CyberScienceLab

University of Guelph

Canada Cyber Foundry

Developing a RAG To Mitigate LLM Hallucinations in Malware Data

Introduction

This project implements a Retrieval-Augmented Generation (RAG) model designed to answer user queries based on the malware information and hash numbers provided in reports or questions. By utilizing Meta's LLaMA 3 and retrieving accurate hash descriptions, the system ensures that the language model has the relevant data to effectively respond to user inquiries.

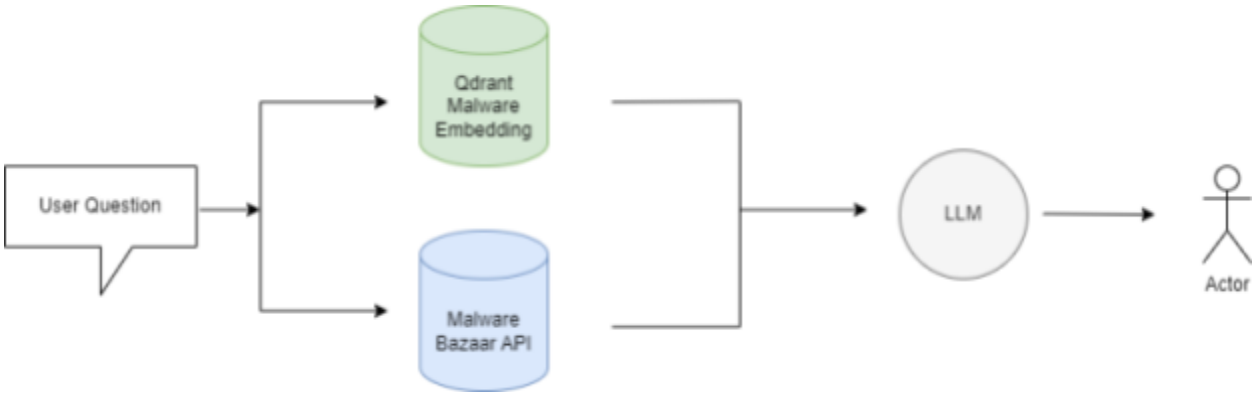
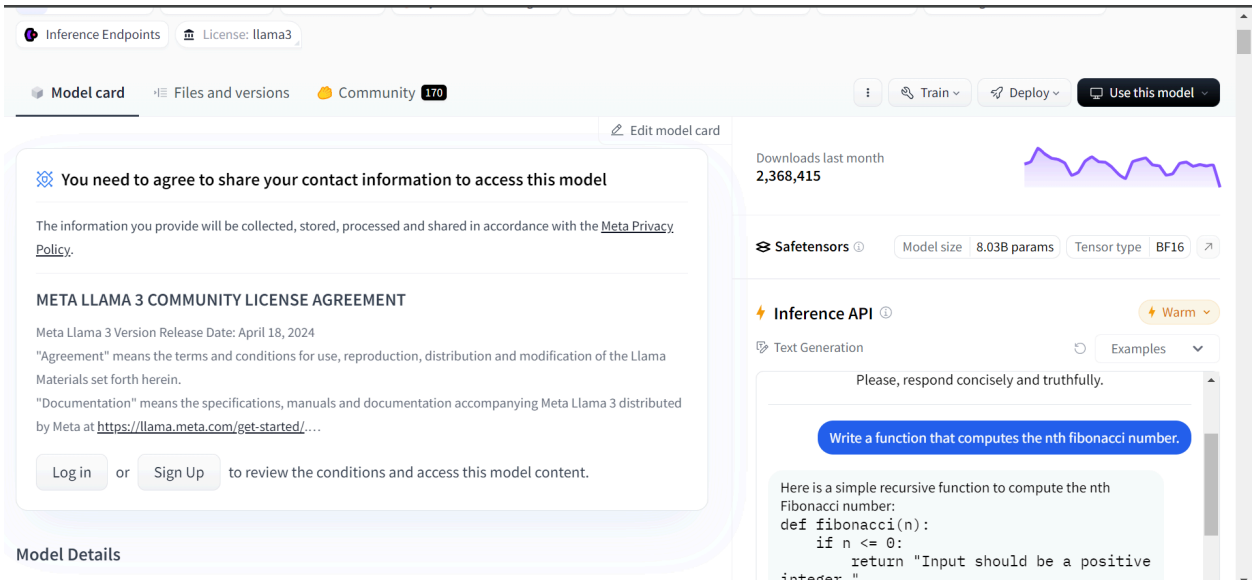


Diagram of Retrieval Augmented Generation with Malware Information
After processing the user's question, the system retrieves information related to any hashes mentioned in the report or query using the Malware Bazaar API. The question is then passed through a vector database to gather relevant data, particularly if the user seeks a malware sample that closely matches their description. Leveraging these two data sources, the system feeds the compiled information to the language model, enabling it to provide an informed and accurate response.

Access Llama3 HuggingFace Token:
[HuggingFace Website]



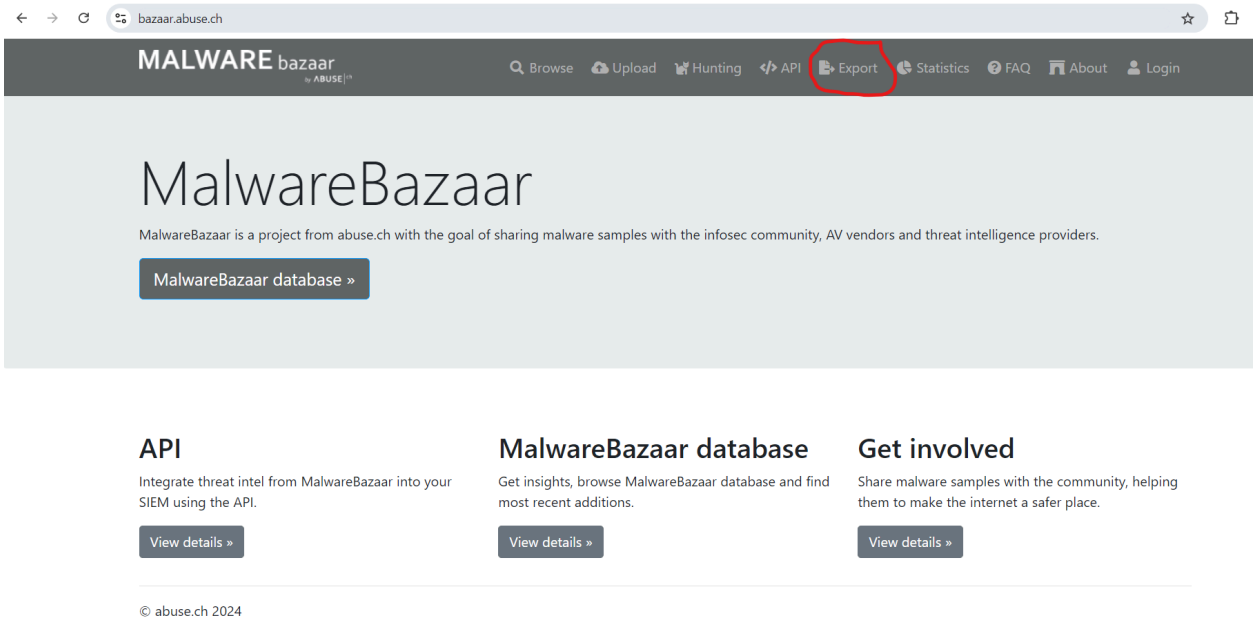
1. Create/Log into your Huggingface account
2. Get access to Llama3's Token
3. Export the token in your python environment by typing the command: export HF_TOKEN=<YOUR TOKEN>

Environment Setup

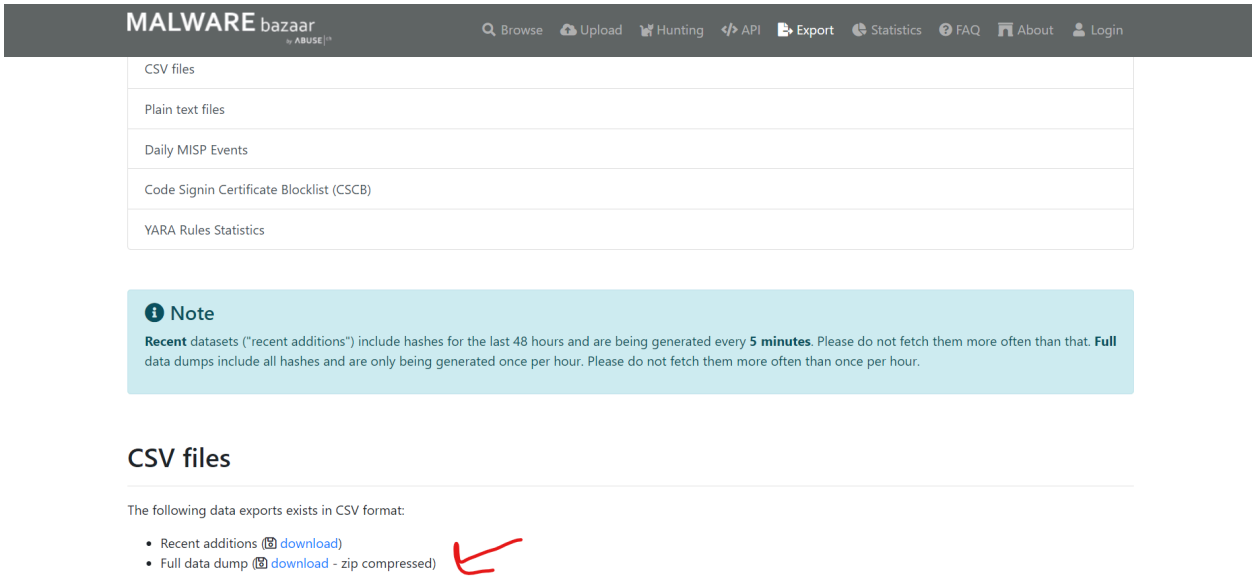
1. **Download the Required Programs**: Docker, Anaconda, Git.
2. **Set Up and Activate the Anaconda Project Environment**:
   - Create a new environment:
     `conda create --name ENV_NAME python=3.12`
   - Activate the environment:
     `conda activate ENV_NAME`
3. **Install Required Dependencies**:
   `pip install -r requirements.txt`
4. **Run Docker**:
   `docker run -p 6333:6333 qdrant/qdrant`

Gather Data:

We'll be using Malware Bazaar as our primary data source. The first step is to download a CSV file containing all the hash numbers available on their website. To do this, visit https://bazaar.abuse.ch/ and click on the export icon.

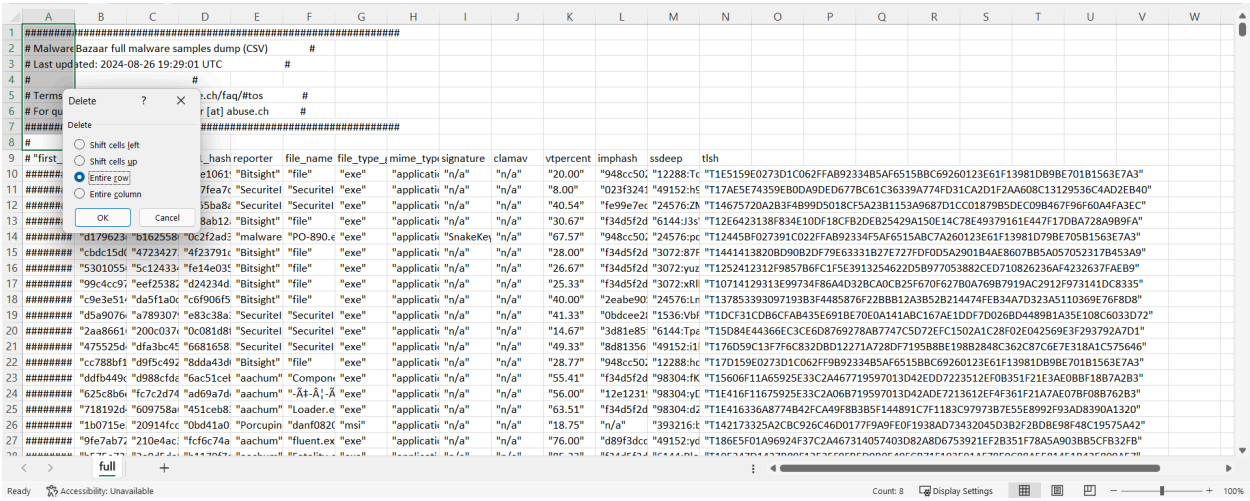


Click on the Full data dump

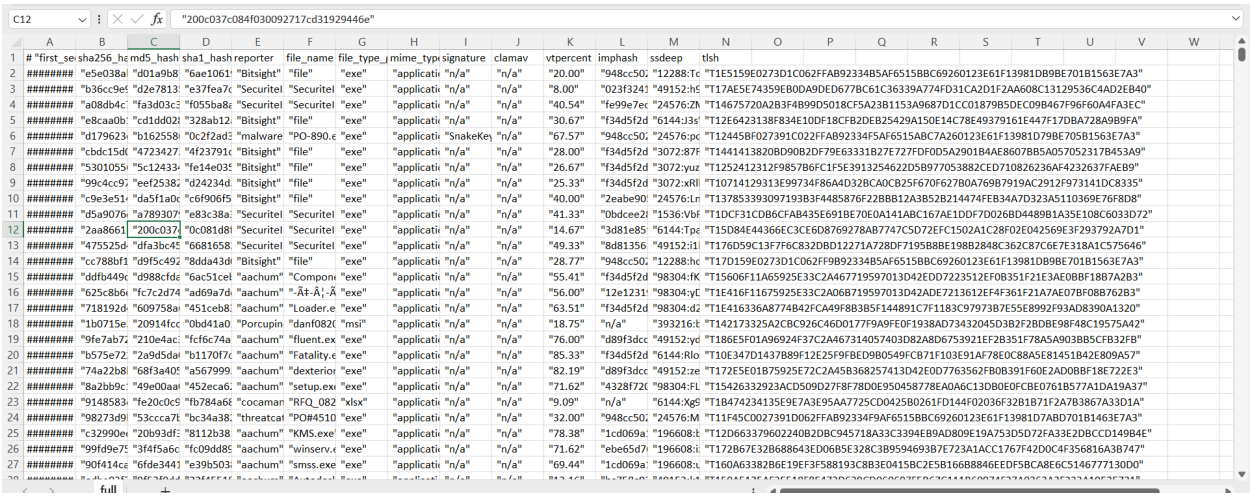


Name	Type	Compressed size	Password p...	Size	Ratio	Date modified
full	Microsoft Excel Comma S...	148,327 KB	No	365,902 KB	60%	8/26/2024 7:42 PM

Open the file and delete the entire rows from lines 1-6

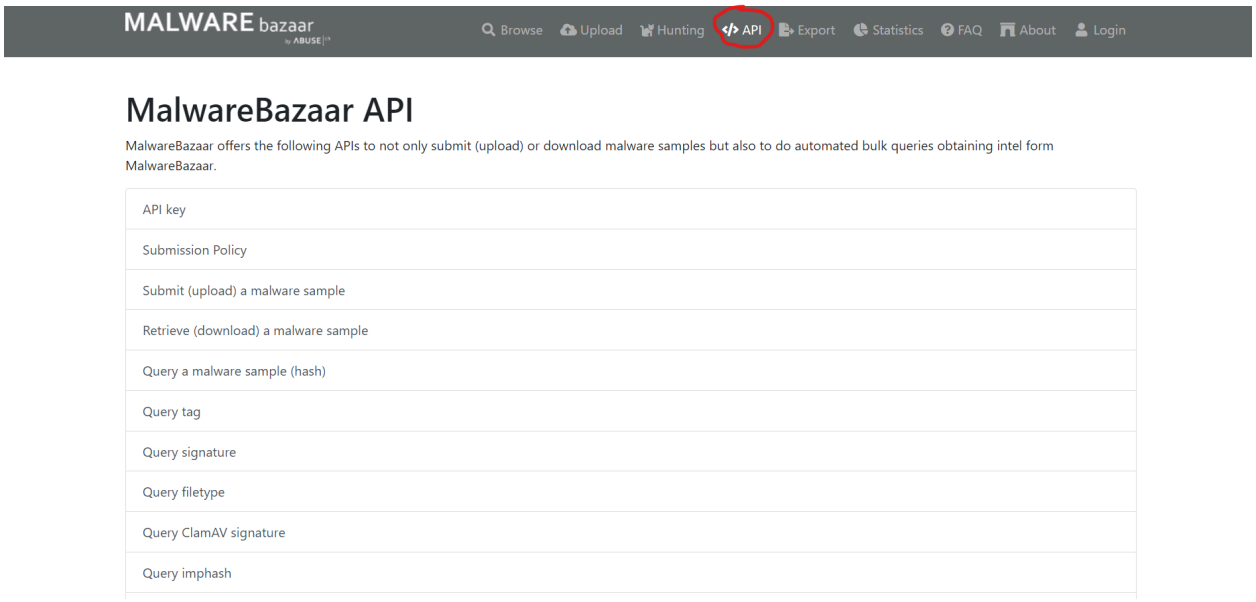


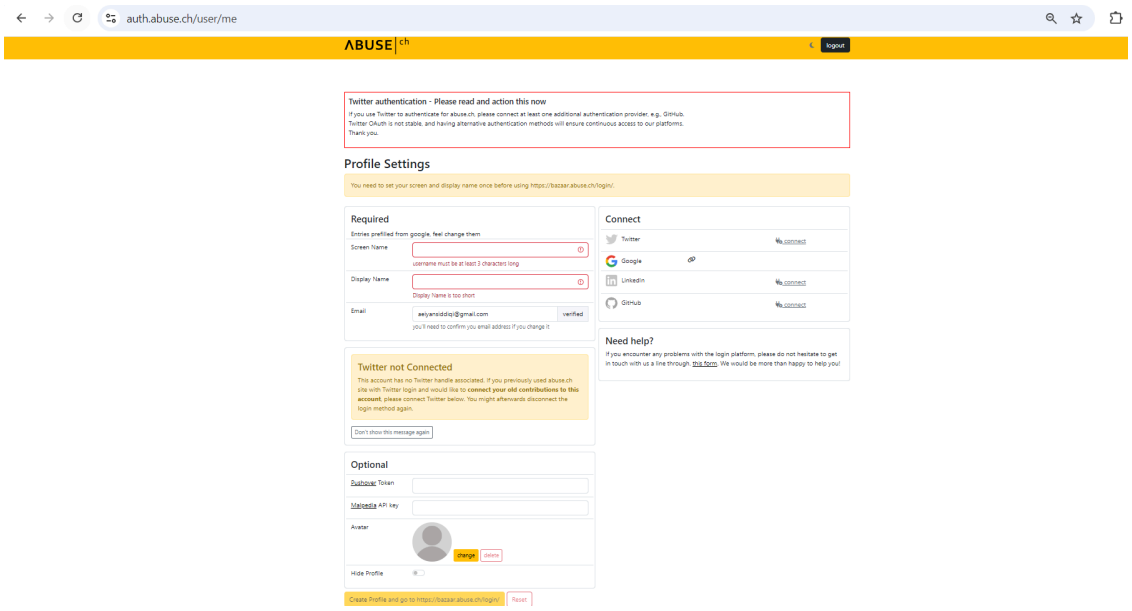
Your data should look something like this:



Save your changes and close the file.

Creating Malware Bazaar Key:





Gathering Data:

1. **Run the Python Script:**
   - Execute the `mbExtractorContinue.py` script to begin the data extraction process.
2. **Open the CSV File:**
   - The script will automatically open the `full.csv` file that you just exported from Malware Bazaar.
3. **Process Each Row:**
   - The script will iterate through each row in the CSV, using the Malware Bazaar API key to create a JSON file containing all relevant data associated with the hash number in that row.
4. **Adjust the Starting Line:**
   - If you need to specify a starting point within the CSV file, modify the script at line 58. Replace the indicated number with the line number from which you want the script to start processing.

```
55
56  # Load the CSV file, skipping bad lines and starting from row 2269
57  csv_file = 'hashFull.csv'  # Replace with your CSV file name
58  df = pd.read_csv(csv_file, on_bad_lines='skip', skiprows=range(1, 74020))  # Skip rows before 2269
```

5. **Customizing Data Extraction:**
   - This adjustment is useful when you want to control the amount of data extracted. For instance, if you only need a subset of data, you can set the script to start from a specific line.
   - Conversely, if you want to expand your database, you can run the script again, starting from where you left off.
6. **Adding your API Key:**
   - On line 11, add your API Key from Malware Bazaar within the quotation marks

```
mbExtractorContinue.py
1   import pandas as pd
2   import requests
3   import json
4   import os
5
6   # Function to make the request and save the response as a JSON file
7   def fetch_json(hash_number, output_dir):
8       hash_number = hash_number.strip().replace('"', '')  # Clean the hash number
9
10      # Replace 'your_api_key_here' with your actual Malware Bazaar API key
11      api_key = 'YOUR API KEY HERe'
12      headers = {
13          'API-KEY': api_key
14      }
```

7. **Track Progress:**

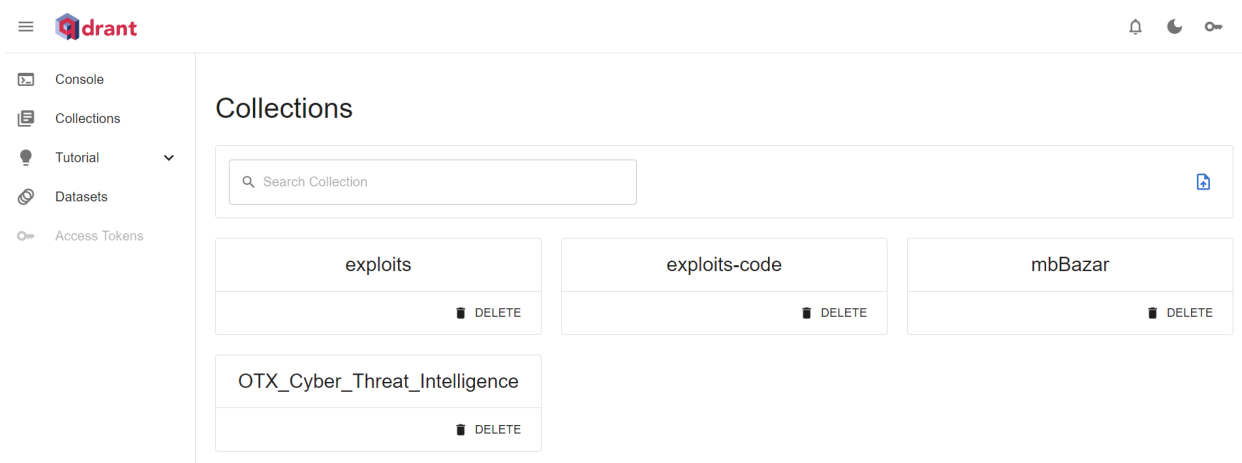
- Keep a record of the last line processed by noting the final output from the previous run. This will help you determine the starting line for the next time you execute the script.

```
have mixed types. Specify dtype option on import or set low_memory=False.
 df = pd.read_csv(csv_file, on_bad_lines='skip', skiprows=range(1, 74020))  # Skip rows before 74020
Line 74020: Response saved to output_json\51f184da384051081835679aee076095873a3e50bb092d0172eedbba2b417c26.json
Line 74021: Response saved to output_json\6a9503cce99eded0e15e83923f65fdbc7d8e5de36cb6f1bd295090eb797bd56a.json
Line 74022: Response saved to output_json\a1f0e5a4b1d39aa0509c231bf5da8c9fe8da5bf66060aeca87cfcbc2dc5dc13d.json
Line 74023: Response saved to output_json\2699d5711c30178c92be712a69600ffb8adc6982b03b1d9cebb8e745ae4fbebf.json
```

-

Embedding the Data:

1. **Prepare Qdrant:**
   - Before embedding your data, start the Qdrant Docker image by running the necessary command in your terminal. This will allow the Qdrant application to be up and running.
   - You can confirm that Qdrant is running by checking the Qdrant UI, which should be accessible at `http://localhost:6333/collections`.



2. **Set Up the Collection:**
   - Open the `qdrantEmbed.py` script in your code editor.
   - If the collection hasn't been named yet, assign it a name within the script. For this case, the collection has already been named `mbBazar`.

```
10  # ==========================================================================
11  QDRANT_URL = 'http://localhost:6333'
12  COLLECTION_NAME = 'mbBazar'
13  VECTOR_SIZE = 1024
14  BATCH_SIZE = 100
15  EMBEDDING_DELAY_SECONDS = 5
16  # ==========================================================================
```

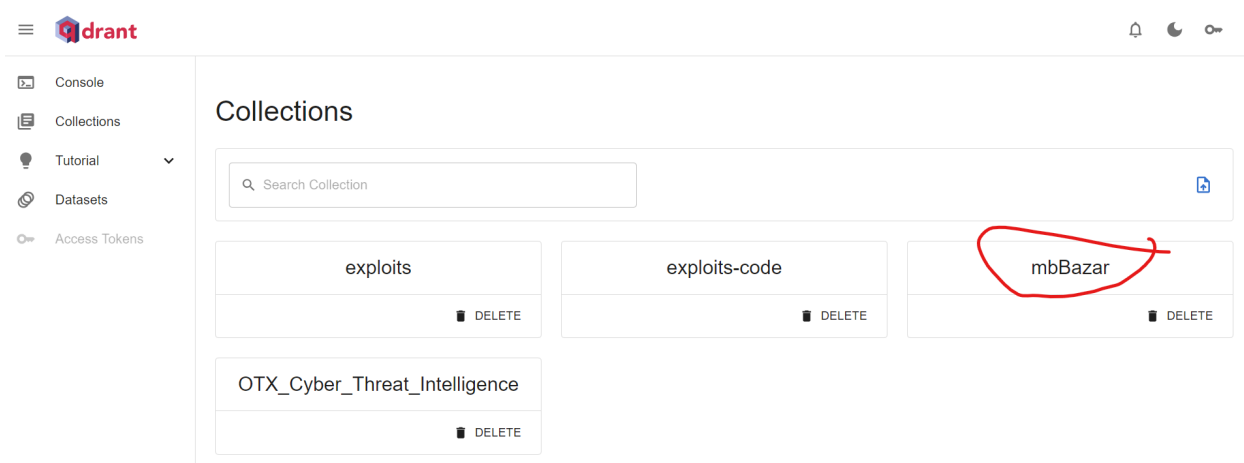
3. **Adjust File Path:**
   - Find the line in the script where the file path is specified (line 323). Update this path to point to the location where your JSON files are stored.

```python
if __name__ == "__main__":



    create_collection()

    text, meta = read_mb_json_("../mbJson/output_json")

    load_embeddings_custom_metadata(text, meta)

```

4. **Run the Embedding Script:**
   - After making the necessary adjustments, run the script by entering the command to execute it. This will begin the process of embedding the data.

```
Malware_Analysis_Rag$ qdrantEmbed.py
```

5. **Monitor Progress:**
   - The embedding process can take some time, ranging from a few minutes to several hours, depending on the amount of data you are working with.
6. **Verify Embeddings:**
   - Once the script has finished running, you should be able to see the embedded data by visiting the Qdrant UI at http://localhost:6333/collections.



-

Point 1

Payload:

metadata


```
{ 15 Items
  "delivery_method": "web_download"
  "file_name": "3bbbc78ebeb54f753c00ae5bed774e4b"
  "file_size": 26000
  "file_type": "elf"
  "file_type_mime": "application/x-executable"
  "first_seen": "2024-05-07 22:10:39"
  "imphash": NULL
  "last_seen": NULL
  "md5_hash": "3bbbc78ebeb54f753c00ae5bed774e4b"
  "origin_country": "FR"
  "reporter": "zbetcheckin"
  "sha1_hash": "3a1fc8b2a4da874a0283d8638e6b020b9b64dfa9"
  "sha256_hash": "0c6b5bff11e2b395c0b1bf3dea5b711ee2a3b7a142bfb4d581 … "
  "sha3_384_hash": "3ccd670d264735dff1924dc4504d1c1d564d014490cbdf768a … "
  "tlsh": "T1F5C2D7D5BCC28A5BC5C403BABF2F4289331267A4D1CF7703 … "
}
```


page_content

YARA Rule Name: setsockoptYARA Rule Name: unixredflags3

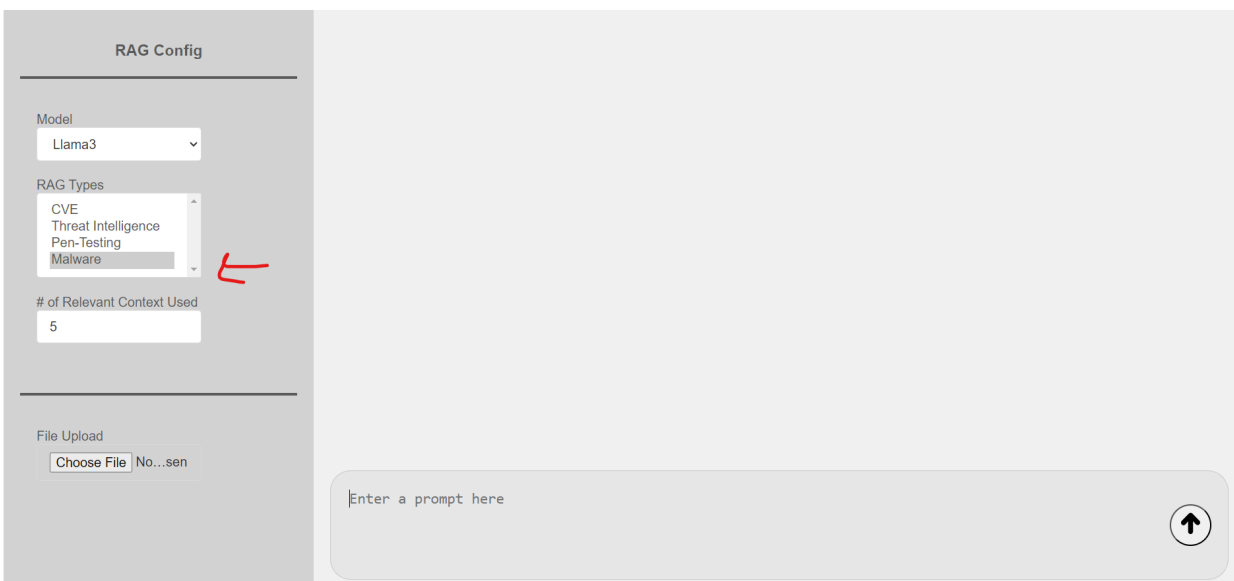
Running the Program:

Your output may vary depending on whether you use the Graphical UI or the text-based UI.

Graphical UI:

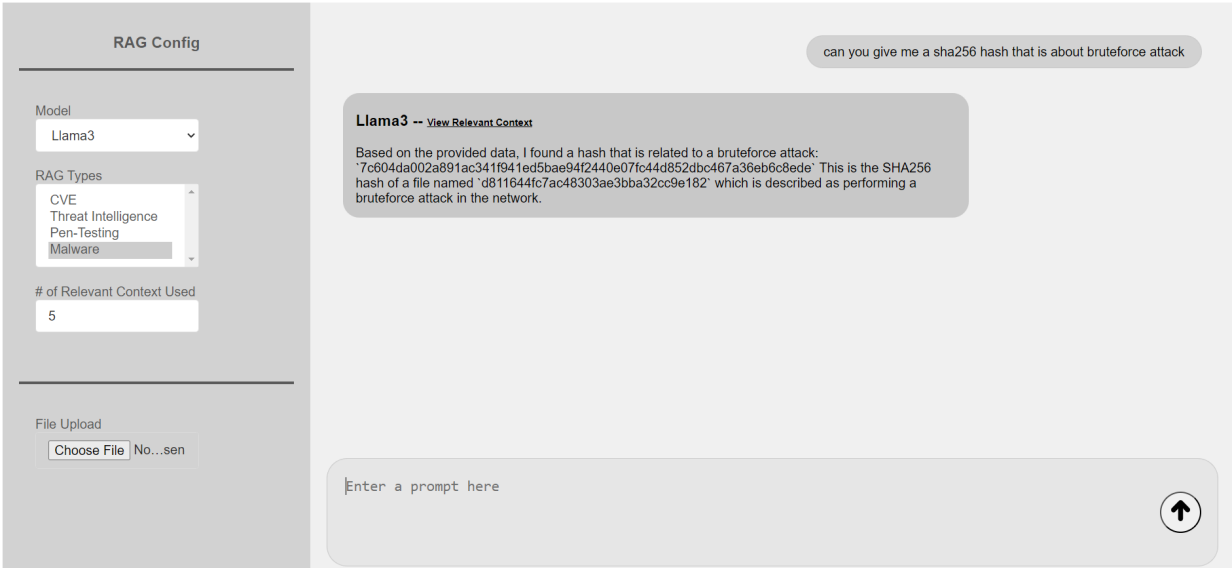
1. To run the UI along with the RAG model, start by cloning the Git repository:
    - (Insert the Git repository link here)
2. Next, follow these steps to launch the server UI:
    - Navigate to the backend directory:
        - `cd RAG_App`
        - `cd backend`
    - Start the server:
        - `python3 server.py`
3. With the server running, you can now launch the frontend by:
    - Navigating to the frontend directory:
        - `cd RAG_App`
        - `cd frontend`
    - Starting the frontend:
        - `npm run dev`
4. Finally, access the UI by visiting the following link in your browser:
    - `http://10.1.74.123:3000/`

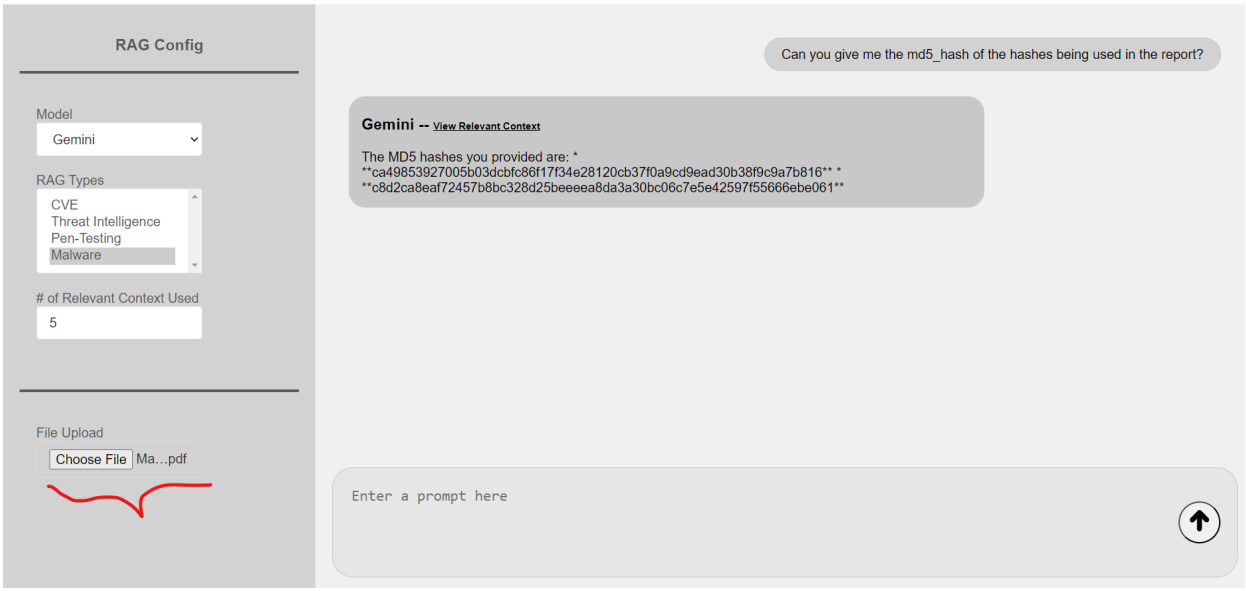
- You should see the following page display:



- You can ask it questions such as:


RAG Config
Model
Llama3
RAG Types
CVE
Threat Intelligence
Pen-Testing
Malware
# of Relevant Context Used
5
File Upload
Choose File No...sen
Can you give me the yara rules on ca49853927005b03dcbfc86f17f34e28120cb37f0a9cd9ead30b38f9c9a7b816
Llama3 -- View Relevant Context
Based on the provided data, the YARA rules associated with the hash `ca49853927005b03dcbfc86f17f34e28120cb37f0a9cd9ead30b38f9c9a7b816` are: 1. `DebuggerCheck__API` 2. `DebuggerCheck__QueryInfo` 3. `iexplorer_remcos` 4. `INDICATOR_SUSPICIOUS_EXE_UACBypass_CMSTPCOM` 5. `maldoc_find_kernel32_base_method_1` 6. `Remcos` 7. `REMCOS_RAT_variants` 8. `ThreadControl__Context` 9. `Windows_Trojan_Remcos_b296e965` 10. `win_remcos_auto` 11. `win_remcos_w0` 12. `yarahub_win_remcos_rat_unpacked_aug_2023` These YARA rules are used to identify malware, specifically the Remcos RAT (Remote Access Trojan), which is a type of malware that allows an attacker to remotely access and control a compromised system.
Enter a prompt here


RAG Config
Model
Llama3
RAG Types
CVE
Threat Intelligence
Pen-Testing
Malware
# of Relevant Context Used
5
File Upload
Choose File No...sen
can you give me a sha256 hash that is about bruteforce attack
Llama3 -- View Relevant Context
Based on the provided data, I found a hash that is related to a bruteforce attack: `7c604da002a891ac341f941ed5bae94f2440e07fc44d852dbc467a36eb6c8ede` This is the SHA256 hash of a file named `d811644fc7ac48303ae3bba32cc9e182` which is described as performing a bruteforce attack in the network.
Enter a prompt here



To Learn More About running the RAG UI Program and accessing a Step-by-step demonstration Visit our [RAG_App Github](#).

UNIVERSITY of GUELPH

CANADA CYBER FOUNDRY

CYBERSCIENCELAB

Secure the future with us

cybersciencelab.com