Implementation of JSON natural-language query

Rishi Jain

Deployment: https://jsondataquery-kl2hgzfgej4fh9eesynqgh.streamlit.app/

Github: https://github.com/rishiraij/JSON_data_query

The implementation of the system that I chose is based on MongoDB+GPT as the database/querying backend and Streamlit for the cloud-deployed frontend. Unlike natural language documents, chunking and embedding does not make sense for JSON files as it does not allow for meaningful analysis of data across groups of objects. While MongoDB was new to me, it is the natural choice for querying large JSON-style data, and draws many parallels to SQL with the aggregation pipelines. Additionally, its popularity also meant that LLMs like GPT 3.5/4 are already trained on data detailing how to write the aggregation pipelines.

The prompt itself was tuned based on live interactions with GPT. In this case, the biggest challenge was getting the output formatting to match the requirements of MongoDB's queries. To make this happen, I had to do a few things. First, I had to explicitly instruct the model to avoid certain common pitfalls. Next, I employed one-shot learning by providing an example of what a sample output should look like. Finally, as the prompt is kept general to adapt to any JSON file, I added an example document from the user-uploaded file so the language model knows what the formatting of keys/values is for the provided file.

The processing pipeline of the application begins with a user being prompted to input a JSON file, which is then pushed to a MongoDB collection. If this file is not formatted correctly, an error message is displayed. Otherwise, users are then directed to provide natural language queries about the data. These queries are then combined with the prompt detailed above and given to a chat-based generative language model with an initial temperature of 0. If the returned JSON aggregation pipeline does not run on the database, then the error message is appended to the chat history along with the failed query. The LLM is then prompted to make a second attempt, this time with a temperature of 0.2 to promote new solutions. If either the new result is not well-formed or the query cannot be answered, then a message explaining this is displayed. Otherwise, the JSON results of the aggregation pipeline are then combined with the original query to generate a natural language response which is returned back to the user.

Overall, I think this method has a lot of promise. For many questions, the pipeline is able to provide an accurate and simple answer. However, there are some limitations. For more complex questions, such as the average age of French members, the aggregation pipeline provided by the LLM is close to the answer, but produces scripts with small bugs in the string comprehension/arithmetic required to convert ‘dob’ to age. Additionally, the method is restricted to JSON files where there are no missing values in documents, and does not account for files where this is not the case. Given more time, there are a few things I would like to do. One is to build a comprehensive evaluation set to test the pipeline on different datasets. This is crucial for benchmarking performance. I

would also like to explore how chain-of-thought prompting can help the model improve with more complex queries that require 4+ aggregation commands chained together. Since the failure cases were not far off from the solution, I believe that this technique can prove successful for multi-step JSON queries as it has in other domains of NLP-based tools.

Note: The assignment was to implement an explicit *run_qa(file_path: str, query: str) -> str* function. However, I chose to implement the project differently since it makes sense that for any given JSON, the user may want to ask multiple questions without reuploading the file. Additionally, the (very basic) frontend is much more intuitive to use. While I have never used Streamlit before, I believe the time it took to learn the framework was worth the eventual ease of use of the tool.

Thank you for taking the time to read through, and please let me know if I can help answer any questions!