Colby Wirth
30 November 2024
COS 398
Professor Tiffany Rad

UTILIZING RETRIEVAL AUGMENTED GENERATION TO PARSE WIKILEAKS ARCHIVES

1. Introduction

This project explores the implementation of a Retrieval Augmented Generation (RAG) pipeline within the context of Ethics in Computing, with a focus on the Wikileaks domain. Given the publicity and notoriety of Julian Assange and Wikileaks, especially regarding the email leaks of the DNC, John Podesta, and Hillary Clinton, this project lies at the intersection of ethics in computing and modern AI practices. This paper highlights the motivation, methodologies, and results from implementing a RAG pipeline to extract information directly from the primary sources within the email archives.

2. Background

Motivation

This project began with the intent to uncover the truth hidden within the depths of the Wikileaks archives. Naively, I assumed that entering a few keywords into the Wikileaks database would quickly yield evidence either exonerating or incriminating Hillary Clinton and, more broadly, the DNC, concerning potential legal wrongdoings during the 2016 Democratic primaries. However, this was not the case as the Clinton archive alone contained over 30,000 emails. It became apparent that my manual search process would take months to complete and would be tediously mundane.

Soon after, it had come to my attention that *if* I could access the content of these pages through a local repository, then I could utilize a large language model's vector embedding functions to build a SQLite Vector Database, and subsequently make my own queries. In addition to the SQLite database, the LLM will aid in the parsing of the returned queries. This is in-fact a high-level overview of what the RAG pipeline would entail. This process would allow me to make semantic queries without the rigid boolean structures of traditional search engines; thus enabling quick retrieval and insightful analysis.

With this discovery, the scope of the project shifted. Instead of focusing on making specific discoveries within the data, the goal would be to build a pipeline that would enable future curious minds to make their own discoveries.

3. Main Content

The methodology behind the pipeline is laid out in broad terms below. The steps are as follows: 1. Scraping the archive. 2. Building the SQLite Vector Database 3. Making Queries. 4. Results

Scraping the Archive

The first step in scraping the archive was to obtain a file with the links that I wanted to search for. I found a public API called called WikiApi Map, which produced a json file of up to 1000 links from a user-specified Wikileaks archive. However, this API was designed to only produce 1,000 links from an archieve. This was a large constraint on the project. Building an API to retrieve all links from an archive was beyond the scope of this project. As a result, the current version can only scrape a maximum of 1,000 pages from any Wikileaks archive.

Before beginning the scraping, I attempted to check Wikileaks' robots.txt file for guidelines regarding scraping. This file contained no relevant information or instructions, so I proceeded with the scraping operations. As stated in the introduction, this project focuses on the implementation of the RAG pipeline, therefore I limited the final project to only contain the content from 10 pages for the sake of maintaining small files and limiting execution times. Nonetheless, the program can be scaled to hanlde thousands of request at a time.

The process of scraping is implemented as an iterative function. In each iteration, a page's content is downloaded, cleaned, and saved to its own individual markdown file. These markdown files serve as the source to build the database from in the second phase of the project.

Building the Database

After all pages have been scraped, the vector database is built. This process is broken into three distinct steps: 1. Creating the 'chunks' and performing vector embedding. 2. Instantiating the database, and 3. Populating the database with the embedded chunks.

The vector embedding function is performed by a pre-trained large language model (LLM) from Hugging Face called "all-mpnet-base-v2". It is intended to be used primarily for vector embedding and text analysis.

For every document retrieved from the archive, its content is embedded as one vector by the LLM. After this is completed for all documents, the database is populated with the documents and their respective vector embeddings. The database is hosted locally as a Chroma SQLite Vector Database, and is persisted to make future queries.

Making Queries

With the database populated with all emails and their vector embeddings, queries can now be made to gain insights into the documents. Firstly the database is activated. Next, the query is received from user-passed arguments, and it is embedded with the same vector embedding function from the LLM. Finally, the embedded query is compared against the database entries with the top k relevant entries are returned.

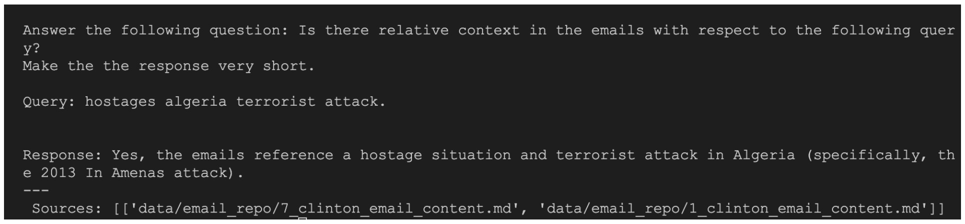
These top entries *and* the query are passed to the LLM with a customized prompt template. This prompt guides the LLM to inform the user if there is any relevant data in the results. The directory to the most relevant file (based upon the vector embeddings is presented to the user, even if there is no seemingly relevant information to the user. This is a limitation of the project. However, the LLM is responsible for telling the user if there is relevant information in the texts, despite this flaw.

Results

The results from two queries are listed below. The first is intended to find a match in the database, and the second is intended to not find one.

User Inputted Query 1:
"hostages algeria terrorist attack"

Output:

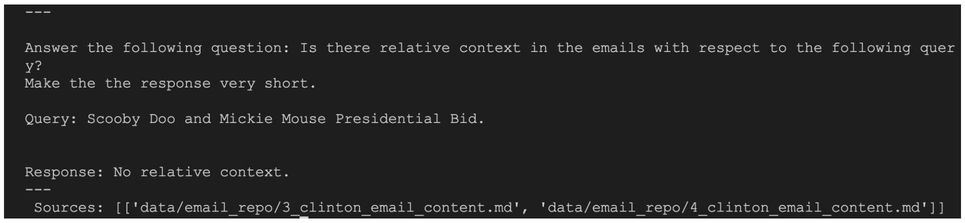


```
Answer the following question: Is there relative context in the emails with respect to the following quer
y?
Make the the response very short.

Query: hostages algeria terrorist attack.


Response: Yes, the emails reference a hostage situation and terrorist attack in Algeria (specifically, th
e 2013 In Amenas attack).
---
 Sources: [['data/email_repo/7_clinton_email_content.md', 'data/email_repo/1_clinton_email_content.md']]
```

Figure 1. Algeria Prompt

User Inputted Query 2:
"Scooby Doo and Mickie Mouse Presidential Bid"

Output:



```
---

Answer the following question: Is there relative context in the emails with respect to the following quer
y?
Make the the response very short.

Query: Scooby Doo and Mickie Mouse Presidential Bid.


Response: No relative context.
---
 Sources: [['data/email_repo/3_clinton_email_content.md', 'data/email_repo/4_clinton_email_content.md']]
```

Figure 2. Scooby Doo Prompt

4. Conclusion

Final Results and Analysis

The results above display the program working in its best capacity. However, the project does have its limitations, as the LLM can hallucinate. I found that under the following conditions, the program performed best: 1. Remove all stop-words from query as they devalue the meaning of the query. 2. Do not format the query as a question, rather use the key words that you wish to find content related. 3. Check the most relevant documents returned the program after making queries.
A properly formed query may look like "State Department Gender Policies". An improper query may be "What are the gender policies of the State Departments Gender Policies?". This program functions as a tool, and to achieve optimal results, its limitations and strengths must be carefully considered and utilized effectively.

Future Advancements

Although only ten files were inserted into the database, this project is highly scalable. The limitations associated with this program are the runtimes involved with scraping, vector embedding and querying. As stated above, the public API only supports up to 1000 HTML pages from each repository. Future projects would likely involve the implementation of a new API that could gather all entries from a specified Wikileaks archive. Additionally, I would encourage other programmers to implement other LLMs that may produce more insightful responses with respect to the queries.