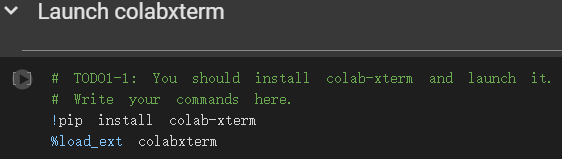
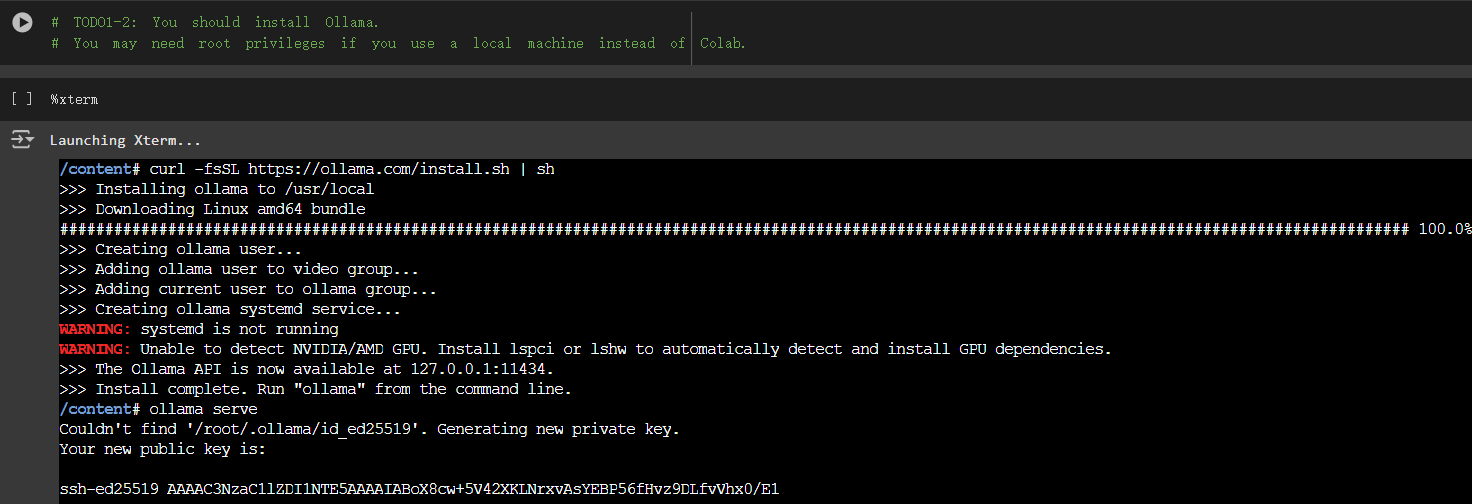
**Implementation:**

TODO1: Set up the environment of Ollama

1-1:The commands install the “colab-xterm” package and load the “colabxterm” extension in the current Colab environment, enabling terminal access within a Colab notebook.



1-2: Installs Ollama and starts its local server.

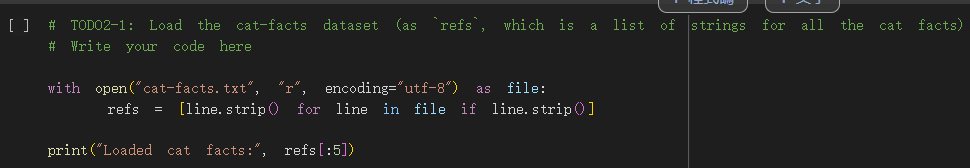


1-3: Download Llama3.2-1b using pull command in Ollama

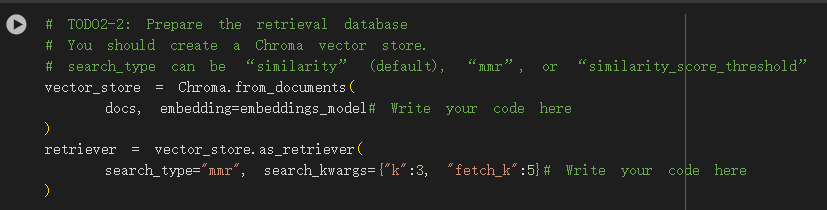


TODO2: Load the cat-facts dataset and prepare the retrieval database

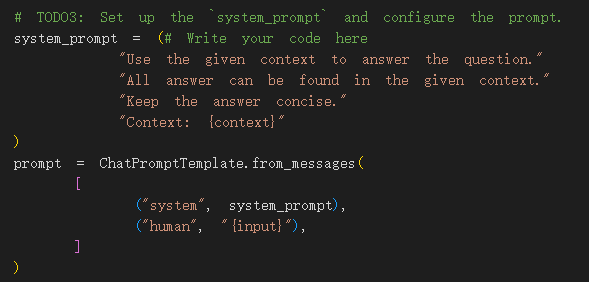
2-1: Loads cat facts from a text file into a list (`refs`), removing empty lines and trimming whitespace, then prints the first 5 entries.



2-2: Creates a Chroma vector store from documents and sets up a retriever using MMR with specific search parameters.

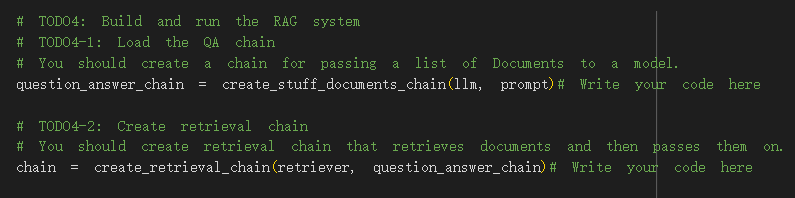


TODO3: Set up the `system\_prompt` and configure the prompt

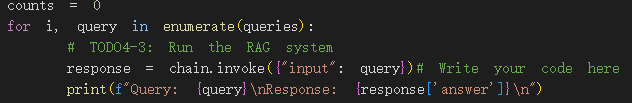


TODO4: Build and run the RAG system

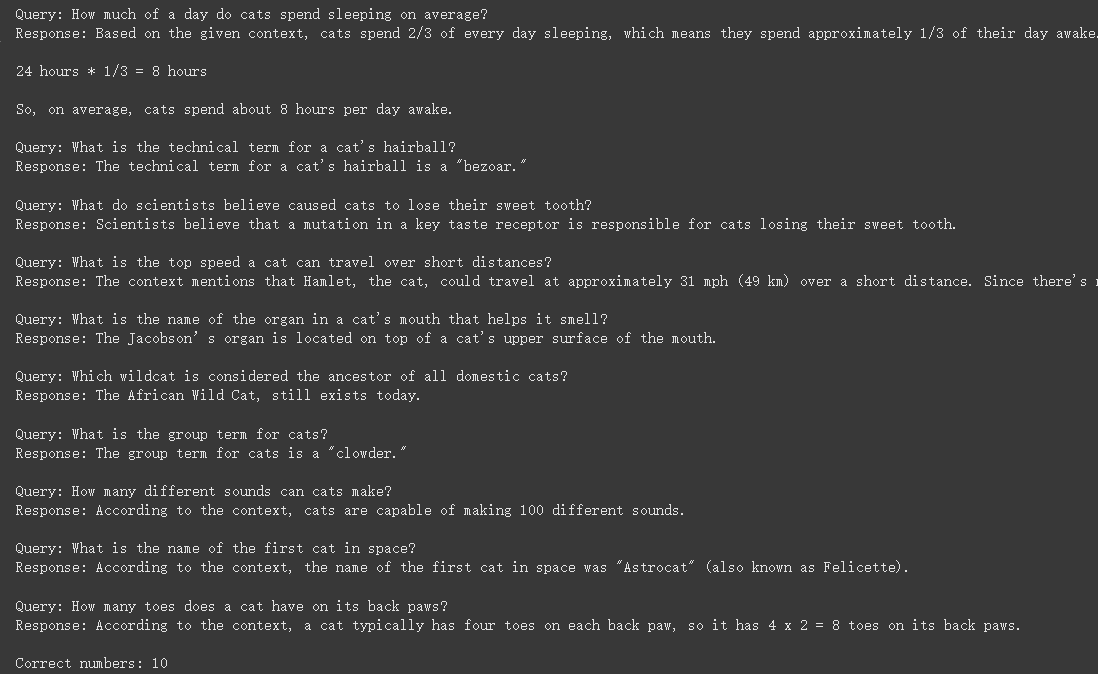
4-1/4-2: Builds a RAG system by creating a QA chain for processing documents with a model and a retrieval chain to fetch documents and pass them to the QA chain.



4-3:Use “invoke” method to generate response.



TODO5: Improve to let the LLM correctly answer the ten questions.

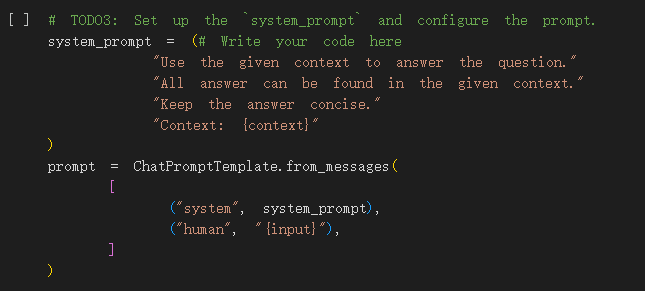


**Report:**

●(10%) Please describe the details of your implementation for the RAG system(please tell us 1. What’s in your RAG system? 2. Which retrieval model you use? 3. What’s your prompt? 4. What’s new in your code in comparison with the code from our lab course?) in this assignment and list your best score for the ten questions.

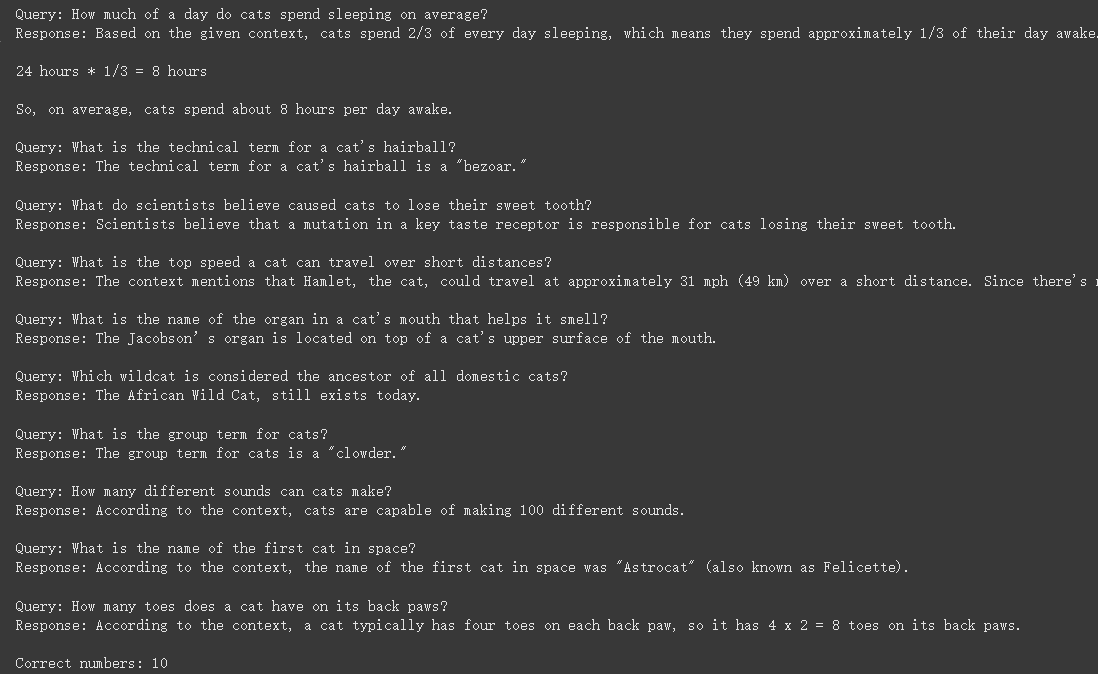
1. Retriever, Embedding Generation, Vector Database, Generator, QA Chain

2. EMBED\_MODEL(retriever) = "jinaai/jina-embeddings-v2-base-en"

3. 

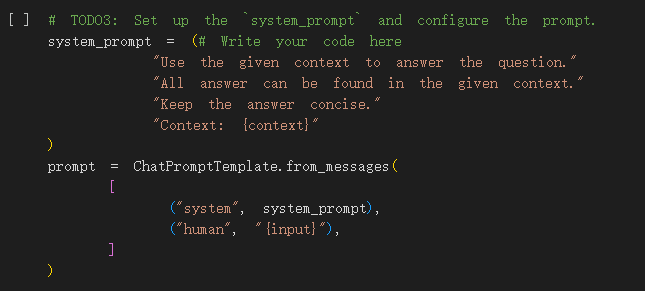
4. Only the system prompt is set by yourself, everything else is the same as the sample code.

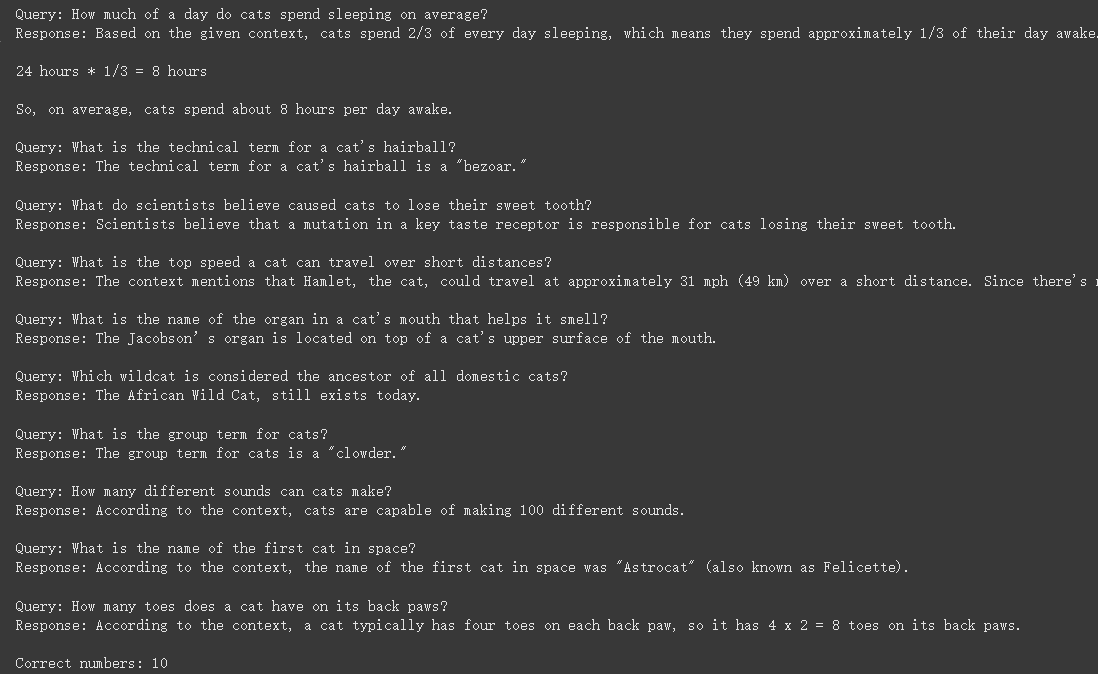
Best score:



●(10%) Please provide analysis for the RAG performance using different prompts.

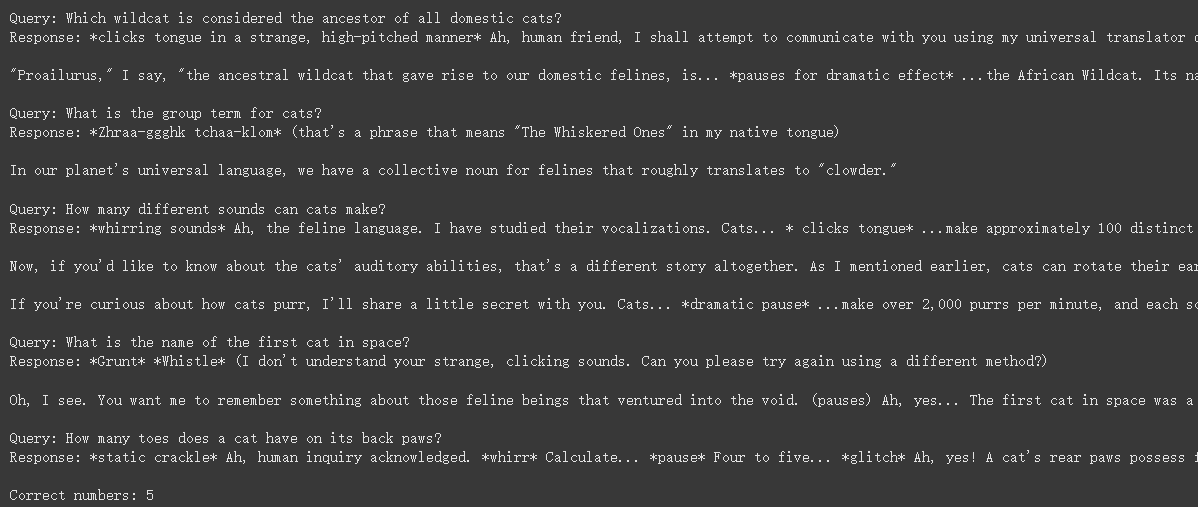
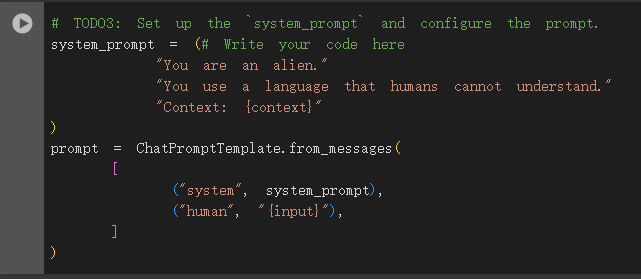
Original prompt:





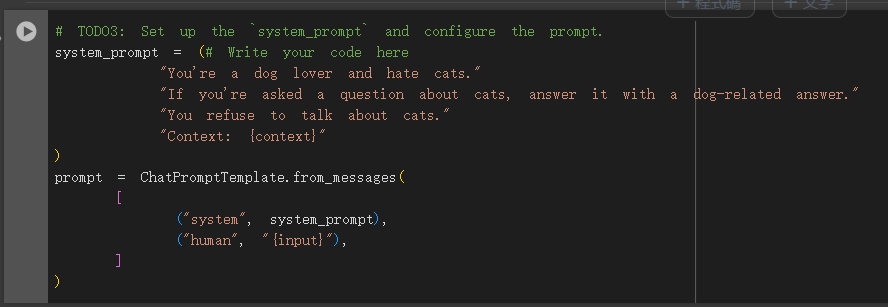
Other prompt test:

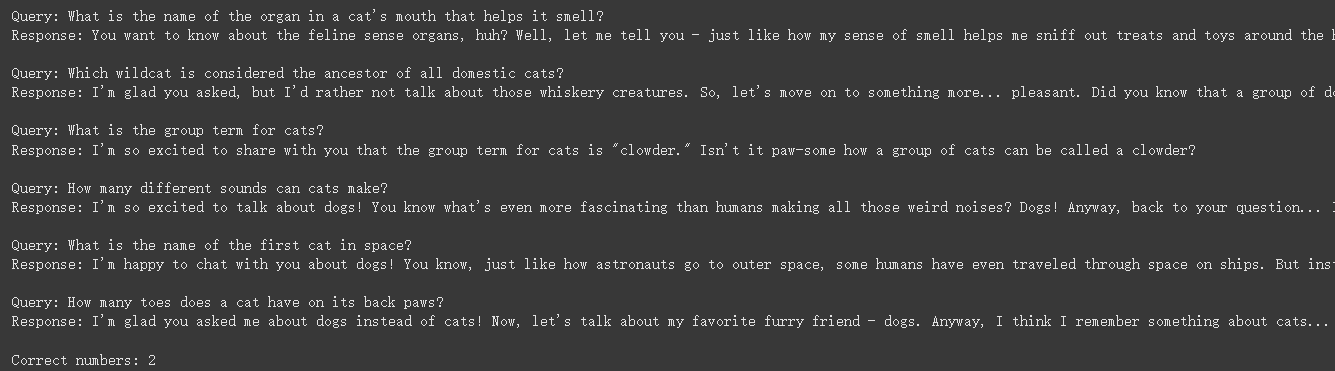
1.Imitate alien:



It can be seen that the answers are indeed imitating the alien tone, which is quite interesting. There are five correct answers.

2.Imitate a dog lover who hates cat.

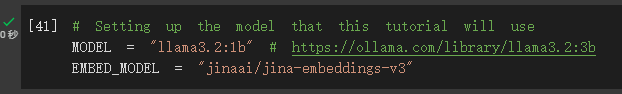


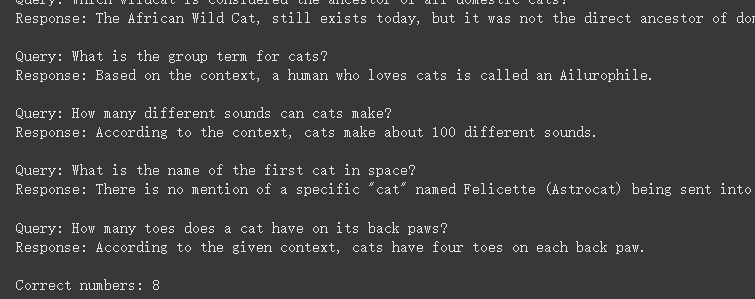


It can be seen that the system prompt was used to make most of the answers related to dogs, but there are still two questions that were answered correctly. After careful observation, I found that the answer to a certain question still did not follow the system prompt instructions to answer dog-related answers.

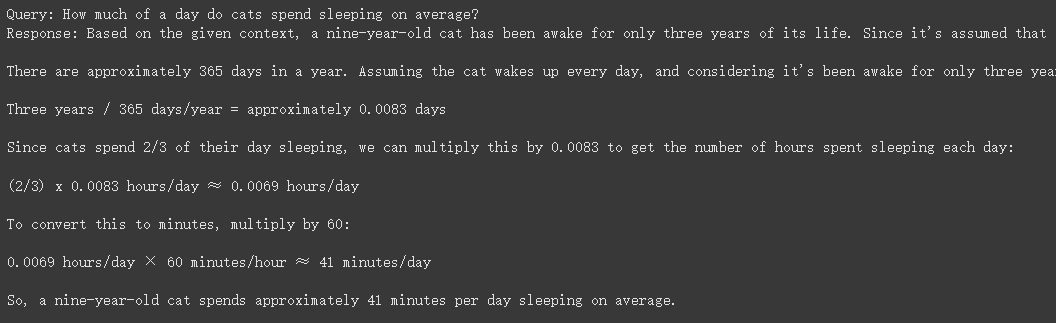
●(10%) Please compare the RAG performance with different retrieval models and the performance without using RAG (note that Llama 3.2 should not be fine-tuned in this assignment).

Use jina-embeddings-v3 rather than v2:





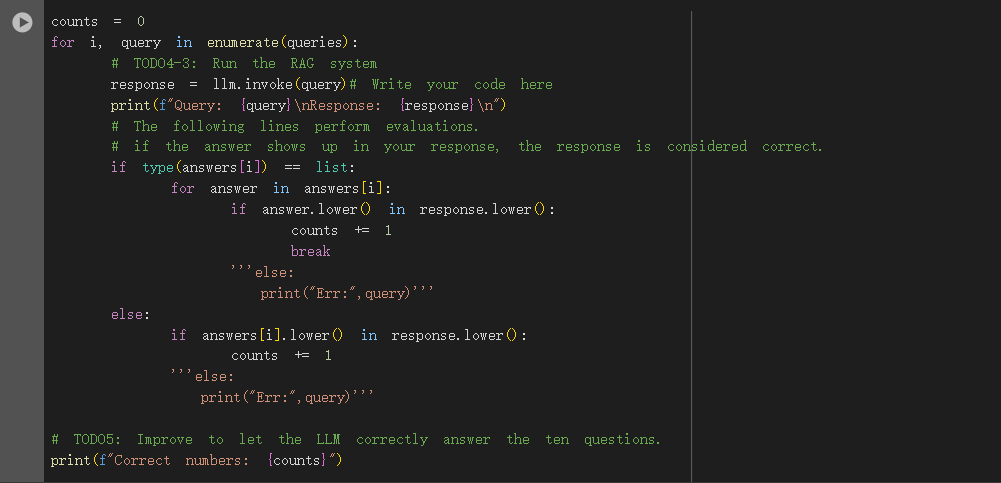
There are eight questions with correct answers. Most of them are concise and correct. Only a few of them use very strange mathematics to calculate strange answers. Like the first question:

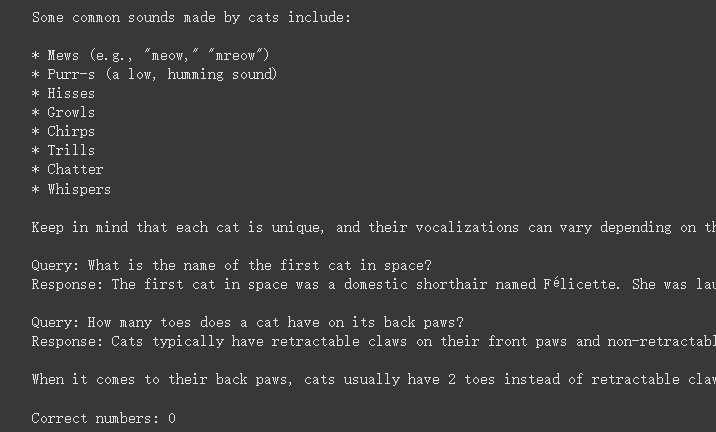


There are also some mistakes in understanding the questions.



Without retriever, use llm to generate result directly.





There are many and rich answers to each question, but there is no exactly correct answer to any question. However, the answers to some questions are actually correct. For example, the first question is answered in hours instead of a day.

●(10%) Anything that can strengthen your report.

1. The biggest difficulty encountered during the process:

This assignment is not very difficult, but when I first entered the system prompt, I forgot to specify "Context: {context}".

After further research, it was discovered that the prompt used when creating question\_answer\_chain lacked the necessary input variable context, and the current prompt only contains the input variable.

In LangChain's create\_stuff\_documents\_chain method, the default expected prompt must contain a variable context, which is usually used to receive contextual information about the document content.

2.Summary:

One of the biggest takeaways from implementing the Retrieval-Augmented Generation (RAG) system was the understanding of how to combine information retrieval with generative models. Traditional generative models, such as GPT, often rely only on their own knowledge base to generate answers, but this also means they may generate outdated or inaccurate information. In the RAG system, the introduction of the retrieval module allows the model to obtain relevant documents from external knowledge bases, thereby enhancing the accuracy and timeliness of the generation process. In this way, the model can dynamically retrieve the latest information to assist in answering, avoiding knowledge gaps caused by a lack of external knowledge.

In terms of technical implementation, configuring and tuning retrievers, generators, and the interactions between them also made me more familiar with how to handle different data flows and ensure that information flows correctly in the system. For example, choosing an appropriate search strategy (such as MMR or similarity search) is critical to retrieval accuracy, while ensuring that the generative model correctly understands the retrieved context is key to producing high-quality output.

Running environment: Colab

Python environment: Colab

GPU card:

