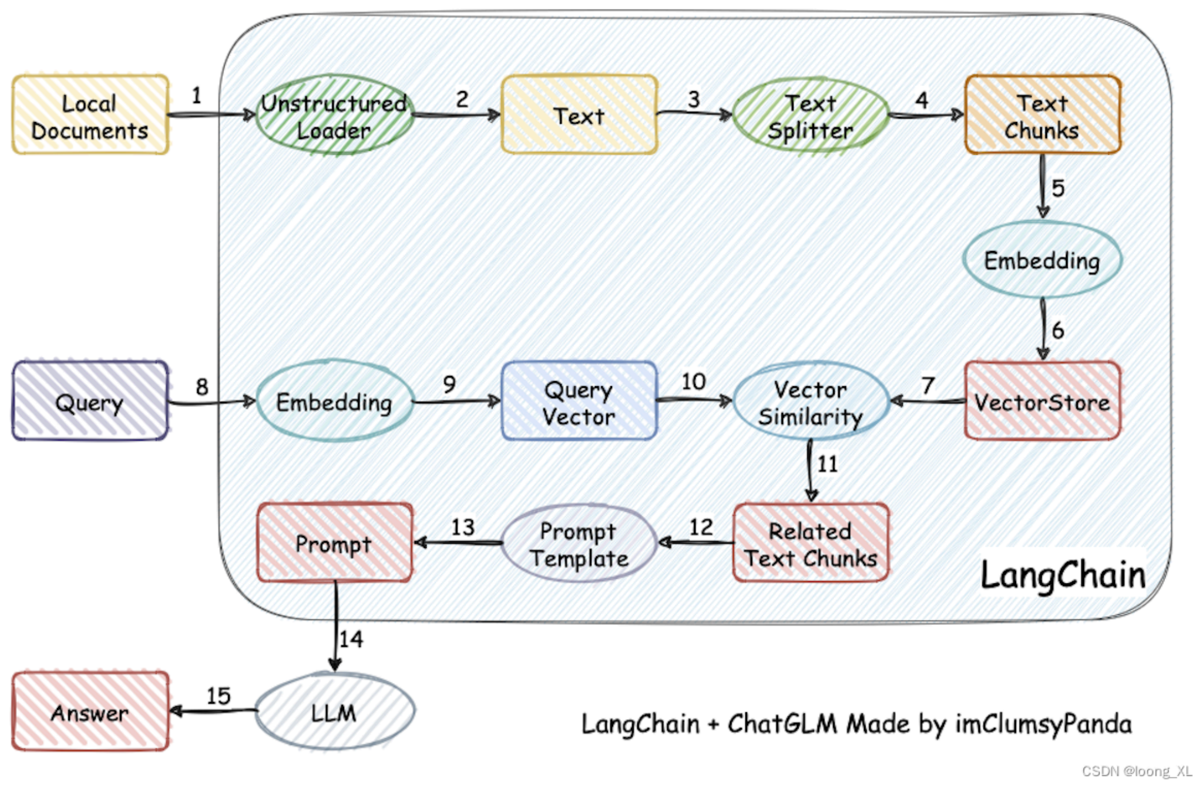
2737 words | 7 minutes

# Evaluate and optimize search parts

In the previous chapter, we explained how to evaluate and optimize Prompt Engineering for the generation part to improve the generation quality of the large model. However, the premise of generation is retrieval. Only when the retrieval part of our application can retrieve the correct answer document according to the user query, the generation result of the large model can be correct. Therefore, the retrieval precision and recall rate of the retrieval part actually affect the overall performance of the application to a greater extent. However, the optimization of the retrieval part is a more engineering and in-depth proposition. We often need to use many advanced, search-derived advanced techniques and explore more practical tools, or even write some tools by hand for optimization. Therefore, in this chapter, we only roughly discuss the ideas of evaluation and optimization of the retrieval part, without in-depth code practice. If readers feel unsatisfied after reading this part and want to learn more advanced techniques to further optimize their applications, welcome to read our upcoming tutorial part 2 "LLM Development Skills".

## 1. Evaluate the search results

First let's review the functionality of the entire RAG system.



For a query entered by a user, the system will convert it into a vector and match the most relevant text segment in the vector database. Then, according to our settings, 3 to 5 text segments will be selected and submitted to the big model together with the user's query. The big model will then answer the question raised in the user's query based on the retrieved text segments. In this entire system, we call the part where the vector database retrieves relevant text segments the retrieval part, and the part where the big model generates answers based on the retrieved text segments the generation part.

Therefore, the core function of the retrieval part is to find text paragraphs that exist in the knowledge base and can correctly answer the questions in the user's query. Therefore, we can define the most intuitive accuracy rate to evaluate the retrieval effect: for N given queries, we ensure that the correct answer corresponding to each query exists in the knowledge base. Assume that for each query, the system finds K text fragments. If the correct answer is in one of the K text fragments, then we consider the retrieval to be successful; if the correct answer is not in one of the K text fragments, our task retrieval fails. Then, the retrieval accuracy of the system can be simply calculated as:

$$a\ cc\ u\ r\ a\ cy = \frac{M}{N}$$

Where M is the number of successfully retrieved queries.

Through the above accuracy, we can measure the retrieval ability of the system. For queries that the system can successfully retrieve, we can further optimize the prompt to improve system performance. For queries that the system fails to retrieve, we must improve the retrieval system to optimize the retrieval effect. However, please note that when we calculate the accuracy defined above, we must ensure that the correct answer to each of our verification queries actually exists in the knowledge base; if the correct answer does not exist, then we should attribute the Bad Case to the knowledge base construction part, indicating that the breadth and processing accuracy of the knowledge base construction need to be improved.

Of course, this is just the simplest way of evaluation. In fact, this evaluation method has many shortcomings. For example:

- Some queries may require the combination of multiple pieces of knowledge to answer. How do we evaluate such queries?
- The order of the retrieved knowledge fragments will actually affect the generation of the large model. Should we take the order of the retrieved fragments into consideration?
- In addition to retrieving the correct knowledge fragments, our system should also try to avoid retrieving wrong and misleading knowledge fragments, otherwise the generation results of the large model are likely to be misled by the wrong fragments. Should we include the retrieved wrong fragments in the indicator calculation?

There are no standard answers to the above questions, and they require comprehensive consideration based on the actual business the project is targeting and the cost of the assessment.

In addition to evaluating the retrieval effect through the above methods, we can also model the retrieval part as a classic search task. Let's take a look at the classic search scenario. The task of the search scenario is to find and sort the relevant content from the given range of content (usually web pages) for the search query given by the user, and try to make the top ranked content meet the user's needs.

In fact, our retrieval tasks are very similar to search scenarios, and they are also aimed at user queries, but we emphasize recall rather than ranking, and the content we retrieve is not web pages but knowledge fragments. Therefore, we can similarly model our retrieval task as a search task, then we can introduce classic evaluation ideas (such as accuracy, recall, etc.) and optimization ideas (such as index building, re-ranking, etc.) in search algorithms to more fully evaluate and optimize our retrieval results. This part will not be elaborated in detail, and interested readers are welcome to conduct in-depth research and share.

## 2. Ideas for optimizing retrieval

The above statement is a few general ideas for evaluating the retrieval effect. When we make a reasonable evaluation of the system's retrieval effect and find the corresponding Bad Case, we can break down the Bad Case into multiple dimensions to optimize the retrieval part in a targeted manner. Note that although in the evaluation section above, we emphasized that the verification query for evaluating the retrieval effect must ensure that its correct answer exists in the knowledge base, here we assume that the knowledge base construction is also part of the retrieval part. Therefore, we also need to solve the Bad Case caused by incorrect knowledge base construction in this part. Here, we share some common Bad Case attributions and feasible optimization ideas.

## 1. The knowledge fragments are fragmented, resulting in the loss of answers

This problem usually manifests itself as follows: for a user query, we can be sure that the question must exist in the knowledge base, but we find that the retrieved knowledge fragments separate the correct answer, resulting in an inability to form a complete and reasonable answer. This problem is more common in queries that require longer answers.

The general optimization idea for this type of problem is to optimize the text segmentation method. The most primitive segmentation method we used in "C3 Building Knowledge Base" is segmentation based on specific characters and chunk size, but this type of segmentation method often fails to take into account the semantics of the text, and easily causes strongly related contexts of the same topic to be segmented into two chunks. For some knowledge documents with unified formats and clear organization, we can build more appropriate segmentation rules in a targeted manner; for documents with chaotic formats and no unified segmentation rules, we can consider incorporating a certain amount of manpower for segmentation. We can also consider training a model dedicated to text segmentation to achieve chunk segmentation based on semantics and topics.

## 2. Query questions require long context and summary answers

This problem also exists in the construction of knowledge bases. That is, some queries require the retrieval of a long context to give a general answer, that is, it is necessary to span multiple chunks to comprehensively answer the question. However, due to the limitations of the model context, it is often difficult for us to provide enough chunks.

The general optimization idea for this type of problem is to optimize the way the knowledge base is built. For documents that may require such answers, we can add a step to summarize long documents using LLM, or pre-set questions for LLM to answer, so as to pre-fill the possible answers to such questions into the knowledge base as a separate chunk, which can solve this problem to a certain extent.

## 3. Misleading keywords

This problem usually manifests itself as: for a user query, the knowledge fragments retrieved by the system have many keywords that are strongly related to the query, but the knowledge fragments themselves are not answers to the query. This situation usually occurs when there are multiple keywords in the query, and the matching effect of the secondary keywords affects the primary keywords.

The general optimization idea for this type of problem is to rewrite the user query, which is also a common idea used in many large model applications. That is, for the user input query, we first use LLM to rewrite the user query into a reasonable form, removing the impact of secondary keywords and possible typos and omissions. The specific form of rewriting depends on the specific business. You can ask LLM to refine the query to form a Json object, or you can ask LLM to expand the query.

## 4. Unreasonable matching relationship

This problem is quite common, that is, the strongly related text segment matched does not contain the answer text. The core problem of this problem is that the vector model we use does not match our initial assumption. When explaining the framework of RAG, we mentioned that RAG works on a core assumption, that is, we assume that the strongly related text segment we match is the answer text segment corresponding to the question. However, many vector models actually construct "pairing" semantic similarity rather than "causal" semantic similarity. For example, for the query "How is the weather today", it is considered that "I want to know the weather today" is more relevant than "The weather is good".

The general optimization ideas for this type of problem are to optimize the vector model or build an inverted index. We can choose a vector model with better results, or collect some data and fine-tune a vector model that is more suitable for our business. We can also consider building an inverted index, that is, for each knowledge fragment in the knowledge base, build an index that can represent the content of the fragment but has a more accurate relative relevance to the query. When searching, match the relevance of the index and the query instead of the full text, thereby improving the accuracy of the matching relationship.

There are many ways to optimize the search part. In fact, the optimization of the search part is often the core engineering part of RAG application development. Due to space limitations, we will not go into more techniques and methods here. Interested readers are welcome to read the second part "LLM Development Techniques" which will be released soon.

‹ Previous chapter

# 2. Evaluate and optimize the generation part