HAAG NLP Summarization Week 6

Michael Bock

September 2024

1 Slack Questions

What did you accomplish this week?

- Finished Training Script with Clearinghouse Data

What are you planning on working on next?

- Get Issue Data from UPenn Database
- If further results are desired on clearinghouse classification task to show that it actually works, run the script on PACE so that I don't run out of memory

What is blocking you from progressing?

- My PACE home directory doesn't exist, we have submitted a support ticket for this and contacted PACE.

2 Abstract

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length. To address this limitation, we introduce the Longformer with an attention mechanism that scales linearly with sequence length, making it easy to process documents of thousands of tokens or longer. Longformer's attention mechanism is a drop-in replacement for the standard self-attention and combines a local windowed attention with a task motivated global attention. Following prior work on long-sequence transformers, we evaluate Longformer on character-level language modeling and achieve state-of-the-art results on text8 and enwik8. In contrast to most prior work, we also pretrain Longformer and finetune it on a variety of downstream tasks. Our pretrained Longformer consistently outperforms RoBERTa on long document tasks and sets new state-of-the-art results on WikiHop and TriviaQA. We finally introduce the Longformer-Encoder-Decoder (LED), a Longformer variant for supporting long document generative sequence-to-sequence tasks, and demonstrate its effectiveness on the arXiv summarization dataset.

Link: https://arxiv.org/abs/2004.05150v2

2.1 Brief Analysis

Longformer Encoder-Decoder is a transformer adapted to handle tasks involving long documents. Transformers typically use self-attention, which has $O(n^2)$ complexity. Longformer's sliding window attention has $O(n)$ complexity. It works by having each token attend only to tokens near it. To increase the size of the receptive field, Longformer dilates the window. Additionally, for a few pre-selected tokens, they get the full self-attention. Then, Longformer uses the same pretraining paradigm that models like BERT use.

Longformer does achieve better performance on long documents than short models. However, in my opinion, the point of attention is that its globally receptive. If I wanted a locally receptive feature extractor, why can't I use a 1D convolutional layer? One reason I can think of is that convolution complexity is $O(nlogn)$, so perhaps a CNN is less efficient than LongFormer's attention mechanism

3 Scripts and Code Blocks

MLM Pipeline:

mistral_datasets.py(updated):

```
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
from copy import deepcopy

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, n = -1):
        self.dataset = {'text': [], 'labels': []}
        print('Retrieving complaints')
        if n == -1:
            cases = get_complaint_only_cases(cases_path)
        else:
            cases = get_complaint_only_cases(cases_path)[:n]
        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):
        print('This is the dataset length:', len(self.dataset['labels']))
        return len(self.dataset['labels'])

    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.
        """
        return self.dataset['text'][idx], self.dataset['labels'][idx]


    def train_test_split(self, pct = 0.8):
        train = deepcopy(self)
        test = deepcopy(self)
        n_train = int(len(self) * pct)
        train.dataset['text'] = train.dataset['text'][:n_train]
        train.dataset['labels'] = train.dataset['labels'][:n_train]
        test.dataset['text'] = test.dataset['text'][n_train:]
        test.dataset['labels'] = test.dataset['labels'][n_train:]

        return train, test
    #Use this in the event of using a DataCollator
    def prepare_corpus(self, vocab, normalize_text, tokenizer, pad, text_len):

        dataset = {'text': [], 'labels': []}
        for text, label in zip(self.dataset['text'], self.dataset['labels']):
            feature = text
            label = label

            feature = pad(vocab(tokenizer(normalize_text(feature))), text_len)
            one_hot = [0] * 26
            one_hot[label] = 1

            dataset['text'].append(feature)
            dataset['labels'].append(one_hot)

        self.dataset = dataset


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
    #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()
```

model.py

```
from torchtext.data.utils import get_tokenizer
from torch.utils.data import DataLoader
from torchtext.vocab import build_vocab_from_iterator
import spacy
import string
import sys
import torch
from torch import nn
from tqdm import tqdm
import time
from torch.utils.data.dataset import random_split
from torch.utils.tensorboard import SummaryWriter
from torchtext.data.functional import to_map_style_dataset
import datetime
import os
from matplotlib import pyplot as plt
import seaborn as sns
from torchmetrics import ConfusionMatrix
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

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

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


class TextClassificationModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

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

```
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text):
        embedded = self.embedding(text, None)
        return self.fc(embedded)

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

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

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

    return total_acc/total_count, total_loss/total_count

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

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

def normalize_text(text):
    # Process the text using SpaCy
```

```
    doc = nlp(text)

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

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

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

    return normalized_text

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

def pad(text_processed, text_len):
    text = text_processed[len(text_processed)//2 - text_len//2 : len(text_processed)//2 + text_len//2]
    while len(text) < text_len:
        text.append(-1)
    return text

if __name__ == '__main__':
    ds = DocumentClassificationDataset(None, cases_path = '../../all_cases_clearinghouse.pkl', n = -1)
    print('DS made, building vocabulary')
    vocab = build_vocab_from_iterator(yield_token(ds), specials = ["<unk>"])
    vocab.set_default_index(vocab["<unk>"])
    text_len = 256

    print('Text pipeline')
    text_preprocessing_pipeline = lambda x: pad(vocab(tokenizer(normalize_text(x))), text_len)
    print(normalize_text(ds[0][0]))

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

    def collate_fn(batch):
        text_batch = []
        label_batch = []
        for text, label in batch:
            text_batch.append(text)
            label_batch.append(label)

        label_batch = torch.tensor(label_batch).double()
        text_batch = torch.tensor(text_batch)

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

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

```

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

    print('Data Loaded, total length = ', len(train_dataloader) + len(val_dataloader))
    num_class = 26#len(set([label for (label, text, offset) in dataloader]))
    vocab_size = len(vocab)
    emsize = 64
    model = TextClassificationModel(vocab_size, emsize, num_class).to(device)
# Hyperparameters
    EPOCHS = 100  # epoch

    #total_accu = None
    #print('Num Train: ', num_train)
    #print(train_dataloader, len(train_dataloader))
    LR = 1e-3  # learning rate
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = LR)

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

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

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

4 Documentation

For this starting task, I took the clearinghouse data and removed every field except the OCR'd complaint and the first issue that it was tagged with. Then, I trained generated a vocabulary and tokenizer with torchtext. Then, I took the middle 256 characters of the text and padded if necessary with blank tokens. I used the tokenized text to train a model that had an embedding and a linear layer to project the embedding to a classification of one of the 26 issue classes. Given that this model does not have any hidden layers, it doesn't perform optimally. I also have been unable to run a full dataset on my GPU due to problems logging into PACE and my local computer not having enough GPU memory. Out of the tiny dataset, the small model has 66% validation accuracy. But most importantly, I was able to show that training runs will reach 0 training loss and 100% training accuracy on a small dataset, indicating that the script is functioning correctly and can have larger datasets run through it to product better tiny models.

5 Scription Validation(Optional)

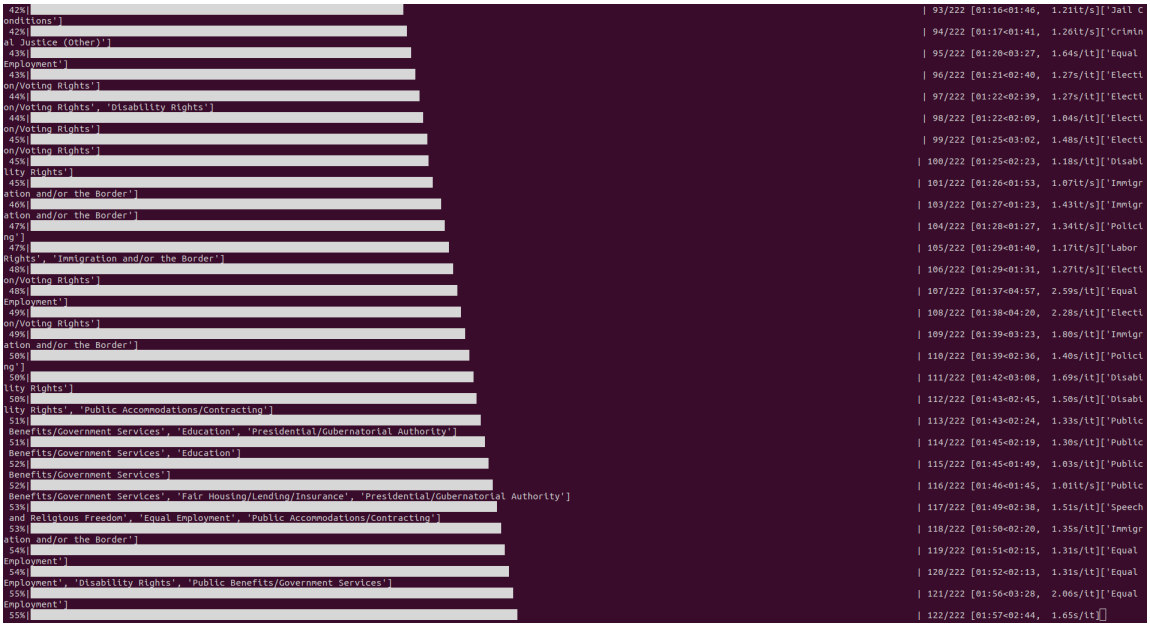


Figure 1: Reading in the complaints and issues

6 Results Visualization

The problem with these results is still the CUDA error. Figure 2 shows the loss and accuracy curves of the tiny model. Even though the validation looks low, realize that this model essentially has no parameters, its only parameters are in the projection to the output space. Importantly, notice that the training loss decreases towards 0. This means that the training script is working and is ready to train larger models on larger datasets.

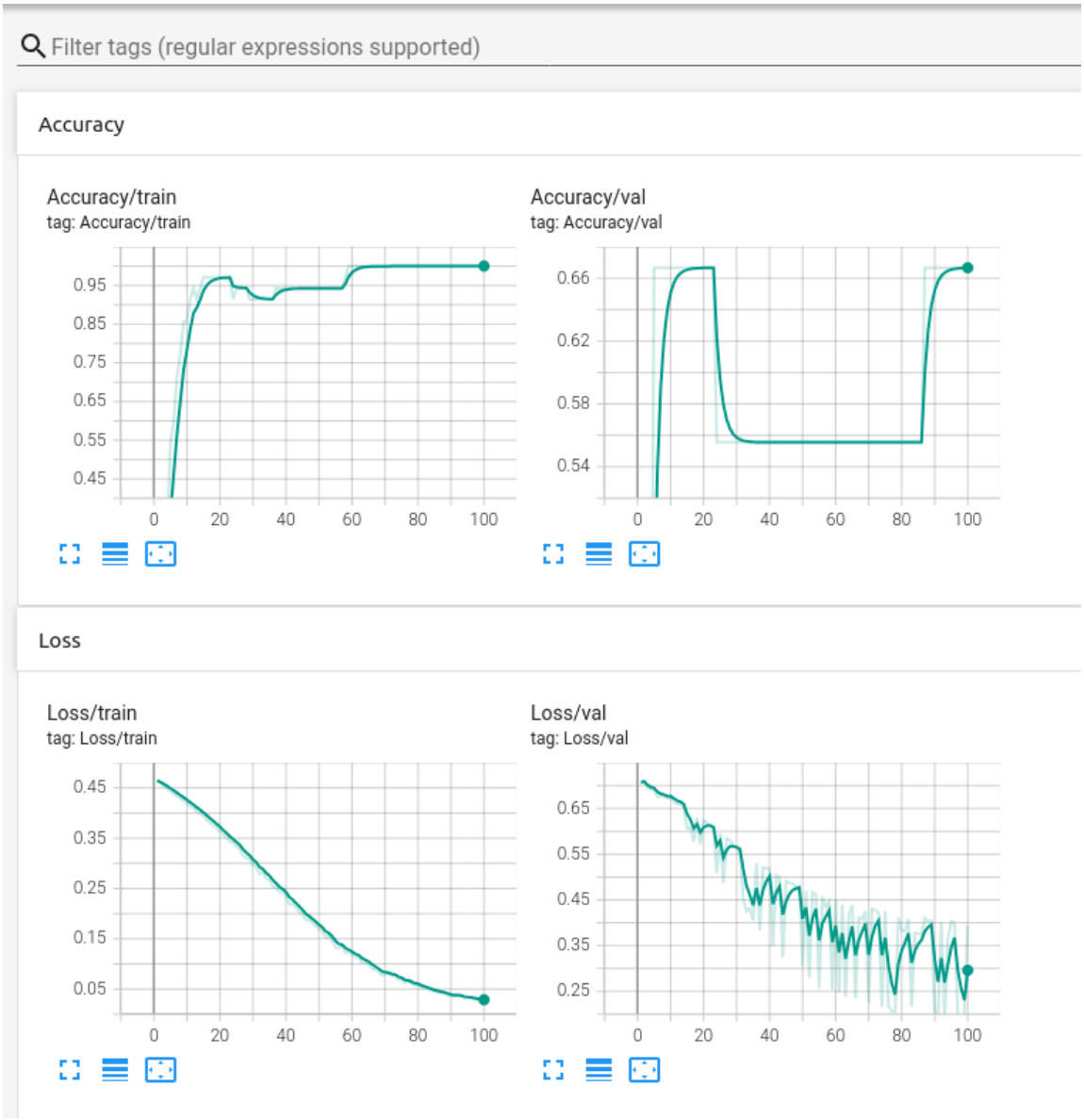


Figure 2: Tensorboard of best model on a tiny dataset

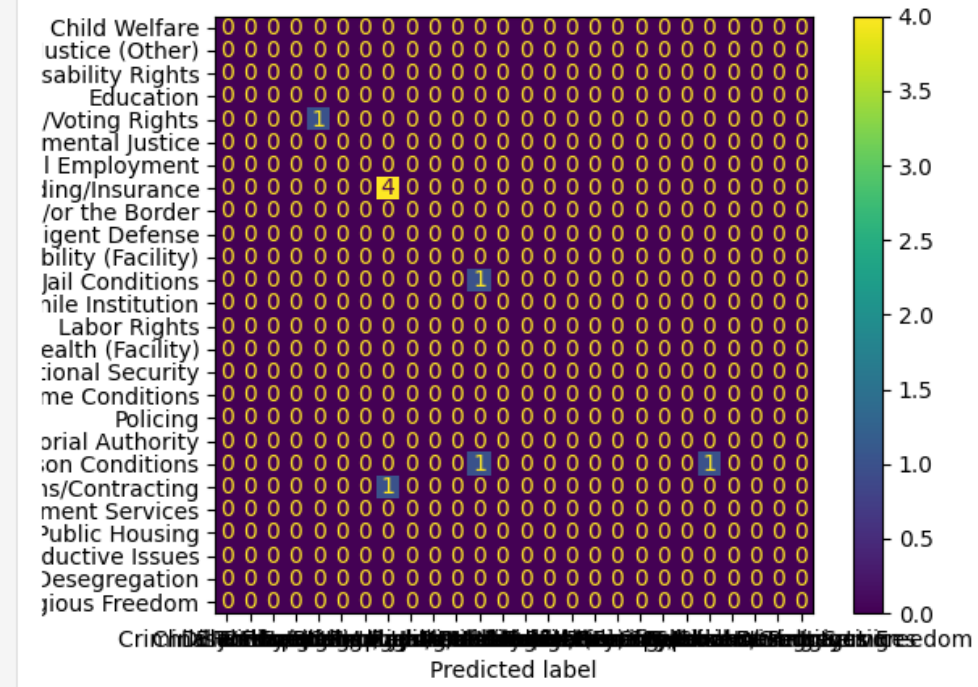


Figure 3: Test Confusion Matrix for Tiny Model on Validation Set

7 Next Week's proposal

- Train on PACE
- Use UPenn Data

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 6

Víctor C. Fernández
September 2024

CONTENTS

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Implemented code for extracting and grouping json objects from models response in order to later use them for the model benchmarking and validation. Models outputs used so far are: llama3.1, gemma2, mistral-nemo, qwen2, deepseek-coder-v2, phi3, and mixtral.
- Created a set of functions to compare model responses contained in JSON file based on the field data type. Short strings such as case number, allow for no error margin, but longer strings such as what is the case about, allow for a wider difference when compared to the validation dataset. Still, this is something that may require further analysis to better validate the models output.
- Created a validation template that simply contains a json object with each of the fields expected in the output and the data type for each of them that will then help identify how to compare the model prediction with the validation data.
- Executed benchmarking process on data generated by the models for input template v1 and output template v1.
- Met with the NLP-Sentencias team on Saturday 21st to align on our goals and distribute our tasks more efficiently.
- Created and distributed surveys for Pathways to PhD seminar carried out by Dr. Lindvall.
- Organized and assisted Dr. Lindvall in the Pathways to PhD seminar on September 24th.
- Meeting with the NLP team on September 27th for our weekly meeting.
- Meeting with Dr. Alexander and Nathan Dahlberg on September 27th to get further insighs on NLP research.
- Reached out to colleagues in my current company to get insights on UIs used for ML projects and contacts for getting more insights around NLP research.

What progress are you making next?

- Enhance prompt engineering techniques to improve data extraction accuracy and completeness.
- Polish automated validation system to address cases like long strings, or data ranges.
- Review scoring logic to correctly compare all models under same conditions.

- Keep conducting comprehensive benchmarking of different Ollama models to identify the most effective model(s) for our specific task. Include latest Llama 3.2 model.
- Retrieve additional insights and perform a second round with modified templates (v2).
- Meet with the NLP team on October 4th for our weekly meeting.

Is there anything blocking you from making progress?

No significant blockers at this time. Already checking out executing code in PACE for faster development and validation.

2 ABSTRACTS

1. **Title:** Semantic Segmentation of Legal Documents via Rhetorical Roles
   - **URL:** https://aclanthology.org/2022.nllp-1.13.pdf

   - **Abstract:** Legal documents are unstructured, use legal jargon, and have considerable length, making them difficult to process automatically via conventional text processing techniques. A legal document processing system would benefit substantially if the documents could be segmented into coherent information units. This paper proposes a new corpus of legal documents annotated (with the help of legal experts) with a set of 13 semantically coherent units labels (referred to as Rhetorical Roles), e.g., facts, arguments, statute, issue, precedent, ruling, and ratio. We perform a thorough analysis of the corpus and the annotations. For automatically segmenting the legal documents, we experiment with the task of rhetorical role prediction: given a document, predict the text segments corresponding to various roles. Using the created corpus, we experiment extensively with various deep learning-based baseline models for the task. Further, we develop a multitask learning (MTL) based deep model with document rhetorical role label shift as an auxiliary task for segmenting a legal document. The proposed model shows superior performance over the existing models. We also experiment with model performance in the case of domain transfer and model distillation techniques to see the model performance in limited data conditions.

   - **Summary:** This paper introduces a corpus of legal documents annotated with rhetorical roles (e.g. facts, arguments, statutes) and proposes methods for automatically predicting these roles in new documents. The authors create a dataset of 100 Indian legal documents from competition law and income tax domains, annotated with 13 fine-grained rhetorical role labels. They experiment with various deep learning models for predicting roles, including a novel multitask learning approach that leverages label shift information.

   - **Relevance:** Even though this paper focuses on Indian legal documents in English, some aspects could be relevant to our work on Spanish legal documents. The concept of rhetorical roles and methods for automatically identi-

fying them could potentially be adapted to Spanish texts. However, the specific roles may differ for Spanish legal documents, and we possibly would need to develop a Spanish-specific corpus and models.

3 SCRIPTS AND CODE BLOCKS

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

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

1. JSON extraction from model outputs code, uploaded here.
   This file extracts the JSON objects that are expected to be returned by the model in the output text plus the one added with execution information when saving the output and convert it into a single JSON file output for each of the obtained outputs from the models. This new extracted JSON will then be used for output validation and model benchmarking. It has been designed to bulk extract JSON files by passing the model outputs parent folder and then extracting these JSON object from all files contained in such folder. The code is the following:

```python
def extract_json_from_text(text):
    """Extract JSON objects from text using a stack-based
    ↪ approach."""
    json_objects = []
    stack = []
    start = -1

    for i, char in enumerate(text):
        if char == '{':
            if not stack:
                start = i
            stack.append(char)
        elif char == '}':
            if stack:
                stack.pop()
                if not stack:
                    try:
                        json_obj = json.loads(text[start:i+1])
                        json_objects.append(json_obj)
                    except json.JSONDecodeError:
                        print(f"Warning: Could not parse JSON
                        ↪ object: {text[start:i+1][:50]}...")
    return json_objects
```

Code 1—Function for extracting the JSON objects from the text file

```python
def process_file(input_path, output_path):
    """Process a single file, extract JSON, and save as new
    ↪ file."""
    with open(input_path, 'r', encoding='utf-8') as file:
        content = file.read()

    json_objects = extract_json_from_text(content)

    if len(json_objects) == 2:
        # Combine the two JSON objects
        combined_json = {**json_objects[0], **json_objects[1]}
    elif len(json_objects) == 1:
        combined_json = json_objects[0]
    else:
        print(f"Warning: Unexpected number of JSON objects
        ↪ ({len(json_objects)}) in {input_path}")
        return

    # Save the combined JSON
    with open(output_path, 'w', encoding='utf-8') as file:
        json.dump(combined_json, file, ensure_ascii=False,
        ↪ indent=2)

    print(f"Processed: {input_path} -> {output_path}")
```

Code 2—Function for processing a file and then storing the result in a new json file

```
def process_model_directory(model_input_dir, model_output_dir):
    """Process all text files in a single model's directory."""
    if not os.path.exists(model_output_dir):
        os.makedirs(model_output_dir)

    for filename in os.listdir(model_input_dir):
        if filename.startswith("output_") and
        ↪  filename.endswith(".txt"):
            input_path = os.path.join(model_input_dir, filename)
            new_name = filename.replace("output_",
            ↪  "").replace(".txt", "_extracted.json")
            output_path = os.path.join(model_output_dir, new_name)
            process_file(input_path, output_path)

def process_parent_directory(parent_input_dir, parent_output_dir):
    """Process all model directories in the parent directory."""
    for model_dir in os.listdir(parent_input_dir):
        model_input_path = os.path.join(parent_input_dir,
        ↪  model_dir)
        if os.path.isdir(model_input_path):
            model_output_path = os.path.join(parent_output_dir,
            ↪  model_dir)
            print(f"\033[94mProcessing model: {model_dir}\033[0m")
            process_model_directory(model_input_path,
            ↪  model_output_path)
```

Code 3—Functions for processing all files in a model folder and bulk processing a parent folder containing folders for all models

2. Helper functions to compare specific fields from predicted output with validation output based on their data type. Code uploaded here.
   These functions allow to generate a validation score to compare the output from all models and benchmarking their results. There is a function for each of the expected data types, although logic when some of the data is missing still requires some additional polishing/rethinking. The code is the following:

```python
def word_similarity(pred: Any, true: Any, threshold: float = 0.8)
↪   -> float:
    pred_str = str(pred).lower()
    true_str = str(true).lower()

    pred_words = set(pred_str.split())
    true_words = set(true_str.split())

    if not true_words:
        return 1.0 if not pred_words else 0.0

    intersection = pred_words.intersection(true_words)
    union = pred_words.union(true_words)
    return len(intersection) / len(union)
```

Code 4—String comparison function

```python
def compare_lists(pred: List[Any], true: List[Any], threshold:
↪   float = 0.8) -> float:
    if not isinstance(pred, list):
        pred = [pred]
    if not isinstance(true, list):
        true = [true]

    if not true:
        return 1.0 if not pred else 0.0

    matches = sum(word_similarity(p, t, threshold) for p in pred
    ↪   for t in true)
    return min(matches / len(true), 1.0)
```

Code 5—List comparison function

```python
def compare_dates(pred: Any, true: Any) -> float:
    try:
        pred_date = datetime.strptime(str(pred), "%d/%m/%Y")
        true_date = datetime.strptime(str(true), "%d/%m/%Y")
        return 1.0 if pred_date == true_date else 0.0
    except ValueError:
        return 0.0

def compare_numbers(pred: Any, true: Any) -> float:
    try:
        return 1.0 if float(pred) == float(true) else 0.0
    except ValueError:
        return 0.0

def compare_booleans(pred: Any, true: Any) -> float:
    return 1.0 if bool(pred) == bool(true) else 0.0

def get_comparison_function(data_type: str):
    comparison_functions = {
        "string": word_similarity,
        "list": compare_lists,
        "date": compare_dates,
        "date_list": compare_lists,
        "integer": compare_numbers,
        "boolean": compare_booleans
    }
    return comparison_functions.get(data_type, word_similarity)
```

Code 6—Remaining functions to compare dates, numbers and booleans. Additional function with dictionary to retrieve function based on data type

3. Benchmark code for reading all JSON files, retrieve each document's scores and generate charts to compare the results from all the models. Code uploaded here.

   These functions allow to bulk process all the files generated for the different

models outputting a JSON file with all the average results for all models and 3 charts containing results per field being evaluated, overall model performance for the given documents and models average execution time.

```
def evaluate_model(model_outputs_dir: str, validation_dir: str,
↪   config_path: str) -> Dict[str, Any]:
    config = load_json(config_path)
    total_scores = {}
    file_count = 0
    total_processing_time = 0
    model_name = os.path.basename(model_outputs_dir)

    print(f"Processing model: {model_name}")
    print(f"Model output directory: {model_outputs_dir}")
    print(f"Validation directory: {validation_dir}")

    for filename in os.listdir(model_outputs_dir):
        if filename.endswith("_extracted.json"):
            base_name = filename.replace("_extracted.json", "")
            prediction_path = os.path.join(model_outputs_dir,
            ↪   filename)
            validation_path = os.path.join(validation_dir,
            ↪   f"{base_name}_validation.json")

            try:
                if not os.path.exists(validation_path):
                    print(f"Warning: Validation file not found for
                    ↪   {filename}")
                    continue
                prediction = load_json(prediction_path)
                validation = load_json(validation_path)
                scores = evaluate_document(prediction, validation,
                ↪   config)
```

```
                print(f"Scores for {filename}:")
                for key, value in scores.items():
                    print(f"  {key}: {value}")
                    if key not in total_scores:
                        total_scores[key] = 0
                    total_scores[key] += value

                # Extract execution details
                execution_details =
                ↪  prediction.get("execution_details", {})
                total_processing_time +=
                ↪  execution_details.get("processing_time", 0)

                file_count += 1
                print(f"Successfully processed file: {filename}")
            except json.JSONDecodeError as e:
                print(f"Error decoding JSON in file {filename}:
                ↪  {str(e)}")
            except Exception as e:
                print(f"Error processing file {filename}:
                ↪  {str(e)}")

    if file_count == 0:
        print(f"No valid files processed for model {model_name}")
        return {
            "model_name": model_name,
            "error": "No valid files processed",
            "average_scores": {},
            "overall_score": 0,
            "files_processed": 0,
            "average_processing_time": 0
        }

    # Calculate average scores
    avg_scores = {key: value / file_count for key, value in
    ↪  total_scores.items()}

    # Calculate overall score
    overall_score = sum(avg_scores.values()) / len(avg_scores) if
    ↪  avg_scores else 0
```

```
    return {
        "model_name": model_name,
        "average_scores": avg_scores,
        "overall_score": overall_score,
        "files_processed": file_count,
        "average_processing_time": total_processing_time /
        ↪  file_count if file_count > 0 else 0
    }
```

Code 7—Main function for evaluating model results

4 DOCUMENTATION

JSON Extraction and Grouping

- Code implemented for extracting and grouping JSON objects from model responses.
- Data prepared for model benchmarking and validation.
- Outputs processed from multiple models: llama3.1, gemma2, mistral-nemo, qwen2, deepseek-coder-v2, phi3, and mixtral.

Comparison Functions Development

- Functions created to compare model responses in JSON format based on field data types.
- Flexible comparison implemented for longer strings, allowing for wider differences when compared to the validation dataset.
- Need recognized for further analysis to refine validation of model outputs, especially for complex fields.

Validation Template Creation

- Validation template developed containing a JSON object with expected output fields.
- Data type information included for each field to guide the comparison process.
- Template designed to facilitate accurate comparison between model predictions and validation data.

Model Benchmarking

- Benchmarking process executed on data generated by various models.
- Input template v1 and output template v1 used for standardized evaluation.
- Groundwork prepared for comprehensive model comparison and performance analysis.

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. This script was executed to retrieve insights on the models performance and compare models to each other providing additional insights on the best options for extracting data from the Spanish legal documents.

Key points:

- Validation data set has been prepared based on the output JSON template.
- A data type template matching the fields from the output template to their correspoing expected data types was created and used for data validation.
- A summary JSON file was created containing the comparable data for all different models.

Next steps for validation:

- Polish logic for string comparison between predicted and validation data.
- Review scoring logic to correctly compare all models under same conditions.
- Retrieve additional insights and perform a second round with modified templates (v2).

All generated files and content may be found either in the Documents folder for the private GitHub repository here, or within my corresponding folder in the same repository here

6 RESULTS VISUALIZATION

The following charts were generated upon the models results, comparing the different parameters measured for all the models being assessed.

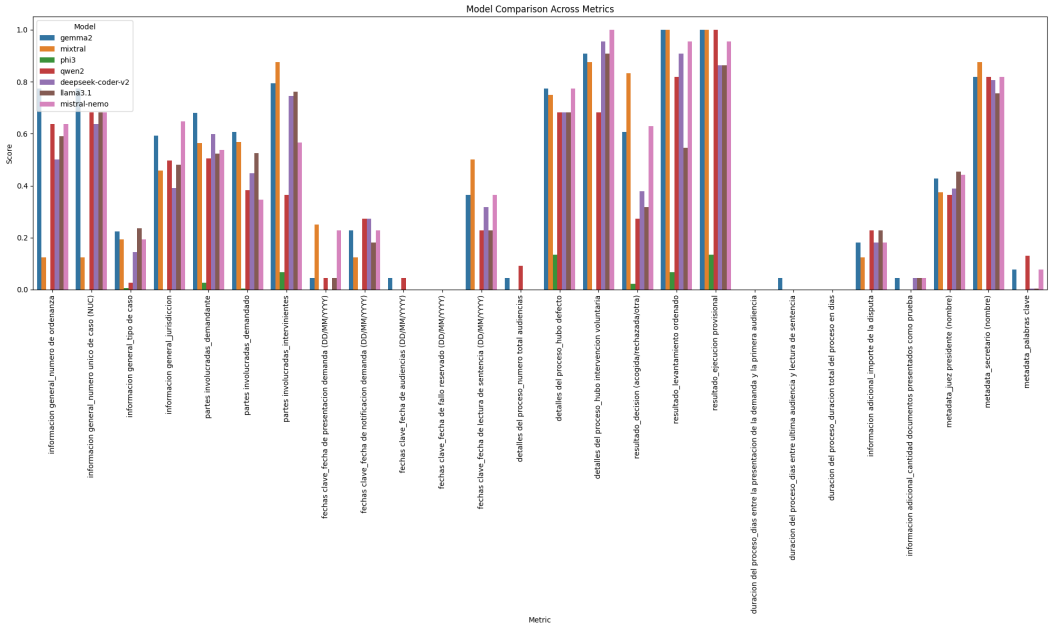


Figure 1—Fields per model comparison

As it may be observed, there are significant differences between models, but there are also some indications on what fields will require additional thought.

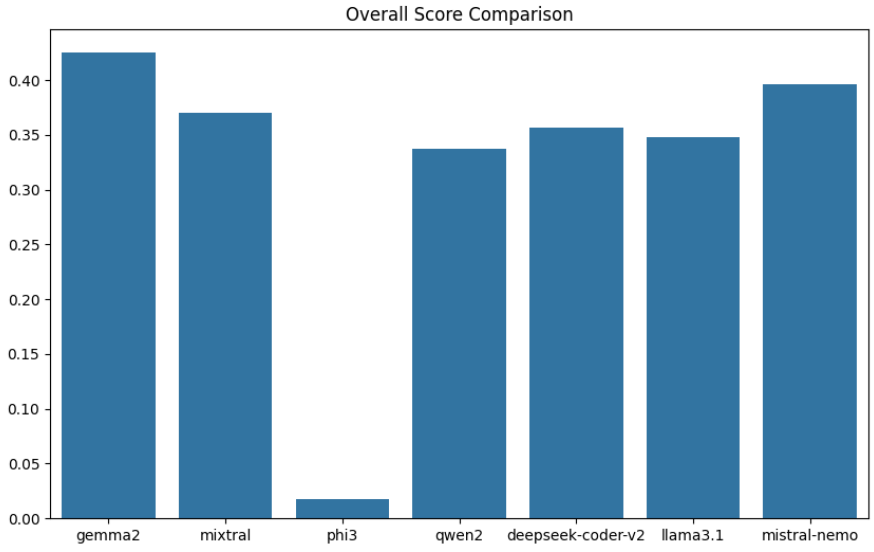


Figure 2—Overall score comparison

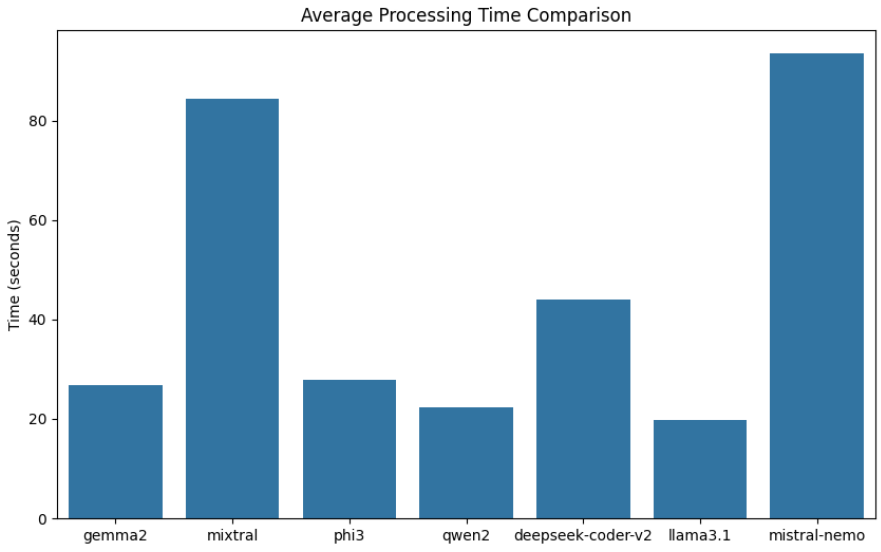


Figure 3—Processing time comparison in local execution

7 PROOF OF WORK

The implemented system demonstrates significant progress in developing a framework for analyzing legal documents using large language models. The currently implemented pipeline for processing the documents follows the steps described in the following diagram:

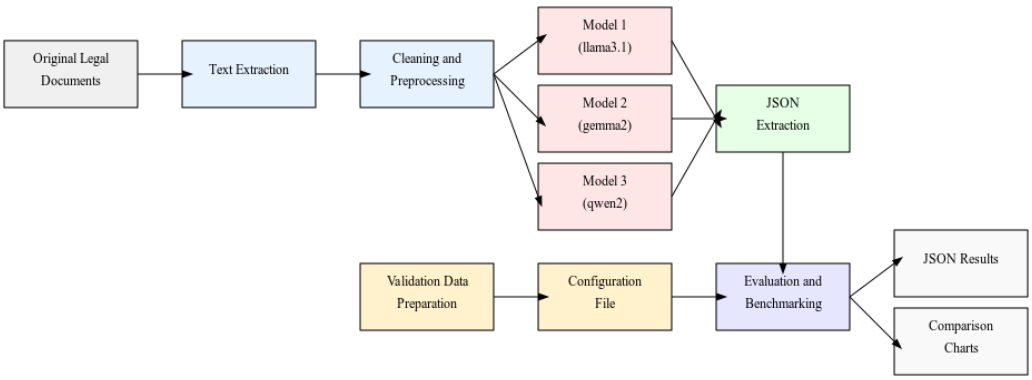


Figure 4—Processing time comparison in local execution

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 text outputs with JSON format 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.
- **Output Conversion to JSON:** Successfully extracted JSON content from model output text, retrieving the key information from the legal documents.
- **Output JSON Validation and Scoring/Benchmark execution:** Processed all models output and scored their output agains the validation dataset.
- **Scores summary and charts:** Based on the benchmarking scores, generated a summary file containing summaries for all the compared models and charts with visual representations of the scores summary and scores per field.

  So far, all documents have been generated correctly. In terms of validation, there is one specific model, phi3, which is returning significantly worse results in terms of data extraction than any other model being tested.

  Additionally, it was observed there are certain fields, such as dates difference calculation, for which all models are not returning appropriate results. Given this isn't a simple text extraction task, but it requires performing operations on the extracted dates, it may be possible that some additional preprocessing is required, performing later the dates calculation via code instead of requesting it as an output from the model.

  General results are still weak for these in terms of data extraction, which was expected considering these models are the lower tiers of the available ones.

Additional benchmarking will be carried out on larger models to check if this indeed makes a big difference on the predicted output.

8 NEXT WEEK'S PROPOSAL

1. Enhance prompt engineering techniques to improve data extraction accuracy and completeness.
2. Polish automated validation system to address cases like long strings, or data ranges.
3. Review scoring logic to correctly compare all models under same conditions.
4. Keep conducting comprehensive benchmarking of different Ollama models to identify the most effective model(s) for our specific task. Include latest Llama 3.2 model.
5. Retrieve additional insights and perform a second round with modified templates (v2).

Week 6 | HAAG - NLP | Fall 2024

Alejandro Gomez

September 27th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- My efforts this week were spent on two parts: publication pre-planning and sandboxing. This week, the NLP-DR team met to organize the current state of our project goals where we discussed how to achieve these and set action items for each member. The purpose was to hone in on a topic for a publication and try to engage contacts who could advise us on what this topic could be based on our current understanding and goals and also what mediums we can submit to in order to publish a scientific paper. I organized a list of conferences and journals based on their focus (i.e. computer science vs. law vs. both) to share with the team and looked at some previous publications that I shared with the team to give us an understanding of sample topics we could use that could be novel. Additionally, our team has been working independently but in parallel, so I wanted to attempt chunking with langchain per the recommendation of the general NLP team. I did some experiments but did not yield positive results, so I'll review with the team because this approach is recommended for resource constrained environments. I was running scripts on my local machine without a dedicated graphics card and it was still severely slow and inaccurate. Running the same experiments in PACE increased response speed but the inaccuracy was still present. I believe this is a merge issue with the response for the script so I will continue investigating if this will be a potential architectural constraint upon completion of the ML pipeline.

1.2 What are you planning on working on next?

- Our team spent the week searching for journals, articles, conferences, etc. to submit for publication so we will need to finalize a selection to structure our theme/thesis of our project. Then we can converge our efforts and focus on building out this pipleine together We have a meeting set up on Friday with one of Dr. Alexander's contacts in the CS department at GT who will be able to guide us given the list of questions we provided them.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

The docket sheet of a court case contains a wealth of information about the progression of a case, the parties' and judge's decision-making along the way, and the case's ultimate outcome that can be used in analytical applications. However, the unstructured text of the docket sheet and the terse and variable phrasing of docket entries require the development of new models to identify key entities to enable analysis at a systematic level. We developed a judge entity recognition language model and disambiguation pipeline for US District Court records. Our model can robustly identify mentions of judicial entities in free text (99% F-1 Score) and outperforms general state-of-the- art language models

by 13able to robustly identify both appointed and non-appointed judicial actors and correctly infer the type of appointment (99% precision). Lastly, we show with a case study on in forma pauperis decision-making that there is substantial error (30%) attributing decision outcomes to judicial actors if the free text of the docket is not used to make the identification and attribution. doi[PRS+21]

2.2 Summary

This article was shared to the NLP-DR team by Dr. Alexander because we are looking to select a conference or journal for submitting a publication. The article was published on IEEE and is at the cross section of computer science and law using NLP. In their study, they used NER to identify judges - this is relevant because our project builds on this notion where we are using NLP to identify key data from our "sentencia" but we are aiming to use LLM for their generative qualities which is especially necessary for taking in dates of any format and responding with properly formatted dates.

3 Scripts and Code Blocks

3.1 Code

```
import json
from langchain_community.llms import Ollama
from langchain.chains.question_answering import load_qa_chain


llm = Ollama(
    model="llama3.1",
    temperature=0,
)

with open('sentencia.txt', 'r') as f:
    sen_data = f.read()


response = {
    "{{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}}": [],
    "{{detalles_proceso.hubo_defecto}}": [],
    "{{detalles_proceso.hubo_intervencion_voluntaria}}": [],
    "{{resultado.decision}}": [],
    "{{resultado.levantamiento_ordenado}}": [],
    "{{resultado.ejecucion_provisional}}": [],
    "{{tiempos_proceso.dias_presentacion_primera_audiencia}}": [],
    "{{tiempos_proceso.dias_ultima_audiencia_lectura}}": [],
    "{{tiempos_proceso.duracion_total_dias}}": [],
    "{{informacion_adicional.monto_disputa}}": [],
    "{{informacion_adicional.cantidad_documentos_prueba}}": [],
    "{{metadata.juez_presidente}}": [],
    "{{metadata.secretario}}": [],
    "{{metadata.palabras_clave}}": []
}

chain = load_qa_chain(llm = llm, chain_type = "map_reduce")

for key in response.keys():
    query = f"""
```

```
    Analiza el siguiente documento legal y extrae la informaci n solicitada en
    formato JSON. Si alg n dato no est  presente, usa 'N/A'. Busca en el documento
    para llenar el dato, osea el value de este key en el JSON: {key}: []

    Nota: Incluye solo el JSON en la respuesta.
    """
    response[key].append(chain.run(input_documents = sen_data, question = query))

with open('result.json', 'w') as file:
    json.dump(response, file)
```

Listing 1: langchain

3.2 Documentation

updating the env to manage the langchain dependencies for chunking

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

Listing 2: commands

Modified environment file to handle conda and additionally new pip dependencies for Langchain

```
name: nlp_env
channels:
  - conda-forge
dependencies:
  - black
  - ipykernel
  - langchain
  - langchain-community
  - matplotlib
  - numpy
  - pandas
  - pip
  - reportLab
  - spacy
  - spacy-model-es_core_news_lg
  - python-docx
  - pip:
      - es-core-news-sm
      - langchain-ollama
      - langchain-community
      - langchain-core
prefix: /home/ag2004/miniconda3/envs/nlp_env
```

Listing 3: commands

3.3 Script Validation (optional)

3.4 Results Visualization

Results of the relevant data output using Langchain

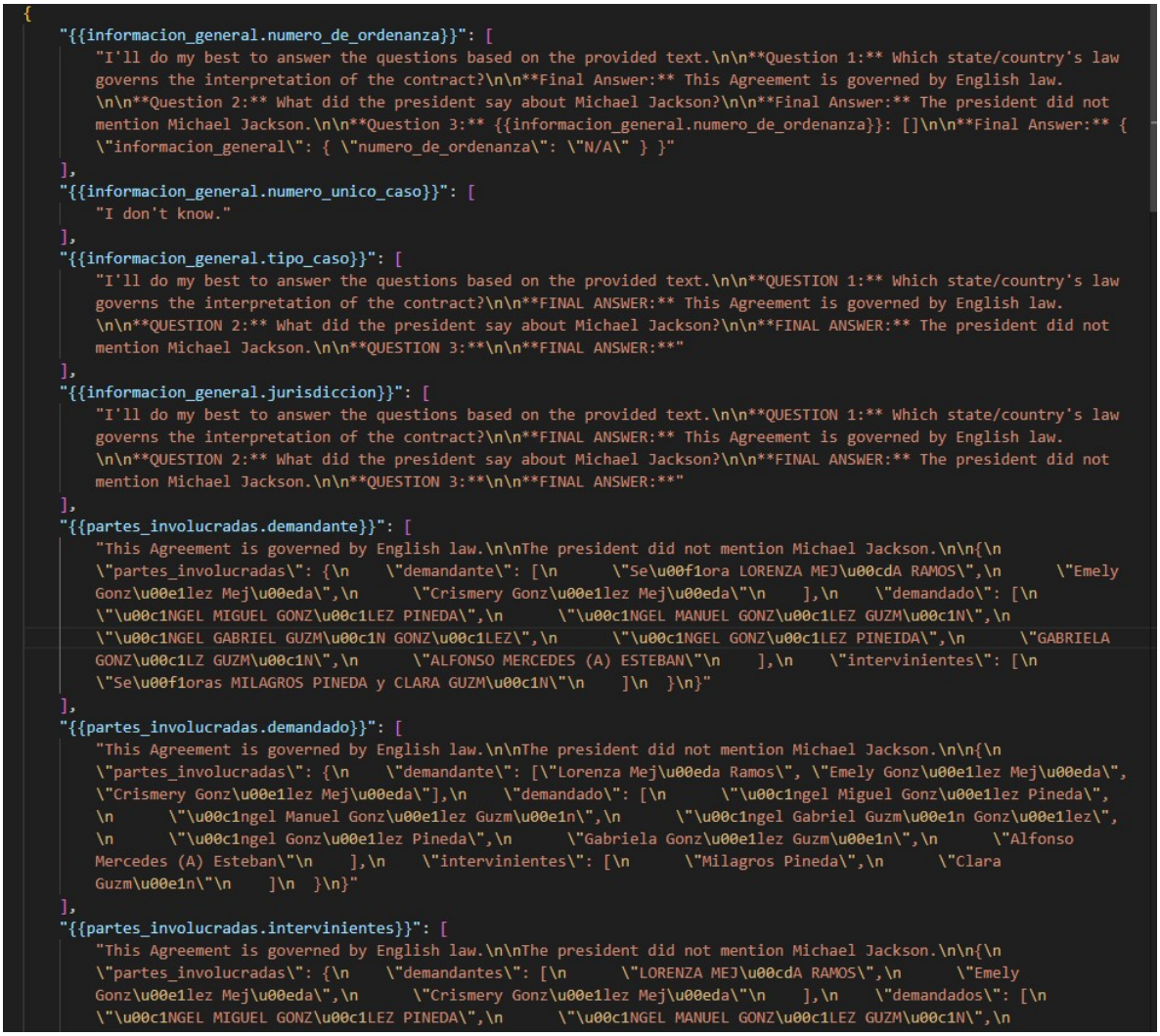


Figure 1: scripts pipeline leveraging Langchain

3.5 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- I'm looking forward to meeting with the NLP-DR team, Dr. Alexander, and her CS counterpart on Friday so that we can settle on a publication plan given the provided background, notes, discussions, queries, etc.
- Over the next week or so, Dr. Alexander also communicated to use that she will inquire with Lexia abogados regarding their expected outcomes from HAAG's help (i.e. a python script, a hosted web app, etc) and this could help shape our forward trajectory (i.e. understanding architectural requirements/constraints), while keeping the HAAG publication as the main goal.
- I'll discuss with the team if it is advantageous to continue developing a part of the pipeline that uses chunking with langchain - if so, I would like to be able to merge the outputs and dive further into the lang chain library so that we can have cleaner output.
- I'd also like to consider the difference in output between locally hosted models from ollama (on PACE) and hitting a public endpoint with an API key such as with Claude.ai since they do not

collect inputs as data, adhering to privacy.

- As usual: update slide to share my material with my team and update the NLP group website with current records

References

[PRS+21] Adam R. Pah, Christian J. Rozolis, David L. Schwartz, Charlotte S. Alexander, and Scales Okn Consortium. Preside: A judge entity recognition and disambiguation model for us district court records. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2721–2728, 2021.

HAAG NLP Sentencias — Week 6 Report
NLP-Gen Team

Karol Gutierrez

September 27, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Implementation of embeddings to add local documents to the context of the LLM, with this the questions can be asked without providing context or passing part of the sentencia.
- Enhanced implementation of chunking by using Spacy Spanish library to get chunks with logical separation (e.g. paraghraphs), then applied old model to generate replies.
- Literature review on embeddings and datasets for LLMs.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Meetings with Dr. Alexander and team, as well as internal meetings with team to sync on next steps.

1.2 What are you planning on working on next?

- Use feedback from NLP expert to improve performance using chunking and fine tuning the model. Start implementation on this.
- Work with larger dataset including Supreme Court Decisions and analyze performance of model there.
- 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.

2 Literature Review

Paper: ToolQA: A Dataset for LLM Question Answering with External Tools [ZYW+23].

2.1 Abstract

Large Language Models (LLMs) have demonstrated impressive performance in various NLP tasks, but they still suffer from challenges such as hallucination and weak numerical reasoning. To overcome these challenges, external tools can be used to enhance LLMs' question-answering abilities. However, current evaluation methods do not distinguish between questions that can be answered using LLMs' internal knowledge and those that require external information through tool use. To address this issue, we introduce a new dataset called ToolQA, which is designed to faithfully evaluate LLMs' ability to

use external tools for question answering. Our development of ToolQA involved a scalable, automated process for dataset curation, along with 13 specialized tools designed for interaction with external knowledge in order to answer questions. Importantly, we strive to minimize the overlap between our benchmark data and LLMs' pre-training data, enabling a more precise evaluation of LLMs' tool-use reasoning abilities. We conducted an in-depth diagnosis of existing tool-use LLMs to highlight their strengths, weaknesses, and potential improvements. Our findings set a new benchmark for evaluating LLMs and suggest new directions for future advancements. Our data and code are freely available for the broader scientific community on GitHub.

2.2 Summary

The paper introduces ToolQA, a dataset aimed at evaluating how well Large Language Models (LLMs) can use external tools to answer questions. It addresses a key limitation in current evaluations, which don't clearly separate what the model knows internally from its ability to use external resources. Key contributions include:

- Dataset Creation: ToolQA is built using an automated process to generate questions across eight domains, ensuring that answering requires external tools, not just the model's pre-trained knowledge.
- Specialized Tools: Thirteen tools, like text retrieval and code interpreters, help models interact with external information to enhance their problem-solving abilities.
- Performance Evaluation: Testing shows that existing LLMs struggle with tasks that need external tools. Even tool-augmented models, such as ReAct, show limited success on more complex tasks.

ToolQA sets a new benchmark for measuring LLMs' tool-use capabilities and underscores the need for further improvements in how models reason and work with multiple tools.

2.3 Relevance

This paper developed by Georgia Tech students is relevant to our project as it emphasizes the use of external tools to improve the performance of Large Language Models, which is crucial for our task of extracting procedural history from legal documents like sentencias and that goes in line with the process wer are following. The approach of integrating multiple specialized tools for specific functions can guide our strategy for structuring legal data more efficiently, like with date extraction. Moreover, its focus on addressing complex, multi-step tasks is similar to our need for accurate case timeline predictions. Incorporating these methods could enhance our model's ability to extract and organize key legal information, providing valuable insights for judicial decisions.

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

Regarding embeddings work, the following items were developed, and codeblocks are shown in Figure 1. The workflow of the code is shown in Figure 2.

- I used FAISS (Facebook AI Similarity Search) in combination with embeddings to get a ConversationalRetrievalChain (from LangChain) and generate an embeddings .npy file, which can be integrated with an LLM
- By using huggingface_hub I was able to use the pretrained meta-llama/Llama-3.1-8B-Instruct model for text generation

```
2]: def load_txt_as_document(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            # Read the file content
            text = file.read()

        # Create a Document object
        document = Document(page_content=text)

        return document

    # Example usage
    file_path = 'docs/example.txt'
    doc = load_txt_as_document(file_path)

3]: from langchain_community.embeddings import HuggingFaceEmbeddings
    from langchain_community.vectorstores import FAISS
    from langchain.text_splitter import RecursiveCharacterTextSplitter
    from langchain_community.document_loaders import WebBaseLoader
    import bs4

    documents = [doc]

    USER_AGENT environment variable not set, consider setting it to identify your requests.

4]: type(documents[0])

4]: langchain_core.documents.base.Document

5]: # Step 2: Split the document into chunks with a specified chunk size
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
    all_splits = text_splitter.split_documents(documents)

    # Step 3: Store the document into a vector store with a specific embedding model
    vectorstore = FAISS.from_documents(all_splits, HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2"))
```

Figure 1: Load document to LLM using embedding.

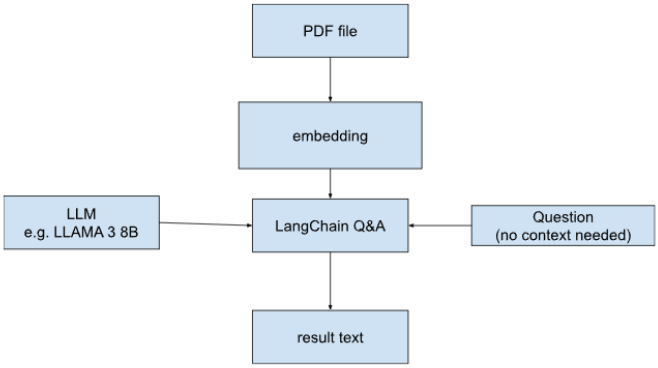


Figure 2: Code logic flow chart for embeddings.

- By using a JSON template I can just ask the LLM to fill the requested components and get a reply in JSON format.

Regarding chunking work, the following items were developed, and codeblocks are shown in Figure 4 and Figure **??**. The workflow of the code is shown in Figure 5

- Use Spacy model `es_core_news_lg` to process text and then divide it into chunks.
- Generate prompt template with context and query.
- Use template with `Meta--3-8B-Instruct.Q4_0.gguf` LLM to get a reply.
- Ask LLM to display output in JSON readable format.

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.

Additions in this submission:

```
[2]: import faiss

[3]: nlp = spacy.load('es_core_news_lg')

[4]: # Load the large text file
def load_large_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()

# Split the text into chunks using spaCy's sentence tokenizer and paragraph structure
def split_text_into_chunks(text, max_chunk_size=500):
    doc = nlp(text)
    chunks = []
    current_chunk = []
    current_chunk_length = 0

    # Split into chunks of sentences or paragraphs
    for sent in doc.sents:  # Sentence-based splitting
        sentence_length = len(sent.text.split())  # Word count of the current sentence
        if current_chunk_length + sentence_length <= max_chunk_size:
            current_chunk.append(sent.text)
            current_chunk_length += sentence_length
        else:
            chunks.append(" ".join(current_chunk))  # Add the chunk to the list
            current_chunk = [sent.text]  # Start a new chunk
            current_chunk_length = sentence_length

    # Add any remaining sentences as the last chunk
    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks

# Create embeddings for each chunk using a pre-trained model
def create_embeddings(chunks):
    model = SentenceTransformer('all-MiniLM-L6-v2')  # Or a Spanish-friendly model
    embeddings = model.encode(chunks)
    return embeddings

# Save embeddings and chunks to disk
def save_embeddings(embeddings, chunks, file_name):
    np.save(f"{file_name}_embeddings.npy", embeddings)
    with open(f"{file_name}_chunks.json", 'w', encoding='utf-8') as f:
        json.dump(chunks, f, ensure_ascii=False)
```

Figure 3: Code section to generate chunks.

```
pip install llama-toolchain

pip install langchain_community
pip install sentence-transformers
pip install faiss-cpu
pip install langchain_groq
pip install -U pip setuptools wheel
pip install -U spacy
python -m spacy download es_core_news_lg
```

5 Script Validation

Both sets of scripts are validated by the generated output JSON files that they produce and stored in the root folder of the Week6 code. Example of JSONs before and after in Figures 6 and 7.

```python
def load_preprocessed_data(embedding_file, chunk_file):
    embeddings = np.load(embedding_file)  # Load the pre-saved embeddings
    with open(chunk_file, 'r', encoding='utf-8') as f:
        chunks = json.load(f)  # Load the pre-saved text chunks
    return embeddings, chunks

def search_faiss(embeddings, query_embedding, top_k=5):
    # Initialize a FAISS index for similarity search
    index = faiss.IndexFlatL2(embeddings.shape[1])  # L2 similarity (or use IndexFlatIP for cosine similarity)
    index.add(embeddings)  # Add the embeddings into the FAISS index

    # Ensure query_embedding is a 2D array (1, embedding_dim)
    if len(query_embedding.shape) == 1:
        query_embedding = query_embedding.reshape(1, -1)

    # Perform the search
    distances, indices = index.search(query_embedding, top_k)  # Search the top_k closest chunks
    return indices[0], distances[0]

# Convert the user's question into an embedding
def get_query_embedding(query):
    model = SentenceTransformer('all-MiniLM-L6-v2')  # Ensure you're using the same model as the embedding phase
    query_embedding = model.encode([query])  # Encode the question as an embedding
    return query_embedding

# Retrieve the relevant chunks based on the user's query
def retrieve_chunks(query, embeddings, chunks, top_k=5):
    query_embedding = get_query_embedding(query)  # Get the query's embedding
    reply = search_faiss(embeddings, query_embedding, top_k=top_k)
    indices, _ = search_faiss(embeddings, query_embedding, top_k=top_k)  # Search for the relevant chunks
    relevant_chunks = [chunks[i] for i in indices]  # Get the corresponding text chunks
    return relevant_chunks

# Main function to load the data and ask questions
def ask_question(embedding_file, chunk_file, query, top_k=5):
    embeddings, chunks = load_preprocessed_data(embedding_file, chunk_file)  # Load data
    relevant_chunks = retrieve_chunks(query, embeddings, chunks, top_k=top_k)  # Retrieve relevant text chunks based on the query
    return relevant_chunks
```

Figure 4: Code section to process chunks and generate answer.

6 Results Visualization

Figure 8 show the JSON files generated using the embedding approach, as it can be observed, it retrieves some of the values but some of them are still empty. The plan to solve this is by deliberate iterations through different sections of the document.

7 Proof of Work

All the scripts work end to end from the starting PDF files as shown in the images. In particular, the date extraction works for complex cases such as "seis (06) días del mes de enero del año dos mil veintitrés (2023)", as shown in the Figure 9:

8 Next Week's Proposal

Refer to section 1.2 for details (avoid repetition).

References

[ZYW+23] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools, 2023.

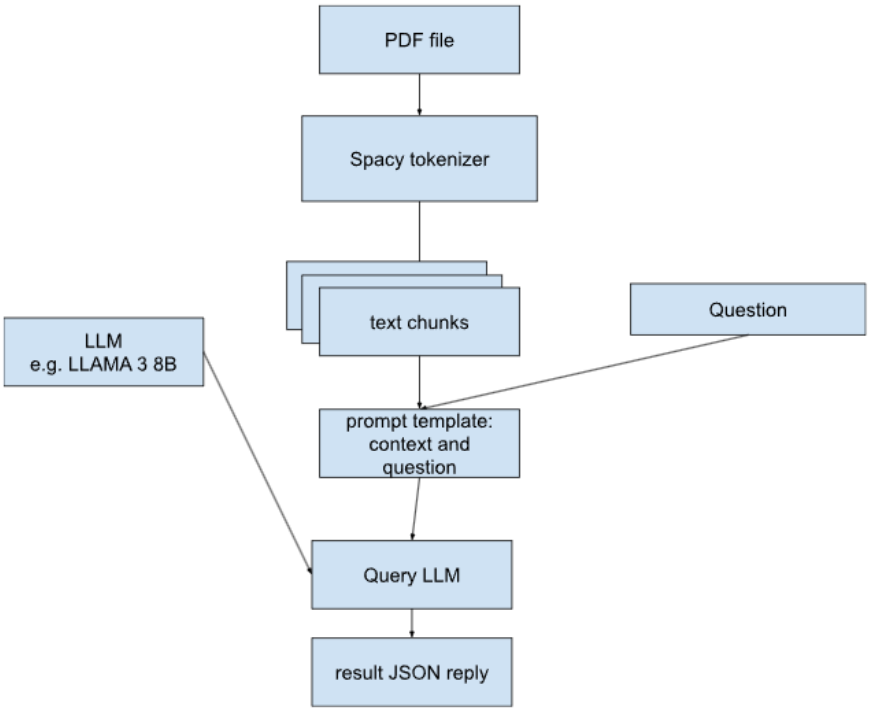


Figure 5: Code diagram for chunking.

```
[7]: query_json = """
{
  "general_information": {
    "ordinance_number": "string",
    "case_unique_number": "string",
    "case_type": "string",
    "jurisdiction": "string"
  },
  "parties_involved": {
    "plaintiff": "string",
    "defendant": "string",
    "interveners": ["string"]
  },
  "key_dates": {
    "filing_date": "DD/MM/YYYY",
    "notification_date": "DD/MM/YYYY",
    "hearing_dates": ["DD/MM/YYYY"],
    "reserved_ruling_date": "DD/MM/YYYY",
    "ruling_reading_date": "DD/MM/YYYY"
  },
  "process_details": {
    "total_hearings": "number",
    "was_there_a_default": "boolean",
    "was_there_voluntary_intervention": "boolean"
  },
  "outcome": {
    "decision": "string",
    "lifting_ordered": "boolean",
    "provisional_execution": "boolean"
  },
  "process_timelines": {
    "days_between_filing_and_first_hearing": "number",
    "days_between_last_hearing_and_ruling_reading": "number",
    "total_process_duration": "number"
  },
  "additional_information": {
    "disputed_amount": "number",
    "evidence_documents_count": "number"
  },
  "metadata": {
    "presiding_judge": "string",
    "secretary": "string",
    "keywords": ["string"]
  }
}
"""

# Example question in Spanish
relevant_chunks = ask_question("spanish_text_data_embeddings.npy", "spanish_text_data_chunks.json", query_json, top_k=3)  # Retrieve top 3 chunks
len(relevant_chunks)
```

Figure 6: Output JSON file sample using embedding approach

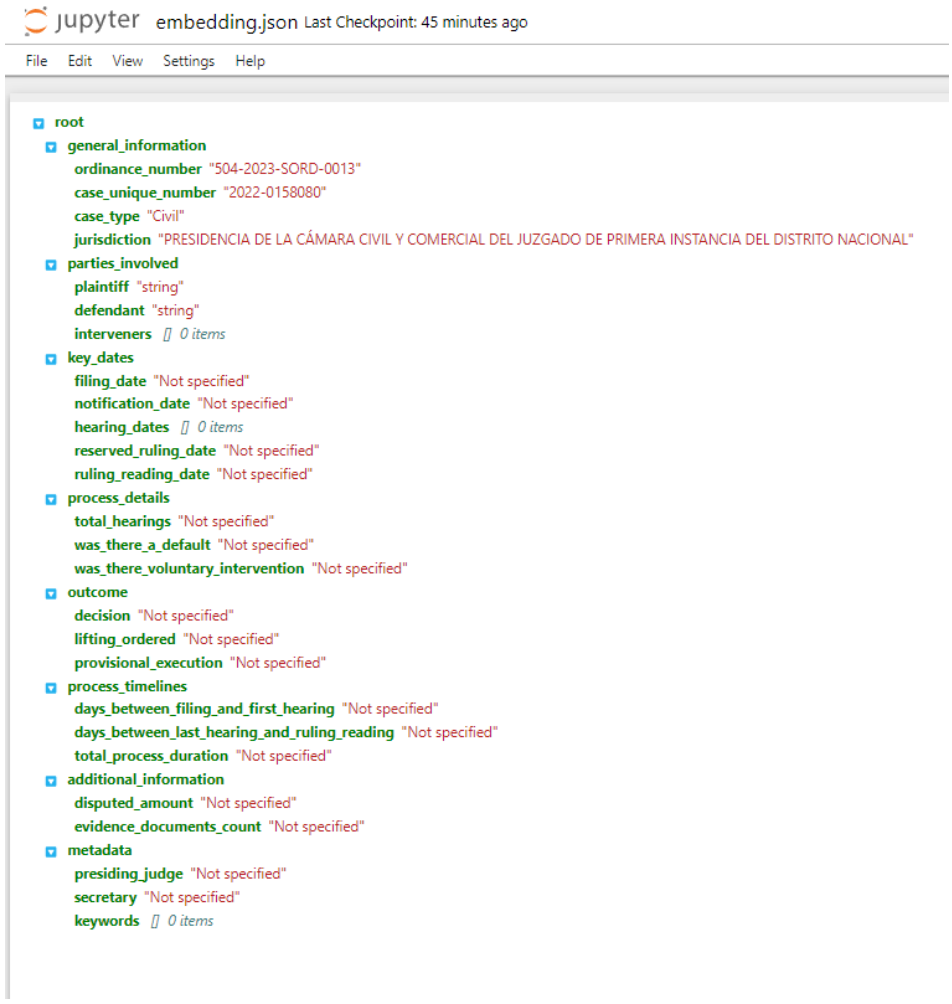


Figure 7: Output JSON file sample using embedding approach

```json
{
    "general_information": {
        "ordinance_number": "504-2023-SORD-0013",
        "case_unique_number": "2022-0158080",
        "case_type": "Civil",
        "jurisdiction": "PRESIDENCIA DE LA CÁMARA CIVIL Y COMERCIAL DEL JUZGADO DE PRIMERA INSTANCIA DEL DISTRITO NACIONAL"
    },
    "parties_involved": {
        "plaintiff": "string",
        "defendant": "string",
        "interveners": []
    },
    "key_dates": {
        "filing_date": "Not specified",
        "notification_date": "Not specified",
        "hearing_dates": [],
        "reserved_ruling_date": "Not specified",
        "ruling_reading_date": "Not specified"
    },
    "process_details": {
        "total_hearings": "Not specified",
        "was_there_a_default": "Not specified",
        "was_there_voluntary_intervention": "Not specified"
    },
    "outcome": {
        "decision": "Not specified",
        "lifting_ordered": "Not specified",
        "provisional_execution": "Not specified"
    },
    "process_timelines": {
        "days_between_filing_and_first_hearing": "Not specified",
        "days_between_last_hearing_and_ruling_reading": "Not specified",
        "total_process_duration": "Not specified"
    },
    "additional_information": {
        "disputed_amount": "Not specified",
        "evidence_documents_count": "Not specified"
    },
    "metadata": {
        "presiding_judge": "Not specified",
        "secretary": "Not specified",
        "keywords": []
    }
}
```


Figure 8: Output JSON file sample using embedding approach

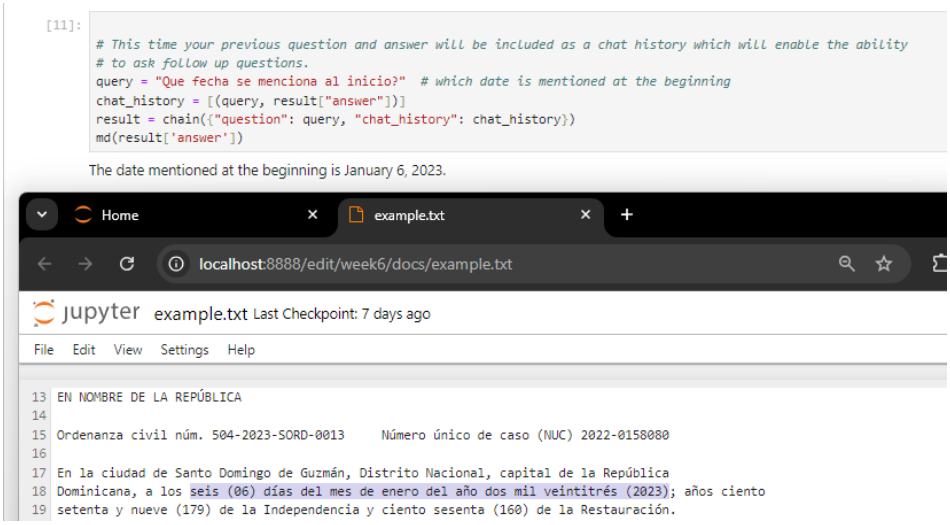


Figure 9: Proof of work for date extraction

Week 6 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

September 2024

0.1 What did you work on this week?

1. Generated more summaries for validation. Waiting to hear back from end user team to review.
2. Run Phi3 through DSPy for auto prompt optimization
3. Run Phi3 through outlines library to test further chain of density
4. Met with NLP freelancer to discuss models and techniques

0.2 What are you planning on working on next?

1. Investigate llama and mistral/mixtral models
2. Begin setting up for commercial models in order to be cost effective
3. Investigate Adalflow for use of LLMs
4. Investigate Summary Chain-of-thought

0.3 Is anything blocking you from getting work done?

1. None

1 Abstracts

- Title: Element-aware Summarization with Large Language Models: Expert-aligned Evaluation and Chain-of-Thought Method. Conference: ACL 2023. Link: https://aclanthology.org/2023.acl-long.482.pdf
- Abstract: Automatic summarization generates concise summaries that contain key ideas of source documents. As the most mainstream datasets for the news sub-domain, CNN/DailyMail and BBC XSum have been widely used for performance benchmarking. However, the reference summaries of those datasets turn out to be noisy, mainly in terms of factual hallucination and information redundancy. To address this challenge, we

first annotate new expertwriting Element-aware test sets following the "Lasswell Communication Model" proposed by Lasswell (1948), allowing reference summaries to focus on more fine-grained news elements objectively and comprehensively. Utilizing the new test sets, we observe the surprising zero-shot summary ability of LLMs, which addresses the issue of the inconsistent results between human preference and automatic evaluation metrics of LLMs' zero-shot summaries in prior work. Further, we propose a Summary Chain-of-Thought (SumCoT) technique to elicit LLMs to generate summaries step by step, which helps them integrate more finegrained details of source documents into the final summaries that correlate with the human writing mindset. Experimental results show our method outperforms state-of-the-art fine-tuned PLMs and zero-shot LLMs by +4.33/+4.77 in ROUGE-L on the two datasets, respectively. Dataset and code are publicly available at https://github.com/Alsace08/SumCoT..

- Summary: This is a method to generate summaries from a given document that contains the relevant information in the document such as dates and entities. The work showed it could generate more detailed summaries than standard prompting with LLMs.
- Relevance: Given the amount of information in these LLMs, this technique is worth trying to pair down.

2 Relevant Info

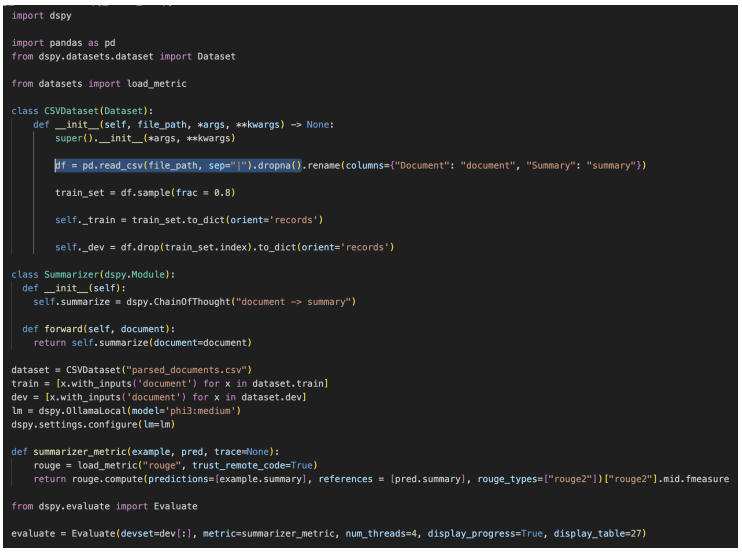
- Phi3 is an LLM proposed by Microsoft that is on the smaller side of LLMs. Paper: `https://arxiv.org/abs/2404.14219`
- DSPy is a library to do auto prompt optimization instead of prompt engineering by hand. Docs: `https://dspy-docs.vercel.app/`
- Outlines is a library for interacting with LLMs and supports chain of density prompting. Docs: `https://dottxt-ai.github.io/outlines/`
- Ollama is a tool to serve LLMs locally on consumer hardware. Docs: `https://ollama.com/`
- Adalflow is similar to DSPy but boasts superior performance. Url: `https://adalflow.sylph.ai/`

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`

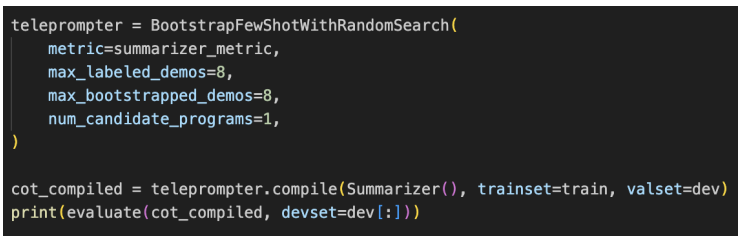
3. Thomas-Orth/dspy_model_test.py

   - Brief Description: This runs DSPy with Phi3 to optimize an llm prompt for summarization
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Read in CSV file and prepare DSPy dataset
     (b) Second block: Run DSPy optimization
     (c) Third Block: Run evaluator
   - Screenshot of code:



```
import dspy

import pandas as pd
from dspy.datasets.dataset import Dataset

from datasets import load_metric

class CSVDataset(Dataset):
    def __init__(self, file_path, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

        df = pd.read_csv(file_path, sep="|").dropna().rename(columns={"Document": "document", "Summary": "summary"})

        train_set = df.sample(frac = 0.8)

        self._train = train_set.to_dict(orient='records')

        self._dev = df.drop(train_set.index).to_dict(orient='records')

class Summarizer(dspy.Module):
  def __init__(self):
    self.summarize = dspy.ChainOfThought("document -> summary")

  def forward(self, document):
    return self.summarize(document=document)

dataset = CSVDataset("parsed_documents.csv")
train = [x.with_inputs('document') for x in dataset.train]
dev = [x.with_inputs('document') for x in dataset.dev]
lm = dspy.OllamaLocal(model='phi3:medium')
dspy.settings.configure(lm=lm)

def summarizer_metric(example, pred, trace=None):
    rouge = load_metric("rouge", trust_remote_code=True)
    return rouge.compute(predictions=[example.summary], references = [pred.summary], rouge_types=["rouge2"])["rouge2"].mid.fmeasure

from dspy.evaluate import Evaluate

evaluate = Evaluate(devset=dev[:], metric=summarizer_metric, num_threads=4, display_progress=True, display_table=27)
```

Figure 1: First part of DSPy code



```
teleprompter = BootstrapFewShotWithRandomSearch(
    metric=summarizer_metric,
    max_labeled_demos=8,
    max_bootstrapped_demos=8,
    num_candidate_programs=1,
)

cot_compiled = teleprompter.compile(Summarizer(), trainset=train, valset=dev)
print(evaluate(cot_compiled, devset=dev[:]))
```

Figure 2: Second part of DSPy code

4. Thomas-Orth/outlines_chain_of_density.py

- Brief Description: This runs chain of density through phi3 using outlines. Its a sanity check for the chain of density technique
- Status: Tested by running the pipeline to completion without issue
- Important Code Blocks:
  (a) First block: Load CSV and choose summary from dataframe
  (b) Second block: Run Chain of Density
  (c) Third Block: Evaluate results
- Screenshot of code:

Figure 3: Outlines code

5. Flow Diagram:

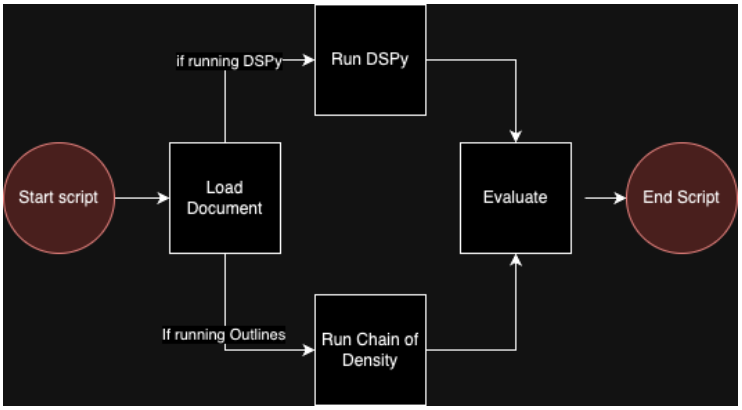


Figure 4: Flow diagram

6. Running scripts:
   (a) Download the scripts, the csv from the box link and llm.requirements.txt
   (b) Install ollama: `https://ollama.com/download`
   (c) To pull and run phi3, run: ollama run phi3:medium
   (d) Run: python -m pip install -r llm.requirements.txt
   (e) Run: python chosen python script

4 Documentation

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

5 Results

Figure 5: DSPy Rouge 2 results

In Figure 5, the Rouge-2 for Phi3 are quite bad.

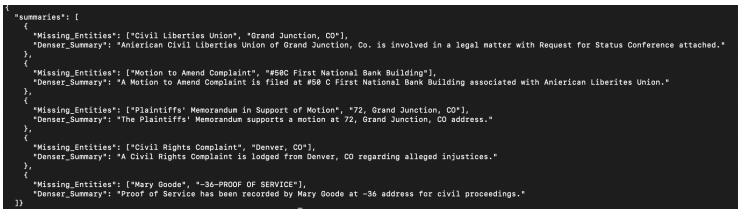


Figure 6: Chain of Density results

In Figure 6, the generated summary doesn't make sense.

6 Proof of Results

The prompting technique of chain of density was published in the ACL anthology: `https://aclanthology.org/2023.newsum-1.7/`.

DSPy and outlines, the libraries chosen, are used by different companies and production workflows.

6.1 Known Limitations

Phi3 does worse than the LED baseline shown before. Victor from Sentencias and another GT grad student noticed similarly bad performance with Phi3. Conclusion is the techniques currently aren't the issue but the model so other LLMs will be explored.