**ASSIGNMENT**

**Your goal is to use LLMs to provide the ability to get financial insights in a chat-like conversation**

**Data sources:** You are provided a single text file per company. In the text file, there are sentences providing financial information about the company. Sentences are separated by a blank line.

Find sources here: <https://drive.google.com/drive/folders/1iawcu_2OCUFhei4PheUunKBftKpbGzX5?usp=sharing>

**Specifications:**

**LLM**

* The user can only talk to one company at a time.
* There should be a dedicated LLM that handles the flow of the conversation:
  + Ensure the conversational LLM only replies to financial questions.
  + Ensure it appropriately handles the memory of the conversation within a session.
* There should be a domain expert LLM that understands the structure of the templates and generates subqueries to get the right templates from a database. The objective of these subqueries is to get the right templates required to provide a satisfying answer to the user.
  + Implement this part to work with GPT-4 and Mistral. Select the model in the project configuration file or environment variable of the project.

**DATABASE FOR RAG:**

* The database can use either semantic match, keyword matching, or a combination of both.
* Use OpenSearch for this purpose.
* Invent company names for each company\_id (the number in the text files) and enable some filtering method in OpenSearch to filter results by either company\_id or company\_name. You can add other metadata if necessary.
* Implement two additional steps:
  + Script for creating a vector index.
  + Script for document upload/update/removal into the vector index.
  + Ensure your data upload script supports adding new documents to an existing index, not just using the from\_documents method call of LangChain vector store.
  + Only basic CRUD operations are required.
  + Note that some of the sentences have been generated on the fly without a pre-known template but follow a characteristic pattern that can be inferred.

**CHAT:**

* Application - a web server with an HTTP endpoint, which should receive the user question and return the response with an answer to that question. Use any convenient framework for that (e.g., Flask, FastAPI).
* Ideally, use a Docker container for running the web server.
* Chat responses should be in plain text or JSON.

**TEST**

* In the shared link you will also find a list of relevant questions you can use for your test. You can add other questions if you need to test anything specific for your solution.

**GENERAL:**

* Please bear in mind that the ability to retrieve different granularity of responses from the system is crucial. The user can both pose specific questions such as “what is the revenue in Q2 2022” or more general or insightful questions such as “Provide a profitability report of the company in the last years”. In the latter case, the relevant subqueries would need to be generated. The system should be “smart enough” to tackle both cases.
* Write clean and commented code.
* The project must contain documentation of all parts in a README file.
* All dependencies should be mentioned in a separate file (e.g., requirements.txt if using pip).
* Code should be runnable end-to-end.
* Use best practices for error logging.
* You can use other frameworks, models if deemed necessary.
* Send your assignment in a ZIP compressed folder.

**Optional improvements:**

* Enable users to pose questions that involve knowledge from different companies.
* Incorporate long-term memory across sessions as an improvement.
* Consider caching responses with a retention period of 4 weeks.