop any chunks that cut off the beginnings of dates
    chunks = chunks[chunks['entities'].apply(lambda x: not any(y['start'] < 0 for y in
     x))]

    # limit how many chunks with no dates that we allow
    if max_empty_example_ratio is not None:
      chunks['has_entity'] = chunks['entities'].apply(len) != 0
      non_empty_chunks = chunks[chunks['has_entity']]
      empty_chunks = chunks[~chunks['has_entity']]
      max_empty_examples = int(max_empty_example_ratio * len(non_empty_chunks))
      if len(empty_chunks) > max_empty_examples:
```

```
        empty_chunks = empty_chunks.sample(max_empty_examples)
      chunks = pd.concat([non_empty_chunks, empty_chunks])
      chunks = chunks.sample(frac=1).reset_index(drop=True)

    all_chunks.append(chunks)

  all_chunks = pd.concat(all_chunks)

  if 0: # print for debug purposes
    for row in all_chunks.to_dict('records'):
      print(row['text'])
      for entity in row['entities']:
        ent_text = row['text'][entity['start']:entity['end']].strip()
        print('*', row['text'][entity['start']:entity['end']])
        if not ent_text:
          print(entity)
          input()
      print()
  return {'text': all_chunks['text'].tolist(), 'entities': all_chunks['entities'].
    tolist()}
```

Listing 1: chunking

This is the chunking script that will chunk the training and validation sets and remove the superflous data as well. Special thanks to the team and Dr. Alexander and Nathan Dahlberg for helping solve this bug.

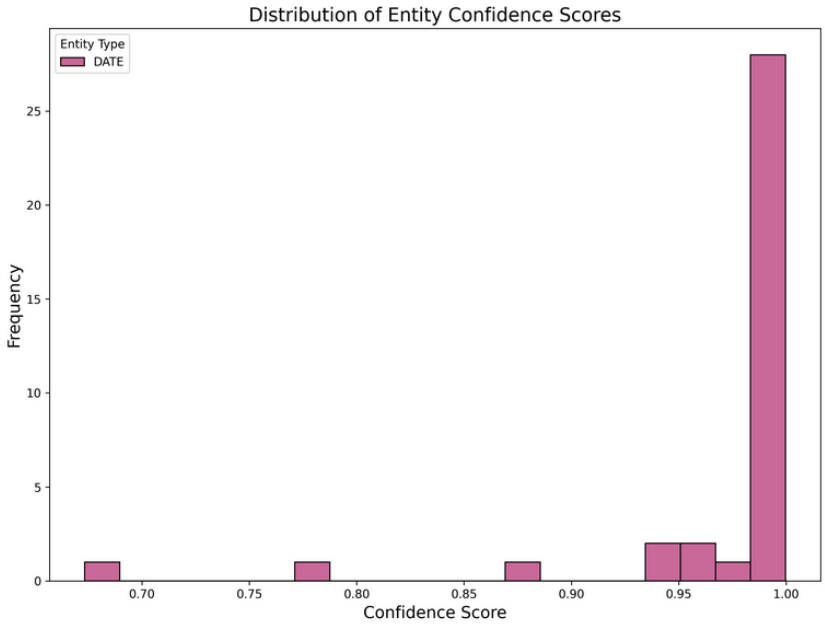


Figure 1: confidence score

Given this new refactor, the confidence score looks still looks good. The model is confident in its output but it is now both accurate AND precise.

3.2 List of Scripts

- Full NER pipeline Scaffolding (wip)
  - Note the new inclusion of the chunking function in the current training file for the NER model
  - This is the entry point for the refactored pipeline but I need to add the chunking to one one the modules.
  - Once the refactor is complete, these are the configs we hope to be able to modify with sliders on the Gradio UI

3.3 Documentation

Figure 2: pipeline visualization for code structure

This is the updated flow given the current standing of the NER model and the upcoming refactor. It shows the current scaffolding for the configs we willbe modifying liberally and shows the NER pipeline from creation to deployment. It also displays the expected intput "txt" and output "json". This will need to be documented on the publicly deployed cloud repo on Hugging Face so that other researchers can continue to build on this knowledge base.

3.4 Script Validation (optional)

N/A

3.5 Results Visualization

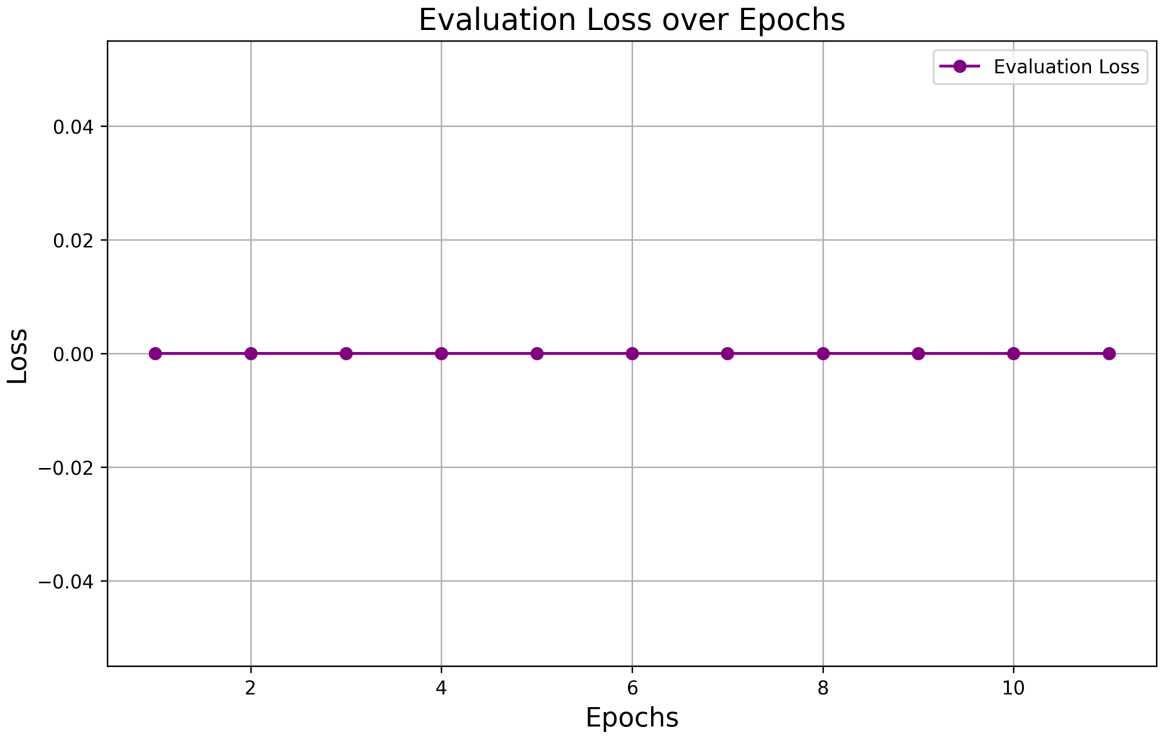


Figure 3: eval loss

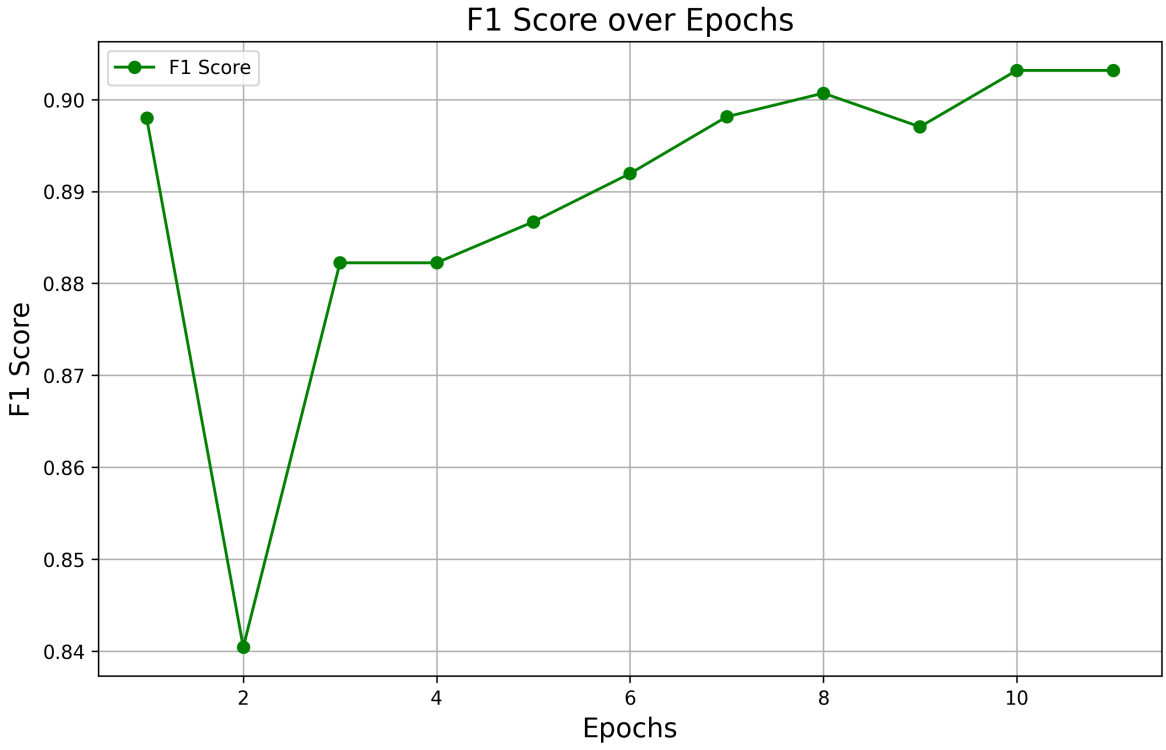


Figure 4: f1 score

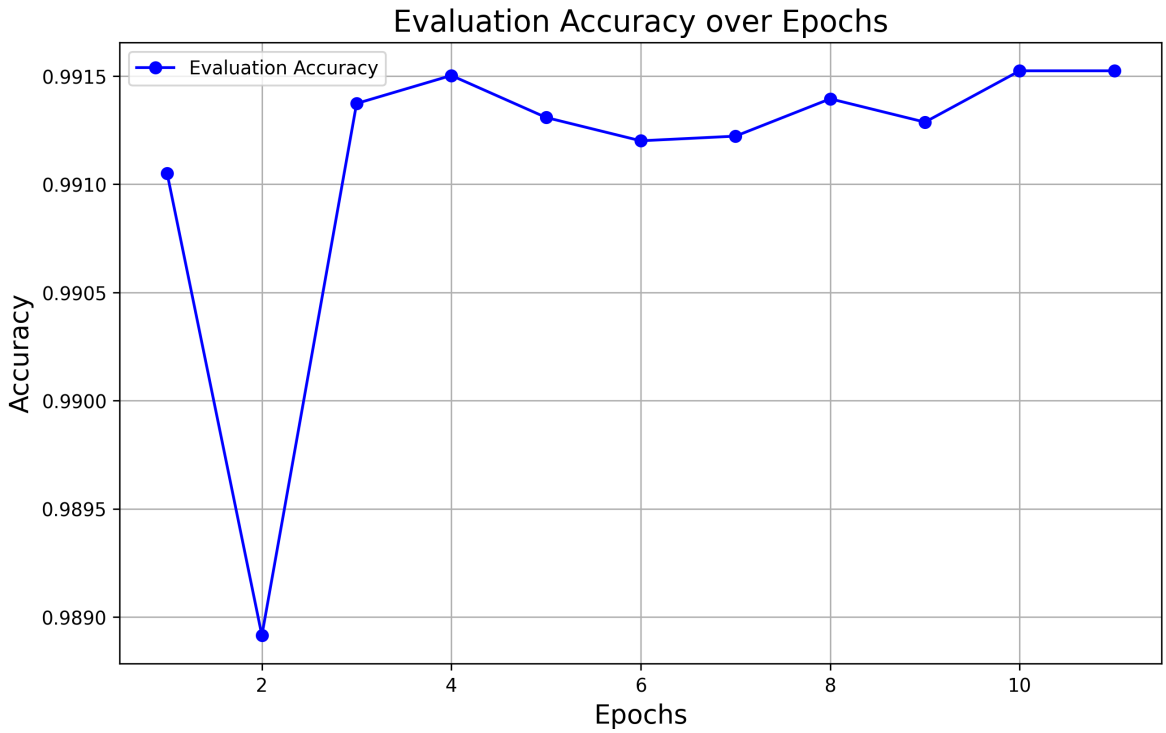


Figure 5: eval accuracy

3.6 Results Visualization Summary

The above graphs have shown massive improvement! The epochs are stable now and the progression of accuracy shows a steady increase! I now feel more confident to play with early exiting if needed so that the further epochs may have negligible results.

3.7 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- (See first section for full list. Brief summary below)
- Present to DR judge
- Set up Proof of Concept for DR judge
- Wrap up coding and shift to publication writing
- Update current documentation, e.g. NLP website.

References

[JY23] Cong Jiang and Xiaolei Yang. Legal syllogism prompting: Teaching large language models for legal judgment prediction. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, ICAIL '23, page 417–421, New York, NY, USA, 2023. Association for Computing Machinery.

HAAG NLP Sentencias — Week 12 Report
NLP-Gen Team

Karol Gutierrez

November 8, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Improved clustering model by using Hugging Face transformer and text semantic similarity.
- Evaluate performance with test set.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Continuous 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?

- continue with iterations of code for context analysis.
- Work on paper with team.
- Work on slides to present to Dr. Miguel, as well as video if necessary.
- Sync on Friday with Dr. Miguel and team to get feedback on the categories to use in our analysis as well as current performance.
- 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: Extracting Business Process Models Using Natural Language Processing (NLP) Techniques [SV17].

2.1 Abstract

This Doctoral Consortium paper discusses how NLP can be applied in the domain of BPM in order to automatically generate business process models from existing documentation within the organization. The main idea is that from the syntactic and grammatical structure of a sentence, the components of a business process model can be derived (i.e. activities, resources, tasks, patterns). The result would be a business process model depicted using BPMN - a dedicated business process modeling technique

2.2 Summary

The authors address the challenge of converting written descriptions of business processes into formal models that can be used for analysis and optimization. Their approach involves several key steps:

- **Text Preprocessing**: Cleaning and preparing the text, including sentence splitting and tokenization.
- **Part-of-Speech Tagging**: Identifying the grammatical role of each word (e.g., noun, verb, adjective) to understand the structure of sentences.
- **Dependency Parsing**: Analyzing the grammatical relationships between words to determine who is performing what action.
- **Entity Extraction**: Identifying key elements such as activities (actions), actors (people or systems), and objects involved in the process.
- **Model Generation**: Mapping the extracted information into a formal business process model using notations like BPMN (Business Process Model and Notation).

By using NLP techniques, the system can automatically generate a visual representation of the business process that reflects the steps described in the text. The authors demonstrate their method with examples and show that it can effectively produce accurate process models from natural language descriptions.

2.3 Relevance

This paper is relevant to our Sentencias project because it shows how certain NLP techniques can be used to extract structured text from unstructured text. In the paper, the authors show how to extract business model process just from the text descriptions, we aim to extract procedural content from the sentencias. Both projects deal with sequences of actions/events, both of them sometimes use especialized terminology and complex sentence structures. The method from this paper can be adapted to ours, thus helping in automating the extraction process and add value to the legal field.

3 Scripts and code blocks

The code is in the private repository repository. The progress for this week is in `./karol/week12/`.

3.1 Code developed

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

- Script to use Hugging Face transformer to evaluate semantic similarity of context and then split it into five buckets, Figure **??**. The method used is cosine similarity.
- Code to show compare results with test set in Fig **??**. This script generates the plots shown in this report.

4 Documentation

The documentation is present in the README.md file in the repository.

For this week, the only added library is: pip install sentence-transformers.

5 Script Validation

Figure 4 shows the generated contexts from the sentencias in Spanish. This document was used to validate the results and provide performance numbers.

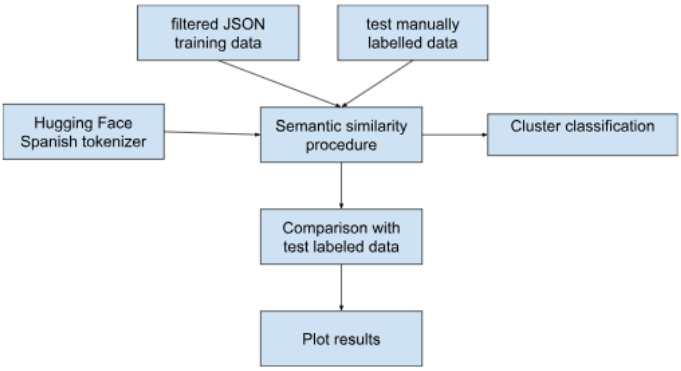


Figure 1: Code logic workflow to process data.

6 Results Visualization

I took a subset of the dates and manually labeled the categories, then compared the performance of my classifier with respect to this set. The results can be seen in Figure 5. The results are better than random by a reasonable margin but still it's not a reliable classifier.

Accuracy: 0.45 Precision: 0.49 Recall: 0.45 F1 Score: 0.42

7 Proof of Work

Figures 6 and Figure 7 show the final distribution of the data and samples of the classification during runtime, thus proving the work.

8 Next Week's Proposal

Refer to section 1.2 for details (avoid repetition).

References

[SV17] Konstantinos Sintoris and Kostas Vergidis. Extracting business process models using natural language processing (nlp) techniques. In *2017 IEEE 19th Conference on Business Informatics (CBI)*, volume 01, pages 135–139, 2017.

sentencias / karol / week12 / **clusterization.py**

Code | Blame | 84 lines (74 loc) · 3.33 KB

```python
folder_path = "./dates_marzo"
output_path = "./plots"
os.makedirs(output_path, exist_ok=True)

# Load Spanish sentence transformer model
model = SentenceTransformer("hiiamsid/sentence_similarity_spanish_es")

# Define representative phrases for each category
category_phrases = {
    "Supreme Court Judgments": "sentencia de la Suprema Corte de Justicia",
    "Appeal Court Decisions": "decisión de la Corte de Apelación",
    "Initial Trial Judgments": "juzgado de primera instancia dicta sentencia",
    "Filing and Appeal Actions": "recurso de apelación interpuesto",
    "Scheduled Dates and Readings": "fecha de lectura de sentencia"
}

# Step 1: Calculate embeddings for each category phrase
print("Calculating category embeddings...")
category_embeddings = {category: model.encode(phrase) for category, phrase in category_phrases.items()}

# Load and encode contexts
contexts = []
context_embeddings = []
print("Loading and encoding contexts...")
for filename in tqdm(os.listdir(folder_path), desc="Files processed"):
    if filename.endswith(".json"):
        with open(os.path.join(folder_path, filename), "r", encoding="utf-8") as file:
            try:
                data = json.load(file)
                if isinstance(data, list):
                    for entry in data:
                        context = entry.get("context", "")
                        if context:
                            contexts.append(context)
                            context_embeddings.append(model.encode(context))
            except json.JSONDecodeError:
                continue

# Step 2: Classify each context based on semantic similarity
classified_contexts = []
print("Classifying contexts based on semantic similarity...")
for i, context_embedding in tqdm(enumerate(context_embeddings), total=len(context_embeddings), desc="Classifying contexts"):
    similarities = {
        category: util.cos_sim(context_embedding, category_embedding).item()
        for category, category_embedding in category_embeddings.items()
    }
    best_category = max(similarities, key=similarities.get)
    classified_contexts.append({"context": contexts[i], "category": best_category})

# Step 3: Save classified data
output_file = os.path.join(output_path, "classified_contexts.json")
with open(output_file, "w", encoding="utf-8") as f:
    json.dump(classified_contexts, f, ensure_ascii=False, indent=4)

# Optional: Visualize category distribution
category_counts = {category: 0 for category in category_phrases.keys()}
for entry in classified_contexts:
    category_counts[entry["category"]] += 1

plt.figure(figsize=(10, 6))
categories, counts = zip(*category_counts.items())
plt.bar(categories, counts, alpha=0.7)
plt.title("Number of Contexts in Each Category")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(os.path.join(output_path, "category_counts.png"))
plt.close()

# Display sample classifications
for category in category_phrases.keys():
    print(f"\nSample contexts for category: {category}")
    sample_contexts = [entry["context"] for entry in classified_contexts if entry["category"] == category][:3]
    for context in sample_contexts:
        print(f" - {context}")
```

Figure 2: Clustering of categories for context

sentencias / karol / week12 / **comparison_test.py**

Code | Blame | 83 lines (71 loc) · 3.27 KB

```python
# Paths for input and output
input_file = "classified_contexts_test.json"  # Combined JSON file with context and actual category
output_path = "./plots"
os.makedirs(output_path, exist_ok=True)

# Load Spanish sentence transformer model
model = SentenceTransformer("hiiamsid/sentence_similarity_spanish_es")

# Define representative phrases for each category
category_phrases = {
    "Supreme Court Judgments": "sentencia de la Suprema Corte de Justicia",
    "Appeal Court Decisions": "decisión de la Corte de Apelación",
    "Initial Trial Judgments": "juzgado de primera instancia dicta sentencia",
    "Filing and Appeal Actions": "recurso de apelación interpuesto",
    "Scheduled Dates and Readings": "fecha de lectura de sentencia"
}

# Step 1: Calculate embeddings for each category phrase
print("Calculating category embeddings...")
category_embeddings = {category: model.encode(phrase) for category, phrase in category_phrases.items()}

# Load contexts and actual categories from the input file
contexts = []
actual_categories = []
print("Loading contexts and actual categories...")
with open(input_file, "r", encoding="utf-8") as f:
    data = json.load(f)
    for entry in data:
        contexts.append(entry["context"])
        actual_categories.append(entry["category"])

# Step 2: Classify each context based on semantic similarity
predicted_categories = []
print("Classifying contexts based on semantic similarity...")
for context in tqdm(contexts, desc="Classifying contexts"):
    context_embedding = model.encode(context)
    similarities = {
        category: util.cos_sim(context_embedding, category_embedding).item()
        for category, category_embedding in category_embeddings.items()
    }
    best_category = max(similarities, key=similarities.get)
    predicted_categories.append(best_category)

# Step 3: Calculate and display metrics
accuracy = accuracy_score(actual_categories, predicted_categories)
precision, recall, f1, _ = precision_recall_fscore_support(actual_categories, predicted_categories, average="weighted")

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")

# Step 4: Plot category comparison
category_counts_actual = {category: actual_categories.count(category) for category in category_phrases.keys()}
category_counts_predicted = {category: predicted_categories.count(category) for category in category_phrases.keys()}

# Bar plot for actual vs predicted category counts
labels = list(category_phrases.keys())
actual_counts = [category_counts_actual[label] for label in labels]
predicted_counts = [category_counts_predicted[label] for label in labels]

x = np.arange(len(labels))
width = 0.35

plt.figure(figsize=(10, 6))
plt.bar(x - width/2, actual_counts, width, label='Actual')
plt.bar(x + width/2, predicted_counts, width, label='Predicted')
plt.xlabel("Category")
plt.ylabel("Count")
plt.title("Actual vs Predicted Category Counts")
plt.xticks(x, labels, rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig(os.path.join(output_path, "category_comparison.png"))
plt.close()
```

Figure 3: Test results and plotting

Figure 4: Sentencias used for validation of results

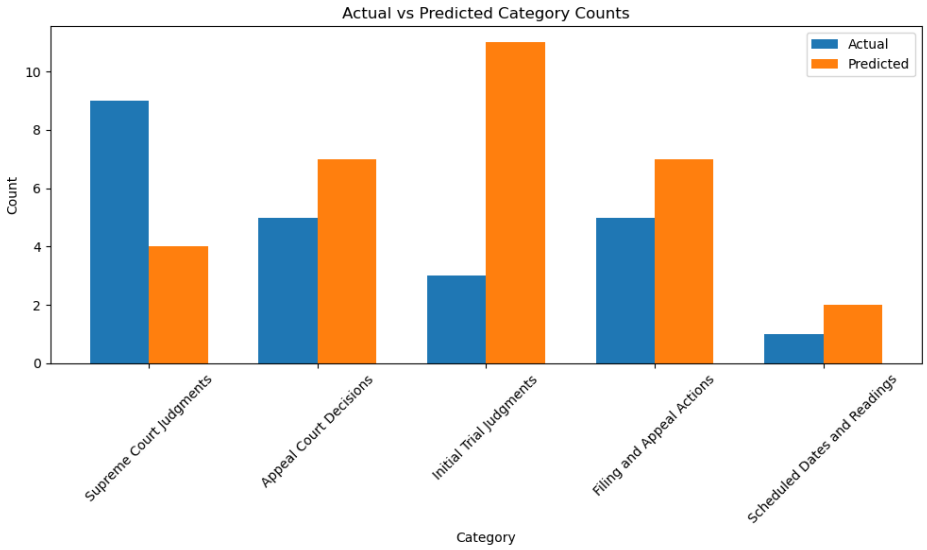


Figure 5: Comparison of results with test data

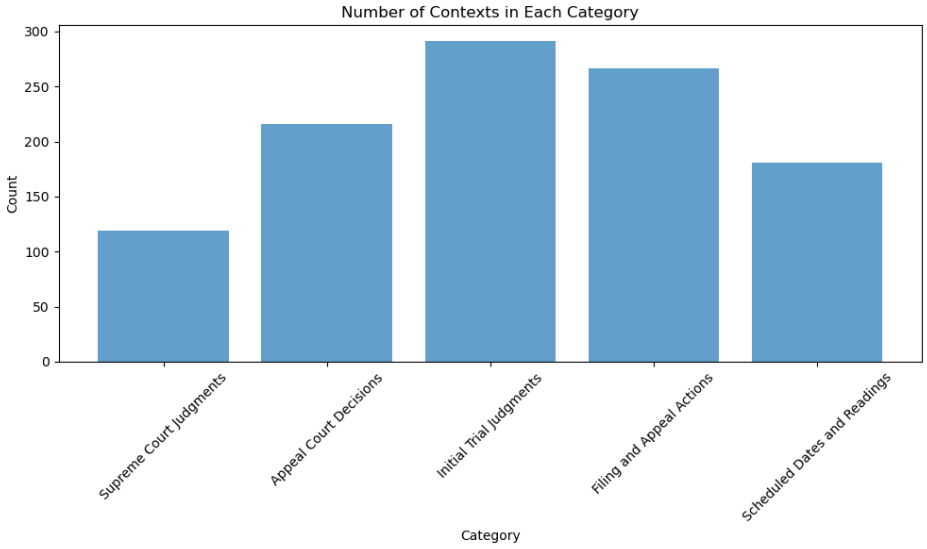


Figure 6: Distribution of categories in available data

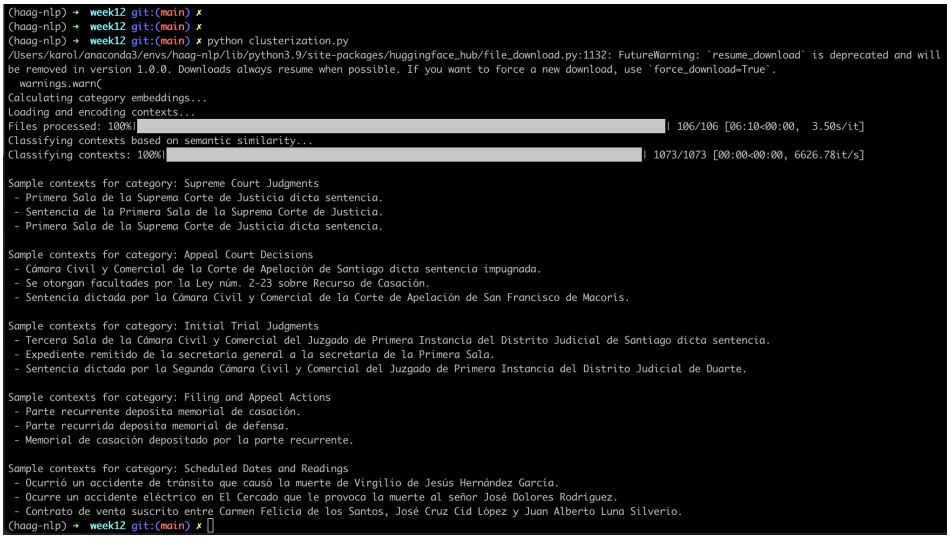


Figure 7: Code working and showing the classified contexts

Week 12 Report

Thuan Nguyen – Clearinghouse Summarization project

Friday, November 8, 2024

Summary

What progress did you make in the last week?

- Developed a proof-of-concept for generating structured, question-based summaries using a large language model (LLM) for various document types.
- Tailored the question list dynamically based on the specific document type (e.g., court opinions, orders).
- Implemented functionality to generate direct links to relevant sections of documents for easier navigation.
- The code for this experiment is still being worked on, so I will upload to GitHub when it's complete.

What are you planning on working on next?

- Yesterday, the team provided the CSV dataset of the orders/opinions. I'll test my model on a bunch of orders to see how good the summaries are.

Is anything blocking you from getting work done?

- Nothing at the moment.

Abstract

INTERPRETABILITY IN THE WILD: A CIRCUIT FOR INDIRECT OBJECT IDENTIFICATION IN GPT-2 SMALL

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris & Jacob Steinhardt

+ The research investigates how the GPT-2 small model performs indirect object identification (IOI) using 26 attention heads, structured into 7 classes, to explain the mechanism of this natural language task.

+ A systematic approach called "path patching" was employed to trace components back from model outputs, complemented by embedding space projections and activation patching for better understanding.

+ The findings highlight the complexity of mechanistic interpretability, revealing instances of redundant behavior in heads and the unexpected utilization of known structures, complicating the search for complete mechanisms.

+ Three criteria—faithfulness, completeness, and minimality—were proposed to validate the identified circuit, demonstrating significant improvements yet highlighting gaps in achieving full understanding.

Work done this week – further details

Developed a proof-of-concept for generating **structured, question-based summaries** using a large language model (LLM) for various document types.

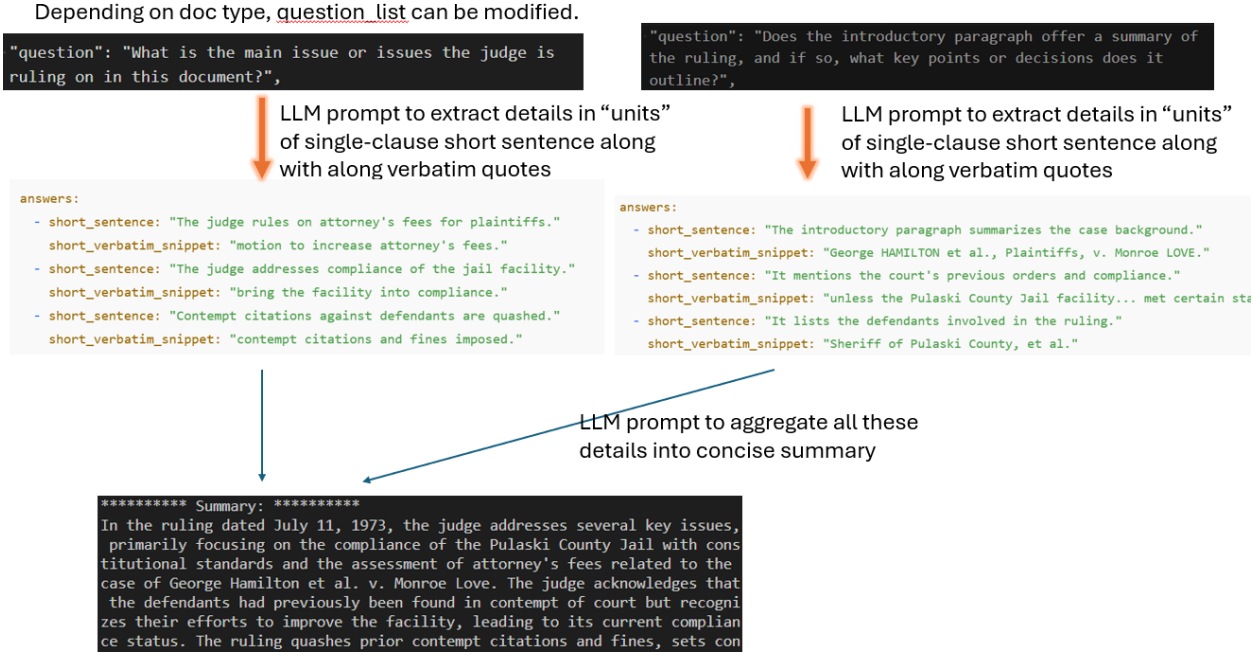
Tailored the **question list** dynamically based on the specific document type (e.g., court opinions, orders).

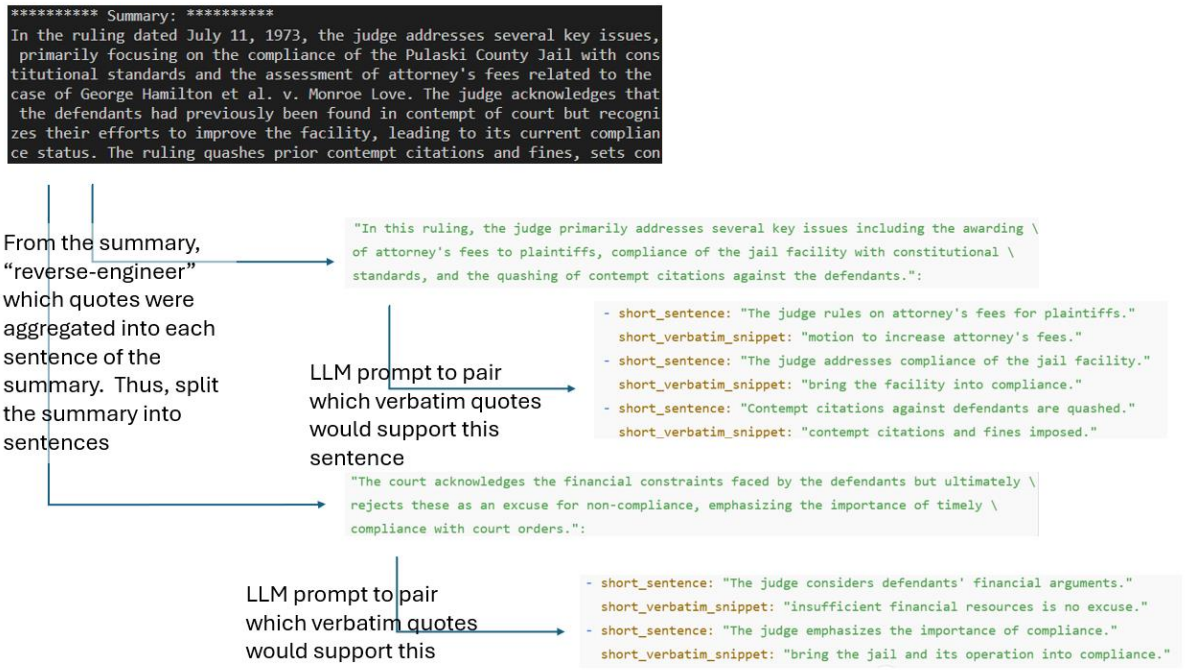
Implemented functionality to **generate direct links** to relevant sections of documents for easier navigation.

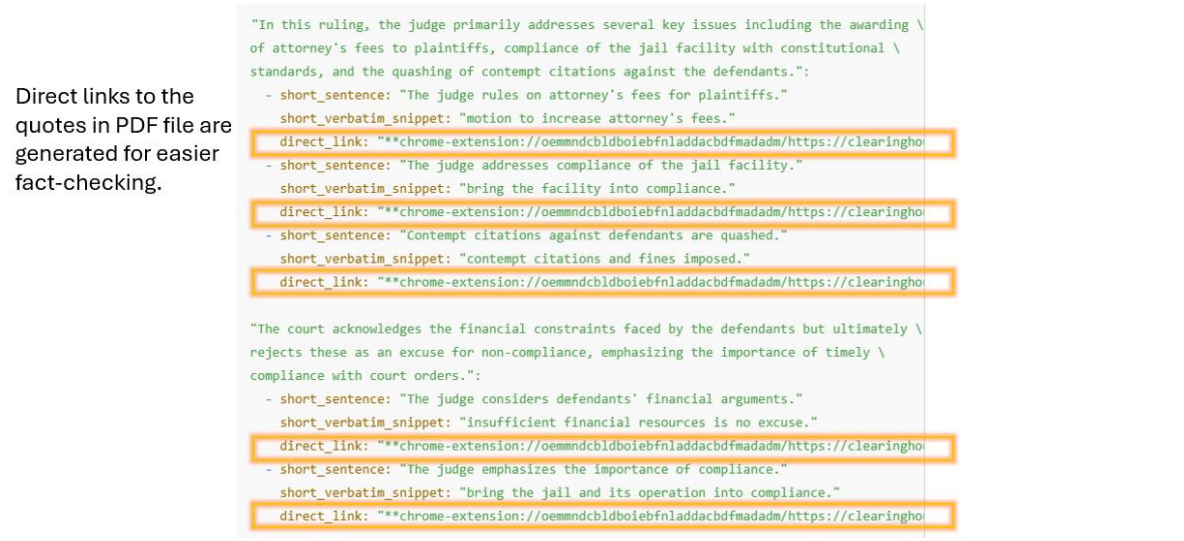
The code for this experiment is still being worked on, so I will upload to GitHub when it's complete.

Thuan – working on summarization with a custom list of queries. Outputs verbatim quotes along with the summary

Depending on doc type, question_list can be modified.







Scripts - Documentation - Script Validation - Results Visualization - Proof of Work

For further details, please refer to my scripts posted on GitHub or the information above.

Next week's proposal

- Yesterday, the team provided the CSV dataset of the orders/opinions. I'll test my model on a bunch of orders to see how good the summaries are.

Week 12 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

November 2024

0.1 What did you work on this week?

1. Reviewed feedback from Interview team, seems like Gemini might be the best contender based on the grad school experiments
2. Adjusted workbench prompts based on feedback from law students to extract more settlement information
3. Started collecting training material for next year's VIP students
4. Reviewed how LLMs do citations in practice to see if there's any sort of standard. Seems like there isn't.
5. Scheduled meeting with Jasmine from clearinghouse to discuss possible integration as summarization work matures
6. I just got the settlement dataset so I am working through the different types of settlements to craft my prompt better

0.2 What are you planning on working on next?

1. Meeting next week with Jasmine
2. Review more of the different settlements to improve prompts
3. Continue working with interview team to determine best model and workflow.

0.3 Is anything blocking you from getting work done?

1. No.

1 Abstracts

- Title: Incorporating Domain Knowledge for Extractive Summarization of Legal Case Documents. Conference / Venue: ICAIL '21: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law. Link: https://dl.acm.org/doi/10.1145/3462757.3466092

- Automatic summarization of legal case documents is an important and practical challenge. Apart from many domain-independent text summarization algorithms that can be used for this purpose, several algorithms have been developed specifically for summarizing legal case documents. However, most of the existing algorithms do not systematically incorporate domain knowledge that specifies what information should ideally be present in a legal case document summary. To address this gap, we propose an unsupervised summarization algorithm DELSumm which is designed to systematically incorporate guidelines from legal experts into an optimization setup. We conduct detailed experiments over case documents from the Indian Supreme Court. The experiments show that our proposed unsupervised method outperforms several strong baselines in terms of ROUGE scores, including both general summarization algorithms and legal-specific ones. In fact, though our proposed algorithm is unsupervised, it outperforms several supervised summarization models that are trained over thousands of document-summary pairs.

- Summary: This paper proposes an unsupervised method of summarization for legal documents that takes into account legal expert feedback. However, this seems to be geared towards specific court type documents instead of a general process. They also compute ROUGE scores as their metrics. However, do not take into account expert reviews. However, based on the information presented they have outperformed many methods present

- Relevance: They do a similar method to how we do summarization by providing context from legal experts. Depending on how hard it is to incorporate new legal guidelines based on the clearinghouse instead of the legal feedback/guidelines used by the

2 Relevant Info

- Summary Chain of Thought (CoT) is a technique to prompt LLMs for information to provide context for summarization. I took a domain centric approach in this experiment to extract entities the Clearinghouse is looking for specifically.

- 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

- Anthropic is a company that produces the Claude family of models that compete with GPT-4.

- The two best models in terms of accuracy and cost tradeoff is Claude 3.5 Sonnet and Claude 3 Haiku

3 Scripts

No scripts as focus was primarily on prompt optimization this week. I wasn't able to translate it to code yet due to still testing and having to deal with move-in stuff for my new apartment.

4 Documentation

1. After making an Anthropic account and obtaining credits, go to https://console.anthropic.com/workbench
2. On the UI, you will see something like this:

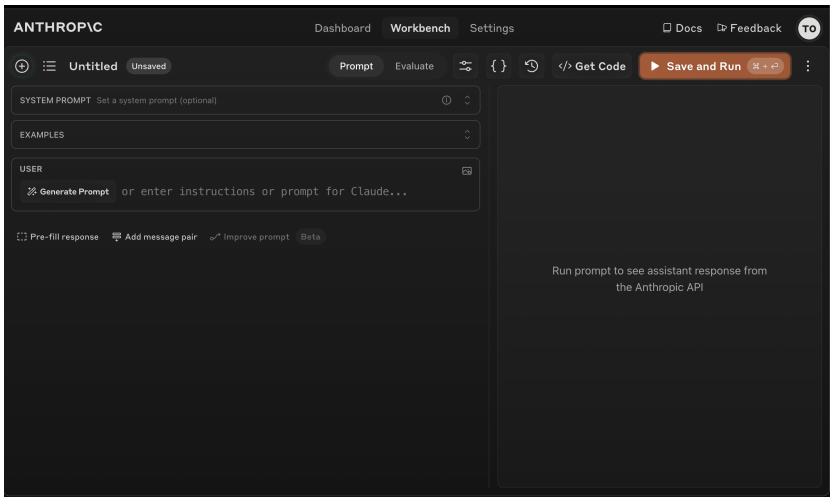


Figure 1: Workbench UI

3. Click "Generate Prompt" to get the following popup:

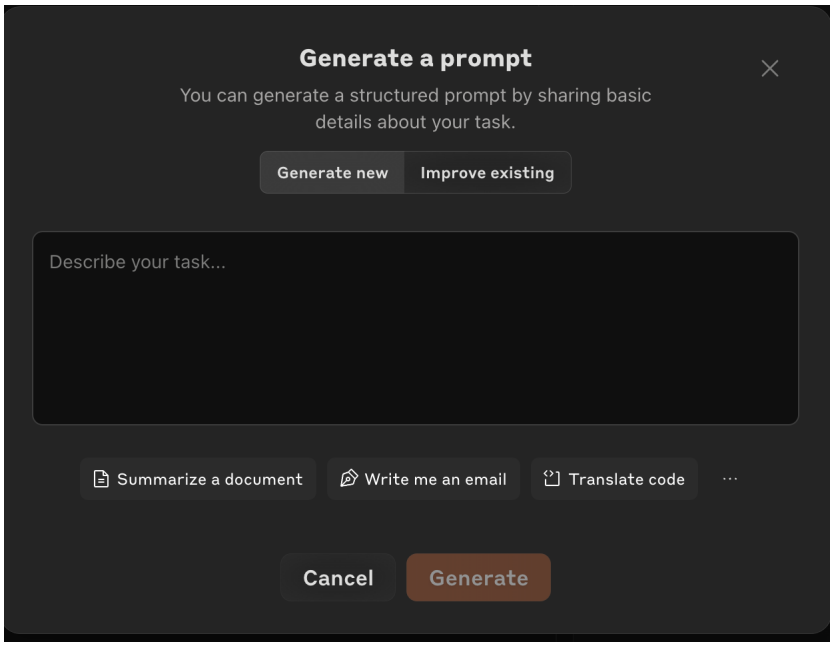


Figure 2: Generate Prompt

4. Enter a basic prompt of the task you are trying to accomplish and workbench will expand it to give you a better starting point.

5 Results

5.1 Prompts

Below are the prompts that were refined based on feedback from law students and workbench. First prompt will extract key details. The second will take that information to make a summary.

First Prompt

You are a law student tasked with extracting key information from a chunk of a settlement agreement. Your goal is to identify and summarize specific elements of the agreement. Here is the settlement chunk you will analyze:

<settlement_chunk>
SETTLEMENT_CHUNK
</settlement_chunk>

Please extract the following information from the settlement chunk:

1. Actions to be Taken by Defendants: Describe who has agreed to do what. Be very detailed in providing this information.

2. Damages (Money): Identify who is paying for what, including attorney fees. For the money to be paid to plaintiffs, do not name the plaintiffs and report the total sum to be paid to plaintiffs
3. Implementation and Enforcement: Note if there's a court-appointed "monitor" or other oversight.
4. Duration: How long the settlement is in effect.
5. Conditional Agreements: Mention any conditions for the settlement (e.g., "will only agree IF ...").
6. Policy Adoptions: Note any agreement to adopt policies and provide any relevant details about those policies. Do not omit important information and describe in detail.
7. The date of the settlement: This is typically the document's filing date, the date the document is dated, or the date of execution

For each piece of information you extract, include a citation of the text from the settlement chunk that supports your conclusion. Use the following format:
<citation>(Exact quote from the text)</citation>
If any of the requested information is not present in the settlement chunk, state "Not Specified" for that item.

If any acronyms are present and their definitions are defined, please spell out the acronym the first time its used.

After extracting the information, provide a brief summary of your findings.

Important: Do not extract or include the following types of information:
- Introductory and Boilerplate Information
- Reporting Information (how parties must report progress)
- Notice for Class Actions (how parties must give notice to consumers for class action suits)
- Giving Up Claims or Admitting Fault (it's a given that settling parties must give up claims)

Present your findings in the following format:
<extracted_information>
1. Actions to be Taken by Defendants:
(Your summary)
(Citation if applicable)

2. Damages (Money):
(Your summary)
(Citation if applicable)

3. Implementation and Enforcement:
(Your summary)
(Citation if applicable)

4. Duration:
(Your summary)
(Citation if applicable)

5. Conditional Agreements:
(Your summary)
(Citation if applicable)

6. Policy Adoptions:
(Your summary)
(Citation if applicable)

7. Date of the settlement:
(Your info)
(Citation if applicable)
</extracted_information>

<summary>
(Your brief summary of the key points found in the settlement chunk)
</summary>

View from workbench:

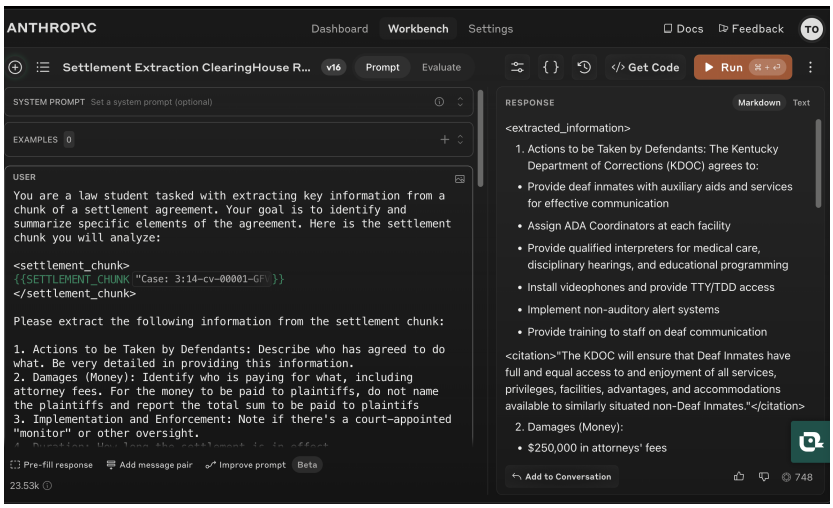


Figure 3: Workbench for Prompt 1

Second Prompt:
You are a law student skilled at distilling extracted information and partial summaries into informative summaries. You will be provided with a set of

extracted information and a partial summary about a legal settlement. Your task is to create a concise, one-paragraph summary of the settlement.

Here is the extracted information and partial summary:

<extracted_info_and_summary>
EXTRACTED_INFO_AND_SUMMARY
</extracted_info_and_summary>

Using the provided information, create a summary of the settlement following these guidelines:

1. Begin with a sentence describing when the settlement was reached, including the specific date.
2. Include information on the following aspects, if available:
- Actions to be Taken by Defendants
- Damages (Money)
- Implementation and Enforcement
- Duration
- Conditional Agreements
- Policy Adoptions
3. Keep the summary to one paragraph. 4. If any information provides a citation, do not use that information in your summary.
5. Do not omit any of the actions or policy adoptions noted.
5. Write the summary in past tense.

Carefully review the extracted information and partial summary to ensure you capture all relevant details. Focus on presenting the most important aspects of the settlement in a clear and concise manner.

Please provide your summary within <summary> tags. Do not include any additional commentary or explanations outside of the summary itself.

Example generate summary:

<summary> A settlement was reached on November 21, 1996, establishing comprehensive requirements for Chicago Public Schools (CPS) and the Illinois State Board of Education regarding homeless students' education. The agreement required CPS to provide immediate enrollment for homeless students, offer transportation assistance including CTA tokens and alternative transportation options, implement dispute resolution processes, provide staff training, coordinate with other agencies serving homeless students, and offer tutoring services. The settlement mandated the adoption of several key policies, including the Policy on Education of Homeless Children and Youth, along with specific policies covering enrollment requirements, non-discrimination, transportation, and dispute resolution procedures. While no direct monetary damages were specified in the settlement beyond attorney's fees (to be negotiated), the agreement established reporting requirements through the 1998-99 school year. The Court retained jurisdiction for enforcement purposes, allowing class members to file motions to enforce the terms of the agreement. </summary>

6 Proof of work

The prompts were generated using Anthropic Workbench and ran using their LLMs, so the results are relatively reliable.

6.1 Known Limitations

With the new dataset I was given, there are different types of settlement such as Consent Decrees which need I believe need to be mentioned in the output summary. So I will be adding that to the prompt.