RAG Analytics Writeup

Introduction:
In this project, I decided to create a process for automatically generating relevant test sets for my RAG model rather than creating one manually. This gave me greater control over the evaluation metrics I could use and is also more useful as a tool for improving RAG models. Below is an explanation of the features I developed, along with a demo video and list of potential future improvements

Features:

- Manual testing interface to interact with chatbot
- Real-time testing of chunk size and prompt
- Evaluation metrics: Recall, Robustness, Brevity, Knowledge Bounding, and Context Matching
- Testset generation customized to LLM purpose
- Summary Dashboard for quick comparison between different settings results
- Ability to save preferred settings

Testing Interface:
When you type a query into the chatbot input field, it will invoke a response from the most recently evaluated retrieval chain. The response time is quick and each message you send will not depend on any other messages in the message history.

Chunk Size and Prompt Testing:
Using the input fields, users can choose the prompt to run their RAG on and the chunk size which the source document is broken into for the vector database

Evaluation metrics:

1. When deciding how to evaluate the model, I decided that in order to test the simple regurgitation of information, we can create simple questions to ensure that readily available information in the document can be found consistently. This is the recall metric.
2. However, in real-world use, these queries are rarely phrased identically to the source material, and often have mistakes. Therefore, I created the robustness metric, which alters the question in one of 3 ways. The first way is to only apply minor syntax and grammatical errors to the question, the second way is to use a LLM model to complicate the question without changing the correct answer, and the third is to add situational context to the question, which may be distracting for the model. These alterations are chosen from randomly for each question.

3. Then, I chose to include a brevity metric, which scales from 0 to 1 as a comparison between the model answer and the answer generated by our RAG model.
4. Additionally, I considered the possibility of false positives/negatives when being asked a question which is not covered in the source. If a question's answer is available in the source material, it should attempt to answer the question. Conversely, if it is not, then it should not attempt to answer. I used a comparison of the response embedding to an embedding of 'I am not aware of that topic' to check for refusal to answer. Then, I added questions to the dataset which were irrelevant to the source doc, and checked that these were refused. The average score of correct answer and refusal of questions is denoted in the Knowledge bounding metric
5. Finally, I wanted to ensure that the LLM actually had the context to make it capable of answering the question. Because I allow for differing chunk sizes, I could not use the index of the document to check that the same information used to generate the question was used in the LLM context. Therefore, I created a system which tracked the individual indices for both the test set generator and the LLM, and multiplied them by their respective chunk lengths in order to get an approximate region where the context was taken from. Although this system is imperfect, it allowed me to verify whether the LLM context contained information in the vicinity of the content used to generate the question, meaning it is likely that the LLM had the necessary context.

Testset Generator

To generate the test set, I randomly choose a style and error for the transformed question, then generate a simple question from a randomly selected part of the source document. Then, I transform the simple question without changing the answer and store the context used, simple, transformed, and answer for use in evaluation.

Summary Dashboard

The summary dashboard keeps track of the average evaluation metric scores for a given prompt and chunk size attempt. To make it easier when determining the ideal chunk size, I added a graph which plots the cumulative score for the model based on the chunk size. The color of the dot is indicative of the prompt used in that trial.

Save Settings

Once you find good settings for the generators and evaluators, you can save the settings so that when the application is reopened, these settings will be preserved for you.

Demo Video:
https://drive.google.com/file/d/1hkj4_ggxOKdhOMrsZ9yTlbC9r0Ee7SZU/view?usp=sharing

Future Improvements:

- Improve the situational question generator, as it is inconsistent in maintaining the original essence of the question
- Split Robustness into multiple metrics
- Cost analysis
- Reduction in runtime for evaluation
- Fine-tuning embedding model, potentially using Sentence Transformers to make context selection and recall/robustness evaluation more accurate
- Allowing for different models to test