HAAG NLP Summarization Week 5

Michael Bock

September 2024

1 Slack Questions

What did you accomplish this week?

- Attempted to get text classification working
- Found cuda error with huggingface trainer, switched to torchtext and have tokenization and text normalization working
- Created dataset for text classification to get text classifier working
- Tested the MLM pipeline, it works but should probably use PACE, this should be kept for later so we can pretrain a large language model(not like an LLM more like just a transformer with a lot of layers)
- Met with UPenn, sent them my ssh key for database access
- Briefly did some prompt engineering with Ollama and Llama3.1, it could only answer 1 question per session, after that it would be wrong.

What are you planning on working on next?

- Continue working on TorchText Training pipeline

What is blocking you from progressing?

- Must download Global Protect for PACE access, many models will be difficult to train outside of the PACE cluster

2 Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

Link: https://arxiv.org/abs/1409.3215

2.1 Brief Analysis

The idea of seq2seq is that some tasks require getting an input sequence of one length and outputting a sequence of another length. They do this using an encoder. The encoder can take in a text of whatever length you want, then it runs an LSTM on that. Then, each word gets added to a hidden state. You don't care about the output sequence of the encoder, which must be of the same length as the input sequence. Instead you pass the encoder's hidden state to a decoder, which then can generate a sequence of any length you want. One way I interpreted this was that the encoder was a feature extractor which mapped features in the text to a fixed dimension latent space. Then, a decoder is like a text generation head which outputs a sequence of words using the extracted features.

3 Scripts and Code Blocks

MLM Pipeline:

mistral_datasets.py(updated):

```python
import torch
import numpy as np
from torch.utils.data import Dataset
from tqdm import tqdm
from datasets import load_dataset, Dataset as HFDataset
import re
from transformers import AutoTokenizer
import string
from langdetect import detect
import random
from transformers import DataCollatorForLanguageModeling
import sys
sys.path.append('../../')
from summarizers.get_complaints import get_complaint_only_cases
from summarizers.ocr import read_doc, extract_text_from_pdf

def normalize(comment, lowercase, remove_stopwords):
    if lowercase:
        comment = comment.lower()
    comment = nlp(comment)
    lemmatized = list()
    for word in comment:
        lemma = word.lemma_.strip()
        if lemma:
            if not remove_stopwords or (remove_stopwords and lemma not in stops):
                lemmatized.append(lemma)
    return " ".join(lemmatized)

ISSUE_IDS = {
    'Child Welfare': 0,
    'Criminal Justice (Other)': 1,
    'Disability Rights': 2,
    'Education': 3,
    'Election/Voting Rights': 4,
    'Environmental Justice': 5,
    'Equal Employment': 6,
    'Fair Housing/Lending/Insurance': 7,
    'Immigration and/or the Border': 8,
    'Indigent Defense': 9,
    'Intellectual Disability (Facility)': 10,
    'Jail Conditions': 11,
    'Juvenile Institution': 12,
    'Labor Rights': 13,
    'Mental Health (Facility)': 14,
    'National Security': 15,
    'Nursing Home Conditions': 16,
    'Policing': 17,
    'Presidential/Gubernatorial Authority': 18,
    'Prison Conditions': 19,
    'Public Accommodations/Contracting': 20,
    'Public Benefits/Government Services': 21,
    'Public Housing': 22,
    'Reproductive Issues': 23,
    'School Desegregation': 24,
    'Speech and Religious Freedom': 25
}

class DocumentClassificationDataset(Dataset):
    def __init__(self, tokenizer, cases_path):
        self.dataset = {'text': [], 'labels': []}
        print('Retrieving complaints')
        cases = get_complaint_only_cases(cases_path)
        print('Iterate over complaints')
        self.text_len = 512
        for entry in tqdm(cases):
            summary = entry.summary
            if not summary or len(summary) < 1 or not entry or not entry.
    case_documents or len(entry.case_documents) < 1:
                continue
            doc = entry.case_documents[0]
            document_text = read_doc(doc)
            print(entry.case_types)
            self.dataset['text'].append(document_text)
            self.dataset['labels'].append(ISSUE_IDS[entry.case_types[0]])

        self.dataset = HFDataset.from_dict(self.dataset)
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        """
        Args:
            idx (int): Index of the sample to retrieve.

        Returns:
            tuple: (data_sample, label) where data_sample is the data at index idx,
                    and label is the corresponding label.
        """
        data_sample = self.dataset[idx]['text']
        while data_sample.isspace() or detect(data_sample) != 'en':
            idx += 1
            data_sample = self.dataset[idx%len(self.dataset)]['text']
        return data_sample, self.dataset[idx%len(self.dataset)]['labels']


    #Use this in the event of using a DataCollator
    def prepare_corpus(self):

        def tokenize(sample):
            return self.tokenizer(sample['text'])

        tokenized_input = self.dataset.map(tokenize, batched = True, num_proc = 4,
    remove_columns = ['text', 'labels'])

        def crop(sample):
            print(len(sample['input_ids']))
            return {'input_ids': sample['input_ids'][:4096], 'attention_mask':
sample['attention_mask'][:4096]}

        tokenized_input = tokenized_input.map(crop, batched = True, num_proc = 4)
        print(tokenized_input)

        return tokenized_input



class MistralMLMDataset(Dataset):
    def __init__(self, tokenizer, split = 'train', text_len = 24):
        """
        Args:
            split (list or ndarray): "train" or "validation".
        """
        self.dataset = load_dataset("pile-of-law/pile-of-law", "nlrb_decisions")
        self.dataset = self.dataset[split]
        print(self.dataset)
        self.text_len = text_len
        self.tokenizer = tokenizer

    def __len__(self):
        """Returns the number of samples in the dataset."""
        return len(self.dataset)

    def __getitem__(self, idx):
        """
        Args:
            idx (int): Index of the sample to retrieve.

        Returns:
            tuple: (data_sample, label) where data_sample is the data at index idx,
                   and label is the corresponding label.
        """
        data_sample = self.dataset[idx]['text']
        lang = ''
        while data_sample.isspace() or lang != 'en':
            try:
                lang = detect(data_sample)
                idx += 1
                data_sample = self.dataset[idx%len(self.dataset)]['text']
            except:
                lang = ''
        #tokens = self.tokenizer(data_sample, return_tensors='pt')
        #text_index = random.randrange(0, len(data_sample) - self.text_len + 1)
        #data_sample = data_sample[text_index: text_index + self.text_len]
        return data_sample

    #Use this in the event of using a DataCollator
    def prepare_corpus(self):
        #concatenated_sequences = []
        #concatenated_masks = []
        #for data_sample in self.dataset:
        #    data_sample = data_sample['text']
        #    tokenized = self.tokenizer(data_sample)
        #    concatenated_samples.extend(tokenized['input_ids'])
        #    concatenated_masks.extend(tokenized['attention_masks'])

        def tokenize(sample):
            return self.tokenizer(sample['text'])

        tokenized_input = self.dataset.map(tokenize, batched = True, num_proc = 4,
    remove_columns = ['text', 'created_timestamp', 'downloaded_timestamp', 'url'])
        print(tokenized_input)
        def group_texts(samples):

            examples = {k: sum(samples[k], []) for k in samples.keys()}
            total_length = len(examples[list(examples.keys())[0]])

            if total_length >= self.text_len:
                total_length = (total_length // self.text_len) * self.text_len

            return {
                    k : [t[ i: i + self.text_len] for i in range(0, total_length,
    self.text_len)] for k, t in examples.items()
                   }

        mlm_dataset = tokenized_input.map(group_texts, batched = True, num_proc = 4)

        return mlm_dataset

# Example usage:
if __name__ == "__main__":
    import numpy as np

    # Create dataset
```

```python
    #train_dataset = MistralMLMDataset(AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased"))
    train_dataset = DocumentClassificationDataset(AutoTokenizer.from_pretrained("allenai/longformer-base-4096"), cases_path = '../../all_cases_clearinghouse.pkl')
    #val_dataset = DocumentClassificationDataset(AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased"), cases_path = '../../all_cases_clearinghouse.pkl')

    #data_collator = DataCollatorForLanguageModeling(tokenizer = train_dataset.tokenizer, mlm_probability = 0.1)
    # DataLoader for batching and shuffling
    mlm_train = train_dataset.prepare_corpus()
    #mlm_val = val_dataset.prepare_corpus()

    #dataloader = torch.utils.data.DataLoader(mlm_train, batch_size=2, shuffle=False)
    # Iterate through the DataLoader
    #for batch_data in dataloader:
    #    #print(batch_data)
    #    quit()
```

```python
from transformers import AutoTokenizer, AutoModelForMaskedLM
from torch import nn
from transformers import RobertaConfig
from transformers import TrainingArguments, Trainer
from mistral_datasets import *

if __name__ == '__main__':
    model = AutoModelForMaskedLM.from_config(RobertaConfig())
    model.cuda()
    print(model)
    # Create dataset
    train_dataset = MistralMLMDataset(AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased"))
    val_dataset = MistralMLMDataset(AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased"), split = 'validation')

    data_collator = DataCollatorForLanguageModeling(tokenizer = train_dataset.tokenizer, mlm_probability = 0.1)
    # DataLoader for batching and shuffling
    mlm_train = train_dataset.prepare_corpus()
    mlm_val = val_dataset.prepare_corpus()

    training_args = TrainingArguments(
        output_dir = "runs",
        eval_strategy = "epoch",
        learning_rate=1e-5,
        num_train_epochs=3,
        weight_decay=0.01,
        report_to='tensorboard'
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=mlm_train,
        eval_dataset=mlm_val,
        data_collator=data_collator
    )
    trainer.train()
```

4 Documentation

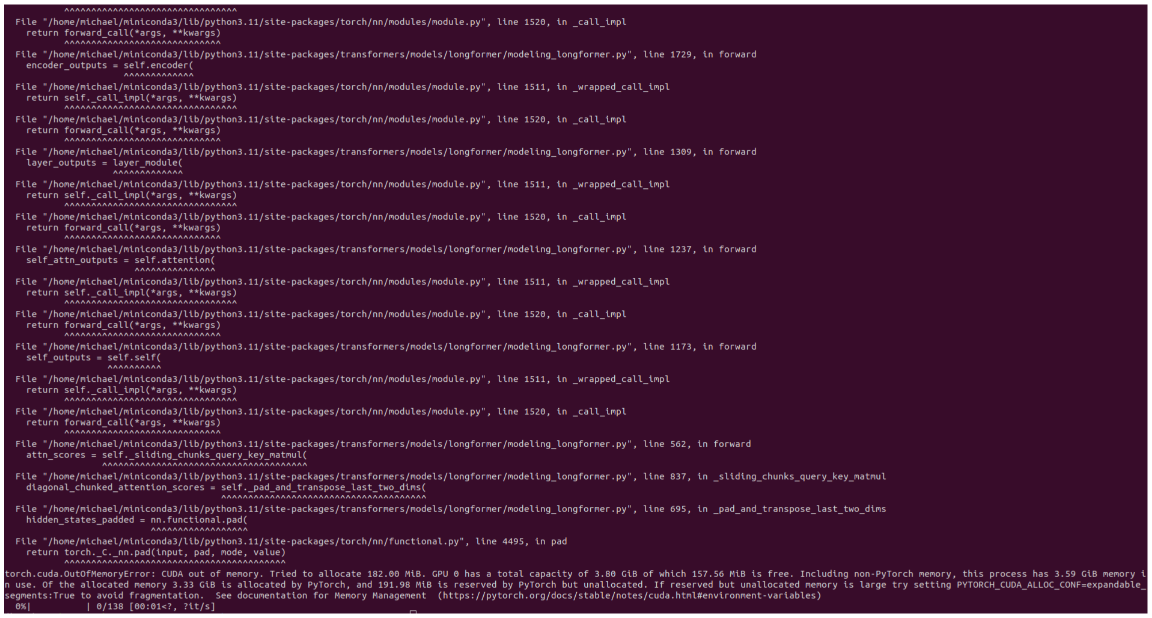
For the Masked Language Modeling pre-training, we first take in a sentence and we randomly remove some words. We record the words we removed and use those as the labels. The model must predict which words should fill in the blanks. After this, we will need to fine tune the model on text classification by freezing the model weights and removing the masked language modeling head. We replace the masked language modeling head with a classification head and we keep that head unfrozen. Then we train the new head on the text classification task.

5 Scription Validation(Optional)

The MLM takes significant time and must be run on the pace cluster. I don't have the pace cluster working yet, so I can't easily record a full run yet.

6 Results Visualization

The core oddity of my results is that there is a CUDA error. I beleive this is because my gpu only has 4 GB of RAM. I think I can fix this in a few ways. First is just using the PACE cluster, which is a good decision, but limits my ability to experiment quickly. Second, I may need more control of the model than hugging face provides. Huggingface doesn't really let you make your own models; you can only finetune existing architectures for new tasks. I believe using normal pytorch with minimal extensions will provide me the ability to better explore techniques because it affords me more control of the model. If we want to later, all the huggingface models are pytorch models, so we can bring that back later if applicable.



```
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1520, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 1729, in forward
    encoder_outputs = self.encoder(
                      ^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1511, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1520, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 1309, in forward
    layer_outputs = layer_module(
                    ^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1511, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1520, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 1237, in forward
    self_attn_outputs = self.attention(
                        ^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1511, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1520, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 1173, in forward
    self_outputs = self.self(
                   ^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1511, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1520, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 562, in forward
    attn_scores = self._sliding_chunks_query_key_matmul(
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 837, in _sliding_chunks_query_key_matmul
    diagonal_chunked_attention_scores = self._pad_and_transpose_last_two_dims(
                                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/transformers/models/longformer/modeling_longformer.py", line 695, in _pad_and_transpose_last_two_dims
    hidden_states_padded = nn.functional.pad(
                           ^^^^^^^^^^^^^^^^^^
  File "/home/michael/miniconda3/lib/python3.11/site-packages/torch/nn/functional.py", line 4495, in pad
    return torch._C._nn.pad(input, pad, mode, value)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 182.00 MiB. GPU 0 has a total capacity of 3.80 GiB of which 157.56 MiB is free. Including non-PyTorch memory, this process has 3.59 GiB memory i
n use. Of the allocated memory 3.33 GiB is allocated by PyTorch, and 191.98 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_
segments:True to avoid fragmentation.  See documentation for Memory Management  (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
  0%|          | 0/138 [00:01<?, ?it/s]
```

Figure 1: CUDA Error with HuggingFace

7 Next Week's proposal

- Continue working on TorchText Training pipeline. The hope here is I can have a basic model trained by next week and I can begin showing results here. Note all of these results will be on the clearinhouse dataset, not the covid dataset, but the same sort of fields are being identified. UPenn has their ”Allegation” field and Clearinghouse has an ”Issue” field, for example.

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 5

Víctor C. Fernández

September 2024

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Developed and implemented the OllamaModelProcessor class:
  - Created functions for direct text querying, file-based querying, and bulk querying of files in a folder.
  - Implemented support for multiple Ollama models including llama3.1, gemma2, mistral-nemo, qwen2, deepseek-coder-v2, phi3, and mixtral.
- Created a new version of the output JSON template based on Dr. Alexander's requirements for data retrieval.
- Generated a set of validation data to assess the accuracy and completeness of extracted information.
- Conducted initial manual spot-checks of the extracted data against original legal documents.
- Meeting with the NLP team on September 20th for our weekly meeting.
- Meeting with Dr. Alexander on September 20th for an update call.
- Organized the webinar events for Dr. Lindvall for the current and coming week.

What progress are you making next?

- Enhance prompt engineering techniques to improve data extraction accuracy and completeness.
- Begin development of an automated validation system:
  - Implement flexible string matching algorithms for comparing extracted data with the validation set.
  - Create a scoring system to quantify extraction accuracy for each field in the output template.
- Start development of visualization tools:
  - Select appropriate data visualization libraries based on project requirements.
  - Create initial prototypes for key visualizations such as case duration time-

lines and jurisdiction distribution charts.
- Conduct comprehensive benchmarking of different Ollama models to identify the most effective model(s) for our specific task.
- Meet with the NLP team on September 20th for our weekly meeting.

Is there anything blocking you from making progress?

No significant blockers at this time. However, we may need to consider computational resources for more extensive model benchmarking and potential fine-tuning in the future. (Larger models would require more storage, RAM and GPU)

2 ABSTRACTS

1. **Title:** Unveiling the Generalization Power of Fine-Tuned Large Language Models
   - **URL:** https://aclanthology.org/2024.naacl-long.51.pdf

   - **Abstract:** While Large Language Models (LLMs) have demonstrated exceptional multitasking abilities, fine-tuning these models on downstream, domain-specific datasets is often necessary to yield superior performance on test sets compared to their counterparts without fine-tuning. However, the comprehensive effects of fine-tuning on the LLMs' generalization ability are not fully understood.This paper delves into the differences between original, unmodified LLMs and their fine-tuned variants. Our primary investigation centers on whether fine-tuning affects the generalization ability intrinsic to LLMs. To elaborate on this, we conduct extensive experiments across five distinct language tasks on various datasets.Our main findings reveal that models fine-tuned on generation and classification tasks exhibit dissimilar behaviors in generalizing to different domains and tasks.Intriguingly, we observe that integrating the in-context learning strategy during fine-tuning on generation tasks can enhance the model's generalization ability.Through this systematic investigation, we aim to contribute valuable insights into the evolving landscape of fine-tuning practices for LLMs.

   - **Summary:** The paper investigates the generalization capabilities of fine-tuned Large Language Models (LLMs), specifically focusing on how fine-tuning affects their ability to perform well on new, unseen data and tasks. The study explores the differences between original, unmodified LLMs and their fine-tuned counterparts, aiming to understand whether fine-tuning enhances or hinders the inherent generalization ability of these models.

   - **Relevance:** It highlights the potential benefits and challenges of fine-tuning LLMs for specific tasks in the legal domain. The findings suggest that while fine-tuning can improve performance on in-domain and similar tasks, it can also lead to negative transfer and reduced generalization on out-of-domain and different tasks. The study also introduces a promising approach called Fine-Tuning with In-Context Learning (FTICL), which shows potential in improving the generalization ability of LLMs for generation tasks.

3 SCRIPTS AND CODE BLOCKS

All scripts have been uploaded to the HAAG NLP Repo.

The following functions are the core parts of the code I have been implementing this week.

First part consists of a class called OllamaModelProcessor, created with the intention of simplifying the process of using multiple models for generating outputs in order to benchmark which model generates better results out of the box. This class contains 3 functions for different prompting methods. One for directly querying text to the model, another one for querying using a file and the third one for bulk querying using all the files in a folder.

The second block of code consists of the code that will be generating new instances of the OllamaModelProcessor class, each of which will be using a different model. Then, for each new instance a defined prompt template file will be used to prompt the model, injecting the text from each file and requesting an output based on a template defined aside. This will then generate 22 files, corresponding to each of the sentencias case, containing the requested information from the files, created according to the model, along with metadata indicating which model was used to generate that file, the time it took to generate it and the hyperparameters used for that specific model.

```python
import json
import requests
import time
import os
import subprocess
from typing import Dict, Any, List

class OllamaModelProcessor:
    def __init__(self, model_name: str, **kwargs):
        self.model_name = model_name
        self.base_url = "http://localhost:11434/api/generate"
        self.kwargs = kwargs
        self.colors = {
            'GREEN':'\033[92m',
            'BLUE': '\033[94m',
            'WHITE': '\033[97m',
            'RED': '\033[91m',
            'YELLOW': '\033[93m',
            'RESET': '\033[0m'
        }

    # ANSI escape codes for colors
    def _print_color(self, text, color = 'WHITE'):
        print(f"{self.colors[color]}{text}{self.colors['RESET']}␣
        ↪   ")
```

Code 1—OllamaModelProcessor class to instantiate models and generate bulk outputs from Sentencias files

```
def _check_model_downloaded(self):
    # Check if the model is installed
    check_model_command = f"ollama list | grep {self.model_name}"
    result = subprocess.run(check_model_command, shell=True,
    ↪   capture_output=True, text=True)

    if self.model_name not in result.stdout:
        # If the model is not installed, download it
        self._print_color(f"Model {self.model_name} not found.
        ↪   Downloading...", color = 'YELLOW')

        try:
            pull_model_command = f"ollama pull {self.model_name}"
            subprocess.run(pull_model_command, check=True,
            ↪   shell=True)
        except subprocess.CalledProcessError as e:
            self._print_color(f"Error running command: {e}", color
            ↪   = 'RED')

def _delete_model(self):
    # Delete the model
    delete_model_command = f"ollama rm {self.model_name}"
    try:
        subprocess.run(delete_model_command, check=True,
        ↪   shell=True)
    except subprocess.CalledProcessError as e:
        self._print_color(f"Error running command: {e}", color =
        ↪   'RED')
```

```python
def _generate_response(self, prompt: str) -> Dict[str, Any]:
    start_time = time.time()
    response = requests.post(self.base_url, json={
        "model": self.model_name,
        "prompt": prompt,
        **self.kwargs
    })
    end_time = time.time()
    processing_time = end_time - start_time

    if response.status_code == 200:
        response_text = ""
        for line in response.text.strip().split('\n'):
            try:
                response_json = json.loads(line)
                if 'response' in response_json:
                    response_text += response_json['response']
            except json.JSONDecodeError:
                self._print_color(text = f"Error decoding JSON:
                ↪  {line}", color = 'RED')
        return {
            "response": response_text,
            "processing_time": processing_time
        }
    else:
        raise Exception(f"Error: {response.status_code} -
        ↪  {response.text}")

def _save_output(self, output: str, output_path: str):
    # Firsth check if the output folder exists, if not, create it
    output_folder = os.path.dirname(output_path)
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    with open(output_path, 'w', encoding='utf-8') as f:
        f.write(output)
```

```python
def _append_execution_details(self, output: str) -> str:
    execution_details = {
        "execution_details": {
            "model_name": self.model_name,
            "hyperparameters": self.kwargs,
            "processing_time": output["processing_time"],
            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")
        }
    }
    return output["response"] + "\n\n" +
    ↪   json.dumps(execution_details, indent=2)

def query_model(self, input_text: str, output_path: str = "",
↪   save_output: bool = False) -> str:
    self._check_model_downloaded()
    output = self._generate_response(input_text)
    final_output = self._append_execution_details(output)

    if save_output and output_path:
        self._save_output(final_output, output_path)

    return final_output
```

```python
def query_model_with_file(self, file_path: str,
↪   prompt_template_path: str, input_placeholder: str,
↪   output_format_path: str, output_placeholder: str,
                          output_path: str = "", save_output: bool
                          ↪   = False) -> str:

    with open(prompt_template_path, 'r', encoding='utf-8') as f:
        input_text = f.read()

    with open(file_path, 'r', encoding='utf-8') as f:
        file_content = f.read()

    with open(output_format_path, 'r', encoding='utf-8') as f:
        output_format = f.read()

    full_input = input_text.replace(input_placeholder,
    ↪   file_content)
    full_input = full_input.replace(output_placeholder,
    ↪   output_format)

    return self.query_model(full_input, output_path, save_output)
```

```
def query_model_bulk(self, prompt_template_path: str,
↪   input_placeholder: str, output_format_path: str,
↪   output_placeholder: str, folder_path: str,
                     output_folder: str, save_output: bool = True)
                     ↪   -> List[str]:

    for filename in os.listdir(folder_path):
        if filename.endswith(".txt"):
            # Print the name of the file being processed
            self._print_color(f"Processing file: {filename}",
            ↪   color = 'GREEN')

            # Define the path for the output file within the
            ↪   folder
            output_path = os.path.join(output_folder,
            ↪   f"output_{filename}") if save_output else ""

            # Query the model with the file content using key
            ↪   arguments
            self.query_model_with_file(file_path=os.path.join(fol⌋
            ↪   der_path,
            ↪   filename),
               prompt_template_path=prompt_template_path,
               input_placeholder=input_placeholder,
               output_format_path=output_format_path,
               output_placeholder=output_placeholder,
               output_path=output_path,
               save_output=save_output)
```

```
from ollama_model_processor import OllamaModelProcessor
import os

def generate_output(ollama_models: list, output_folder: str,
↪   model_hyperparameters: dict = {}):
    for model in ollama_models:
        processor = OllamaModelProcessor(model,
        ↪   **model_hyperparameters)
        sentencias_path = "../../sentencias_txt"

        prompt_template_path =
        ↪   "../../model_input_templates/input_text_v1.txt"
        input_placeholder = "{{DOCUMENT_CONTENT}}"
        output_placeholder = "{{MODEL_OUTPUT_FORMAT}}"

        output_template_folder = "../../model_output_templates"

        version_number = 1
        for filename in os.listdir(output_template_folder):
            if filename.endswith(".txt"):
                output_template_file_name =
                ↪   f"output_json_v{version_number}.txt"
                output_file_template_path =
                ↪   os.path.join(output_template_folder,
                ↪   output_template_file_name)
                output_folder_path =
                ↪   f"{output_folder}/{model}/v{version_number}"
                processor.query_model_bulk(prompt_template_path=p⌋
                ↪   rompt_template_path,
                        input_placeholder=input_placeholder,
                        output_format_path=output_file_template_p⌋
                        ↪   ath,
                        output_placeholder=output_placeholder,
                        folder_path=sentencias_path,
                        output_folder=output_folder_path,
                        save_output=True)
                version_number += 1
```

Code 2—Function for processing all documents with multiple models to benchmark the results

```
generate_output(
    ollama_models=[
        "llama3.1",
        "gemma2",
        "mistral-nemo",
        "qwen2",
        "deepseek-coder-v2",
        "phi3",
        "mixtral"
        ],
    output_folder="../../model_output",
    model_hyperparameters={"temperature": 0.7, "top_p": 0.9}
        )
```

Code 3—Code execution using multiple models with Ollama

```
{
"informacion general": {
                "numero de ordenanza": "",
                "numero unico de caso (NUC)": "",
                "tipo de caso": "",
                "jurisdiccion": ""
        },
        "partes involucradas": {
                "demandante": "",
                "demandado": "",
                "intervinientes": []
        },
        "fechas clave": {
                "fecha de presentacion demanda (DD/MM/YYYY)": "",
                "fecha de notificacion demanda (DD/MM/YYYY)": "",
                "fecha de audiencias (DD/MM/YYYY)": [],
                "fecha de fallo reservado (DD/MM/YYYY)": "",
                "fecha de lectura de sentencia (DD/MM/YYYY)": ""
        },
        "detalles del proceso": {
                "numero total audiencias": 0,
                "hubo defecto": false,
                "hubo intervencion voluntaria": false
        },
        ...
}
```

Code 4—Updated JSON template preview (access repository to view full detail)

4 DOCUMENTATION

1. **Data Collection and Preprocessing:**
   - Judicial decisions (sentencias) in PDF and DOC format were obtained from Dr. Alexander, originating from the National School of the Judiciary in the Dominican Republic.

- Text was extracted from PDF and DOC files using the PyMuPDF library and converted to TXT format.
- The extracted text was processed to remove headers, footers, pagination, and other repetitive elements in the corpus.
- Clean text files were generated with the processed content.

2. **Text Analysis and Feature Extraction:**
   - Instead of identifying named entities, a new approach using LLM models was adopted to directly identify specific content in the cleaned text files.
   - LLM models such as LLama3.1 were run locally using GPT4All and Ollama.
   - A template file was created to verify correct installation and use of the Ollama model.
   - An output JSON template was developed based on Dr. Alexander's requirements for data retrieval.
   - A process was implemented to bulk extract data from all given "sentencias" and store results in JSON format according to the template, including benchmark information such as processing time and token count.

3. **Model Implementation:**
   - An OllamaModelProcessor class was created to simplify the process of using multiple models for generating outputs and benchmarking.
   - The class includes functions for direct text querying, file-based querying, and bulk querying of files in a folder.
   - A generate_output function was implemented to process all documents with multiple models for benchmarking results.
   - Various Ollama models were utilized, including llama3.1, gemma2, mistral-nemo, qwen2, deepseek-coder-v2, phi3, and mixtral.

5 SCRIPT VALIDATION

A set of validation data has been generated to assess the accuracy and completeness of the information extracted by the LLM models. However, the validation process is still in progress due to the nature of the data and the required flexibility in response interpretation.

Key points:

- Validation data set has been prepared based on the output JSON template.
- The validation process aims to check the accuracy of extracted information rather than entity recognition.
- Code for automated validation is pending development, as some fields require string-based comparison with a degree of flexibility.
- Manual spot-checks have been performed to gauge initial performance, but a comprehensive validation is yet to be implemented.

Next steps for validation:

- Develop a flexible string matching algorithm to compare extracted data with the validation set.
- Implement a scoring system to quantify the accuracy of extracted information for each field in the output template.
- Create a validation report generator to summarize the performance of different models and configurations.
- Establish thresholds for acceptable accuracy levels for each type of extracted information.

6 PROOF OF WORK

The implemented system demonstrates significant progress in developing a framework for analyzing legal documents using large language models. Key achievements and current status include:

- **Data Processing Pipeline:** Successfully implemented a pipeline for extracting text from PDF and DOC files, cleaning the data, and preparing it for analysis.
- **LLM Integration:** Developed the OllamaModelProcessor class, enabling the use of multiple Ollama-based language models for text analysis.
- **Bulk Processing:** Implemented functionality for bulk processing of legal documents, generating individual JSON files for each document containing extracted key information.
- **Model Versatility:** The system can utilize multiple Ollama models (llama3.1, gemma2, mistral-nemo, qwen2, deepseek-coder-v2, phi3, mixtral), allowing for comprehensive benchmarking and comparison of model performance.
- **Output Generation:** Successfully generating structured JSON outputs based on the predefined template, capturing key information from legal documents.
- **Benchmark Information:** Including processing time and token count for each

document analysis, providing insights into model efficiency.

- **Validation Data:** Generated a set of validation data to assess the accuracy and completeness of extracted information, laying the groundwork for future automated validation.

Challenges and ongoing work:

- **Accuracy Refinement:** While initial results show improvement over previous methods, further prompt engineering and model fine-tuning are needed to enhance data extraction accuracy.
- **Validation Process:** Development of an automated validation system is pending, requiring the implementation of flexible string matching algorithms to handle variations in text-based responses.
- **Visualization Tools:** Current result analysis relies on manual inspection of JSON outputs. Development of comprehensive visualization tools is planned but not yet implemented.

Next steps:

1. Enhance prompt engineering techniques to improve data extraction accuracy and completeness.
2. Develop and implement the automated validation system, including flexible string matching and a scoring mechanism for assessing extraction accuracy.
3. Create visualization tools and an interactive dashboard for easier analysis and presentation of extracted data and model performance metrics.
4. Conduct a comprehensive benchmarking of different Ollama models to identify the most effective model(s) for this specific legal document analysis task.
5. Explore potential integration of other NLP techniques to complement the LLM-based approach and further improve information extraction capabilities.
6. Possibility to develop a user-friendly interface for interacting with the system, allowing for easy document upload, model selection, and result visualization.

This proof of work demonstrates substantial progress in developing a robust system for legal document analysis using state-of-the-art language models. The foundation laid sets the stage for more advanced analysis, improved accuracy, and comprehensive insights in the coming phases of the project. The combination of bulk processing capabilities, multi-model support, and the groundwork for validation and visualization positions this project for significant advancements in legal document processing and analysis.

Week 5 | HAAG - NLP | Fall 2024

Alejandro Gomez

September 20th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- I worked on a script for the part of the pipeline that will take a JSON template and fill out a Microsoft Word docx file. The Lexia Abogados team had requested for a step in the pipeline that could funnel the data summarized from the model into a Word document. So I setup a Word document that takes in variables from a mock JSON output file and insersts them into the Word doc to have a summarized cover page on the sentencias.
- I worked on sandboxing different models on a PACE cluster that had access to GPU. Last week I had run Llama3.1:8b on my local machine without a GPU and it was very slow and I was having issues with the context window and processing documents using a GUI (Open WebUI). By setting up a High Performance Computing (HPC) node I was able to run models locally and see instant changes in response speed from the LLM's. A response that would take a few minutes on my machine locally could be delivered by the HPC node in less than 10 seconds. Using my team's scripts I was able to learn how to use the PACE cluster and understand how this can affect the architecture decisions for our ML pipeline. My team's scripts had time metrics for benchmarking, but I can say from using many of the available LLM's that I feel mistral-nemo could be a good fit when compared to other small/mid-size models as it was very fast to response and generally had the most response fields filled out. I also learned the PACE is more of a sandboxing platform as I understand and not a long-running machine to serve our model so although we can test/prototype on there, we need to consider other cloud resources and their respective costs.

1.2 What are you planning on working on next?

- My teammates and I had honed in on the decision to leverage LLM's and we have many scripts working independently at this point. I think it would be good to start designing an API with a simple GUI that can take a sentencias PDF through the ML pipeline and return the docx cover page. This would require some collaboration to design this system, some refactoring, and developing out new features.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

GPT is receiving increasing attention and has a variety of application scenarios in clinical practice. In clinical decision support, ChatGPT has been used to generate accurate differential diagnosis lists, support clinical decision-making, optimize clinical decision support, and provide insights for cancer screening decisions. In addition, ChatGPT has been used for intelligent question-answering to provide reliable information about diseases and medical queries. In terms of medical documentation, ChatGPT has proven effective in generating patient clinical letters, radiology reports, medical notes, and discharge summaries, improving efficiency and accuracy for health care providers. Future research directions include real-time monitoring and predictive analytics, precision medicine and personalized treatment, the role of ChatGPT in telemedicine and remote health care, and integration with existing health care systems. Overall, ChatGPT is a valuable tool that complements the expertise of health care providers and improves clinical decision-making and patient care. However, ChatGPT is a double-edged sword. We need to carefully consider and study the benefits and potential dangers of ChatGPT. In this viewpoint, we discuss recent advances in ChatGPT research in clinical practice and suggest possible risks and challenges of using ChatGPT in clinical practice. It will help guide and support future artificial intelligence research similar to ChatGPT in health doi[Liu23]

2.2 Summary

This article felt extremely relevant to the ongoing work by the NLP team because in the same way the researchers tested LLM's to generate patient letters, reports, notes, and summaries, the NLP DR team is working on developing a pipeline to summarize specific information from sentencias. The paper also brings up an important point:

"To ensure the safe and reliable use of ChatGPT, a rigorous human review process and human involvement in the workflow are essential. Adherence to relevant standards and criteria, such as accuracy, reliability, interpretability, explainability, and user acceptance benchmarks, is necessary...Security measures must be taken to safeguard patient information while using ChatGPT, including encryption, access control, secure data storage, and compliance with privacy regulations."[Liu23]

These QA and security considerations are important for the NLP team as well because we are working on a judicial system where mistakes by LLM's that are accepted into the system can cause massive entropy in the affected's life and further legal hurdles. We also work with confidential personal identification data that we need to be mindful of for the pipeline we're building. One consideration we took was making our codebase private to protect this information.

3 Scripts and Code Blocks

3.1 Code

```json
{
  "{{informacion_general.numero_de_ordenanza}}": "",
  "{{informacion_general.numero_unico_caso}}": "",
  "{{informacion_general.tipo_caso}}": "",
  "{{informacion_general.jurisdiccion}}": "",
  "{{partes_involucradas.demandante}}": "",
  "{{partes_involucradas.demandado}}": "",
  "{{partes_involucradas.intervinientes}}": [],
  "{{fechas_clave.presentacion_demanda}}": "",
  "{{fechas_clave.notificacion_demanda}}": "",
  "{{fechas_clave.audiencias}}": [],
  "{{fechas_clave.fallo_reservado}}": "",
  "{{fechas_clave.lectura_sentencia}}": "",
  "{{detalles_proceso.total_audiencias}}": 0,
  "{{detalles_proceso.hubo_defecto}}": false,
  "{{detalles_proceso.hubo_intervencion_voluntaria}}": false,
  "{{resultado.decision}}": "",
  "{{resultado.levantamiento_ordenado}}": false,
  "{{resultado.ejecucion_provisional}}": false,
  "{{tiempos_proceso.dias_presentacion_primera_audiencia}}": 0,
  "{{tiempos_proceso.dias_ultima_audiencia_lectura}}": 0,
  "{{tiempos_proceso.duracion_total_dias}}": 0,
  "{{informacion_adicional.monto_disputa}}": "",
  "{{informacion_adicional.cantidad_documentos_prueba}}": 0,
  "{{metadata.juez_presidente}}": "",
  "{{metadata.secretario}}": "",
  "{{metadata.palabras_clave}}": []
}
```

Listing 1: flattened json for sentencia summary

```python
# methods not shown - visit github repo to see implementations

def main():
    # manage inputs and outputs
    template_file_path: str = "nueva_plantilla.docx"
    tmp_dir: str = "./tmp/"
    data_file_path: str = os.path.join(tmp_dir, os.listdir(tmp_dir)[0])  # takes first
     file in tmp dir assuming the API in the pipeline will upload here
    data_file_name: str = Path(data_file_path).stem
    output_file_path: str = os.path.join(tmp_dir, f"{data_file_name}_resumen_de_datos.
    docx")

    new_doc = Document(template_file_path)
    paragraphs_in_doc = new_doc.paragraphs

    # load json data in to python dict
    try:
        data: Dict[str, Any] = load_json_file(data_file_path)
        logging.info(f"Success! Loaded JSON file.")
    except Exception as e:
        logging.error(f"Error loading JSON data: {e}")
        return

    # replace placeholders keys in the document
    try:
        replace_text_in_paragraph(paragraphs_in_doc, data)
        logging.info(f"Success! added input into template")
    except Exception as e:
        logging.error(f"Error replacing text in document: {e}")
        return

    # saves the new doc
    try:
        new_doc.save(output_file_path)
        logging.info(f"New document with data inputs created and saved as {
    output_file_path}")
    except Exception as e:
        logging.error(f"Error saving the document: {e}")

    ######################################################
    ## debugging by printing the new doc to terminal     #
    ######################################################
    # dd = Document(output_file_path)
    # for p in dd.paragraphs:
    #     print(p.text)
    ##################
    ## end debugging #
    ##################

    # converts to pdf
    try:
        new_pdf_path: str = convert_to_pdf(tmp_dir, output_file_path, new_doc)
        logging.info(f"new pdf created as {new_pdf_path}")
    except Exception as e:
        logging.error(f"Error converting the document: {e}")


    # can be used to remove json data if file was only added temporarily
    # os.remove(f'./tmp/{data_file_path}')

if __name__ == "__main__":
    main()
```

Listing 2: main method for processing JSON into cover page

3.2 Documentation

Running the program to generate the word doc

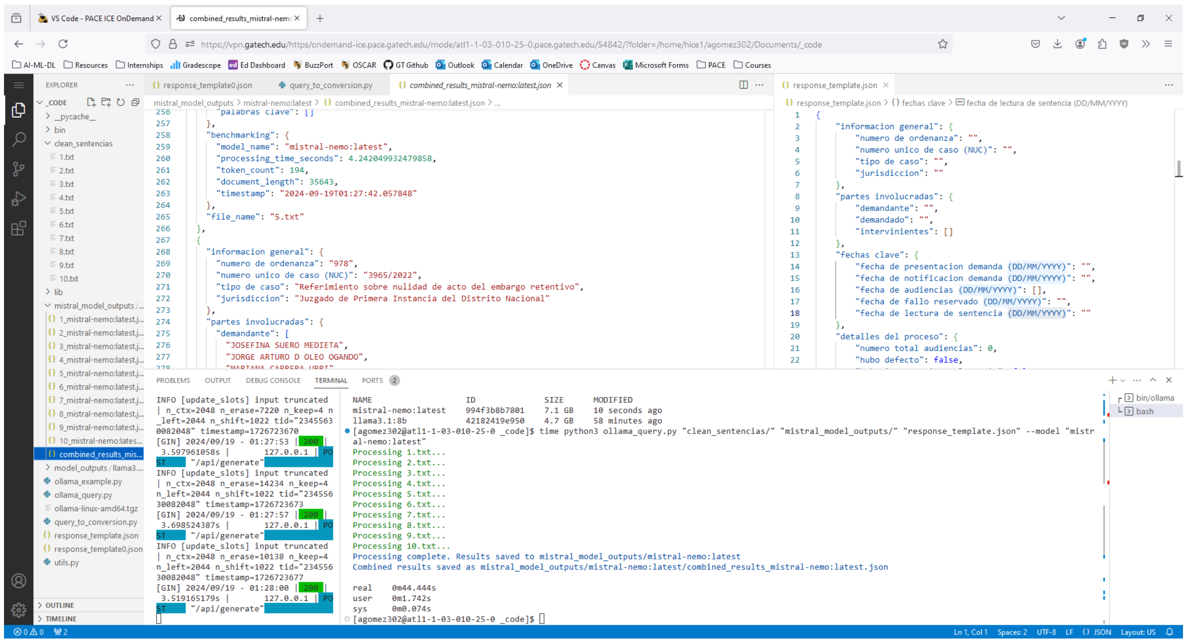
```
conda activate nlp_env

conda env export --no-builds > current_environment.yml

conda env update --name nlp_env --file environment.yml --prune

python generate_plantilla.py

python generate_sentencia_summary.py


cd tmp && explorer.exe sentencia_504-2022-SORD-0529_resumen_de_datos.docx # opens the
    new word doc from WSL2 onto windows host machine
```
Listing 3: commands

Running Ollama and LLM models on PACE

```
conda activate nlp_env

curl -fsSL https://ollama.com/install.sh | sh # had to modify this as I didn't have
    root access on PACE

ollama serve # typically would be ollama run but bc of the non-root download method, I
     had to use this in one terminal and let it run

ollama pull llama3.1:8b

ollama pull llama3.1:70b

ollama pull llama3.1:405b

ollama pull mistral-nemo

# etc...

ollama run mistral-nemo # for prompting onthe command line

python3 query_model_main.py # this was my teammates script with benchmarking metrics
```
Listing 4: commands

3.3 Script Validation (optional)

Working in PACE


VS Code - PACE ICE OnDemand
combined_results_mistral-nemo
https://vpn.gatech.edu/https/ondemand-ice.pace.gatech.edu/rnode/atl1-1-03-010-25-0.pace.gatech.edu/54842/?folder=/home/hice1/agomez302/Documents/_code
response_template0.json
query_to_conversion.py
combined_results_mistral-nemo:latest.json
response_template.json
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2
Ln 1, Col 1 Spaces: 2 UTF-8 LF {} JSON Layout: US

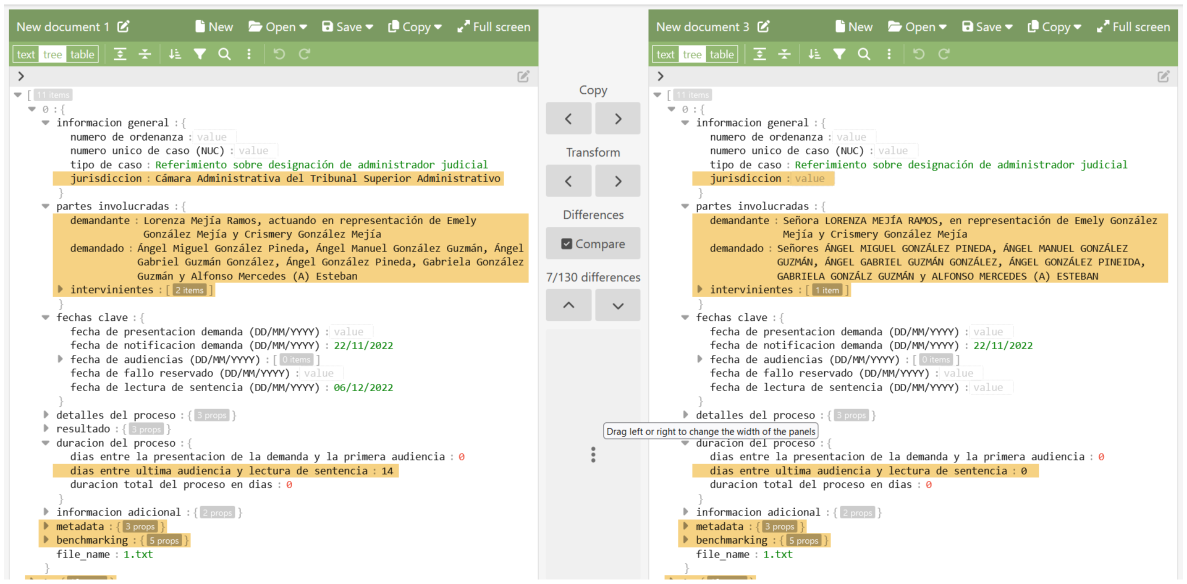
```
    "palabras clave": []
  },
  "benchmarking": {
    "model_name": "mistral-nemo:latest",
    "processing_time_seconds": 4.242049932479858,
    "token_count": 194,
    "document_length": 35643,
    "timestamp": "2024-09-19T01:27:42.057848"
  },
  "file_name": "5.txt"
},
{
  "informacion general": {
    "numero de ordenanza": "978",
    "numero unico de caso (NUC)": "3965/2022",
    "tipo de caso": "Referimiento sobre nulidad de acto del embargo retentivo",
    "jurisdiccion": "Juzgado de Primera Instancia del Distrito Nacional"
  },
  "partes involucradas": {
    "demandante": [
      "JOSEFINA SUERO MEDIETA",
      "JORGE ARTURO D OLEO OGANDO",
```

```
{
    "informacion general": {
        "numero de ordenanza": "",
        "numero unico de caso (NUC)": "",
        "tipo de caso": "",
        "jurisdiccion": ""
    },
    "partes involucradas": {
        "demandante": "",
        "demandado": "",
        "intervinientes": []
    },
    "fechas clave": {
        "fecha de presentacion demanda (DD/MM/YYYY)": "",
        "fecha de notificacion demanda (DD/MM/YYYY)": "",
        "fecha de audiencias (DD/MM/YYYY)": [],
        "fecha de fallo reservado (DD/MM/YYYY)": "",
        "fecha de lectura de sentencia (DD/MM/YYYY)": ""
    },
    "detalles del proceso": {
        "numero total audiencias": 0,
        "hubo defecto": false,
```

```
INFO [update_slots] input truncated
| n_ctx=2048 n_erase=7220 n_keep=4 n
_left=2044 n_shift=1022 tid="2345563
0082048" timestamp=1726723670
[GIN] 2024/09/19 - 01:27:53 | 200 |
 3.597961058s |       127.0.0.1 | PO
ST     "/api/generate"
INFO [update_slots] input truncated
| n_ctx=2048 n_erase=14234 n_keep=4
n_left=2044 n_shift=1022 tid="234556
30082048" timestamp=1726723673
[GIN] 2024/09/19 - 01:27:57 | 200 |
 3.698524387s |       127.0.0.1 | PO
ST     "/api/generate"
INFO [update_slots] input truncated
| n_ctx=2048 n_erase=10138 n_keep=4
n_left=2044 n_shift=1022 tid="234556
30082048" timestamp=1726723677
[GIN] 2024/09/19 - 01:28:00 | 200 |
 3.519165179s |       127.0.0.1 | PO
ST     "/api/generate"
```

```
NAME                  ID              SIZE      MODIFIED
mistral-nemo:latest   994f3b8b7801    7.1 GB    10 seconds ago
llama3.1:8b           42182419e950    4.7 GB    58 minutes ago
[agomez302@atl1-1-03-010-25-0 _code]$ time python3 ollama_query.py "clean_sentencias/" "mistral_model_outputs/" "response_template.json" --model "mistr
al-nemo:latest"
Processing 1.txt...
Processing 2.txt...
Processing 3.txt...
Processing 4.txt...
Processing 5.txt...
Processing 6.txt...
Processing 7.txt...
Processing 8.txt...
Processing 9.txt...
Processing 10.txt...
Processing complete. Results saved to mistral_model_outputs/mistral-nemo:latest
Combined results saved as mistral_model_outputs/mistral-nemo:latest/combined_results_mistral-nemo:latest.json

real    0m44.444s
user    0m1.742s
sys     0m0.074s
[agomez302@atl1-1-03-010-25-0 _code]$
```

Figure 1: PACE VM



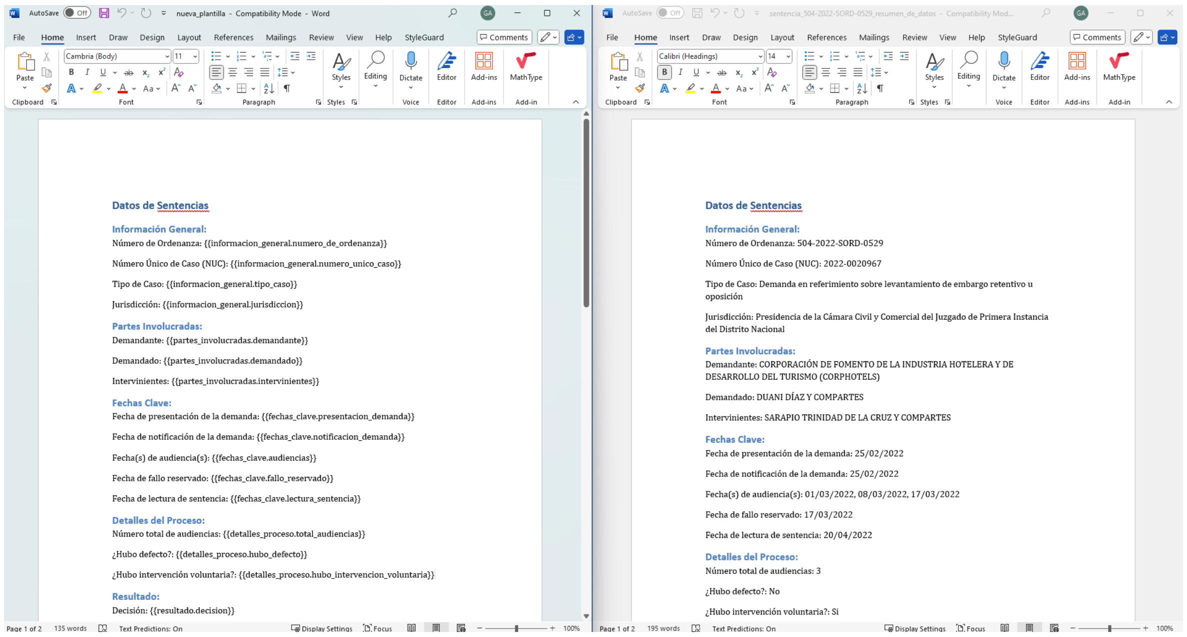
```
[ 11 items
  0 : {
    informacion general : {
      numero de ordenanza : value
      numero unico de caso (NUC) : value
      tipo de caso : Referimiento sobre designación de administrador judicial
      jurisdiccion : Cámara Administrativa del Tribunal Superior Administrativo
    }
    partes involucradas : {
      demandante : Lorenza Mejía Ramos, actuando en representación de Emely
                   González Mejía y Crismery González Mejía
      demandado : Ángel Miguel González Pineda, Ángel Manuel González Guzmán, Ángel
                  Gabriel Guzmán González, Ángel González Pineda, Gabriela González
                  Guzmán y Alfonso Mercedes (A) Esteban
      intervinientes : [ 2 items ]
    }
    fechas clave : {
      fecha de presentacion demanda (DD/MM/YYYY) : value
      fecha de notificacion demanda (DD/MM/YYYY) : 22/11/2022
      fecha de audiencias (DD/MM/YYYY) : [ 0 items ]
      fecha de fallo reservado (DD/MM/YYYY) : value
      fecha de lectura de sentencia (DD/MM/YYYY) : 06/12/2022
    }
    detalles del proceso : { 3 props }
    resultado : { 3 props }
    duracion del proceso : {
      dias entre la presentacion de la demanda y la primera audiencia : 0
      dias entre ultima audiencia y lectura de sentencia : 14
      duracion total del proceso en dias : 0
    }
    informacion adicional : { 2 props }
    metadata : { 3 props }
    benchmarking : { 5 props }
    file_name : 1.txt
  }
```

```
[ 11 items
  0 : {
    informacion general : {
      numero de ordenanza : value
      numero unico de caso (NUC) : value
      tipo de caso : Referimiento sobre designación de administrador judicial
      jurisdiccion : value
    }
    partes involucradas : {
      demandante : Señora LORENZA MEJÍA RAMOS, en representación de Emely González
                   Mejía y Crismery González Mejía
      demandado : Señores ÁNGEL MIGUEL GONZÁLEZ PINEDA, ÁNGEL MANUEL GONZÁLEZ
                  GUZMÁN, ÁNGEL GABRIEL GUZMÁN GONZÁLEZ, ÁNGEL GONZÁLEZ PINEIDA,
                  GABRIELA GONZÁLZ GUZMÁN y ALFONSO MERCEDES (A) ESTEBAN
      intervinientes : [ 1 item ]
    }
    fechas clave : {
      fecha de presentacion demanda (DD/MM/YYYY) : value
      fecha de notificacion demanda (DD/MM/YYYY) : 22/11/2022
      fecha de audiencias (DD/MM/YYYY) : [ 0 items ]
      fecha de fallo reservado (DD/MM/YYYY) : value
      fecha de lectura de sentencia (DD/MM/YYYY) : value
    }
    detalles del proceso : { 3 props }
    duracion del proceso : {
      dias entre la presentacion de la demanda y la primera audiencia : 0
      dias entre ultima audiencia y lectura de sentencia : 0
      duracion total del proceso en dias : 0
    }
    informacion adicional : { 2 props }
    metadata : { 3 props }
    benchmarking : { 5 props }
    file_name : 1.txt
  }
```

Figure 2: Comparing mistral-nemo vs. llama3.1:8b sentencia summary

3.4 Results Visualization

Working with word docs


nueva_plantilla - Compatibility Mode - Word
sentencia_504-2022-SORD-0529_resumen_de_datos - Compatibility Mod...

Datos de Sentencias

Información General:
Número de Ordenanza: {{informacion_general.numero_de_ordenanza}}

Número Único de Caso (NUC): {{informacion_general.numero_unico_caso}}

Tipo de Caso: {{informacion_general.tipo_caso}}

Jurisdicción: {{informacion_general.jurisdiccion}}

Partes Involucradas:
Demandante: {{partes_involucradas.demandante}}

Demandado: {{partes_involucradas.demandado}}

Intervinientes: {{partes_involucradas.intervinientes}}

Fechas Clave:
Fecha de presentación de la demanda: {{fechas_clave.presentacion_demanda}}

Fecha de notificación de la demanda: {{fechas_clave.notificacion_demanda}}

Fecha(s) de audiencia(s): {{fechas_clave.audiencias}}

Fecha de fallo reservado: {{fechas_clave.fallo_reservado}}

Fecha de lectura de sentencia: {{fechas_clave.lectura_sentencia}}

Detalles del Proceso:
Número total de audiencias: {{detalles_proceso.total_audiencias}}

¿Hubo defecto?: {{detalles_proceso.hubo_defecto}}

¿Hubo intervención voluntaria?: {{detalles_proceso.hubo_intervencion_voluntaria}}

Resultado:
Decisión: {{resultado.decision}}

Datos de Sentencias

Información General:
Número de Ordenanza: 504-2022-SORD-0529

Número Único de Caso (NUC): 2022-0020967

Tipo de Caso: Demanda en referimiento sobre levantamiento de embargo retentivo u oposición

Jurisdicción: Presidencia de la Cámara Civil y Comercial del Juzgado de Primera Instancia del Distrito Nacional

Partes Involucradas:
Demandante: CORPORACIÓN DE FOMENTO DE LA INDUSTRIA HOTELERA Y DE DESARROLLO DEL TURISMO (CORPHOTELS)

Demandado: DUANI DÍAZ Y COMPARTES

Intervinientes: SARAPIO TRINIDAD DE LA CRUZ Y COMPARTES

Fechas Clave:
Fecha de presentación de la demanda: 25/02/2022

Fecha de notificación de la demanda: 25/02/2022

Fecha(s) de audiencia(s): 01/03/2022, 08/03/2022, 17/03/2022

Fecha de fallo reservado: 17/03/2022

Fecha de lectura de sentencia: 20/04/2022

Detalles del Proceso:
Número total de audiencias: 3

¿Hubo defecto?: No

¿Hubo intervención voluntaria?: Si

Figure 3: ML pipeline step that fills summary page from Word doc variables

3.5 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- Start working toward a cohesive ML pipeline that can meet the acceptance criteria for the NLP team's first deliverable: extracting specific information from the sentencias to be used as a cover page for previous sentencias.
- Explore architectural designs and resources for the model deployments - this can help us understand the resources we should be constrained to when prototyping (e.g. llama3.1 :8b vs. llama3.1:405b have vastly different system requirements).
- As usual: update slide to share my material with my team and update the NLP group website with current records

References

[Liu23] Jialin Liu. Utility of chatgpt in clinical practice. *Journal of Medical Internet Research*, 25(1):e48568–e48568, 2023.

HAAG NLP Sentencias — Week 5 Report
NLP-Gen Team

Karol Gutierrez

September 20, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Performance experiments with Llama 3 using local setup. I experimented with prompting engineering through iterations and feedback. I generated several output files and compared performance vs. real fields extracted from the sentencias.
- Using the experiments, I generated several instances of data containing the fields that Dr. Alexander requested for the next milestone.
- I also coded an analogous version of the GPT4ALL functionality in Python, however, it doesn't have the embedding capabilities to load the documents.
- I did research and experimentation in embeddings models in Spanish but need to check how to add it to my implementation using existing Llama model.
- Literature review on LLMs and embeddings.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Attended seminar for pathways to PhD.

1.2 What are you planning on working on next?

- Complete implementation with embeddings for Spanish language, and integrate it in the existing workflow that uses Llama 3.
- Further literature review on embeddings and applications of them.
- Sync with team on how to present results to stakeholders.
- Design pipeline to iterate on specific fields that are harder to extract.
- 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, but as mentioned by Dr. Alexander, it would be helpful to have access to some experts on NLP for advice or general office hours.

2 Literature Review

Paper: Clinical Flair: A Pre-Trained Language Model for Spanish Clinical Natural Language Processing [RDV22].

2.1 Abstract

Word embeddings have been widely used in Natural Language Processing (NLP) tasks. Although these representations can capture the semantic information of words, they cannot learn the sequence-level semantics. This problem can be handled using contextual word embeddings derived from pre-trained language models, which have contributed to significant improvements in several NLP tasks. Further improvements are achieved when pre-training these models on domain-specific corpora. In this paper, we introduce Clinical Flair, a domain-specific language model trained on Spanish clinical narratives. To validate the quality of the contextual representations retrieved from our model, we tested them on four named entity recognition datasets belonging to the clinical and biomedical domains. Our experiments confirm that incorporating domain-specific embeddings into classical sequence labeling architectures improves model performance dramatically compared to general-domain embeddings, demonstrating the importance of having these resources available.

2.2 Summary

The paper discusses the development of a domain-specific language model for Spanish clinical narratives, designed to outperform general-purpose models in medical NLP tasks like Named Entity Recognition (NER). It incorporates several key enhancements to achieve this:

- Character-Level Language Model: Built on Flair, Clinical Flair captures word-level context through character embeddings, improving performance on medical text.
- Domain-Specific Fine-Tuning: The model is trained on Chilean clinical data to adapt to medical terminology.
- Performance Evaluation: Tested on four datasets, Clinical Flair outperforms other models on tasks involving clinical and biomedical data.

This model demonstrates the potential of fine-tuned, domain-specific NLP models for processing clinical narratives in Spanish, particularly in identifying key medical entities. However, future work is required to apply these techniques to broader NLP tasks and further enhance the model's robustness in clinical applications.

2.3 Relevance

This paper is highly relevant to our project because it directly talks about the need for domain-specific language models in Spanish, which is crucial for processing Spanish-language legal and clinical documents in NLP tasks. It also highlights the usage of embeddings. By using domain-specific embeddings, I can load and analyze these documents more effectively, extracting meaningful information such as named entities and dates. The emphasis on fine-tuning with clinical narratives provides a model to adapt it to my needs of extracting and structuring key data.

3 Scripts and code blocks

As previously mentioned, the existing code is in a private repository. Since we are handling private information from the PDF files, it was decided alongside Dr. Alexander that we should add all of our code work here from now on.

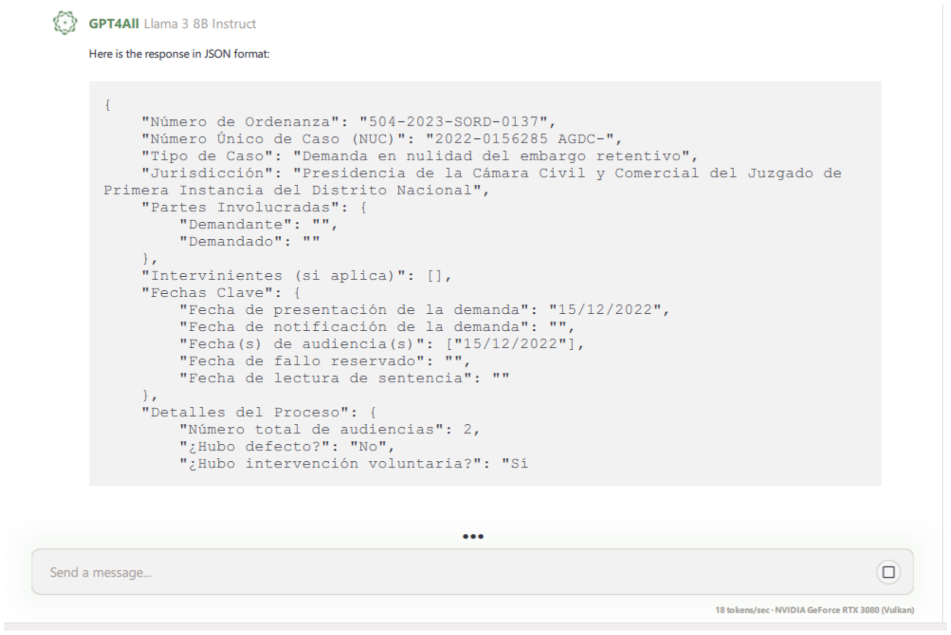
3.1 Code developed

- Sample code using `sentence-transformers/all-distilroberta-v1` as embedding. This allows us to extract context for a specific answer.
- Embed sentencias documents into Llama 3 model by using UI with `nomic-embed-text-v1.5` embedding.

- Experiments on Llama 3 to extract the requested fields from the documents and display them in JSON format. Different experimentation by providing feedback to LLM and iterative improvement.
- Analogous implementation in Python, to retrieve JSON components by chunking the text into blocks such that the prompt is below the limit of 2040 tokens.
- Compare results vs manual classification

By using GPT4All[AI24], I experimented with prompting engineering by using some of the following strategies:

- Sample code using `sentence-transformers/all-distilroberta-v1` as embedding. This allows us to extract context for a specific answer.
- Embed sentencias documents into Llama 3 model by using UI with `nomic-embed-text-v1.5` embedding.
- Give feedback to the LLM by mentioning that some attributes doesn't have the look of a valid asnwer, e.g. giving a placeholder instead of a name. This proved to be effective after several tries for some examples.
- Ask to display output in JSON readable format.


GPT4All Llama 3 8B Instruct
Send a message...
18 tokens/sec · NVIDIA GeForce RTX 3080 (Vulkan)

Here is the response in JSON format:

```
{
    "Número de Ordenanza": "504-2023-SORD-0137",
    "Número Único de Caso (NUC)": "2022-0156285 AGDC-",
    "Tipo de Caso": "Demanda en nulidad del embargo retentivo",
    "Jurisdicción": "Presidencia de la Cámara Civil y Comercial del Juzgado de
Primera Instancia del Distrito Nacional",
    "Partes Involucradas": {
        "Demandante": "",
        "Demandado": ""
    },
    "Intervinientes (si aplica)": [],
    "Fechas Clave": {
        "Fecha de presentación de la demanda": "15/12/2022",
        "Fecha de notificación de la demanda": "",
        "Fecha(s) de audiencia(s)": ["15/12/2022"],
        "Fecha de fallo reservado": "",
        "Fecha de lectura de sentencia": ""
    },
    "Detalles del Proceso": {
        "Número total de audiencias": 2,
        "¿Hubo defecto?": "No",
        "¿Hubo intervención voluntaria?": "Sí
```

Figure 1: Output from Llama 3

The current Python code is limited by the size of the prompt, then the process splits the original string and tries to fill the required fields as much as possible in each iteration. This workflow is explained in 2, a code block for this is found in 3.

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 the progress of this week, a new version of the `environment.yml` file was added, which include more of the libraries and dependencies.

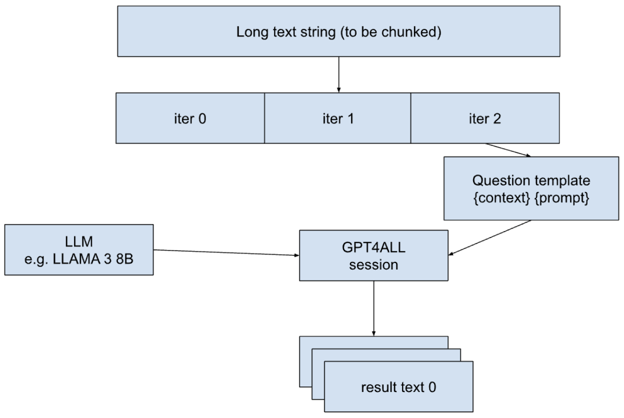


Figure 2: Code logic flow chart.

5 Script Validation

It doesn't apply at this point of the development of the project.

6 Results Visualization

The replies to the prompts can be seen in the next figures. First, Figure 4 show the replies using the visual interface. Figure 5 show the replies by calling the models and providing the context in Python.

7 Proof of Work

All the scripts work end to end from the starting PDF files as shown in the images from the visualization. The quality of the results is yet to be examined quantitatively it showed promising results that can be further improved.

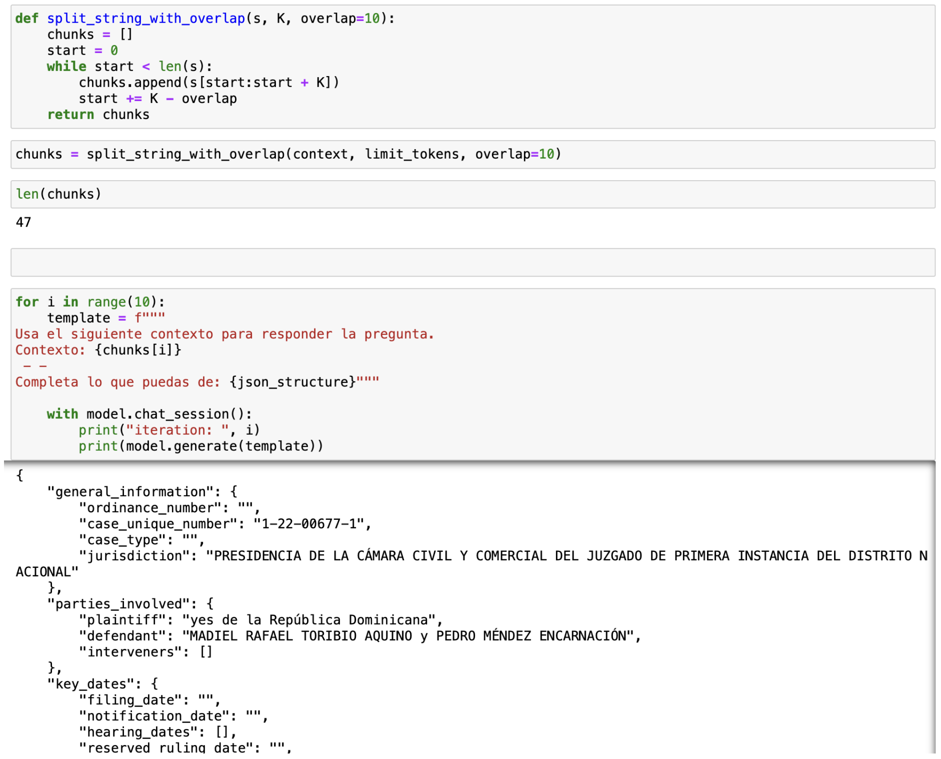
8 Next Week's Proposal

Refer to section 1.2 for details (avoid repetition).

References

[AI24] Nomic AI. Gpt4all. https://www.nomic.ai/gpt4all, 2024. Accessed: 2024-09-13.

[RDV22] Matías Rojas, Jocelyn Dunstan, and Fabián Villena. Clinical flair: A pre-trained language model for Spanish clinical natural language processing. In Tristan Naumann, Steven Bethard, Kirk Roberts, and Anna Rumshisky, editors, *Proceedings of the 4th Clinical Natural Language Processing Workshop*, pages 87–92, Seattle, WA, July 2022. Association for Computational Linguistics.



```
def split_string_with_overlap(s, K, overlap=10):
    chunks = []
    start = 0
    while start < len(s):
        chunks.append(s[start:start + K])
        start += K - overlap
    return chunks
```

```
chunks = split_string_with_overlap(context, limit_tokens, overlap=10)
```

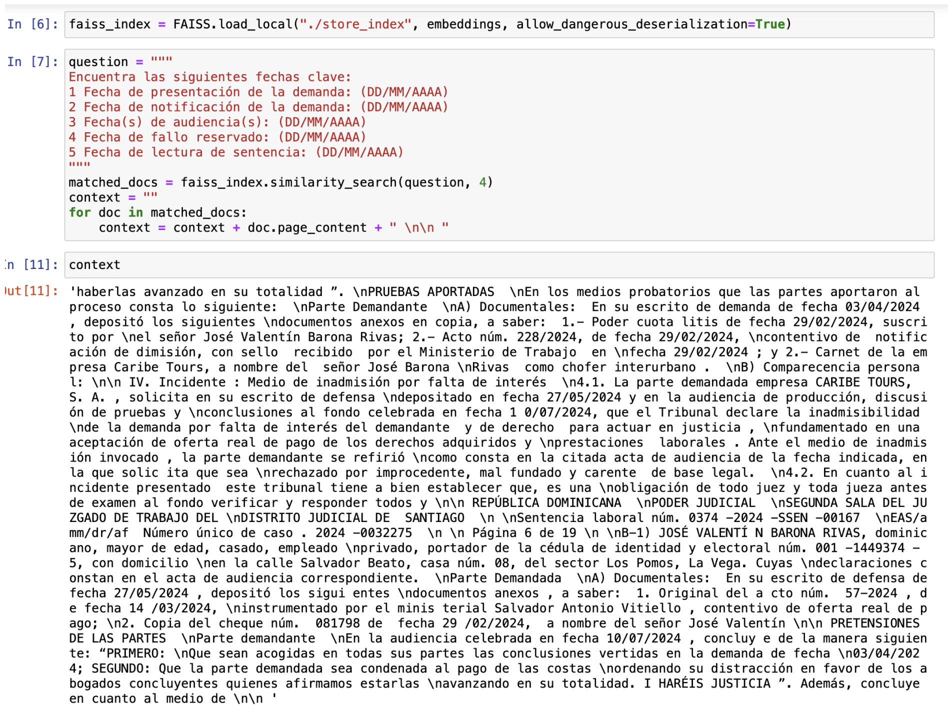
```
len(chunks)
```

```
47
```

```
for i in range(10):
    template = f"""
Usa el siguiente contexto para responder la pregunta.
Contexto: {chunks[i]}
 - -
Completa lo que puedas de: {json_structure}"""

    with model.chat_session():
        print("iteration: ", i)
        print(model.generate(template))
```

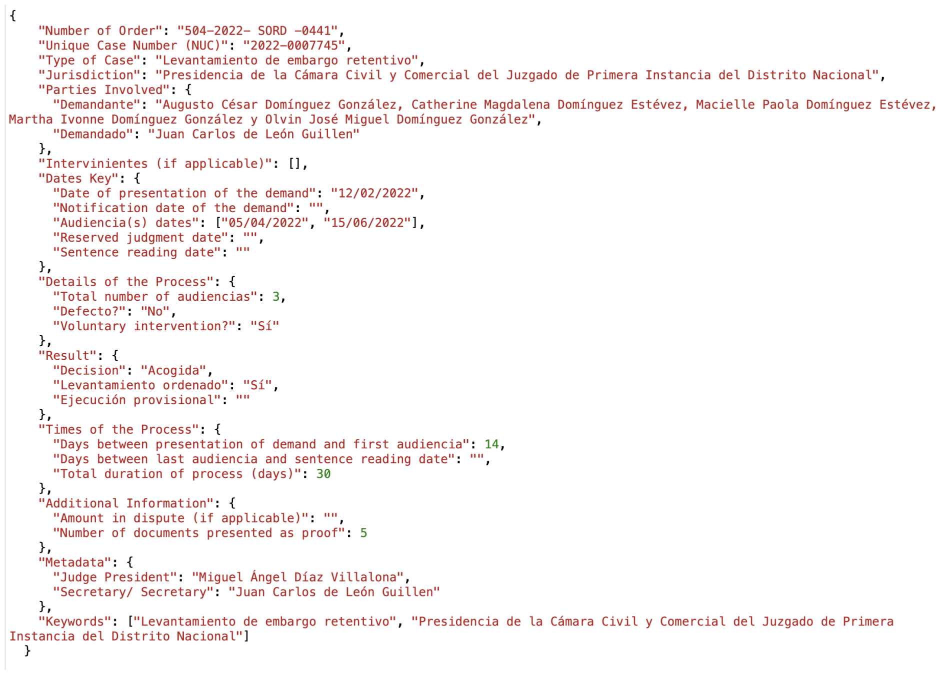
```
{
    "general_information": {
        "ordinance_number": "",
        "case_unique_number": "1-22-00677-1",
        "case_type": "",
        "jurisdiction": "PRESIDENCIA DE LA CÁMARA CIVIL Y COMERCIAL DEL JUZGADO DE PRIMERA INSTANCIA DEL DISTRITO N
ACIONAL"
    },
    "parties_involved": {
        "plaintiff": "yes de la República Dominicana",
        "defendant": "MADIEL RAFAEL TORIBIO AQUINO y PEDRO MÉNDEZ ENCARNACIÓN",
        "interveners": []
    },
    "key_dates": {
        "filing_date": "",
        "notification_date": "",
        "hearing_dates": [],
        "reserved_ruling_date": "",
```

Figure 3: Code section from Jupyter notebook



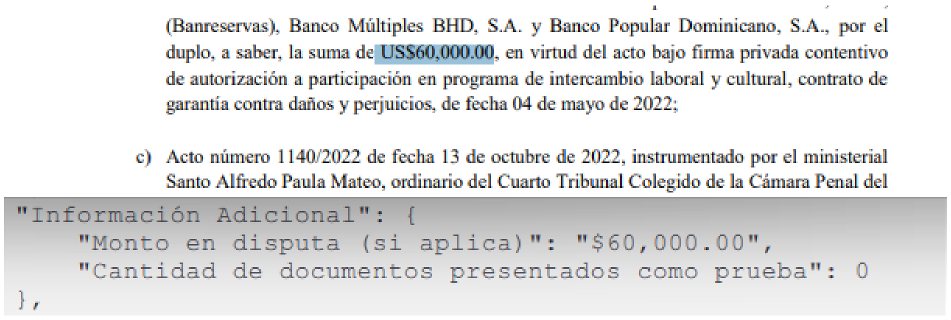
```
In [6]: faiss_index = FAISS.load_local("./store_index", embeddings, allow_dangerous_deserialization=True)

In [7]: question = """
        Encuentra las siguientes fechas clave:
        1 Fecha de presentación de la demanda: (DD/MM/AAAA)
        2 Fecha de notificación de la demanda: (DD/MM/AAAA)
        3 Fecha(s) de audiencia(s): (DD/MM/AAAA)
        4 Fecha de fallo reservado: (DD/MM/AAAA)
        5 Fecha de lectura de sentencia: (DD/MM/AAAA)
        """
        matched_docs = faiss_index.similarity_search(question, 4)
        context = ""
        for doc in matched_docs:
            context = context + doc.page_content + " \n\n "

:n [11]: context

Out[11]: 'haberlas avanzado en su totalidad ”. \nPRUEBAS APORTADAS  \nEn los medios probatorios que las partes aportaron al
        proceso consta lo siguiente:  \nParte Demandante  \nA) Documentales:  En su escrito de demanda de fecha 03/04/2024
        , depositó los siguientes \ndocumentos anexos en copia, a saber:  1.- Poder cuota litis de fecha 29/02/2024, suscri
        to por \nel señor José Valentín Barona Rivas; 2.- Acto núm. 228/2024, de fecha 29/02/2024, \ncontentivo de  notific
        ación de dimisión, con sello  recibido  por el Ministerio de Trabajo  en \nfecha 29/02/2024 ; y 2.- Carnet de la em
        presa Caribe Tours, a nombre del  señor José Barona \nRivas  como chofer interurbano .  \nB) Comparecencia persona
        l: \n\n IV. Incidente : Medio de inadmisión por falta de interés  \n4.1. La parte demandada empresa CARIBE TOURS,
        S. A. , solicita en su escrito de defensa \ndepositado en fecha 27/05/2024 y en la audiencia de producción, discusi
        ón de pruebas y \nconclusiones al fondo celebrada en fecha 1 0/07/2024, que el Tribunal declare la inadmisibilidad
        \nde la demanda por falta de interés del demandante  y de derecho  para actuar en justicia , \nfundamentado en una
        aceptación de oferta real de pago de los derechos adquiridos y \nprestaciones  laborales . Ante el medio de inadmis
        ión invocado , la parte demandante se refirió \ncomo consta en la citada acta de audiencia de la fecha indicada, en
        la que solic ita que sea \nrechazado por improcedente, mal fundado y carente  de base legal.  \n4.2. En cuanto al i
        ncidente presentado  este tribunal tiene a bien establecer que, es una \nobligación de todo juez y toda jueza antes
        de examen al fondo verificar y responder todos y \n\n REPÚBLICA DOMINICANA  \nPODER JUDICIAL  \nSEGUNDA SALA DEL JU
        ZGADO DE TRABAJO DEL \nDISTRITO JUDICIAL DE  SANTIAGO  \n \nSentencia laboral núm. 0374 -2024 -SSEN -00167  \nEAS/a
        mm/dr/af  Número único de caso . 2024 -0032275  \n \n Página 6 de 19 \n \nB-1) JOSÉ VALENTÍ N BARONA RIVAS, dominic
        ano, mayor de edad, casado, empleado \nprivado, portador de la cédula de identidad y electoral núm. 001 -1449374 -
        5, con domicilio \nen la calle Salvador Beato, casa núm. 08, del sector Los Pomos, La Vega. Cuyas \ndeclaraciones c
        onstan en el acta de audiencia correspondiente.  \nParte Demandada  \nA) Documentales:  En su escrito de defensa de
        fecha 27/05/2024 , depositó los sigui entes \ndocumentos anexos , a saber:  1. Original del a cto núm.  57-2024 , d
        e fecha 14 /03/2024, \ninstrumentado por el minis terial Salvador Antonio Vitiello , contentivo de oferta real de p
        ago; \n2. Copia del cheque núm.  081798 de  fecha 29 /02/2024,  a nombre del señor José Valentín \n\n PRETENSIONES
        DE LAS PARTES  \nParte demandante  \nEn la audiencia celebrada en fecha 10/07/2024 , concluy e de la manera siguien
        te: “PRIMERO: \nQue sean acogidas en todas sus partes las conclusiones vertidas en la demanda de fecha \n03/04/202
        4; SEGUNDO: Que la parte demandada sea condenada al pago de las costas \nordenando su distracción en favor de los a
        bogados concluyentes quienes afirmamos estarlas \navanzando en su totalidad. I HARÉIS JUSTICIA ”. Además, concluye
        en cuanto al medio de \n\n '
```

Figure 4: Example of embedding usage to get context.



```
{
    "Number of Order": "504-2022- SORD -0441",
    "Unique Case Number (NUC)": "2022-0007745",
    "Type of Case": "Levantamiento de embargo retentivo",
    "Jurisdiction": "Presidencia de la Cámara Civil y Comercial del Juzgado de Primera Instancia del Distrito Nacional",
    "Parties Involved": {
      "Demandante": "Augusto César Domínguez González, Catherine Magdalena Domínguez Estévez, Macielle Paola Domínguez Estévez,
Martha Ivonne Domínguez González y Olvin José Miguel Domínguez González",
      "Demandado": "Juan Carlos de León Guillen"
    },
    "Intervinientes (if applicable)": [],
    "Dates Key": {
      "Date of presentation of the demand": "12/02/2022",
      "Notification date of the demand": "",
      "Audiencia(s) dates": ["05/04/2022", "15/06/2022"],
      "Reserved judgment date": "",
      "Sentence reading date": ""
    },
    "Details of the Process": {
      "Total number of audiencias": 3,
      "Defecto?": "No",
      "Voluntary intervention?": "Sí"
    },
    "Result": {
      "Decision": "Acogida",
      "Levantamiento ordenado": "Sí",
      "Ejecución provisional": ""
    },
    "Times of the Process": {
      "Days between presentation of demand and first audiencia": 14,
      "Days between last audiencia and sentence reading date": "",
      "Total duration of process (days)": 30
    },
    "Additional Information": {
      "Amount in dispute (if applicable)": "",
      "Number of documents presented as proof": 5
    },
    "Metadata": {
      "Judge President": "Miguel Ángel Díaz Villalona",
      "Secretary/ Secretary": "Juan Carlos de León Guillen"
    },
    "Keywords": ["Levantamiento de embargo retentivo", "Presidencia de la Cámara Civil y Comercial del Juzgado de Primera
Instancia del Distrito Nacional"]
  }
```

Figure 5: Output JSON file sample



(Banreservas), Banco Múltiples BHD, S.A. y Banco Popular Dominicano, S.A., por el duplo, a saber, la suma de US$60,000.00, en virtud del acto bajo firma privada contentivo de autorización a participación en programa de intercambio laboral y cultural, contrato de garantía contra daños y perjuicios, de fecha 04 de mayo de 2022;

c) Acto número 1140/2022 de fecha 13 de octubre de 2022, instrumentado por el ministerial Santo Alfredo Paula Mateo, ordinario del Cuarto Tribunal Colegido de la Cámara Penal del

```
"Información Adicional": {
    "Monto en disputa (si aplica)": "$60,000.00",
    "Cantidad de documentos presentados como prueba": 0
},
```

Figure 6: Example of field extraction

Week 5 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

September 2024

0.1 What did you work on this week?

1. Run LED legal tuned model for finetuning
2. Conclude LongT5 experiment issues
3. Experiment with Phi3 for initial experimentation
4. Generate initial summaries with LED legal tuned

0.2 What are you planning on working on next?

1. Continue LLM testing and tools
2. Generate more summaries for validation
3. Prepare questions for Dr. Alexanders NLP freelancer

0.3 Is anything blocking you from getting work done?

1. None

1 Abstracts

- Title: From Sparse to Dense: GPT-4 Summarization with Chain of Density Prompting. Conference: ACL 2023, Proceedings of the 4th New Frontiers in Summarization Workshop. Link: https://aclanthology.org/2023.newsum-1.7/
- Abstract: Selecting the "right" amount of information to include in a summary is a difficult task. A good summary should be detailed and entity-centric without being overly dense and hard to follow. To better understand this tradeoff, we solicit increasingly dense GPT-4 summaries with what we refer to as a "Chain of Density" (CoD) prompt. Specifically, GPT-4 generates an initial entity-sparse summary before iteratively incorporating missing salient entities without increasing the length. Summaries generated by CoD are more abstractive, exhibit more fusion, and have less

of a lead bias than GPT-4 summaries generated by a vanilla prompt. We conduct a human preference study on 100 CNN DailyMail articles and find that humans prefer GPT-4 summaries that are more dense than those generated by a vanilla prompt and almost as dense as human written summaries. Qualitative analysis supports the notion that there exists a tradeoff between informativeness and readability. 500 annotated CoD summaries, as well as an extra 5,000 unannotated summaries, are freely available on HuggingFace (https://huggingface.co/datasets/griffin/chain_of_density).

- Summary: Chain of Density was proposed to allow fo iterative summarization by creating denser and denser summaries after each iteration according tot he prompt. This let to more abstractive and less biased summaries than with vanilla prompting from GPT-4.
- Relevance: This is a wildly popular prompting technique for summarization that, if it works for OSS LLMs as well, will be very useful.

2 Relevant Info

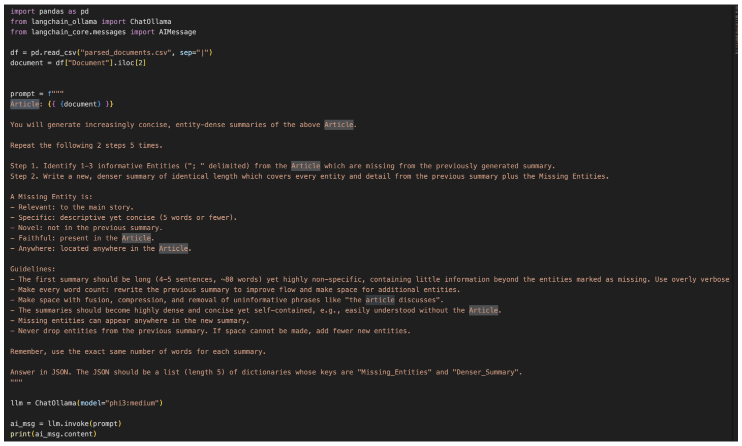
- LED model is a proposed transformer technique from 2020 for the use in long document tasks in NLP. The paper on it can be found here.
- Legal documents are very long, making models meant for long form documents the ideal candidates.
- Phi-3 is an LLM created by Microsoft that was meant to be on the smaller end and thus uses less compute.

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/hv70flwkm977gky004l5vz15rpgfdmir`
3. Thomas-Orth/ollam_test.py
   - Brief Description: This runs Chain of Density prompting on Phi-3 with a chosen summary
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Read in and set up the dataset and choose a summary
     (b) Second block: Build prompt
     (c) Third Block: Apply to Ollama

   (d) Fourth Block: Print output

- Screenshot of code:



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

df = pd.read_csv("parsed_documents.csv", sep="|")
document = df["Document"].iloc[2]


prompt = f"""
Article: {{ {document} }}

You will generate increasingly concise, entity-dense summaries of the above Article.

Repeat the following 2 steps 5 times.

Step 1. Identify 1-3 informative Entities ("; " delimited) from the Article which are missing from the previously generated summary.
Step 2. Write a new, denser summary of identical length which covers every entity and detail from the previous summary plus the Missing Entities.

A Missing Entity is:
- Relevant: to the main story.
- Specific: descriptive yet concise (5 words or fewer).
- Novel: not in the previous summary.
- Faithful: present in the Article.
- Anywhere: located anywhere in the Article.

Guidelines:
- The first summary should be long (4-5 sentences, ~80 words) yet highly non-specific, containing little information beyond the entities marked as missing. Use overly verbose
- Make every word count: rewrite the previous summary to improve flow and make space for additional entities.
- Make space with fusion, compression, and removal of uninformative phrases like "the article discusses".
- The summaries should become highly dense and concise yet self-contained, e.g., easily understood without the Article.
- Missing entities can appear anywhere in the new summary.
- Never drop entities from the previous summary. If space cannot be made, add fewer new entities.

Remember, use the exact same number of words for each summary.

Answer in JSON. The JSON should be a list (length 5) of dictionaries whose keys are "Missing_Entities" and "Denser_Summary".
"""

llm = ChatOllama(model="phi3:medium")

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

Figure 1: Phi3 Code

4. Flow Diagram:

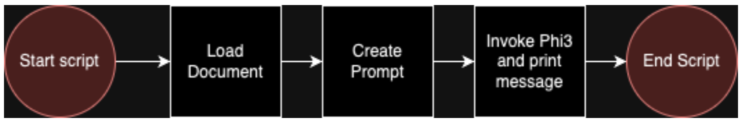


Figure 2: Flow diagram

5. Running scripts:
   (a) Download the script and the csv from the box link.
   (b) Download ollama
   (c) Run: ollama pull phi3:medium
   (d) Run: python -m pip install langchain pandas langchain_ollama
   (e) Run: python ollama_test.py

4 Documentation

1. Download CSV file, with two columns: Document and Summary
2. Update script to point to the CSV file
3. Prompt Phi3 with Chain of Density
4. Manually evaluate summary

5 Results

5.1 LED Summary Comparison

5.1.1 Ground truth

"COVID-19 Summary: This is a class action petition for a writ of mandamus alleging that the Minnesota Department of Corrections failed in its legal duty to protect individuals under its custody or control from COVID-19. On October 22, 2020, thirteen individuals in the custody and control of the Minnesota Department of Corrections (MNDOC) filed this petition for a writ of mandamus in the Second Judicial District of Minnesota on behalf of themselves and all others similarly situated. Represented by the ACLU of Minnesota and individual public defenders, the petitioners asked the court to find that the MNDOC had failed and refused to perform its legal duty to protect the petitioners and those similarly situated from COVID-19 and to issue a peremptory writ of mandamus compelling the MNDOC to do so.The petitioners alleged that the COVID-19 virus was allowed to spread rapidly at the Moose Lake Correctional Facility because the MNDOC had failed to implement reasonable measures to slow or stop the transmission of the virus. They further alleged that in the period between June 2, 2020, and June 23, 2020, Faribault Correctional Facility saw its case count explode from three confirmed cases to 205 confirmed positive COVID-19 cases, including two deaths. The petition names six other facilities, five of which saw similar exponential increases in confirmed COVID-19 cases.The petitioners–most of whom have pre-existing conditions making them particularly vulnerable to a severe infection with COVID-19–alleged that social distancing was impossible within correctional facilities; that cleaning procedures and supplies were inadequate to prevent the spread of the virus; and that staff failed to comply with proper protective procedures for COVID-19, including mask wearing. Various petitioners also alleged that they were denied adequate medical care for other conditions because of the pandemic, that they were denied conditional medical release, that they were denied COVID-19 testing except when exhibiting the most serious symptoms, that they were forced to work despite being symptomatic, and that they were forced into or threatened with punitive segregation.The petitioners argued that MNDOC had a duty to protect incarcerated people from the foreseeable harm of COVID-19 that arose at least as early as President Trump's March 13, 2020, acknowledgment of the pandemic and announcement of a national emergency. They further argued that failure and refusal to protect incarcerated people from COVID-19 constituted cruel and unusual punishment in violation of the Minnesota Constitution and also violated Minnesota state statutes. They contended that halting the spread of the virus in correctional settings would be best achieved through population reduction. The case was assigned to Judge Sara R. Grewing. On December 10, 2020, the petitioners filed an amended and supplemental petition for a writ of mandamus. The amended petition added six new petitioners and requested the court order MNDOC to obtain sufficient quantity of vaccine to vaccinate staff and incarcerated people. It otherwise maintained most aspects of the original petition.On March 31, 2021, the court granted the motion for class certification and motion to amend the petition to add new respondent and new causes of action. The certified class consisted of all prisoners either in the custody or under the supervision of the Minnesota Department of Corrections. However, the court denied the request for a writ of mandamus. In her order, Judge Grewing expressed concern that the petitioners requested the court direct MNDOC's exercise of discretion, noting that the response to COVID-19 involved the exercise of thousands of discretionary acts. In denying the request, the court noted that there is no law mandating specifically how the Commissioner of MNDOC should be responding to COVID-19 and that MNDOC had not failed to act, but had engaged in population reduction, testing, and quarantining of some individuals.The next day, the petitioners filed a second amended and supplemental petition requesting an injunction enjoining MNDOC from continuing to violate their legal duty; declaratory judgment that MNDOC had violated the Cruel or Unusual Punishment Clause, the Equal Protection Clause, and the Due Process Clause of the Minnesota Constitution and a permanent injunction preventing MNDOC from continuing to do so; and an affirmative order that MNDOC prioritize COVID-19 vaccinations for all class members.In July, the respondents filed a motion to dismiss for failure to state a claim. On September 30, 2021 the court denied the petitioners' motion for a temporary injunction and granted the respondents' motion for failure to state a claim, entering judgment in favor of the respondents."

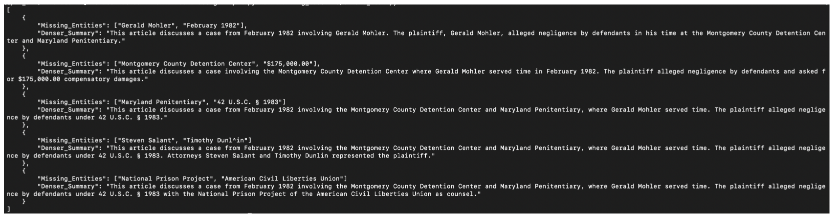
5.1.2 Generated

"[This summary is temporary while we research the case.] This is a class action lawsuit in which the American Civil Liberties Union of Minnesota (ACLU-MN) and the Minnesota State Public Defender (MSPD) filed suit against the Minnesota Department of Corrections (DOC) on behalf of themselves and all others similarly affected by COVID-19. The plaintiffs, represented by the ACLU-MN, alleged that the DOC failed to take preventative measures to slow or stop the spread of the disease resulting from the novel coronavirus. They sought declaratory judgment, preliminary and permanent injunctions, and a writ of mandamus to require the DOC to perform its legal duty to keep prisoners reasonably safe from the disease. The case was originally brought in 2020, and the plaintiffs filed a second amended and Supplemental Petition on March 1, 2021. The second amended petition was filed on April 28, 2021, in the U.S. District Court for the District of Minnesota. The third amended and supplemental petition, filed in the Minnesota Supreme Court on April 29, 2021 and unsealed on May 1, 2022, alleged violations of the Minnesota Constitution and Minnesota Statutes, Sections 586, 586 and 7 of the State Constitution. The amended petition asked the court for a Writ of Mandamus, a declarator judgment, and to grant the parties' motion for a writ to show cause why they should not be ordered to perform their duty to protect prisoners from the new disease. The amended amended petition also included a provision for the parties to show probable cause why the defendants should not perform their duties to keep the disease contained within their prisons. In the amended amended complaint, the parties agreed to the granting of a Writ to Mandamus and to give the parties the right to dismiss the amended petition. The parties also agreed to pay a penalty of $100,000 each. On May 2, 2021 the parties reached a settlement agreement, which is subject to court approval. This settlement is ongoing."

5.1.3 Summary Comparison

The summary is coherent and able to pick up some facts. Although it seems to get some facts incorrect

5.2 Phi-3 Chain of Density



```
[
    {
        "Missing_Entities": ["Gerald Mohler", "February 1982"],
        "Denser_Summary": "This article discusses a case from February 1982 involving Gerald Mohler. The plaintiff, Gerald Mohler, alleged negligence by defendants in his time at the Montgomery County Detention Cen
ter and Maryland Penitentiary."
    },
    {
        "Missing_Entities": ["Montgomery County Detention Center", "$175,000.00"],
        "Denser_Summary": "This article discusses a case involving the Montgomery County Detention Center where Gerald Mohler served time in February 1982. The plaintiff alleged negligence by defendants and asked f
or $175,000.00 compensatory damages."
    },
    {
        "Missing_Entities": ["Maryland Penitentiary", "42 U.S.C. § 1983"]
        "Denser_Summary": "This article discusses a case from February 1982 involving the Montgomery County Detention Center and Maryland Penitentiary, where Gerald Mohler served time. The plaintiff alleged neglige
nce by defendants under 42 U.S.C. § 1983."
    },
    {
        "Missing_Entities": ["Steven Salant", "Timothy Dunl^in"]
        "Denser_Summary": "This article discusses a case from February 1982 involving the Montgomery County Detention Center and Maryland Penitentiary, where Gerald Mohler served time. The plaintiff alleged neglige
nce by defendants under 42 U.S.C. § 1983. Attorneys Steven Salant and Timothy Dunlin represented the plaintiff."
    },
    {
        "Missing_Entities": ["National Prison Project", "American Civil Liberties Union"]
        "Denser_Summary": "This article discusses a case from February 1982 involving the Montgomery County Detention Center and Maryland Penitentiary, where Gerald Mohler served time. The plaintiff alleged neglige
nce by defendants under 42 U.S.C. § 1983 with the National Prison Project of the American Civil Liberties Union as counsel."
    }
]
```

Figure 3: Phi-3 CoD output

The output of the Chain-of density seems to be not super impressive. But that could be a limitation of langchain when applying the Chain-of-density prompt.

6 Proof of Results

The LED model used for finetuning was initially trained from legal documents and was using the already validated pipeline.

6.1 Known Limitations

The initial work is done with an early version of the dataset so further results with this dataset may be sub-par to start.

Additionally, Phi-3 does not have the same effect with Chain-of-Density prompting that ChatGPT did.