Multiple Document Comparison

Table of Contents

1. Overview

The objective of this project is to automate the process of comparing multiple reports of patients.

The raw input to the workflow is a csv file with rows of medical reports and each column contains report details. **Pre-processing** of the raw csv file will be first carried out to output the medical reports in a JSON file format. Each report object will also have 2 new attributes after pre-processing:

1. Laymen summary of the report
2. Extracted report text content organised in 3 sub-sections:
    - Diseases Mentioned
    - Organs Mentioned
    - Symptoms / Phenomena of Concern

Next, the pre-processed JSON file will be uploaded to the database. Subsequently, the pre-processed JSON file will be fetched and used as an input to the **comparison** workflow. In the comparison workflow, all the medical reports will be organised by patients and arranged in descending chronological order such that the latest report comes first. Up to 5 latest reports will be considered for comparison. The comparisons are done pairwise between 2 reports. Hence, the maximum number of pairwise comparison for each patient is 4. The latest report will be used as the basis for comparison. For example, Patient 1 has Report 1 (latest report), Report 2, Report 3, Report4, and Report 5. There will be a total of 4 comparison between Report 1 and 2, Report 1 and 3, Report 1 and 4, and lastly, Report 1 and 5.

The pairwise comparison result will be outputted to the database in JSON file format.

Finally, the frontend application will display the raw reports for each patient and all possible comparison results for each patient.

After generating comparison output using Gemini, evaluation on the performance of the output will be required. This documentation will also include the generation of comparison output using GPT. The GPT comparison output will be serves as the ground truth and marks the beginning to an evaluation workflow which will be developed in due course.

2. Getting started

2.1. Local Installation

Ensure that you have installed VSC and MongoDB compass locally by following the instructions in the links below:

- https://code.visualstudio.com/download
- https://www.mongodb.com/try/download/compass

2.2. Prerequisites

After the installations, make sure that you are using the correct python version for this project and have set up a virtual environment. Feel free to use any virtual environment you are comfortable with, and the packages required are in the requirements.txt file. Alternatively, you can follow the steps below for venv setup:

1. Clone the remote repository into your local directory
    a. Open VSC
    b. Change directory to where you would like to clone the remote repository
    c. Go to gitlab repository
    d. Copy the html link from gitlab repository
    e. Paste the html link copied into the terminal in this format:
    Git clone <html link>
    f. Change directory to the cloned project folder
2. If you have more than 1 python versions installed locally, check python version
    a. Window+R
    b. Type “sysdm.cpl”
    c. Click on “Advanced”
    d. Click on “Edit” under “System variables”
    e. Make sure the path of the python version 3.10.5 is placed before all other versions
    f. Then save the changes
    g. Close ALL VSC windows and restart the application
    h. Double check the current python version by typing “python –version” into command prompt
    i. List the paths of python versions installed by typing “where python” into command prompt
3. Set up virtual environment
    a. Make sure you are using the python version 3.10.5 locally before you start creating virtual env
    b. Click on any existing .py files in your VSC project folder.
    c. Ctrl + Shift + P to open palette

d. Then click on “Create virtual environment” → Choose python interpreter.
e. Wait for the virtual environment to get created
f. Note that once this is completed successfully, it would also mean that requirements.txt had been ran successfully too
g. To check what is installed so far in the environment: pip list
h. To rerun requirements.txt if there are any changes: pip install -r requirements.txt

2.3. Connection to APIs

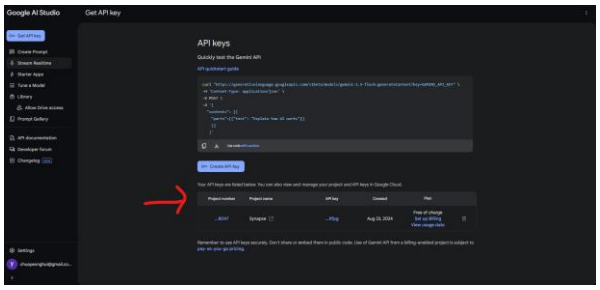
2.3.1. MongoDB

The database application for this project is MongoDB atlas. Establish a connection with the project database by following the steps below:

1. Do a web search for “MongoDB atlas”
2. Create an account with MongoDB atlas
3. Share your email with the database owner to receive the invite to join the project database.
4. Once in the project overview page, add your IP address under the “Network Access” tab found in the sidebar
5. Then, add yourself as a User for connection with the database under the “Database Access” tab found in the sidebar
6. Next, establish the connection to the database under the “Clusters” tab found in the sidebar
    a. Select “MongoDB compass” connection method.
7. You can only establish a connection with the database once the previous steps are completed and your IP address is added
    a. IP address changes every time you connect to a different Wi-Fi.

2.3.2. Gemini

We will require connection to Gemini API to generate responses. Hence, obtain your unique API key from Google AI Studio. The model we will be using is open sourced and free of charge. Once you obtain the API Key, it will be reflected in the webpage.



3. Methodology using Gemini

3.1. Preprocessing Reports

Firstly, we will run the script called **pre_processing.py**. This script processes radiology reports to generate structured summaries and layman-friendly explanations using the Gemini Generative AI model. To generate the structured summaries, it extracts and categorizes information into three sections: diseases mentioned, organs mentioned, and symptoms or phenomena of concern, ensuring no duplication or inference occurs between categories. The input radiology reports are stored in a CSV file and the script will converts each row into a JSON structure with both raw and processed data to include the structured summaries and laymen explanation. The script incorporates a 30-second delay between API calls to manage rate limits and includes error handling to ensure robustness during processing and data upload. Finally, the script will upload the JSON file output to a MongoDB collection.

Steps to run the script:

1. cd to the folder containing the script
2. Add the input csv file containing the raw reports into the same folder containing the script.
3. Ctrl+f the script for "IMPORTANT" and edit the portion of the code stated in the docstring.
    a. Replace Gemini API key with yours
    b. Edit MongoDB credentials and check the names of "db" and "collection" to be used
    c. Replace "df" to the correct filename to be used as input (if incorrect)
4. In the terminal, run "python pre_processing.py"

Once the script is successfully run, you will be able to see a new collection in MongoDB compass with the name defined in the "collection" variable in the script. We will be using this collection as our input to the workflow for comparing multiple reports next.

3.2. Comparing Reports

Next, we will perform comparison between reports from the same patient using the pre-processed reports obtained from the workflow above. In general, the script in this section compares de-identified radiology reports grouped by patient ID to identify changes over time. It retrieves reports from a MongoDB collection and organizes them chronologically. For each patient with more than five medical reports, it compares only the most recent five reports to detect Differences, New Developments, or descriptions that are No Longer Mentioned, following structured comparison logic. These comparisons are generated using the Gemini Generative AI model, which categorizes observations across sections such as "Diseases Mentioned," "Organs Mentioned," and "Symptoms/Phenomena of Concern." The output is stored in a MongoDB collection as a structured comparison, with additional logic to handle cases where only one report exists or when no new reports require processing. This workflow acts as a follow-up to the previous pre-processing workflow, leveraging its output to provide insights into longitudinal changes in radiology findings.

There are two versions of the script to perform the above logic. The main difference between the two scripts lies in the structure of the output.

For the script called **comparison_gemini_table.py**, the JSON output file outputs all comparison by `PatientID`.

For the script called **comparison_gemini_sectioned.py**, the JSON output file has an additional level to categorise the comparison by “PatientID” and the “Sections” (ie Diseases Mentioned, Organs Mentioned, Symptoms/Phenomena of Concern).

Please refer to the sample JSON file output stored in MongoDB database for reference.

3.2.1. Table Output

The script to the table output structure is **comparison_gemini_table.py**. The JSON file output to this script is suitable for translation to a table format in frontend application because each object in the "Comparisons" component of the JSON file corresponds to a row in the comparison output table.

Steps to run the script:

1. cd to the folder containing the script
2. Ctrl+f the script for "IMPORTANT" and edit the portion of the code stated in the docstring.
    a. Replace Gemini API key with yours
    b. Edit MongoDB setup portion of the code and check if the declaration to all the variables in this section is accurate. The "collection" variable stores the input to the script and the "comparison_collection" variable defines the name of the output collection in MongoDB.
3. In the terminal, run "python comparison_gemini_table.py"

Once the script is successfully run, you will be able to see a new collection in MongoDB compass with the name defined in the "comparison_collection" variable in the script.

3.2.2. Section Output

The script to the section output structure is **comparison_gemini_table.py**. The JSON file output to this script is suitable for filtering the comparison output by section names in frontend application because for each patient, the comparison object in the "ComparisonResults" is further categorised by section name and is not outputted all at once like in the table output structure.

Steps to run the script:

1. cd to the folder containing the script
2. Ctrl+f the script for "IMPORTANT" and edit the portion of the code stated in the docstring.
    a. Replace Gemini API key
    b. Edit MongoDB setup portion of the code and check if the declaration to all the variables in this section is accurate. The "collection" variable stores the input to the script and the "comparison_collection" variable defines the name of the output collection in MongoDB.
3. In the terminal, run "python comparison_gemini_sectioned.py"

Once the script is successfully run, you will be able to see a new collection in MongoDB compass with the name defined in the "comparison_collection" variable in the script.

4. Evaluation using GPT

To evaluate the performance of the comparison output generated by Gemini LLM, we will evaluate based on accuracy and faithfulness.

4.1. Accuracy

To evaluate accuracy, we will obtain ground truth using GPT LLM. The prompt utilised remains consistent and the only variable is the type of LLM used.

4.1.1. Generating using GPT

In this section, we will obtain the comparison output using GPT LLM by using the notebook called comparison_gpt_table.ipynb. The input to this notebook is the same as the input to the comparison_gemini_table.py script. Likewise, MongoDB will store the output to the notebook.

Steps to run the script:

1. Open the notebook called comparison_gpt_table.ipynb
2. Select the correct kernel with the necessary imports downloaded
3. Ctrl+f the script for “IMPORTANT” and edit the portion of the code stated in the docstring.
    a. Edit MongoDB setup portion of the code and check if the declaration to all the variables in this section is accurate. The “collection” variable stores the input to the script and the “comparison_collection” variable defines the name of the output collection in MongoDB.
4. Open the .env file and replace the following information:
    a. OPENAI_API_KEY = "YOUR_API_KEY”
    b. OPENAI_API_BASE = "https://genai-openai-eus.openai.azure.com/"
5. Run all the cells

Once the notebook is successfully ran, you will be able to see a new collection in MongoDB compass with the name defined in the “comparison_collection” variable in the notebook.

The next step will be to evaluate the accuracy of Gemini against GPT generated ground truth.

5. Resources

5.1. Gitlab

The relevant scripts/notebooks documented in this word document, requirements.txt, and this word document have been uploaded to Synapxe's Gitlab repository here:

https://gitlab.com/c0desnippet/multiple_document_comparison.git

In addition to the above resources, you will require the raw patient reports in csv file format. Due to data sensitivity, the raw data **will not be** uploaded to Gitlab and will be attached in the handover email via Synapxe's email.