HAAG NLP Summarization Week 4

Michael Bock

September 2024

1 Slack Questions

What did you accomplish this week?

- Read Seq2Seq, Decoder Only Transformers, BERT, and Mistral
- Supplied my list of papers to read to undergraduates who were looking for more involvement in NLP tasking
- Supplied PDFs of Cases to OCR subteam
- Began coding a Seq2Seq example, but stopped due to task switch
- Began trying to get a BERT Masked Language Modeling example
- Switched over Text Classification Team working with COVID-19 Cases

What are you planning on working on next?

- Finish the Masked Language Modeling Example
- We currently don't have the COVID-19 data yet, but I want to get a classification baseline down with my BERT example. For that I found text classification datasets on paperswithcode

What is blocking you from progressing?

- Need clarification on problem type for covid dataset. I've been told its named entity recognition, but actually the way the covid data website has it it looks like document classification
- Need to meet with COVID-19 team at UPenn to ask questions(scheduled for 1 preliminary meeting this week and then another meeting with the whole Law Data and Design Team on 9/25.

2 Abstract

We introduce a new language representa- tion model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language repre- sentation models (Peters et al., 2018a; Rad- ford et al., 2018), BERT is designed to pre- train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a

re- sult, the pre-trained BERT model can be fine- tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task- specific architecture modifications. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art re- sults on eleven natural language processing tasks, including pushing the GLUE score to 80.5MultiNLI accuracy to 86.7improvement), SQuAD v1.1 question answer- ing Test F1 to 93.2 (1.5 point absolute im- provement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

Link: https://arxiv.org/pdf/1810.04805

2.1 Brief Analysis

In the BERT paper, they draw a line between feature based NLP methods and fine tuning NLP methods. Feature Based NLP methods are models who's parameters are specialized to a specific task. So for example, Pegasus is a summarization model, you can't really retrain it to work on text classification. Fine tuning based models have 2 tasks: a pretraining task and a fine tuning task. Fine tuning model's first learn a general language representation using a task like Masked Langauge Modeling, where a model must fill in the blank in a sentence, or Next Sentence Prediction, where the model must generate the next sentence in a sequence. These pretraining tasks are very easy to conduct in a self-supervised manner, for example in masked langauge modeling we know what word we took out so we can make that word the label. That means that fine tuning based models can take advantage of large, unlabeled datasets, before fine tuning on smaller, task specific datasets.

At the core of BERT's contribution is that they use the *same* model for many different downstream fine tuning tasks. BERT used Masked Langauge Modeling to train a transformer, and then took the MLM head off that the transformer and replaced it with task specific heads. Then, they froze the weights of the transformer to keep the language representation while training the head to do the task. Using this approach, BERT was able to achieve SoTA results on 11 NLP tasks, all with one backbone and limited labeled data.

In our tasks we have several things to consider with BERT. First, BERT can be used on our classification tasking in multiple ways, but we may want to change things about the BERT model to make it specific to legal NLP. This means we may need to re-run Masked Language Modeling on BERT for legal NLP. I've found well OCR'd unlabeled legal text datasets for this purpose. At the same time, we may not want to trust the weights of BERT. Names and places in a legal document may be long. Long names are bad for BERT's tokenizer, which relies on merging commonly associated tokens together. I believe our best shot at improving NER or text classification(they are really nearly the same thing) is to "bully" the tokenizer into including certain strings by pre-populating it with names of attorneys, judges, hospitals, insurance terms, laws, and issues.

Another item of note is joint training. Last week I discussed how RAG reminded me of extractive-abstractive summarization approaches. NER and text classification tasks are a type of text extraction. If we desired, we'd be able to include the NER/Text Classification output as an input to a summarization model to help it be more factual. Taking this idea further, it may be worth it for the team to investigate training our text classifier and our summarization model jointly. This joint training of the retriever and a decoder was prescribed by the RAG paper.

3 Scripts and Code Blocks

My Seq2Seq and Masked Language Modeling examples aren'y yet finished. However here is what I currently have for them:

mistral_datasets.py:

```
import torch
from torch.utils.data import Dataset
from datasets import load_dataset
import re
from transformers import AutoTokenizer
import string
from langdetect import detect
import random

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)

class MistralMLMDataset(Dataset):
    def __init__(self, tokenizer, split = 'train', text_len = 24):
        """
        Args:
            split (list or ndarray): "train" or "val".
        """
        self.dataset = load_dataset("pile-of-law/pile-of-law", "nlrb_decisions")
        self.dataset = self.dataset[split]
        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']
        while detect(data_sample) != 'en':
            idx += 1
            data_sample = self.dataset[idx%len(self.dataset)]['text']
        tokens = self.tokenizer(data_sample, return_tensors='pt')
        print(tokens)
        #data_sample = data_sample.split()
```

```
        text_index = random.randrange(0, len(data_sample) - self.text_len + 1)
        data_sample = data_sample[text_index: text_index + self.text_len]
        data_sample = data_sample
        return data_sample

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

    # Create dataset
    dataset = MistralMLMDataset(AutoTokenizer.from_pretrained("distilbert/distilbert
    -base-uncased"))

    # DataLoader for batching and shuffling
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=2, shuffle=False)

    # Iterate through the DataLoader
    for batch_data in dataloader:
        #print(batch_data)
        quit()
```

mistral.py:

```
from transformers import AutoTokenizer, AutoModelForTokenClassification
from torch import nn
from transformers import MistralConfig, MistralModel
import torch

#bidirectional mistral class
class BiMistral(nn.Module):
    def __init__(self, num_entities, num_blocks, hidden_size):
        super(BiMistral, self).__init__()
        self.mistral = MistralModel(MistralConfig(num_hidden_layers = num_blocks,
    hidden_size = hidden_size))
        print('Decoder built: ', self.mistral)
        self.classifier = nn.Linear(hidden_size, num_entities)

    def forward(self, x, labels = None):
        x = self.mistral(x)[0]
        logits = self.classifier(x)
        if labels is not None:
            loss = torch.nn.cross_entropy(logits, labels)
            return (loss, logits)
        return logits


if __name__ == '__main__':
    model = BiMistral(5, num_blocks = 4, hidden_size = 16)
    input = torch.LongTensor([[1, 2, 4]])
    print(model(input), model(input).shape)
```

For the pdf extraction, this was already coded in previous weeks, we just never had any cause to run it the way we did this week so I will present that code again: generate_pdfs.py:

```
from summarizers.get_complaints import get_complaint_only_cases
from summarizers.ocr import read_doc, extract_text_from_pdf
from tqdm import tqdm
import pandas as pd
import os
```

```
from urllib.request import urlretrieve

os.mkdir('pdfs')

data = get_complaint_only_cases("all_cases_clearinghouse.pkl")

data_entries = []
for entry in tqdm(data):
    if not entry or not entry.case_documents or len(entry.case_documents) < 1:
        continue

    for i, case_doc in enumerate(entry.case_documents):
        #doc = entry.case_documents[-1]
        urlretrieve(case_doc.file, f'pdfs/{entry.id}_{i}.pdf')
```

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 script runs quickly this week.

6 Results Visualization

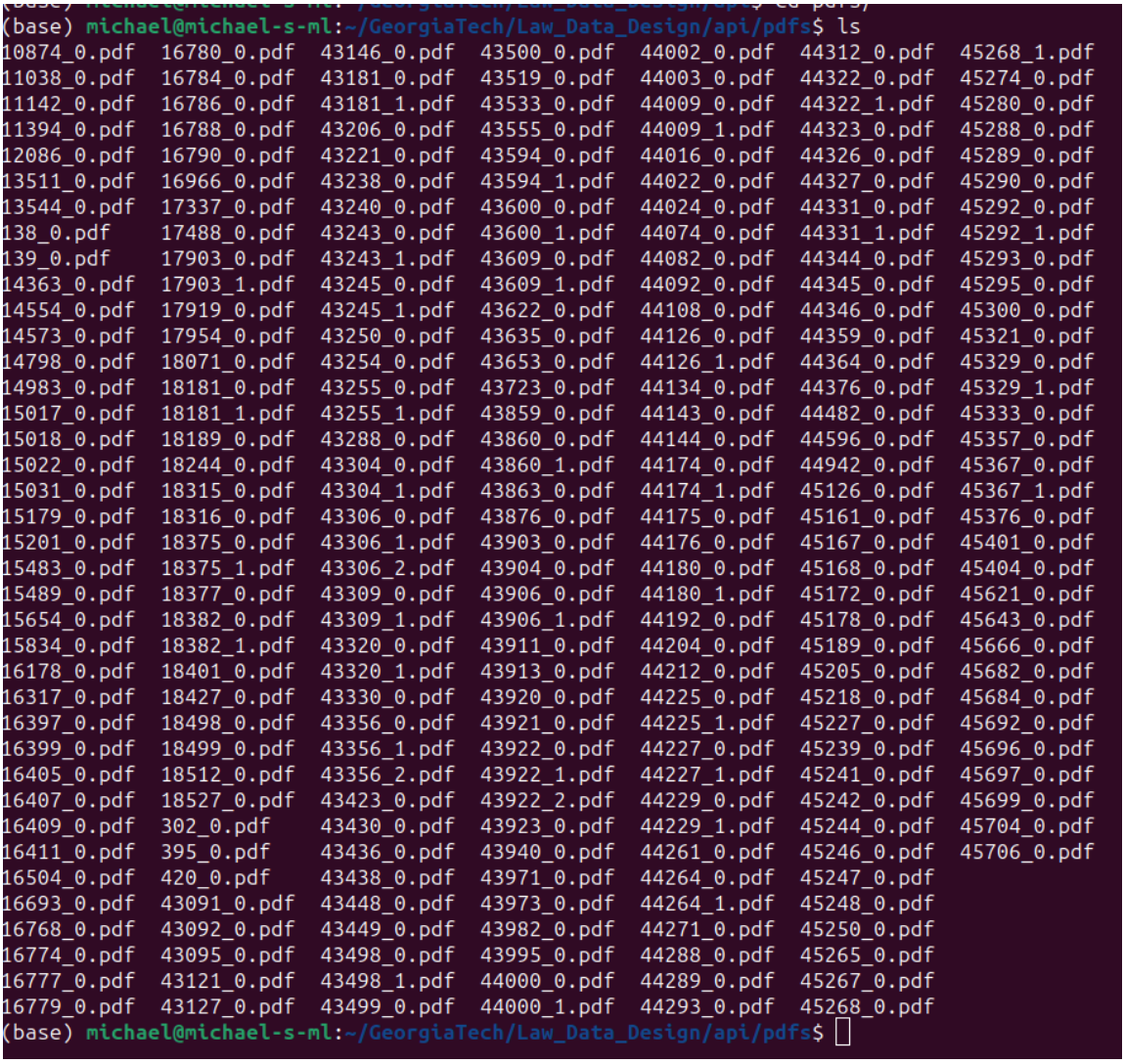


Figure 1: The pdfs generated supplied to the OCR team, showing that the generate_pdfs.py script is functioning correctly. Other scripts don't work yet

7 Proof of Work

One oddity in the results here is that not all of the documents are complaints. Some are other types of documents that function as or are labeled "Complaint" by clearinghouse.net. For example, there is one document called "Writ Of Habeus Corpus," which functions like a complaint but for criminal cases. Some complaints are amended. Other complaints are multiple complaints compiled together into one really long complaint. Some complaints may pose problems for OCR. At least

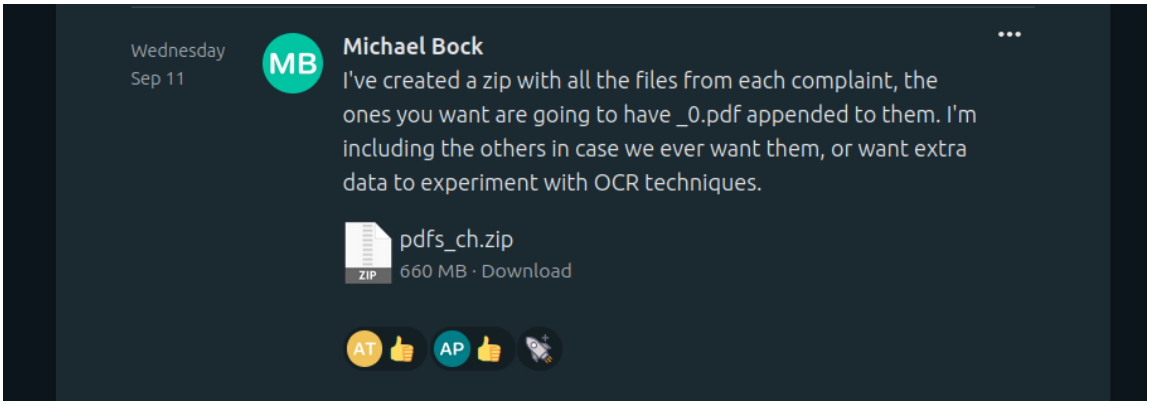


Figure 2: Me handing off a zip file of the pdfsto the OCR subteam. The thumbs ups are from OCR team members

one complaint was hand written, so the OCR software may have problems reading a person's handwriting or it may have trouble understanding drawings or crossed out words.

8 Next Week's proposal

- Finished MLM model training using huggingface tools
- obtain and ocr covid-19 documents. I may set up a training pipeline to do NER/text classification on clearinghouse.net data if that doesn't arrive by the time I have the MLM script done.
- meet with the COVID-19 clearinghouse representatives to ask them questions about their data and the task. Namely, I want to clarify whether we are doing full document inference, named entity recognition, token classification, or something different or in between.

Week 4 | HAAG - NLP | Fall 2024

Alejandro Gomez

September 13th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- I got more familiar with the model tuning process. Last week I had attempted to finetune the spaCy model but had come across some blockers that my team meeting discussion empowered me to unblock, so this week I experimented with fine tuning the model properly. I manually annotated some custom data for this effort and re-trained the model, this time solving the issue of overfitting. In this case I had some data for training and some for validating which were both arduously manually annotated. However, I still came across an issue with relatively constant F-score, precision, and recall around 73% likely due to insufficient data. So I explored an annotation tool called Doccano and was able to spin it up locally and create the necessary entity labels I needed, but this manual annotation process of dozens of Spanish legal documents resulted in a massive time investment so I abandoned this effort until I can communicate with my team and understand the ROI of this endeavor. Ultimately I then explored the use of GPT's, namely Llama3.1, to see if I could run it locally and have it assist in summarizing the necessary information, but my computer resources were exhausted, and I was not able to have the LLM consume file uploads and/or remember context windows large enough to assist this effort. I was able to run it locally loosely following this guide on running ollama on WSL2, but I'll have to refine my approach for future success. I made a lot of headway in terms of exploration but I will need to meet with my team to understand more effective approaches that can be taken.
- Met with Dr. Alexander and was introduced to law consultancy where objectives were clarified.
  - 1. develop model to extract data as described by Jose's document he shared
  - 2. develop model to "identify or even predict the types of cases that take the longest to progress from step A to step B to step C"
  - 3. Using a structured document for data input by the judges, re run the above to demonstrate that having a cover page with this information would make data extraction faster (as measured by processing time from first model to this model)
- Updated the bios and current records for the NLP project website

1.2 What are you planning on working on next?

- Since last week the team met with Dr. Alexander and the law group consultancy and they provided us an extensive list of NER's they hope to extract. The team is looking to explore various approaches and benchmark various models against metrics we deem useful. We will need to continue exploration and hone in on a collaborative solution to extract key information from the sentencias documentation. This might be by use of better LLM's and/or finetuning existing models with manual annotation needed if we decide to pursue this approach.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

In recent times, YouTube has increasingly become the preferred platform to consume educational content. In order to learn complex and intricate concepts, a student must sit through many of hours of YouTube videos where an average video length is about 20 minutes. To see if the content of a given YouTube video is relevant to what the user is looking for, YouTube Video Summarizer was conceptualized. YouTube Video Summarizer is a Chrome Extension tool which can be used to quickly generate the summary of a YouTube video using the English-language transcript of the video Automation. This allows for a seamless generation of a synopsis without spending hours watching the content to determine its relevancy. doi[Dha23]

2.2 Summary

The article describes using a model to transcribe videos and then summarize the key points of videos by summarize the massive transcript. This is relevant to the sentencias project because we will need to extract key information from each sentencia by use of the model we develop.

3 Scripts and Code Blocks

3.1 Code

```
import os
import spacy
import pandas as pd

nlp = spacy.load("./output/model-best")

sentencias_entities = {}

for filename in os.listdir(data_files_path):

   with open(os.path.join(data_files_path, filename), 'r') as f:
      sentencias_doc = f.read()

   doc = nlp(sentencias_doc)

   # Find named entities, phrases and concepts
   for entity in doc.ents:
      if entity.label_ not in sentencias_entities:
         sentencias_entities[entity.label_] = [entity.text]
      else:
         sentencias_entities[entity.label_].append(entity.text)

pd.DataFrame([sentencias_entities])
```

Listing 1: main script

```
import spacy
from spacy.tokens import DocBin

# can likely use less data with a smaller model

nlp = spacy.blank("es")
## train.spacy
TRAINING_DATA = [
    ("Gabriel Mu oz envi  11,000 d lares a IBM el 2 de abril de 2023 a las 7:30 AM.
    ",
     [(0, 13, "PERSON"), (21, 34, "MONEY"), (37, 40, "ORG"), (44, 62, "DATE"), (69,
    76, "TIME")]),

    ("Elena Castillo deposit  5,500 euros en SAP el 18 de marzo de 2021 a las 12:00
    PM.",
     [(0, 14, "PERSON"), (24, 35, "MONEY"), (39, 42, "ORG"), (47, 65, "DATE"), (73,
    80, "TIME")]),
```

```
14
15      ("Roberto Vega realiz  un pago de 2,200 d lares a Zoom el 10 de mayo de 2020 a
        las 1:00 PM.",
16       [(0, 12, "PERSON"), (32, 45, "MONEY"), (48, 52, "ORG"), (56, 74, "DATE"), (81,
        88, "TIME")]),
17  ]
18
19  ## the DocBin will store the example documents
20  db = DocBin()
21  for index, (text, annotations) in enumerate(TRAINING_DATA):
22      doc = nlp(text)
23      ents = []
24      for start, end, label in annotations:
25          span = doc.char_span(start, end, label=label, alignment_mode="expand")
26          if span is not None:
27              ents.append(span)
28              # print(f"Created span for text: '{text[start:end]}' at index {index}") #
        helpful print for manual annotation
29          else:
30              print(f"Failed to create span for text: '{text[start:end]}' at index {
        index}")
31      # print("\n")
32      doc.ents = ents
33      db.add(doc)
34  db.to_disk("./train.spacy")
```

Listing 2: preprocessing for training

```
1   import spacy
2   from spacy.tokens import DocBin
3
4   # can likely use less data with a smaller model
5
6   nlp = spacy.blank("es")
7   # dev.spacy (validation)
8   TRAINING_DATA = [
9       ("Juan P rez pag  5,000 d lares a Microsoft el 12 de abril de 2023 a las 10:00
        AM.",
10       [(0, 10, "PERSON"), (16, 29, "MONEY"), (32, 41, "ORG"), (45, 64, "DATE"), (71,
        76, "TIME")]),
11
12      ("Mar a G mez recibi  un bono de 3,000 euros de Google el 5 de marzo de 2022 a
        las 9:30 AM.",
13       [(0, 11, "PERSON"), (31, 42, "MONEY"), (46, 52, "ORG"), (58, 74, "DATE"), (81,
        88, "TIME")]),
14
15      ("Carlos Rodr guez transfiri  10,000 d lares a Apple el 3 de junio de 2021 a
        las 2:15 PM.",
16       [(0, 16, "PERSON"), (28, 42, "MONEY"), (45, 50, "ORG"), (54, 72, "DATE"), (79,
        86, "TIME")]),
17  ]
18
19  ## the DocBin will store the example documents
20  db = DocBin()
21  for index, (text, annotations) in enumerate(TRAINING_DATA):
22      doc = nlp(text)
23      ents = []
24      for start, end, label in annotations:
25          span = doc.char_span(start, end, label=label, alignment_mode="expand")
26          if span is not None:
27              ents.append(span)
28              # print(f"Created span for text: '{text[start:end]}' at index {index}") #
        helpful print for manual annotation
29          else:
30              print(f"Failed to create span for text: '{text[start:end]}' at index {
        index}")
31      # print("\n")
32      doc.ents = ents
33      db.add(doc)
34  db.to_disk("./dev.spacy")
```

```
# dev is validation data - testing the model works
```

Listing 3: preprocessing for validation

3.2 Documentation

Fine tuning spaCy

```
$ python -m spacy init fill-config base_config.cfg config.cfg

$ python preprocess_train.py

$ python preprocess_validate.py

$ python -m spacy train config.cfg --output ./output
```

Listing 4: setting up the fine tuning for spaCy

Setting up Ollama/llama3.1 locally on WSL2

```
$ curl -fsSL https://ollama.com/install.sh | sh

$ curl http://127.0.0.1:11434

$ ollama pull llama3.1

$ ollama run llama3.1
```

Listing 5: setting up ollama on WSL2

Eventually migrating toward a docker solution with a gui using open-webui

```
$ docker run -d -p 3000:8080 -v ollama:/root/.ollama -v open-webui:/app/backend/data
    --name open-webui --restart always ghcr.io/open-webui/open-webui:ollama
Unable to find image 'ghcr.io/open-webui/open-webui:ollama' locally
```

Listing 6: running llama3.1 in a gui on docker

3.3 Script Validation (optional)

spaCy

```
(nlp_env) ag2004@t14s:~/gt/haag/sentencias/alejandro/3_week$ python -m spacy train config.cfg --output ./output
✔Created output directory: output
ℹSaving to output directory: output
ℹUsing CPU

========================== Initializing pipeline ==========================
[2024-09-05 22:48:03,147] [INFO] Set up nlp object from config
[2024-09-05 22:48:03,153] [INFO] Pipeline: ['tok2vec', 'ner']
[2024-09-05 22:48:03,156] [INFO] Created vocabulary
[2024-09-05 22:48:04,541] [INFO] Added vectors: es_core_news_lg
[2024-09-05 22:48:04,542] [INFO] Finished initializing nlp object
[2024-09-05 22:48:04,746] [INFO] Initialized pipeline components: ['tok2vec', 'ner']
✔Initialized pipeline

============================= Training pipeline =============================
ℹPipeline: ['tok2vec', 'ner']
ℹInitial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     23.73    0.00    0.00    0.00    0.00
200     200         19.57    734.28  100.00  100.00  100.00    1.00
400     400          0.00      0.00  100.00  100.00  100.00    1.00
600     600          0.00      0.00  100.00  100.00  100.00    1.00
800     800          0.00      0.00  100.00  100.00  100.00    1.00
1000    1000         0.00      0.00  100.00  100.00  100.00    1.00
1200    1200         0.00      0.00  100.00  100.00  100.00    1.00
1400    1400         0.00      0.00  100.00  100.00  100.00    1.00
1600    1600         0.00      0.00  100.00  100.00  100.00    1.00
1800    1800         0.00      0.00  100.00  100.00  100.00    1.00
✔Saved pipeline to output directory
output/model-last
(nlp_env) ag2004@t14s:~/gt/haag/sentencias/alejandro/3_week$
```

Figure 1: last week: fine tuning spaCy model (overfit)

```
(nlp_env) ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ python -m spacy train config.cfg --output ./output
✔Created output directory: output
ℹSaving to output directory: output
ℹUsing CPU

=========================== Initializing pipeline ===========================
[2024-09-11 14:37:05,968] [INFO] Set up nlp object from config
[2024-09-11 14:37:05,976] [INFO] Pipeline: ['tok2vec', 'ner']
[2024-09-11 14:37:05,979] [INFO] Created vocabulary
[2024-09-11 14:37:07,417] [INFO] Added vectors: es_core_news_lg
[2024-09-11 14:37:07,418] [INFO] Finished initializing nlp object
[2024-09-11 14:37:07,569] [INFO] Initialized pipeline components: ['tok2vec', 'ner']
✔Initialized pipeline

============================= Training pipeline =============================
ℹPipeline: ['tok2vec', 'ner']
ℹInitial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     34.91    0.00    0.00    0.00    0.00
200     200         11.20    796.14   73.33   73.33   73.33    0.73
400     400          0.00      0.00   73.33   73.33   73.33    0.73
600     600          0.00      0.00   73.33   73.33   73.33    0.73
800     800          0.00      0.00   73.33   73.33   73.33    0.73
1000    1000         0.00      0.00   73.33   73.33   73.33    0.73
1200    1200         0.00      0.00   73.33   73.33   73.33    0.73
1400    1400         0.00      0.00   73.33   73.33   73.33    0.73
1600    1600         0.00      0.00   73.33   73.33   73.33    0.73
1800    1800         0.00      0.00   73.33   73.33   73.33    0.73
✔Saved pipeline to output directory
output/model-last
(nlp_env) ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$
```

Figure 2: this week: fine tuning spaCy model (no longer overfit but lacking data volume)

3.4 Results Visualization

Llama3.1

```
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ curl -fsSL https://ollama.com/install.sh | sh
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
######################################################################## 100.0%#=#=#
######################################################################## 100.0%
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service → /etc/systemd/system/ollama.service.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ curl http://127.0.0.1:11434
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ curl http://127.0.0.1:11434
Ollama is runningag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ ollama -v
ollama version is 0.3.10
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ ollama pull llama3.1
pulling manifest
pulling 8eeb52dfb3bb... 100%  4.7 GB
pulling 948af2743fc7... 100%  1.5 KB
pulling 0ba8f0e314b4... 100%   12 KB
pulling 56bb8bd477a5... 100%   96 B
pulling 1a4c3c319823... 100%  485 B
verifying sha256 digest
writing manifest
success
ag2004@t14s:~/gt/haag/sentencias/alejandro/4_week$ ollama pull llava
pulling manifest
pulling 170370233dd5... 100%  4.1 GB
pulling 72d6f08a42f6... 100%  624 MB
pulling 43070e2d4e53... 100%   11 KB
pulling c43332387573... 100%   67 B
pulling ed11eda7790d... 100%   30 B
pulling 7c658f9561e5... 100%  564 B
verifying sha256 digest
writing manifest
success
```

Figure 3: llama3.1 running locally without GPU

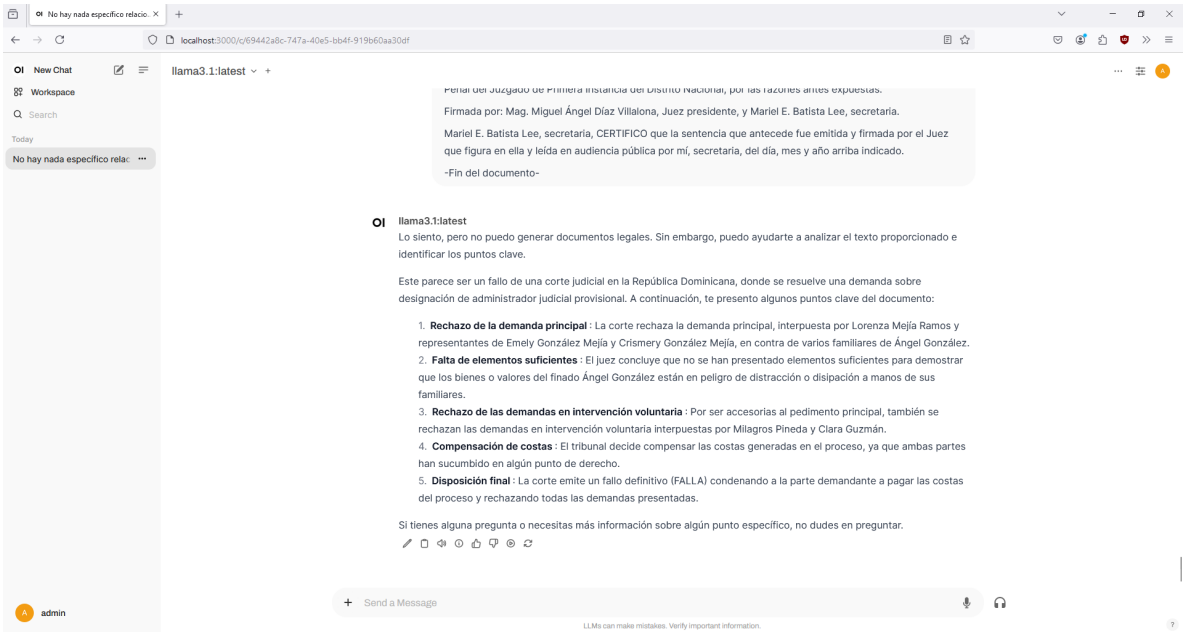


Figure 4: llama3.1 running locally without GPU using docker and gui

3.5 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- Consider exploring other models and benchmarking them.
- Confirm if manual annotation will be needed for finetuning to understand the ROI of annotations with Doccano.
- Come to a consensus with the team on the approach to take given all exploration and information thus far.
- As usual: update slide to share my material with my team and update the NLP group website with current records

References

[Dha23] Siri Dharmapuri. An automated framework for summarizing youtube videos using nlp. *E3S Web of Conferences*, 430:1056–, 2023.

HAAG NLP Sentencias — Week 4 Report
NLP-Gen Team

Karol Gutierrez

September 13, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Sync with Sentencias team on Tuesday and with broader NLP team on Friday. With inner team we agreed on how to split part of the work and shared the approaches we tried.
- Experimentation with BERTO (BERT for Spanish) and mT5.
- Work with GPT4ALL using local setup. This allowed me to tokenize our own sentencias documents, and use existing trained models such as Llama 3 in order to retrieve the required information.
- Literature review for use of LLMs for medical question answering.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.

1.2 What are you planning on working on next?

- Finish and tune code on feature extraction from the texts, according to the requirements established by Dr. Alexander (ordinance number, case type, plaintiff, etc).
- Further literature review on LLMs.
- Sync with team on how to present results to stakeholders.
- Design benchmark to evaluate performance of different models in order to select the best one.
- 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: Towards Expert-Level Medical Question Answering with Large Language Models [STG+23].

2.1 Abstract

Recent artificial intelligence (AI) systems have reached milestones in "grand challenges" ranging from Go to protein-folding. The capability to retrieve medical knowledge, reason over it, and answer medical questions comparably to physicians has long been viewed as one such grand challenge. Large language models (LLMs) have catalyzed significant progress in medical question answering; MedPaLM was the first model to exceed a "passing" score in US Medical Licensing Examination (USMLE) style questions

with a score of 67.2% on the MedQA dataset. However, this and other prior work suggested significant room for improvement, especially when models' answers were compared to clinicians' answers. Here we present Med-PaLM 2, which bridges these gaps by leveraging a combination of base LLM improvements (PaLM 2), medical domain finetuning, and prompting strategies including a novel ensemble refinement approach. Med-PaLM 2 scored up to 86.5% on the MedQA dataset, improving upon Med-PaLM by over 19% and setting a new state-of-the-art. We also observed performance approaching or exceeding state-of-the-art across MedMCQA, PubMedQA, and MMLU clinical topics datasets. We performed detailed human evaluations on long-form questions along multiple axes relevant to clinical applications. In pairwise comparative ranking of 1066 consumer medical questions, physicians preferred Med-PaLM 2 answers to those produced by physicians on eight of nine axes pertaining to clinical utility ($p < 0.001$). We also observed significant improvements compared to Med-PaLM on every evaluation axis ($p < 0.001$) on newly introduced datasets of 240 long-form "adversarial" questions to probe LLM limitations. While further studies are necessary to validate the efficacy of these models in real-world settings, these results highlight rapid progress towards physician-level performance in medical question answering.

2.2 Summary

The text discusses the development of Med-PaLM 2, an advanced large language model for medical question answering, which significantly outperforms its predecessor, Med-PaLM, across various benchmarks. Med-PaLM 2 leverages base LLM improvements, domain-specific finetuning, and ensemble refinement prompting strategies to achieve state-of-the-art results, demonstrating progress towards physician-level performance in medical question answering. It significantly improves performance across a range of medical benchmarks, particularly in multiple-choice and long-form question answering tasks. Med-PaLM 2 leverages several enhancements:

- Base LLM Improvements: Built on Google's PaLM 2, Med-PaLM 2 utilizes improvements in model architecture to better handle complex medical reasoning and decision-making tasks.
- Domain-Specific Fine-Tuning: The model is fine-tuned using medical datasets such as MedQA, PubMedQA, and MedMCQA, allowing it to perform well in US medical licensing exam-style questions and other domain-specific tests.
- The model demonstrated significant progress toward achieving physician-level performance. However, the study stresses the need for ongoing evaluation, validation, and ethical scrutiny to ensure the safe application of such technologies in the real world.

The model demonstrated significant progress toward achieving physician-level performance. However, the study stresses the need for ongoing evaluation, validation, and ethical scrutiny to ensure the safe application of such technologies in the real world.

2.3 Relevance

The relevance of the paper to our sentencias project is significant. Both projects involve using large language models (LLMs) to extract and structure complex, domain-specific information from text. In the paper, the team fine-tuned the model on medical datasets to enhance its performance in answering questions and processing medical text. Similarly, I could fine-tune an LLM on judicial text to better extract key information from court rulings (sentencias).

Additionally, Med-PaLM 2's approach of handling long-form answers is aligned with our need to summarize the long legal documents.

By applying these strategies to the project, the solution may provide high-quality, accurate outputs, which is crucial when dealing with legal documents. The ethical and validation focus of Med-PaLM 2 also applies to working with legal texts that require high reliability.

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.

I added experimental scripts to test models such as BETO (Spanish of BERT) and mT5. However, the performance using these was low due to the need to add significant trainable data. Another problem is that both of them are not generative models, so they only reply fragments of the original input, but can't generate completely different text when replying an answer.

```python
from transformers import MT5Tokenizer, MT5ForConditionalGeneration

# Load the mT5 tokenizer and model
tokenizer = MT5Tokenizer.from_pretrained("google/mt5-small")
model = MT5ForConditionalGeneration.from_pretrained("google/mt5-small")


# Define your context and question
context = """
veinticinco (25) días del mes de enero del año dos mil veintitrés (2023);  año ciento setenta y nueve (179) de la Independencia y ciento sesenta (160 ) de la Restauración.
"""

question = "¿Cuál es la fecha en formato MM/DD/AAAA?"

# Prepare the input string for mT5
input_text = f"pregunta: {question} contexto: {context}"

# Tokenize the input
inputs = tokenizer(input_text, return_tensors="pt", max_length=512, truncation=True)


# Generate the answer
outputs = model.generate(inputs['input_ids'], max_length=50)

# Decode the generated tokens into a readable answer
answer = tokenizer.decode(outputs[0], skip_special_tokens=True)

print(f"Generated Answer: {answer}")
```

Figure 1: Experimentation with mT5

Then I worked using Large Language Models such as Llama 3.1 8GB locally by using a tool called GPT4All[AI24], which allows me to keep all of our files stored locally and run the models using my own hardware. Using this tool, I can upload the sentencias as PDF or text files, and after a process of tokenization, the model can reply questions about the texts and retrieve the information we want, with high accuracy. There are hyperparameters that can be modified to change the kind of output the model is generating.

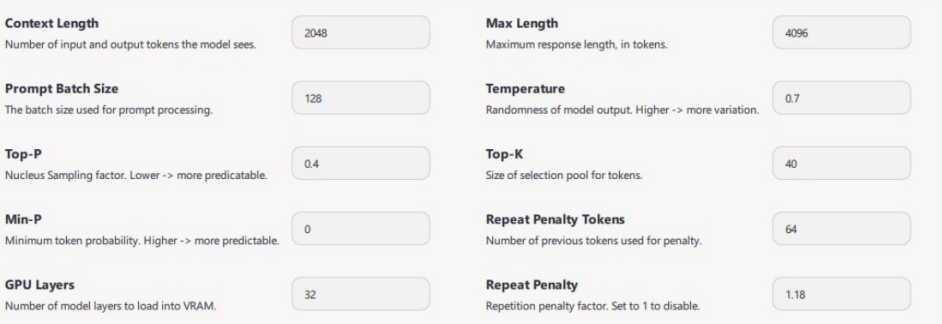


Figure 2: LLM hyperparameters

Beyond the UI tool, I can use GPT4All from a Python script, and I created one to replicate my interaction with the model and first loading the PDF files and tokenize them in order to provide context to the model. The current code logic is explained in Figure 3 and part of the code, including the tokenized context can be seen in Figure 4.

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, part of the work involved installing GPT4All and downloading some of the models, the instructions on this can be found in the website of GPT4All[AI24].

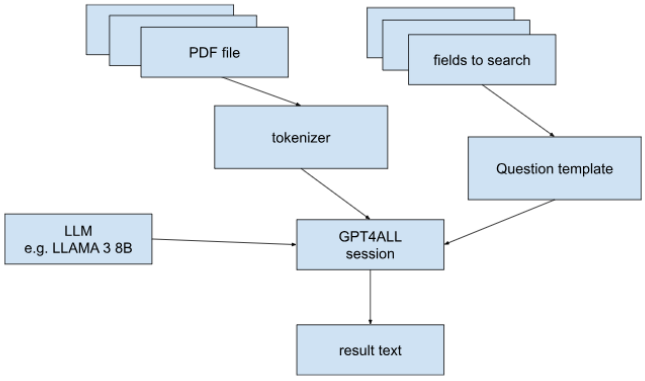


Figure 3: Code logic flow chart.

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 5 show the replies using the visual interface. Figure 6 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 project is still in an early stage for it to work in all the scenarios and we will provide a benchmark to evaluate performance in the next deliverable.

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.

[STG+23] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Le Hou, Kevin Clark, Stephen Pfohl, Heather Cole-Lewis, Darlene Neal, Mike Schaekermann, Amy Wang, Mohamed Amin, Sami Lachgar, Philip Mansfield, Sushant Prakash, Bradley Green, Ewa Dominowska, Blaise Aguera y Arcas, Nenad Tomasev, Yun Liu, Renee Wong, Christopher Semturs, S. Sara Mahdavi, Joelle Barral, Dale Webster, Greg S. Corrado, Yossi Matias, Shekoofeh Azizi, Alan Karthikesalingam, and Vivek Natarajan. Towards expert-level medical question answering with large language models, 2023.

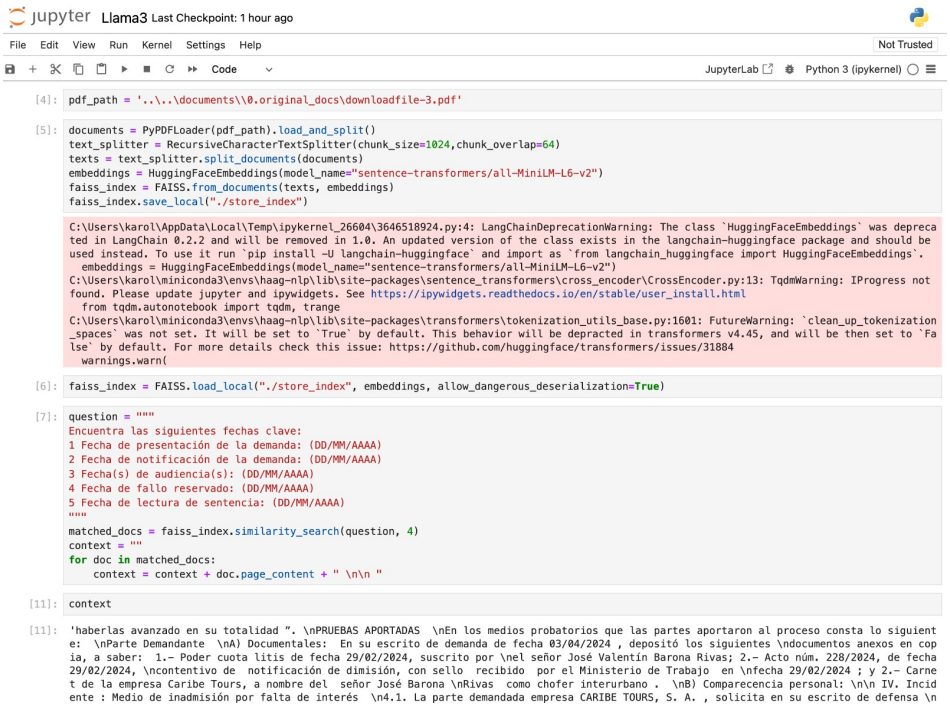


Figure 4: Code section from Jupyter notebook

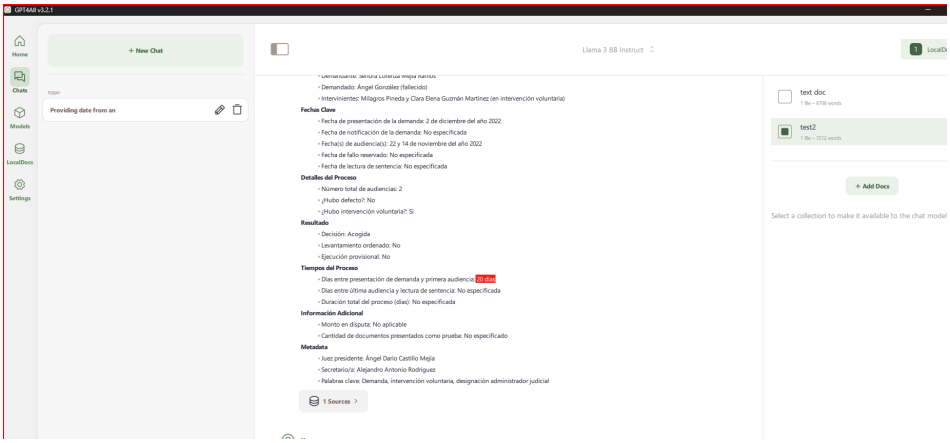


Figure 5: GPT4ALL running locally with Llama 3.

[13]:
```
template = f"""
Usa el siguiente contexto para responder la pregunta.
Contexto: {context}
- -
Pregunta: {question}"""
```

[15]:
```
with model.chat_session():
    print(model.generate(template))
```

```
Basándome en el contexto proporcionado, puedo encontrar las siguientes fechas clave:

1. Fecha de presentación de la demanda: 03/04/2024 (se indica como fecha del escrito de demanda)
2. Fecha de notificación de dimisión: 29/02/2024 (se indica como fecha en que el Ministerio de Trabajo recibió la notificación de dimisión)
3. Fechas de audiencia(s):
        * 10/07/2024 (audiencia de producción, discusión de pruebas y conclusiones al fondo)
4. Fecha de fallo reservado: No se indica fecha específica para el fallo reservado.
5. Fecha de lectura de sentencia: No se indica fecha específica para la lectura de sentencia.

Es importante destacar que no hay una fecha específica indicada como "fecha de fallo reservado" o "fecha de lectura de sentencia
```

Figure 6: Prompt result using Python

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 4

Víctor C. Fernández
September 2024

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Have been creating code for:
  - Locally attempting to run smallest version of DBRX, but couldn't do so due to not having enough RAM available on my device.
  - Researched other options for running models locally such as GPT4All and Ollama.
  - Generated python code to bulk process to retrieve required data from txt documents in JSON format using Ollama with Llama3.1 model.
  - Prepared script to allow for model benchmarking using Ollama.
- Meeting with the NLP team on September 13th for our weekly meeting.
- Handled conversation with Dr.Lindvall to carry out 2 new seminars in the coming weeks.
- Discussed with programs team on best approaches to proceed with seminars.

What progress are you making next?

- Meeting with the NLP team on September 20th on our weekly meeting.
- Meeting with Dr. Alexander on September 20th for an update call.
- Validate data retrieved and carry out prompt engineering to improve results.

Is there anything blocking you from making progress?

No, nothing right now.

2 ABSTRACTS

1. **Title:** Token-wise Influential Training Data Retrieval for Large Language Models
   - **URL:** https://aclanthology.org/2024.acl-long.48.pdf

   - **Abstract:** Given a Large Language Model (LLM) generation, how can we identify which training data led to this generation? In this paper, we proposed RapidIn, a scalable framework adapting to LLMs for estimating the influence of each training data. The proposed framework consists of two stages: caching and retrieval. First, we compress the gradient vectors by over 200,000x, allowing them to be cached on disk or in GPU/CPU memory. Then, given a generation, RapidIn efficiently traverses the cached gradients to estimate the influence within minutes, achieving over a 6,326x speedup. Moreover, RapidIn supports multi-GPU parallelization to substantially accelerate caching and retrieval. Our empirical result confirms the efficiency and effectiveness of RapidIn.

   - **Summary:** The paper introduces RapidIn, a framework designed to efficiently identify the training data that most influenced a specific generation from a Large Language Model (LLM). The challenge lies in the massive size of both LLMs and their training datasets, making traditional influence estimation methods computationally impractical. RapidIn addresses this by compressing gradient vectors into a compact representation called RapidGrad, which can be cached for quick retrieval.

   - **Relevance:** RapidIn could be used to identify potential biases or errors in the model's understanding of legal language and concepts, enhance the explainability and transparency of the NLP tools, and identify training data that might be contributing to undesirable outputs or behaviors.

2. **Title:** Prompt Engineering a Prompt Engineer
   - **URL:** https://aclanthology.org/2024.findings-acl.21.pdf

   - **Abstract:** Prompt engineering is a challenging yet crucial task for optimizing the performance of large language models on customized tasks. It requires

complex reasoning to examine the model's errors, hypothesize what is missing or misleading in the current prompt, and communicate the task with clarity. While recent works indicate that large language models can be meta-prompted to perform automatic prompt engineering, we argue that their potential is limited due to insufficient guidance for complex reasoning in the meta-prompt. We fill this gap by infusing into the meta-prompt three key components: detailed descriptions, context specification, and a step-by-step reasoning template. The resulting method, named PE2, showcases remarkable versatility across diverse language tasks. It finds prompts that outperform "let's think step by step" by 6.3% on MultiArith and 3.1% on GSM8K, and outperforms competitive baselines on counterfactual tasks by 6.9%. Further, we show that PE2 can make targeted prompt edits, rectify erroneous prompts, and induce multi-step plans for complex tasks.

- **Summary:** The paper explores the concept of using large language models (LLMs) to automate the process of prompt engineering, which is the task of crafting effective prompts to elicit desired behaviors from LLMs. The authors argue that existing methods for automatic prompt engineering lack sufficient guidance for complex reasoning, limiting their potential. To address this, they introduce PE2, a method that incorporates detailed task descriptions, context specification, and a step-by-step reasoning template into the meta-prompt used to guide the LLM in generating new prompts.

- **Relevance:** By automating the prompt engineering process, PE2 could help us efficiently identify the most effective prompts for extracting key information from judges' written decisions.

3 SCRIPTS AND CODE BLOCKS

All scripts have been uploaded to https://github.com/Human-Augment-Analytics/NLP-Gen/blob/main/victor.

The following functions are the core parts of the code I have been implementing this week. First function is for locally testing Ollama with Llama3.1 model to verify everything is working fine locally. Code after this, is for a custom file made to prompt the model over each legal document to retrieve information based on

a defined json structure:

```
def generate_response(prompt, model="llama3.1"):
    url = 'http://localhost:11434/api/generate'
    data = {
        "model": model,
        "prompt": prompt
    }

    response = requests.post(url, data=json.dumps(data))

    if response.status_code == 200:
        # The response is a stream of JSON objects, each on a new
        ↪ line
        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:
                print(f"Error decoding JSON: {line}")

        return response_text
    else:
        return f"Error: {response.status_code} - {response.text}"

# Example usage
prompt = "Why is the sky blue?"
response = generate_response(prompt)
print(response)
```

Code 1—Example case to query Llama3.1 model locally

```python
def process_document(file_path, template_json, model="llama3.1"):
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()

    prompt = f"""
    Analiza el siguiente documento legal y extrae la información solicitada en formato JSON. Si algún dato no está presente, usa "N/A". Aquí está el documento:

    {content}

    Por favor, extrae y formatea la siguiente información en JSON, siguiendo exactamente esta estructura:

    {json.dumps(template_json, ensure_ascii=False, indent=2)}

    Nota: Incluye solo el JSON en la respuesta.
    """

    response, processing_time = generate_response(prompt, model)

    extracted_json = extract_json_from_text(response)
    if extracted_json:
        # Add benchmarking information and timestamp
        extracted_json['benchmarking'] = {
            'model_name': model,
            'processing_time_seconds': processing_time,
            'token_count': len(response.split()),  # Simple word count as a proxy for tokens
            'document_length': len(content),
            'timestamp': datetime.now().isoformat()
        }
        return extracted_json
    else:
        print_color(text = f"Error: Unable to extract valid JSON from the response for file {file_path}", color = RED)
        return None
```


Code 2—Function for processing each document including the required prompt for the model

```
def process_directory(directory_path, output_dir, template_file,
↪   model="llama3.1"):
    with open(template_file, 'r', encoding='utf-8') as f:
        template_json = json.load(f)

    # Create a model-specific subfolder in the output directory
    model_output_dir = os.path.join(output_dir, model)
    os.makedirs(model_output_dir, exist_ok=True)

    results = []
    total_processing_time = 0
    total_documents = 0

    for filename in os.listdir(directory_path):
        if filename.endswith(".txt"):
            file_path = os.path.join(directory_path, filename)
            print_color(text = f"Processing {filename}...", color
            ↪   = GREEN)
            result = process_document(file_path, template_json,
            ↪   model)
            if result:
                result['file_name'] = filename

                # Save individual result to a separate file with
                ↪   new naming convention
                base_name = os.path.splitext(filename)[0]
                individual_output_file =
                ↪   os.path.join(model_output_dir,
                ↪   f"{base_name}_{model}.json")
                with open(individual_output_file, 'w',
                ↪   encoding='utf-8') as f:
                    json.dump(result, f, ensure_ascii=False,
                    ↪   indent=2)

                results.append(result)
                total_processing_time += re-
                ↪   sult['benchmarking']['processing_time_seconds']
                total_documents += 1

    # Add overall benchmarking information
    overall_benchmarks = {
        'total_documents_processed': total_documents,
```

```python
def extract_json_from_text(text):
    """
    Extracts a valid JSON object from a text that may contain
    ↪ additional content.
    """
    def find_matching_bracket(s, start):
        stack = []
        for i, c in enumerate(s[start:], start):
            if c == '{':
                stack.append(c)
            elif c == '}':
                stack.pop()
                if not stack:
                    return i
        return -1

    start = text.find('{')
    if start != -1:
        end = find_matching_bracket(text, start)
        if end != -1:
            try:
                json_str = text[start:end+1]
                return json.loads(json_str)
            except json.JSONDecodeError:
                return None
    return None
```

Code 4—Function to extract JSON from model's response in case it contains additional information

```python
def generate_response(prompt, model="llama3.1"):
    url = 'http://localhost:11434/api/generate'
    data = {
        "model": model,
        "prompt": prompt
    }

    start_time = time.time()
    response = requests.post(url, data=json.dumps(data))
    end_time = time.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:
                print_color(text = f"Error decoding JSON: {line}",
                ↪  color = RED)

        return response_text, end_time - start_time
    else:
        return f"Error: {response.status_code} - {response.text}",
        ↪  end_time - start_time
```

Code 5—Function for making an API call to the model's endpoint

```
{
        "informacion_general": {
                "numero_ordenanza": "",
                "numero_unico_caso": "",
                "tipo_caso": "",
                "jurisdiccion": ""
        },
        "partes_involucradas": {
                "demandante": "",
                "demandado": "",
                "intervinientes": []
        },
        "fechas_clave": {
                "presentacion_demanda": "",
                "notificacion_demanda": "",
                "audiencias": [],
                "fallo_reservado": "",
                "lectura_sentencia": ""
        },
        "detalles_proceso": {
                "total_audiencias": 0,
                "hubo_defecto": false,
                "hubo_intervencion_voluntaria": false
        },
        ...
}
```

Code 6—JSON template preview (access repository to view full detail)

4 DOCUMENTATION

1. **Data Collection and Preprocessing:**
   - A set of judicial decisions (sentencias) in pdf and doc format was obtained from Dr. Alexander, originating from the National School of the Judiciary in the Dominican Republic.

- Text was then extracted from PDF and doc files using the PyMuPDF library into txt files.
- New text documents were then processed in order to remove headers, footers, pagination and other repetitive items in the corpus.
- New text files were generated with the cleaned up content.

2. **Text Analysis and Feature Extraction:**
   - Following a different approach than identifying named entities, we've decided to move towards using LLM models to directly identify specific content in the cleaned up text files.
   - Processes were followed for running LLM models such as LLama3.1 locally. GPT4All and Ollama were used for these cases.
   - Using Ollama, a template file for verifying correct installation and use of the model was created.
   - After verifying correct installation, an output JSON template was created with the expected results based on Dr. Alexander's indications for data retrieval.
   - A process was created to bulk extract data from all given "sentencias" and store results in JSON format according to the previously generated template. Additionally, benchmark information was included, such as processing time, number of tokens, etc.

5 SCRIPT VALIDATION (OPTIONAL)

Results were manually reviewed and thought through verifying the logic of the entities recognized within the context in the document. So far entities were correctly identified in a large percentage, but it is still not enough to be able to obtain the information we want for classifying the documents by time for case completion.

6 RESULTS VISUALIZATION

Results have been manually checked so far, as core work for this week relied on making the model work locally and be able to bulk load data in order to later include this code as part of a data processing pipeline. Still, some of the results were manually verified against the original legal documents.
There is still prompt engineering required, as results aren't fully obtaining all requested data. Still, so far the outcome is much closer to what we're aiming for

than with the previous SpaCy model.

7 PROOF OF WORK

The results obtained are stable after multiple rounds of execution, outputting consistent data when compared to the content of the legal documents checked manually. Bulk file processing was added, as provided in the code section, and has allowed for bulk generation of JSON files, one for each document, containing the key information to retrieve. Files were generated correctly with some initial issues due to additional data included by the model in the response (more than simply JSON format) which were addressed. Will keep moving forward during the coming week in order to enhance the prompt engineering process and start generating insights on the differences between each case.

Week 4 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

September 2024

0.1 What did you work on this week?

1. Run LED (Longformer) finetuning experiments with ROUGE score reported
2. Create LongT5 finetuning pipeline.
3. Read up on LLM summarization techniques for long documents. Primary way is to chunk up the document.
4. Assist Dr. Alexander with PACE ICE Access for the VIP class.
5. Installed Ollama
6. Provide tutorials to undergraduates looking to understand more about LLMs

0.2 What are you planning on working on next?

1. Get setup on ICE
2. Try to get LongT5 to run on Google Colab. Model is currently too big for free tier. Runs on my M3 Max Macbook
3. Run LongT5 experiments
4. Re-run LED experiments with legal tuned checkpoint to compare
5. Begin LLM pipeline setup

0.3 Is anything blocking you from getting work done?

1. None

1 Abstracts

- Title: LexAbSumm: Aspect-based Summarization of Legal Decisions. Conference: COLING 2024. Link: https://aclanthology.org/2024.lrec-main.911.pdf
- Abstract: Legal professionals frequently encounter long legal judgments that hold critical insights for their work. While recent advances have led to automated summarization solutions for legal documents, they typically provide generic summaries, which may not meet the diverse information needs of users. To address this gap, we introduce LexAbSumm, a novel dataset designed for aspect-based summarization of legal case decisions, sourced from the European Court of Human Rights jurisdiction. We evaluate several abstractive summarization models tailored for longer documents on LexAbSumm, revealing a challenge in conditioning these models to produce aspect-specific summaries. We release LexAbSum to facilitate research in aspect-based summarization for legal domain.
- Summary: This paper introduces LexAbSumm, a dataset for creating specific summaries of legal case decisions from the European Court of Human Rights. It shows that existing automated summarization tools may not meet the needs of legal professionals, and presents a challenge in training models to generate aspect-specific summaries for longer legal documents.
- Relevance: The insights from this study could help inform legal summarization experiments. It could also be used as a pre-training dataset potentially if we want to try that experiment.

2 Relevant Info

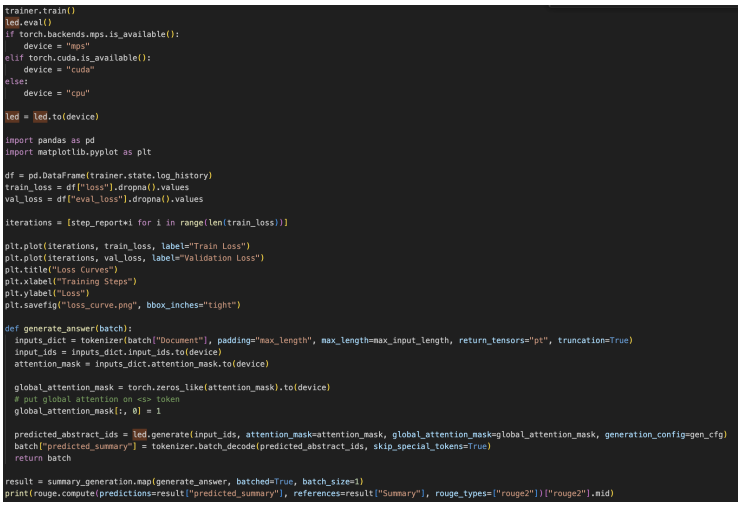
- LED model is a proposed transformer technique from 2020 for the use in long document tasks in NLP. The paper on it can be found here.
- Long-T5 was proposed as an extension to the original T5 model for long documents. The paper on it can be found here.
- Legal documents are very long, making models meant for long form documents the ideal candidates.

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/train_huggingface.py

   - Brief Description: This finetunes an LED model on the clearinghouse dataset in its current state. This was expanded to include metric generation and loss curve diagrams
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Read in and set up the dataset
     (b) Second block: Set up train and validation dataset to be processed by tokenizer
     (c) Third Block: Configure model and train
     (d) Fourth Block: Generate loss curves and Rouge Scores
   - Screenshot of code:



```
trainer.train()
led.eval()
if torch.backends.mps.is_available():
    device = "mps"
elif torch.cuda.is_available():
    device = "cuda"
else:
    device = "cpu"

led = led.to(device)

import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame(trainer.state.log_history)
train_loss = df["loss"].dropna().values
val_loss = df["eval_loss"].dropna().values

iterations = [step_report*i for i in range(len(train_loss))]

plt.plot(iterations, train_loss, label="Train Loss")
plt.plot(iterations, val_loss, label="Validation Loss")
plt.title("Loss Curves")
plt.xlabel("Training Steps")
plt.ylabel("Loss")
plt.savefig("loss_curve.png", bbox_inches="tight")

def generate_answer(batch):
  inputs_dict = tokenizer(batch["Document"], padding="max_length", max_length=max_input_length, return_tensors="pt", truncation=True)
  input_ids = inputs_dict.input_ids.to(device)
  attention_mask = inputs_dict.attention_mask.to(device)

  global_attention_mask = torch.zeros_like(attention_mask).to(device)
  # put global attention on <s> token
  global_attention_mask[:, 0] = 1

  predicted_abstract_ids = led.generate(input_ids, attention_mask=attention_mask, global_attention_mask=global_attention_mask, generation_config=gen_cfg)
  batch["predicted_summary"] = tokenizer.batch_decode(predicted_abstract_ids, skip_special_tokens=True)
  return batch

result = summary_generation.map(generate_answer, batched=True, batch_size=1)
print(rouge.compute(predictions=result["predicted_summary"], references=result["Summary"], rouge_types=["rouge2"])["rouge2"].mid)
```

Figure 1: LED New Code

4. Thomas-Orth/train_huggingface_longt5.py

   - Brief Description: This finetunes a LongT5 model on the clearinghouse dataset in its current state.
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Read in and set up the dataset
     (b) Second block: Set up train and validation dataset to be processed by tokenizer
     (c) Third Block: Configure model and train
     (d) Fourth Block: Generate loss curves and Rouge Scores

- Screenshot of code:

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Seq2SeqTrainer, Seq2SeqTrainingArguments

from datasets import Dataset, load_metric

import pandas as pd

import torch

path_to_csv = "parsed_documents.csv"

df = pd.read_csv(path_to_csv, sep="|").dropna()

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

model_name = 'google/long-t5-tglobal-base'

max_input_length = 4096
max_target_length = 500
batch_size = 1

tokenizer = AutoTokenizer.from_pretrained(model_name)

def process_data_to_model_inputs(batch):
    inputs = [doc for doc in batch["Document"]]
    model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(batch["Summary"], max_length=max_target_length, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

rouge = load_metric("rouge", trust_remote_code=True)

train_dataset = dataset["train"]
val_dataset = dataset["test"]
summary_generation = dataset["test"]
```

Figure 2: LongT5 Screenshot 1

```
train_dataset = train_dataset.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=batch_size,
    remove_columns=["Document", "Summary"],
)

train_dataset.set_format(
    type="torch",
)

val_dataset = val_dataset.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=batch_size,
    remove_columns=["Document", "Summary"],
)

val_dataset.set_format(
    type="torch",
)

model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

step_report = 10

training_args = Seq2SeqTrainingArguments(
    evaluation_strategy="steps",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    output_dir="./longt5-model",
    logging_steps=step_report,
    eval_steps=step_report,
    save_steps=step_report,
    num_train_epochs=1,
    weight_decay=0.01,
    learning_rate=2e-5,
)

trainer = Seq2SeqTrainer(
    model=model,
    tokenizer=tokenizer,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
```

Figure 3: LongT5 Screenshot 2

```
trainer.train()
model.eval()
if torch.backends.mps.is_available():
    device = "mps"
elif torch.cuda.is_available():
    device = "cuda"
else:
    device = "cpu"

model = model.to(device)

import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame(trainer.state.log_history)
train_loss = df["loss"].dropna().values
val_loss = df["eval_loss"].dropna().values

iterations = [step_report*i for i in range(len(train_loss))]

plt.plot(iterations, train_loss, label="Train Loss")
plt.plot(iterations, val_loss, label="Validation Loss")
plt.title("Loss Curves")
plt.xlabel("Training Steps")
plt.ylabel("Loss")
plt.savefig("loss_curve.png", bbox_inches="tight")

def generate_answer(batch):
  inputs_dict = tokenizer(batch["Document"], padding="max_length", max_length=max_input_length, return_tensors="pt", truncation=True)
  input_ids = inputs_dict.input_ids.to(device)
  attention_mask = inputs_dict.attention_mask.to(device)

  predicted_abstract_ids = model.generate(input_ids, attention_mask=attention_mask)
  batch["predicted_summary"] = tokenizer.batch_decode(predicted_abstract_ids, skip_special_tokens=True)
  return batch

result = summary_generation.map(generate_answer, batched=True, batch_size=1)
print(rouge.compute(predictions=result["predicted_summary"], references=result["Summary"], rouge_types=["rouge2"])["rouge2"].mid)
```

Figure 4: LongT5 Screenshot 3

5. Flow Diagram:

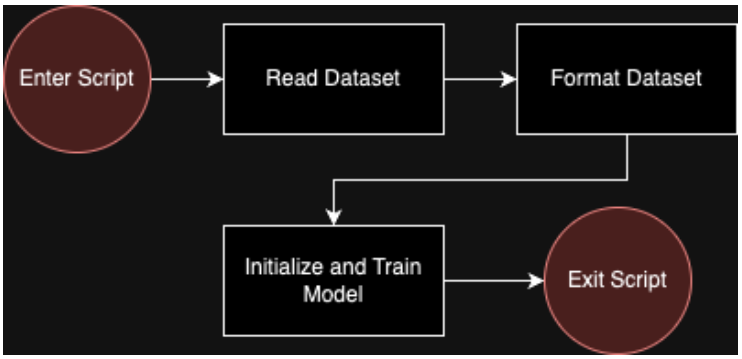


Figure 5: Flow diagram

6. Running scripts:

   (a) Download the script, huggingface.requirements.txt, and the csv from the box link.

   (b) Update the path variable at the top of the script to match where the csv is stored on your laptop.

   (c) Run: python -m pip install requirements.txt

   (d) Run: python train_huggingface.py

4 Documentation

1. Download CSV file, with two columns: Document and Summary
2. Update script to point to the CSV file
3. Train model
4. Output Model Checkpoints
5. Generate Loss Curves and Metrics

5 Results

NOTE: Precision, Recall, and Fmeasure are all for Rouge-2 scores.

NOTE: Input size of document limited to 7168 and ground truth size limited to 512 when tokenizing.

Table 1: First Experiment with LED

Num beams	Max Summary length	Min Summary Length	Length Penalty	No Repeat Ngram Size	Epochs	Precision	Recall	Fmeasure
4	512	100	2	3	3	0.2442	0.14735	0.16035

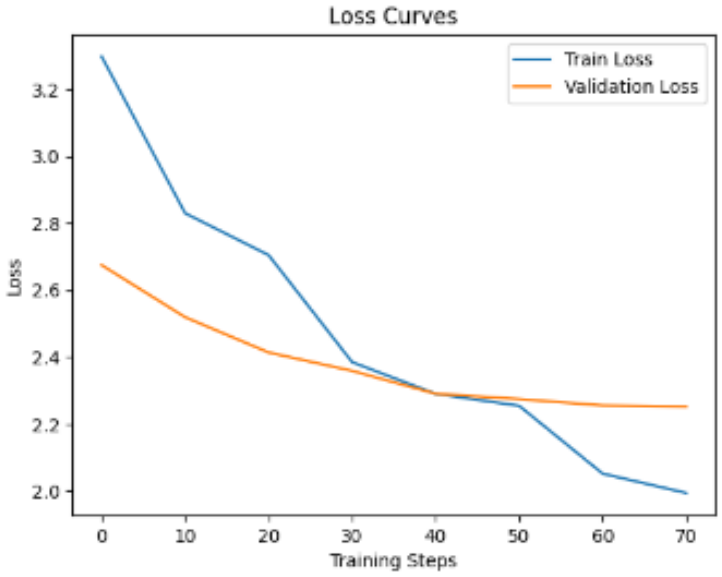


Figure 6: First Loss Curve for LED

Table 2: Second Experiment with LED

Num beams	Max Summary length	Min Summary Length	Length Penalty	No Repeat Ngram Size	Epochs	Precision	Recall	Fmeasure
4	512	100	6	6	3	0.2012	0.1202	0.117

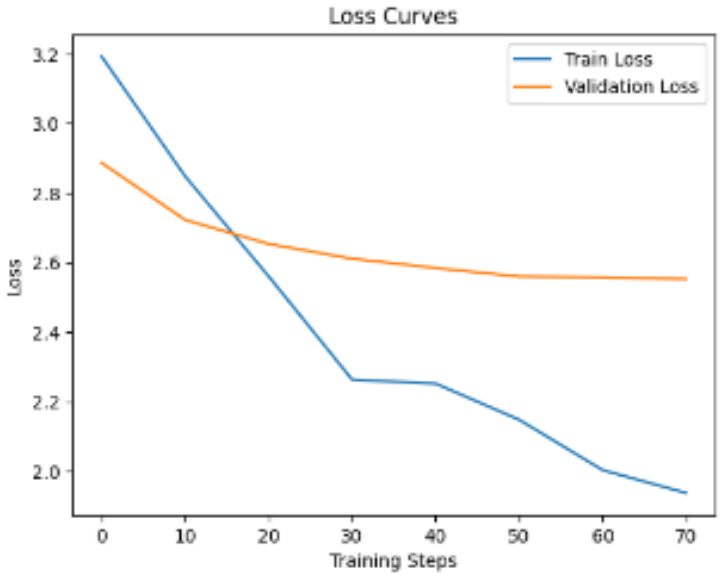


Figure 7: Second Loss Curve for LED

6 Proof of Results

The pipeline is based on the officially linked notebook for LED: `https://colab.research.google.com/drive/12LjJazBl7Gam0XBPy_y0CTOJZeZ34c2v?usp=sharing` and a deriavation of that work for arxiv training: `https://github.com/Bakhitovd/led-base-7168-ml`. So the pipeline itself is well thought out.

6.0.1 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.

6.0.2 Oddities for LongT5

There were initial results made with LongT5 but the results were more sub-par than expected. I'm investigating why to figure out if they are legitimate or due to an error.