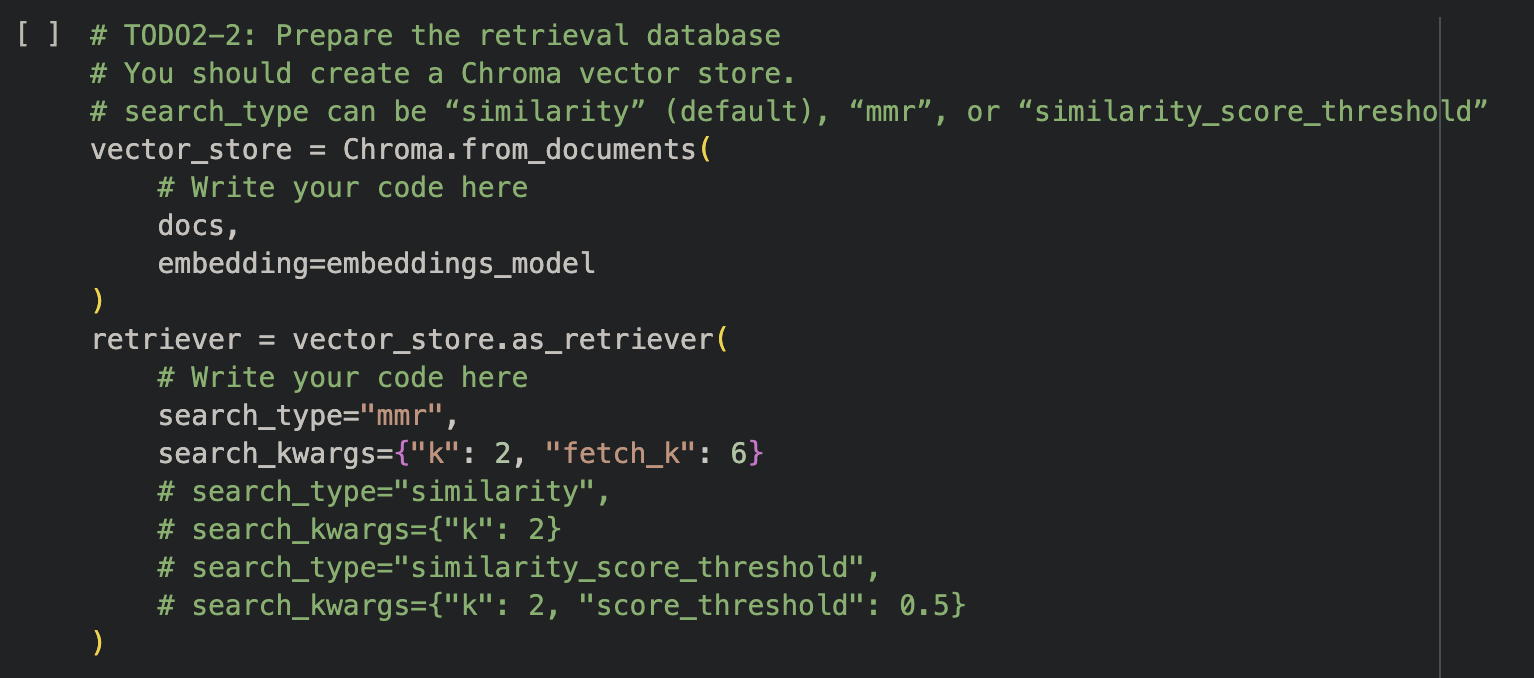
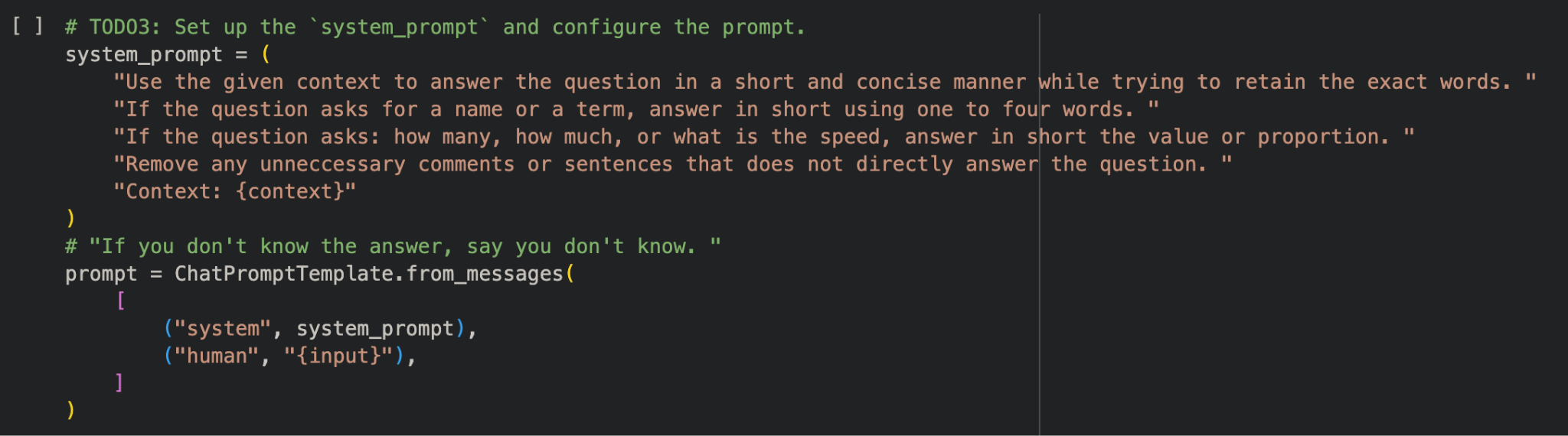
NLP HW4 Report

## Environment

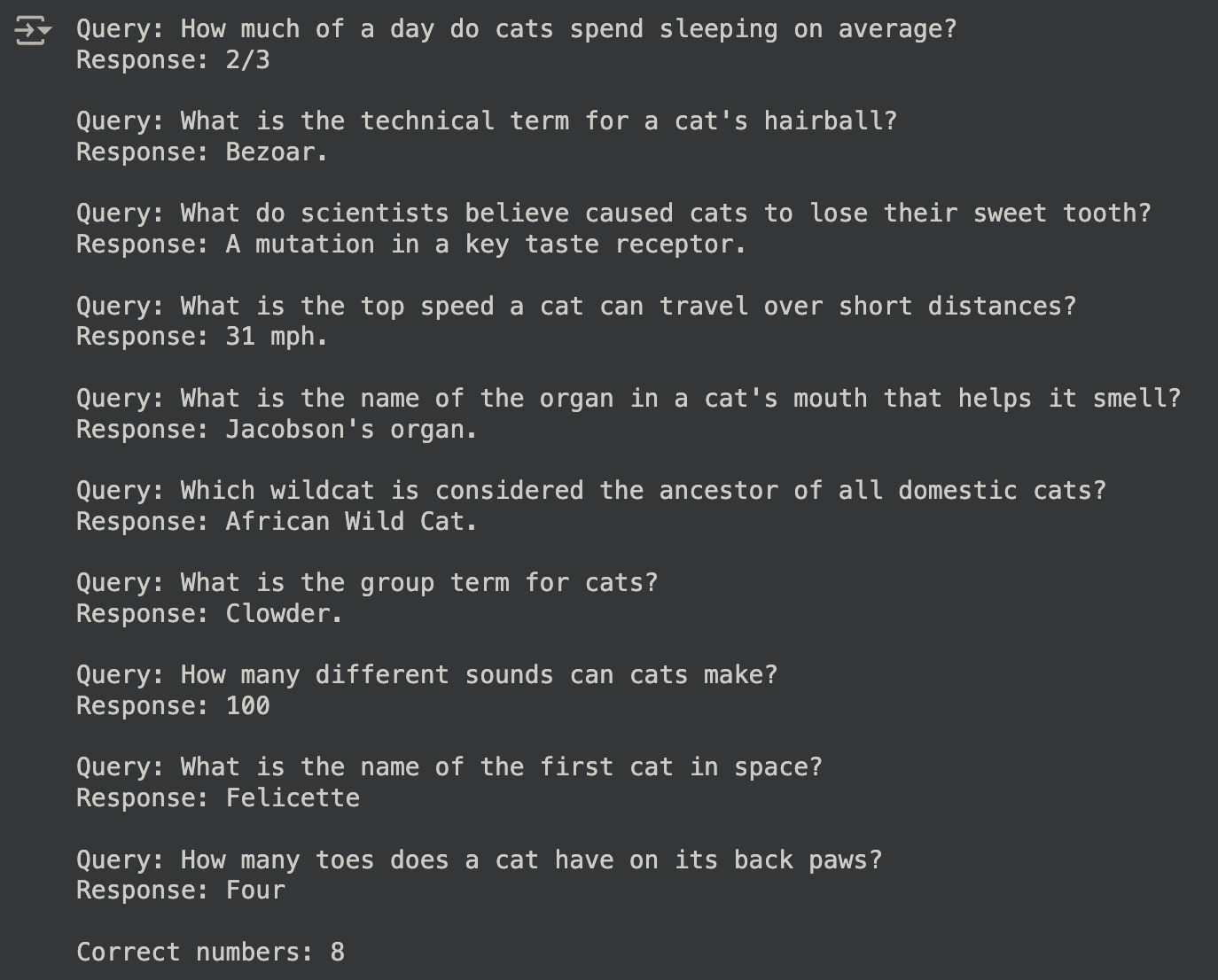
| Running environment | Colab |
| --- | --- |
| Python version | Colab |
| GPU(s) used | T4 GPU |

## Discussions

**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 did 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. There are several components in my RAG system to generate a concise and accurate responses to user queries:
   1. First, read the cat facts dataset from the text file and do some simple preprocessing. Turn each fact into an independent document with metadata (ID) so that LangChain can handle it.
   2. Use HuggingFace embedding model ("jinaai/jina-embeddings-v2-base-en") to encode the documents and queries into dense embeddings for semantic similarity matching.
   3. Vector Store: Chroma, a high-performance vector database, was utilized to store document embeddings and perform retrieval operations.
   4. Retriever: Configured with the mmr (Maximal Marginal Relevance) retrieval model to ensure a balance between relevance and range in the retrieved documents.
   5. Finally, Prompt: A custom system prompt guides the language model to answer concisely using technical terms or exact values where possible. A retrieval chain passes the retrieved context and user query to the language model for final output generation.
2. Maximal Marginal Relevance (mmr) with parameter: {k: 2, fetch\_k: 6}. As it balances the relevance and range of the retrieved documents  
   
3. Custom system prompt designed to guide the model toward generating concise and accurate responses.  
   
4. Main differences
   1. Simple preprocessing of the cat fact dataset
   2. Added custom dynamic system prompts that can adapt to different question types (Explained later in the report)
   3. Used a method (mentioned later in the report) to find the most suitable retrieval model search type and its parameter values. This can help strike a good balance between search speed/range, and response quality (relevance)

**Best score:**



* As you can see, the response is mainly accurate and concise.
* Upon taking a closer look, the answers are actually all correct, but due to the exact matching evaluation, some responses were marked incorrect.

**Please provide analysis for the RAG performance using different prompts.**

I’ve experimented around with many different prompts. Here are my key observations:

**General vs. Specific Prompts**

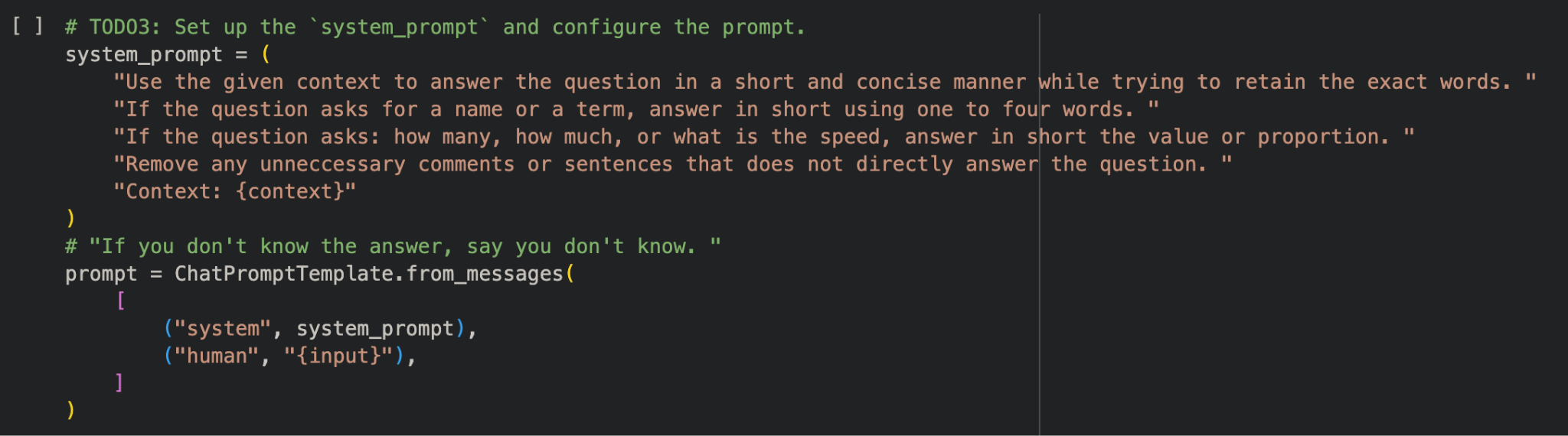
* General prompts like “Use the given context to answer the question concisely” struggle to consistently generate concise answers across all types of questions.
* Specific prompts like “Use 1~2 words” perform better for simple questions (e.g., "how many"), but fail for questions requiring explanations, as they restrict the model's output length.

**Question-Type Sensitivity**

* Questions involving naming terms like “What’s the technical term for …” or values like “How many …” perform relatively well with concise and structured prompts.
* Open-ended questions (e.g. “Why did …,” “What caused …”) require a flexible response format, which conflicts with overly constrained prompts.

**Need for Question-Specific Handling**

* These observations led me to think of the idea of separately handling question types to explicitly guide the model and allow it to better adapt to all query types.

This is my final custom system prompt designed to guide the model toward generating concise and accurate responses.  


1. First prompt tells the model to use given context to answer concisely and retain exact words used in the database (as the evaluation method is exact matching). Without this, the model may provide additional information.
2. Second prompt tells the model to answer in short when asked for a term or a name. Without this, the model may answer in full sentences (e.g. Response: Cats can make over 100 sounds. Expected: 100)
3. Third prompt guides the model to answer ‘how many’ or ‘how much’ questions using a value. Without this, sometimes the model doesn’t respond with numbers, it instead comes up with some random rubbish.
4. Last prompt tells the model to not include irrelevant comments which it sometimes does.

**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).**

I’ve experimented with three different retrieval model search types: Maximal Marginal Relevance (mmr), Similarity, and Similarity Score Threshold

| **Question Types** | **Naming/Technical Terms** | **Quantitative (How many/much)** | **Explanatory (Why/How, cause)** |
| --- | --- | --- | --- |
| MMR | Performs reasonably well, but diversity might introduce slightly less-relevant documents, affecting accuracy for concise questions. | Effective when multiple contexts provide distinct pieces of evidence (e.g., combining units or terms like "mph" and "km"). | Can retrieve diverse perspectives, which is helpful for complex causal answers but might dilute precision. |
| Similarity | Performs well, especially for precise matching tasks where concise terms are found in the most similar document. | Reliable if the closest document directly contains the numeric value. | Tends to retrieve highly relevant but less diverse documents, which may limit the depth of the explanation. |
| Similarity Score Threshold | Works well if the threshold filters irrelevant documents, ensuring concise and accurate answers. | Effective when the relevant document is above the threshold but fails if numeric data is distributed across multiple documents. | Similar to similarity, but the strict threshold might exclude valuable explanatory context, leading to incomplete answers. |
| Model without RAG | Very poor, as the model lacks the domain knowledge. (The terms are never before seen in the pre-trained dataset) | Very poor and inconsistent, as it also lacks the domain knowledge | Decent for general topics within the model’s pre-training but fails when topic extends to deep specific domains |

In conclusion: Maximal Marginal Relevance (mmr) with parameters: {k: 2, fetch\_k: 6} (relatively high fetch\_k ensures search range and diversity, smaller k ensures model focuses on a few data to increase precision) performs best with balanced performance across all question types, particularly when the dataset is dense with overlapping information.

## Explored Methods (Anything that can strengthen your report)

I’ve explored several methods to try to improve my model’s response. However, I came to the conclusion that prompt engineering (ofcourse, paired with a good embedding and retriever model) is the most important factor, a good system prompt directly reflects the relevance and quality of the response. Nevertheless, let me demonstrate some methods I’ve experimented with:

**Example Prompting**

* Including a few examples of how answers for each question type should look like, to guide the model’s response format
* Example for naming: "Q: What is the term for a cat's hairball? A: Bezoar."
* Example for numeric: "Q: How many hours do cats sleep on average? A: 2/3."
* However, the model’s response sometimes looks like: “Response: Expected: 2/3“
* So this method doesn’t really seems to work

**Threshold-Based Filtering**

* Discard responses that deviate significantly from the embeddings of the expected answers:

similarity\_score = cosine\_similarity(embedding\_response, embedding\_expected)

if similarity\_score < threshold:

response['answer'] = "I don't know."

* I use this method to help me find the most suitable retrieval model search type and their parameter values. For example, when the similarity score is low, the retriever may not have fetched the correct fact, then I know I have to increase the search range or change a search type.

**Preprocessing Cat Facts Dataset**

* I tried many preprocessing techniques such as lowercase, remove stopwords, and stemming/lemmatization.
* But these have minimal improvements to the model’s response
* On the other hand, it actually worsens the evaluation as the evaluation method is exact matching…

## References

* TA’s Lab class’ pdf
* <https://python.langchain.com/v0.2/docs/tutorials/rag/#retrieval-and-generation-retrieve>
* ChatGPT to help assist the understanding of the RAG system