RAG Analytics Writeup

Introduction:

There are currently many RAGAS-style (RAG Assessment) tools which test metrics such as Faithfulness, Groundedness, Answer Relevance, Context Precision, and Context Recall. These evaluations are fairly standard and can be imported using a python library. Some tools even allow for automated test set generation, which will save time and human error in building question sets which cover the source material of the RAG app. In this project, I decided to also create a process for automatically generating relevant test sets for my RAG model rather than creating one manually. However, to make this project valuable my goal was to differentiate it from existing tools in two ways: First, I wanted to change the format of the produced test set by including a “simple” version and a “complex” version of each question. Each of these questions will have the same answer, and should be based on the same context. This allows us to isolate the information gathering capability and the tolerance to human imperfect input in our model, which is more blurred in existing tools as there is no direct comparison. Secondly, I sought to test the ability of the RAG model to know its own limitations. In many applications, it is critical that a model does not “hallucinate” or return false information when presented with a question which it does not have the source material to answer. A prime example of this is when a customer is inquiring about their coverage on their insurance policy, and is relying on a RAG model for help. This is something which I found to be lacking in many current tools for LLM evaluation, as metrics such as “groundedness” are useless for avoiding hallucinations when there is not adequate context to ground a response to. Therefore, I focused this project on highlighting these two improvements, and built a full-stack proof of concept of this platform. My final product is a full-stack application which allows for easy editing of RAG model prompt and chunk size, and generates a customized dataset with the priorities mentioned above. It will also evaluate the RAG model on this dataset, and can return evaluation metrics on 30 question pairs (simple and complex) in around 2 minutes.

Evaluation Metric Choice

When deciding how to evaluate the model, I wanted to prioritize creating metrics which could not simply be evaluated using existing libraries such as giskard. However, standard baselines are also necessary for proper evaluation. 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. To create these questions, I prompted an LLM to generate a question about a randomly selected chunk of the source document, and instructed it to return the result in JSON format. This made interpreting the query output very consistent, and allowed me to avoid parsing errors.

However, in real-world use, these queries are rarely phrased very similarly to the source material, and often have mistakes. Therefore, I created the robustness metric, which is based off of the transformed version of the simple question. This process is explained below in “Test Set Comprehensiveness”

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, prioritizing a shorter answer. I chose to include this metric because in an ideal world, we seek to convey as accurate and complete of an answer as possible with as little redundancy as possible. Any response shorter than or equal to the model response in the test set will have a perfect score for this metric.

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, the model 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. In this example, I made the problem slightly contrived by instructing my RAG model to return the sentence ‘I am not aware of that topic’ explicitly when it does not know the response, but this can be generalized by loosening the similarity threshold and potentially catching a number of variations of refusal to answer.

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. This is comparable to the commonly used context precision metric, but I found it to be critical for my evaluation purposes so I implemented it in my own application as well.

Test Set Comprehensiveness

The Test set generator randomly selects regions of the source document, meaning it is very likely that the question set will comprehensively cover the documents for a large enough test set. Additionally, the simple questions are manipulated by a complexity model which alters the question in one of 3 ways and then applies minor syntax and grammatical errors to the question to simulate human error. The first way is to simply

rephrase the question slightly, the second way is to use a LLM model to significantly 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 randomly for each question. The value of the combination of these two metrics is the ability to explicitly compare the responsiveness of our RAG model to simpler retrievals versus more “human” queries. This allows us to simulate most common complications due to human input, and this ensures that we are covering the relevant source material and asking difficult questions about the content in a variety of ways. Therefore, I believe that the generated test sets are relatively comprehensive, but can always be improved with more variation and addition to complexity factors such as the ones mentioned above.

Improving the RAG model

After building a baseline RAG model, I pursued several methods for improving its performance. Initially, I focused on the features which can be quickly updated in my frontend app. This is the prompt and the chunk size which the document is split into. With my application, I am able to quickly iterate over potential settings and then quickly save them as my preference when they score well. I also decided to add overlap between the chunks when breaking up the document, as this helped to ensure that the context did not cut off any important information which could confuse the model. Another consideration I made was utilizing either query transformations or reranking for context lookup. However, in this specific instance, I was unable to improve performance by implementing HyDE transformations or RankGPTRerank. I believe it is possible that either the document set is too small for it to make a significant impact, or the method I am using to generate my test set does not stray far enough from the source content to merit more complex lookups. This is something which I would continue to explore and improve in future iterations.

Features

Below is an explanation of the features I developed, along with a demo video and list of potential future improvements

- 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
- Detailed evaluation metrics by question in csv file format
- 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:
The evaluation metrics will be generated for each question set, with a value between 0 and 1 inclusive. The specific results for each question are populated into the eval.csv file automatically.
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. The question set is loaded to the output.csv file automatically.
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(currently evaluates 30 questions in about 2 minutes)
- Fine-tuning embedding model, potentially using Sentence Transformers to make context selection and recall/robustness evaluation more accurate
- Continue exploring the merits of query transformation/reranking
- Allowing for different models to test
- “Smart chunking” which breaks documents into intuitive sections to avoid splitting similar sections