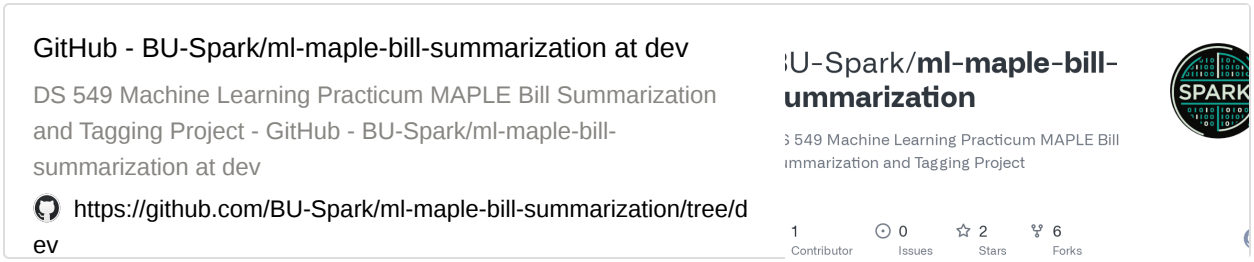
Documentation for MAPLE

A thorough documentation for team MAPLE

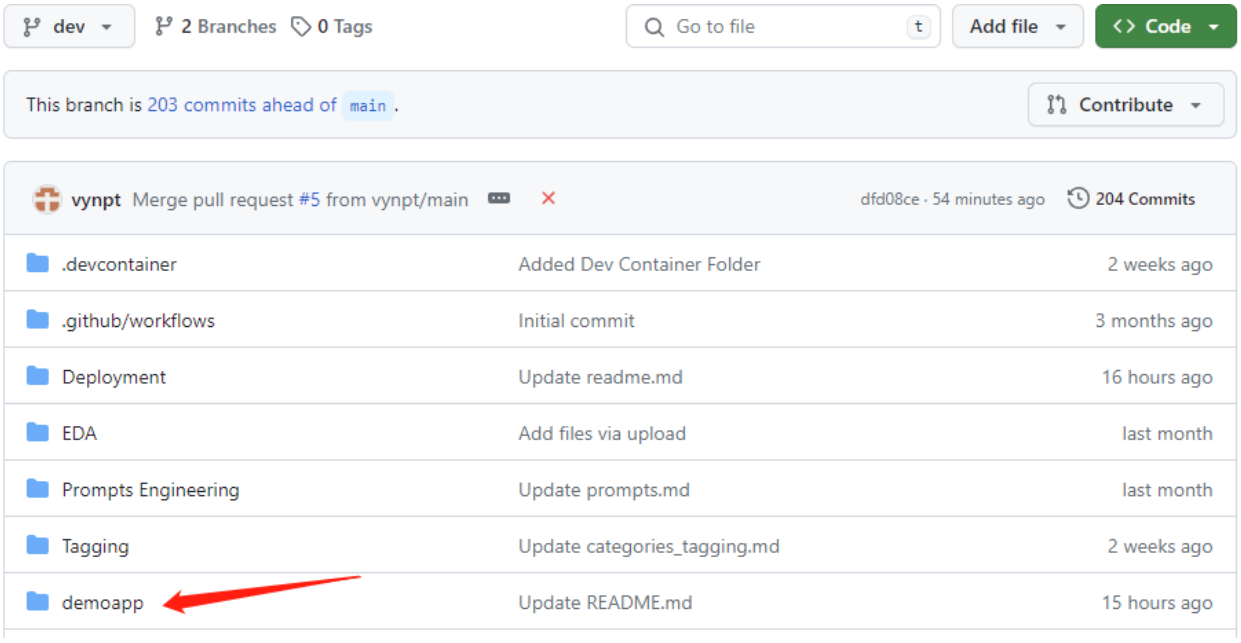
Access information/code

The code we wrote is in the dev branch. Here is the link to the spark repo.

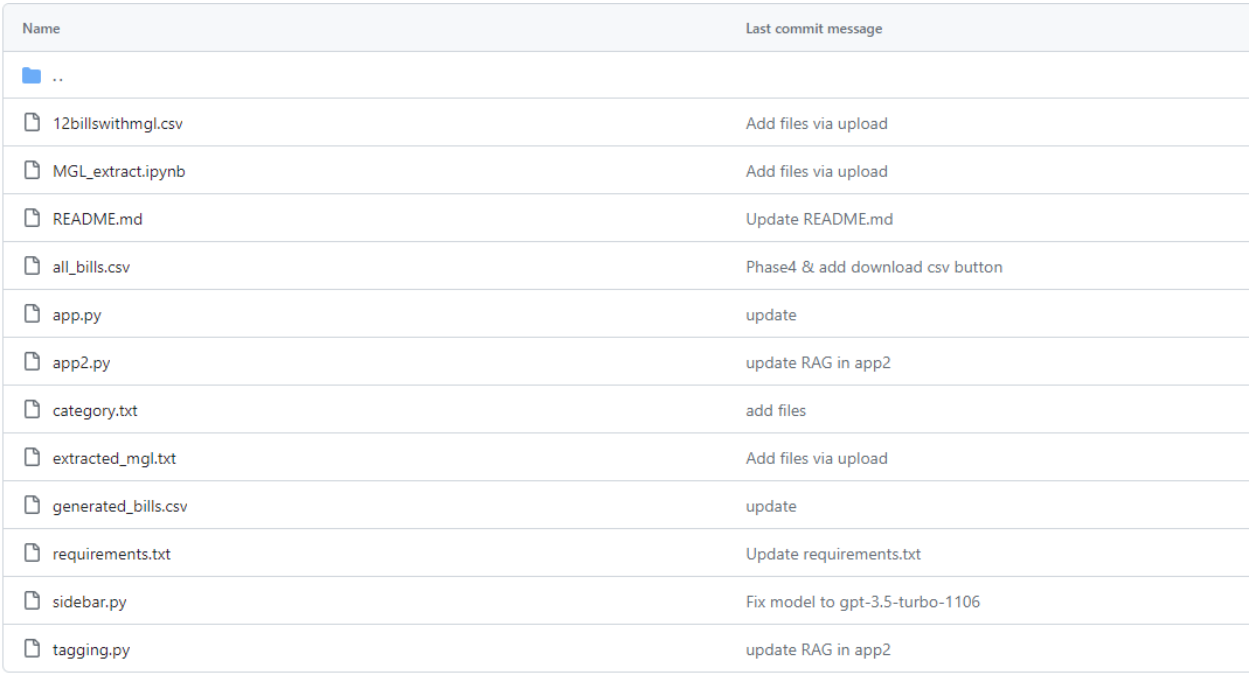


Explanation of the repo

Majority of the work lies inside of demoapp folder.



inside the folder, you should see a list of files like this.



Name	Last commit message
..	
12billswithmgl.csv	Add files via upload
MGL_extract.ipynb	Add files via upload
README.md	Update README.md
all_bills.csv	Phase4 & add download csv button
app.py	update
app2.py	update RAG in app2
category.txt	add files
extracted_mgl.txt	Add files via upload
generated_bills.csv	update
requirements.txt	Update requirements.txt
sidebar.py	Fix model to gpt-3.5-turbo-1106
tagging.py	update RAG in app2

12billswihmgl.csv → csv files that contain the 12 requested bills with relevant mass general law sections.

MGL_extract.ipynb → a python notebook that shows how we make api_call to get mass general law sections

all_bills.csv → csv files that contain around 6000-7000 bills from making api request.

app.py → python file that has all the bills loaded with summary responses.

app2.py → python file on the 12 bills

category.txt → categories and tagging received from Matt, but turn into a text file

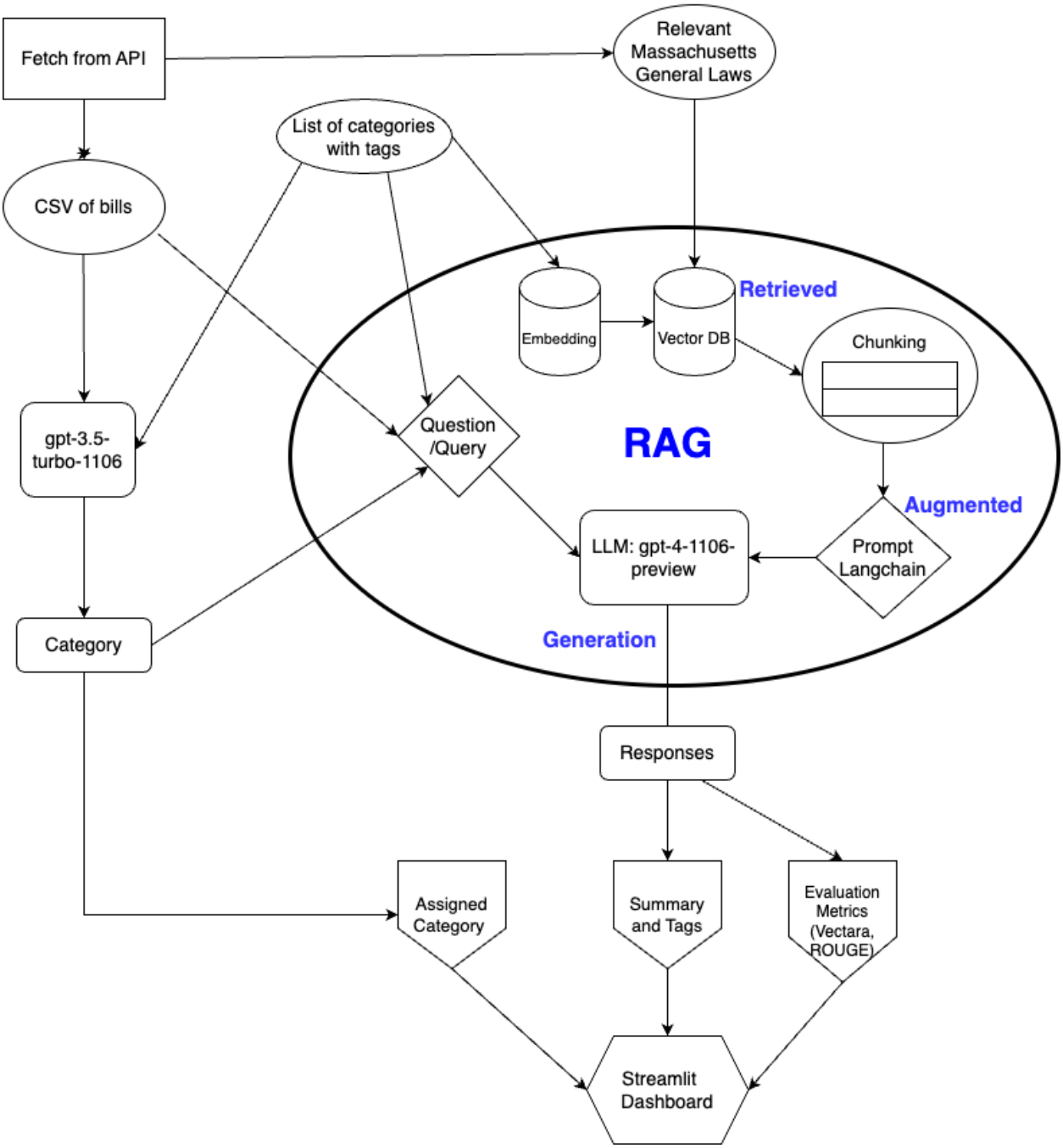
extracted_mgl.txt → extracted mgl from 12 bills and put into text file

requirements.txt → all the python packages / libraries used for the project

sidebar.py → UI for the webapp to enter openai api key

tagging.py → python file on the categories and tagging

High-level architecture



There are four main components to our work this semester:

1. Categorization

2. Summarization and Tagging
3. Evaluation
4. Deployment

Getting the data

Everything begins with getting the bills from the API with these two endpoints:

- /Documents
    - Returns all of documents publicly available in the current General Court. Chain this call to:
- /Documents/{documentNumber}
    - Returns a document by the Bill or Docket Number

With this, we can construct a csv file of all the bills in the current General Court, at the time having ~6,500 rows.

Categorization

We first give gpt-3.5-turbo the contents of category.txt, along with the bill text, and ask it to return a single category that the bill is about.

Summarization and Tagging

Two main things go into generating summaries and tags:

1. The query, which includes:
    a. The bill text itself
    b. The generated category
    c. Our prompt-engineered query (telling it how to summarize, return tags, format, etc)
2. The relevant MGLs, achieved through RAG

3. The tags, achieved through RAG

Giving all of this to GPT4 Turbo (specifically, the gpt-4-preview-1106 model), it will return the summaries and tags in one call.

Evaluation

After generating the summaries, these are the two quantitative evaluation metrics that we used:

1. ROUGE score
    a. ROUGE checks how many words, phrases, or sentences in the generated summary match with those in the reference summary (the original bill). More information about the type of ROUGE scores is available in the research document.
2. Vectara Factual Consistency
    a. This is an open source model designed to check if the summary is factually consistent with the original bill.

These are displayed on the right hand side of the streamlit app

Deployment

We used streamlit to deploy everything. Streamlit is a Python framework for creating interactive web applications.

Deployment link: https://maple-billsummarization.streamlit.app/

The link above is the deployment of “demoapp/app.py”. Regarding this file, since it has all bills loaded, the deployment allows user to filter and search for all bills. Though this version does not contain our latest code, it contains all user-friendly functionalities of the webapp that we did show you in the meetings.

One obstable of using Streamlit is the limited resources (specifically, Streamlit Cloud allows only 1GB resource). Therefore, we did not include the Vectara in “app.py” when deploying since this model is heavy and occupies large memory. However, you can still

uncomment the “Vectara model” and “Factual Consistency Value” parts in the code and run it in your local.

As we were unable to extract the entire MGL to integrate into our model, the generated response when you running this file does not include the context info. However, feel free to refine the prompt and modify GPT models to see their generation performances without context info.

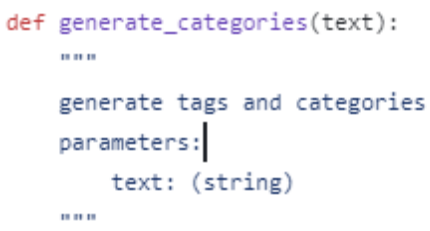
At the moment, we just extracted MGL sections relevant to the popular 12 bills, implemented RAG, and created vector bases for these 12 bills. Hence, “demoapp/app2.py” contains our latest code with significant improvement in the quality of summaries. In the next part, we will provide detailed operation of our model development in this file.

Explanation of the code

We go in the details on app2.py and focus on certain part of the code.

To generate categories.

The function to generate the categories is in the generate_categories function



we used the following text prompt to generate categories.

```
"""According to this list of category {category}.

        classify this bill {context} into a closest relevant cat

        Do not output a category outside from the list
    """
```

{category} and {context} are the required inputs to feed into the prompt. In this case, context is the bill contents and category is the list of categories given by Matt which in the tagging.py.

Before generating response, you can change a few things.

- temperature → how creative you want GPT to response, we set 0 for our project
- model → available model used ‘gpt-3.5-turbo-1106, gpt-4, gpt-4-preview-1106’
- openai_api_key → api_key to access the models, https://openai.com/blog/openai-api

Note: To generate predictions, you would need openai_api_key. With free subscription, you have access to gpt-3.5-turbo-1106. If you plan to use gpt-4 model, you would be required to upgrade your subscription. Here is the pricing for all the types of model. https://openai.com/pricing

To generate response.

The function to generate response is in generate_response. Essentially, we take advantage of **Retrieval Augmented Generation,** fancy word for feeding external knowledge to the GPT to make better summaries. You can find more details in this link.

https://python.langchain.com/docs/use_cases/question_answering/

This method comes with many benefits:

- Saving cost on prompt tokens like in context learning
- Reduction of hallucinations/lies
- Not reaching limits for some GPT models

Essentially, we provide the GPT model with the text inside the **extracted_mgl.txt**

To generate summaries with RAG, we will need multiple things:

1. prompt: telling GPT to use the extracted_mgl information to output summaries.

```
template = """You are a trustworthy assistant for question-answe
Use the following pieces of retrieved context to answer the ques
Question: {question}
```

```
Context: {context}
Answer:
"""
```

2. query: telling GPT to output a summary and tags

```
query = f""" Can you please explain what the following MA bill r
            Please provide a one paragraph summary in 4 sentence
            Note that the bill refers to specific existing secti
Use the information from those sections in your context to const
            Summarize the bill that reads as follows:\n{text}\n\

            After generating summary, output Category: {category
            Then, output top 3 tags in this specific category fr
```

from this string, you see multiple inputs: {text}, {category}, {tags_for_bill}

{text} → bill content

{category} → output from the generate_categories function

{tags_for_bill} → the string variable in tagging.py

TO alter the response, feel free to change the query, instead of top 3 tags, you can change it to top 1 tags that are most relevant.

Final Note:

If you would like to run the code, you will need the streamlit python library. simply by entering

```
streamlit run path/app2.py
```

Model Selections

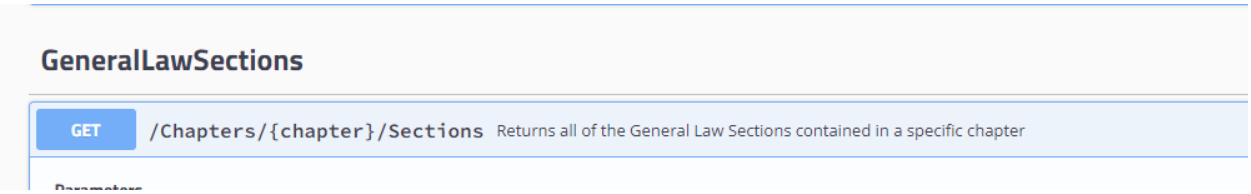
We found that GPT-4-preview-1106 had the best overall quality in summaries.

Unfinished work

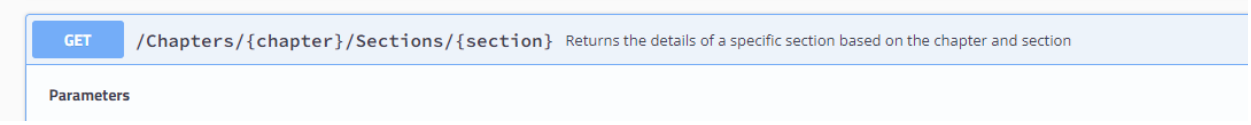
Scraping the entire Massachusetts General Laws section was a difficult task in a limited amount of time. Although we were not able to scrape the entirety of the MGL in short period of time, we do have the parts, titles, Chapters in the json file below.

https://drive.google.com/file/d/1LNddQiY8SpBTWh8krS76816s27ZZk08W/view

With the information, we figured there must be a way to feed chapter in the API first, and this will give us the sections.



Next, we can use those outputs and the chapters to get details about a specific section.



Next, you can save all the mass general law sections inside a text or any kind of file such that you can perform RAG on it.

Integrations/Recommendations

We know that the MAPLE website is likely built with typescript and JavaScript. Fortunately, it is possible to use langchain and openai in javascript. More information can be found here.

https://js.langchain.com/docs/get_started/introduction

As for what's next, we recommend next step to try is to experiment if RAG can work with the entirety of the MGL sections and continue to alter the prompt engineering. If the above method does not seem to help with the summary, the last step is to fine-tune a large language model.

Resources / What's next?

As mentioned during our meeting on 12/08, here is the blog post written by ***Armand Ruiz,*** *about a trend of many smaller companies/startup shifting to smaller language model and fine-tune for a specific field. They found that smaller language model have similar performance compare to large language model when fine-tuning. Not only that, smaller language model is easy on the computing cost (GPUs). You can find more in huggingface, think it as a github for AI.*

Models - Hugging Face

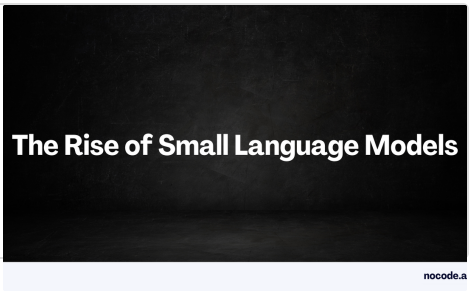
We're on a journey to advance and democratize artificial intelligence through open source and open science.

🤗 https://huggingface.co/models

The Emergence of Small Language Models

Explore Small Language Models (SLMs): efficient, resource-friendly AI for business. This blog covers key concepts and tuning strategies.

🚀 https://www.nocode.ai/the-emergence-of-small-language-models/



This is the resource we used to track hallucinations for evaluation. Also, we believe that it is important that the summaries are also viewed by lawyers. Lawyers judging the responses and verifying factual consistency is essential in process of continuous development.

Measuring Hallucinations in RAG Systems

Blog Post

https://vectara.com/measuring-hallucinations-in-rag-systems/

To learn more about using large language models, Deeplearning.ai offers many short courses (1hr to complete)

Short Courses | Learn Generative AI from DeepLearning.AI

Take your generative AI skills to the next level with short courses from DeepLearning.AI. Enroll today to learn directly from industry leaders, and practice generative AI concepts via hands-on

https://www.deeplearning.ai/short-courses/