RAG(检索增强生成)基础调研

研究背景

使用LLM进行简单的微调后可以进行一些简单的具体领域任务，例如情绪分析、实体分析等等，这些任务不需要额外的背景知识。要完成更**复杂和知识密集型**的任务，可以基于语言模型构建一个系统，访问外部知识源来做到。这样的实现与事实更加一性，生成的答案更可靠，还有助于缓解“幻觉”问题。

Meta AI 的研究人员引入了一种叫做检索增强生成（Retrieval Augmented Generation，RAG）(opens in a new tab)的方法来完成这类知识密集型的任务。RAG 把一个信息检索组件和文本生成模型结合在一起。

优势：RAG的内部知识可以很容易地被改变，甚至即时补充，使研究人员和工程师能够控制RAG知道和不知道的东西，而不会浪费时间或计算能力重新训练整个模型。（灵活性）

工作原理简介

知识来源：seq2seq生成模型参数中的知识（参数记忆）+语料库中的知识（非参数记忆）。——“开卷”与“闭卷”

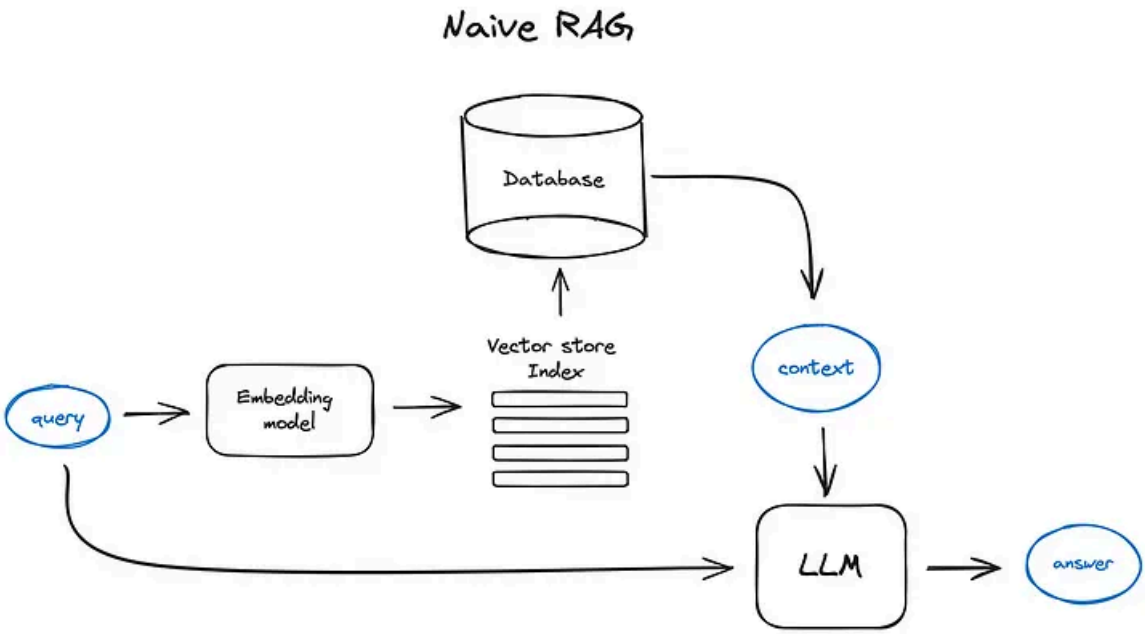
输入：RAG 结合了搜索技术和大语言模型，向模型提出问题，并以搜索算法找到的信息作为背景上下文，这些查询和检索到的上下文信息都会被整合进发送给大语言模型的Prompt中。

评估指标

RAG评估资料大全 (techdiylife.github.io)

工作流程介绍

简单RAG



输入为query，分为两路进行：

①将query编码为向量之后，在索引index中进行寻找，找到数据库中最相关的前m个结果，作为context。

②将query与context整合起来作为LLM的Prompt输入，输出得到结果。

简单的Prompt可以是这样的：

```
def question_answering(context, query):
    prompt = f"""
                Give the answer to the user query delimited by triple backticks
```{query}```\
                using the information given in context delimited by triple
backticks ```{context}```.\
                If there is no relevant information in the provided context, try
to answer yourself,
                but tell user that you did not have any relevant context to base
your answer on.
                Be concise and output the answer of size less than 80 tokens.
                """

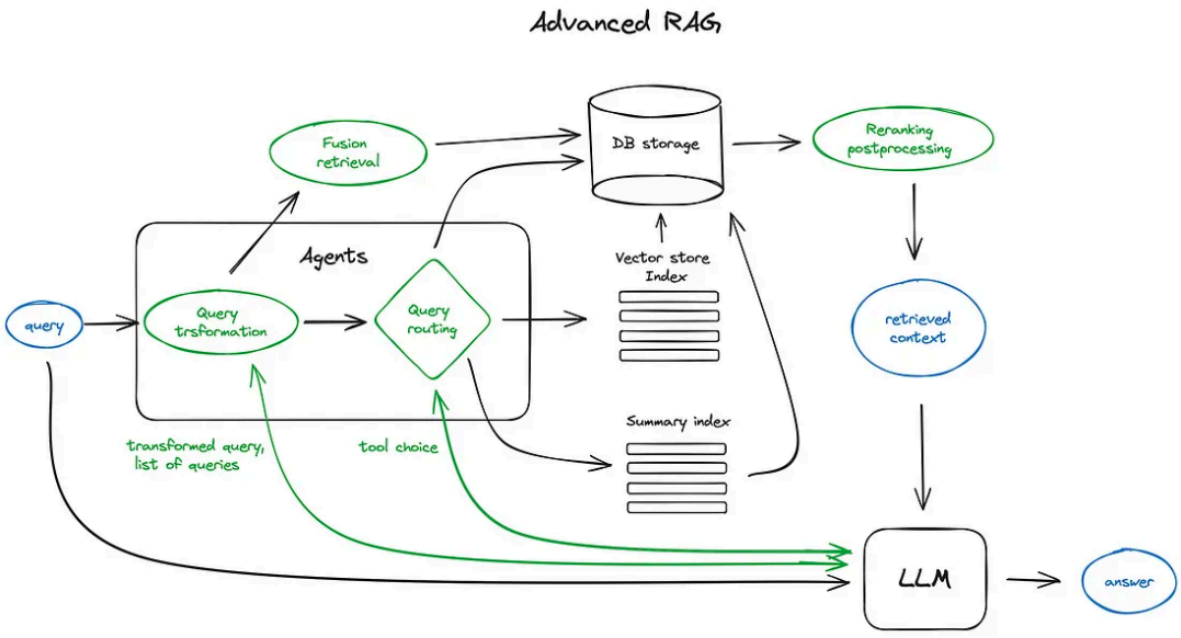
    response = get_completion(instruction, prompt, model="gpt-3.5-turbo")
    answer = response.choices[0].message["content"]
    return answer
```

注：在改进RAG管道的方法中，优化提示词（Prompt Engineering）是最具成本效益的尝试。

[提示工程指南 | Prompt Engineering Guide (promptingguide.ai)]
```

高级RAG

有更多可用的工具。



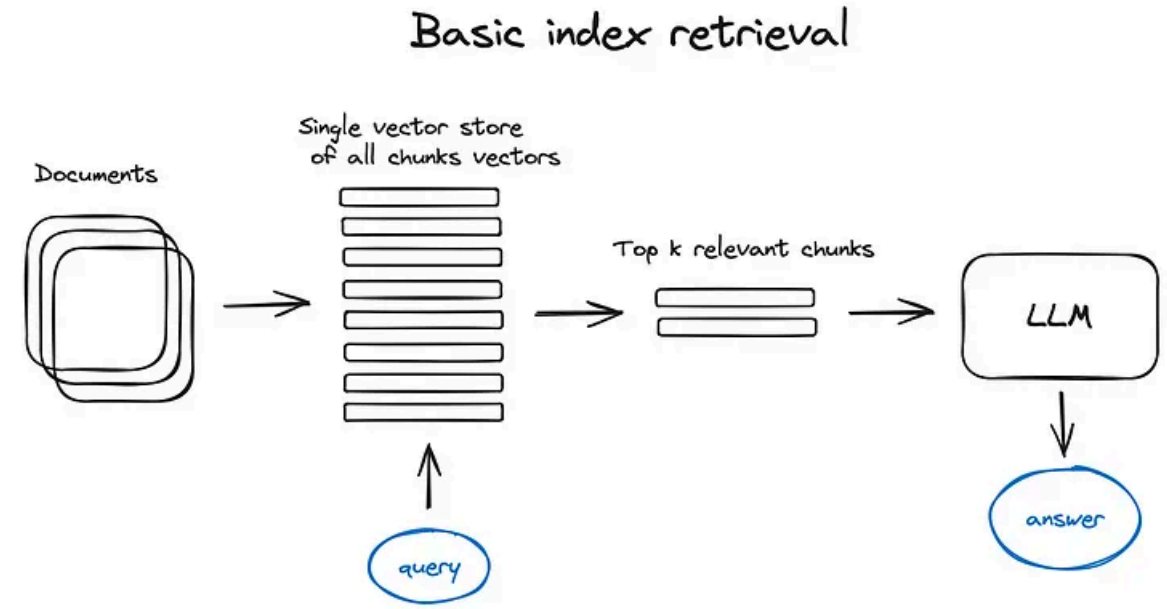
分块与向量化

分块：Transformer 模型的输入序列长度是固定的，即使输入上下文窗口很大，用一个句子或几个句子的向量来代表它们的语义含义，通常比对几页文本进行平均向量化更为有效。因此，需要对数据进行分块处理，将文档切分成合适大小的段落，同时保持其原有意义不变。

向量化：将这些分块的段落Embedding成向量，即Encoder。

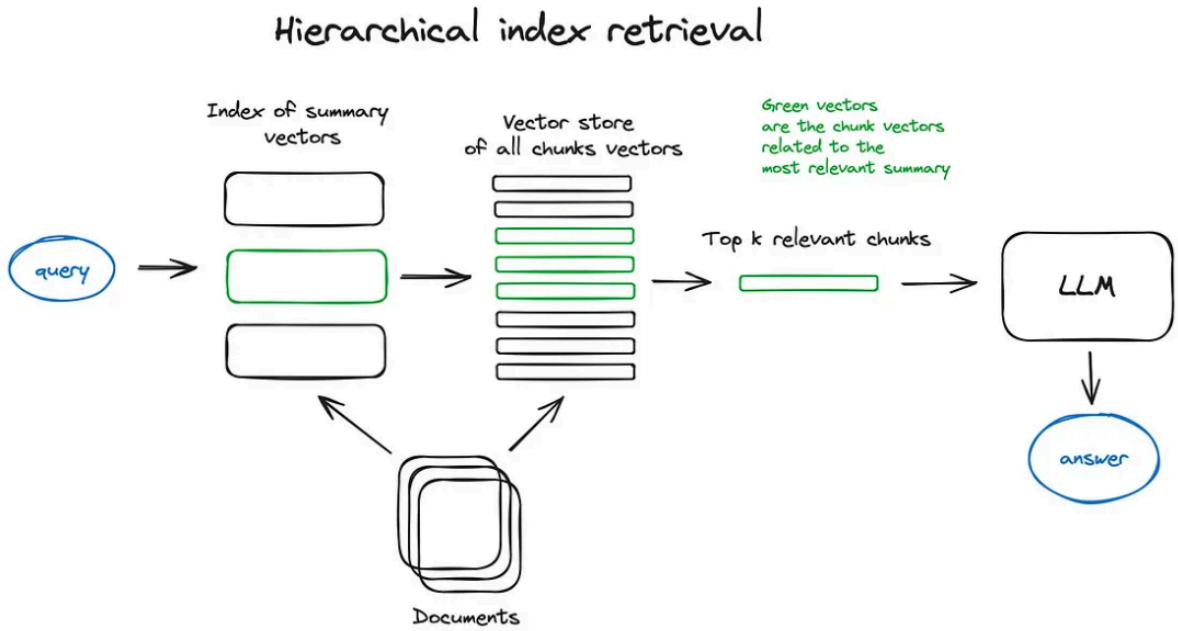
搜索索引（Important）

向量存储索引



比较基础的搜索索引方式，将query向量与数据库中各种数据的向量比对找出前K个最相似的。可以有多种搜索方式。

层次索引



引入了一个分步操作，先检索摘要索引，再对应检索相应的文档。对于处理大型数据库更高效。

假设性问题和HyDE

假设性问题：让LLM为每个分好的块生成一个假设性问题，将其以向量形式Embedding，代替Index中的向量。检索后将原始query文本作为上下文发给LLM获取答案。在这个方法中query和假设性问题件语义相似性更高，故搜索质量提高。

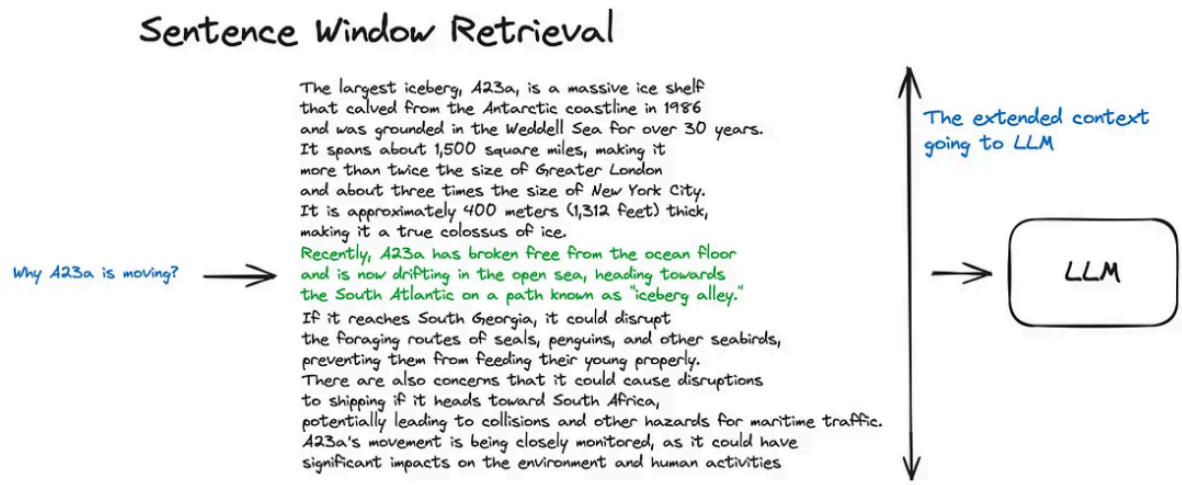
HyDE：与假设性问题逻辑相反，让LLM根据query生成一个假设性回答，然后将该回答Embedding的向量与query向量一起提高搜索质量。

上下文增强

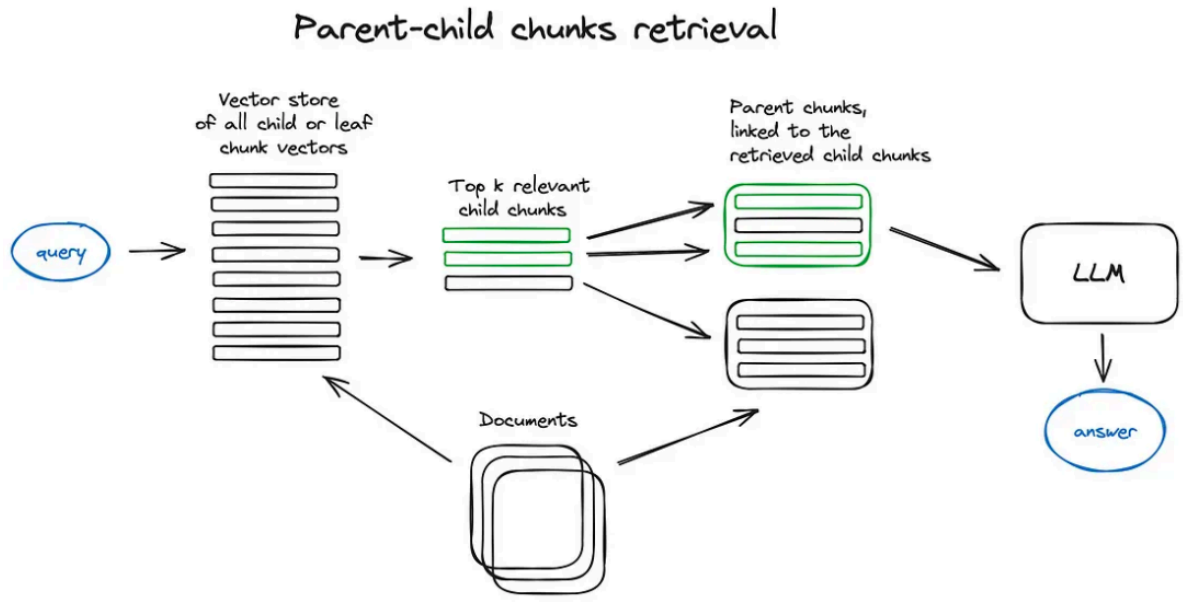
在数据库检索相关内容的过程中，检索较小的块以提高搜索质量，但同时增加周围的上下文以供LLM进行推理分析。

有两种方法：一是通过在检索到的较小块周围添加句子来扩展上下文，二是递归地将文档分割成多个包含较小子块的大型父块。

①句子窗口检索：文档中的每个句子都被单独嵌入向量，这样做可以提高查询到上下文的余弦距离搜索的准确度。在检索到的关键句子前后各扩展k个句子，然后将这个扩展的上下文发送给LLM。



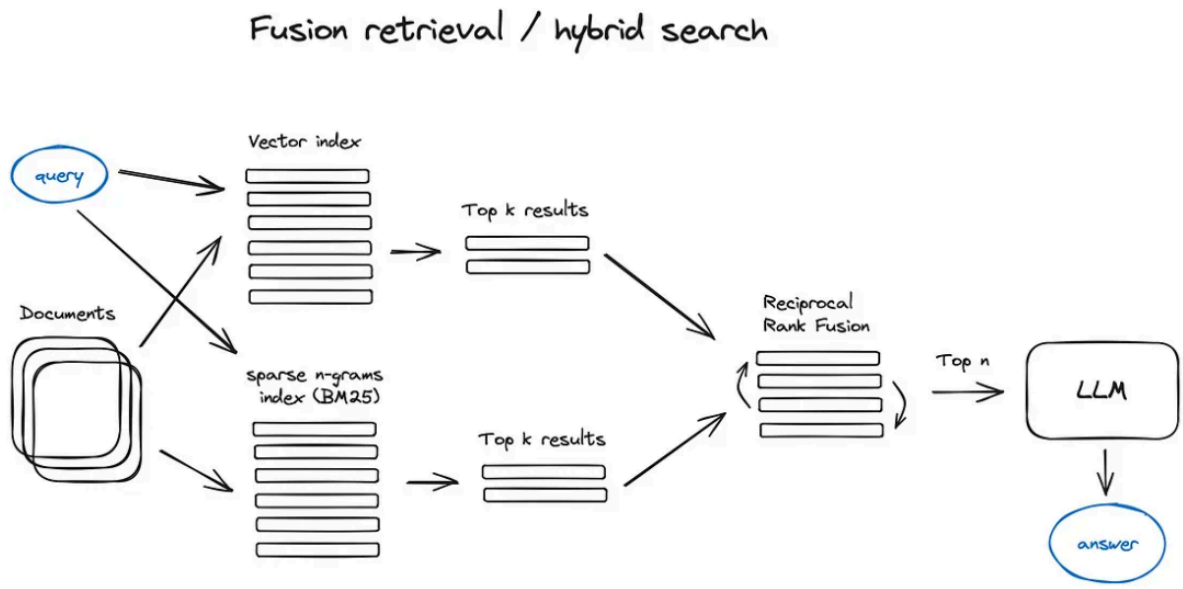
②自动合并检索器：文档被分割成较小的子块，这些子块又与更大的父块相对应。首先搜索更精细的信息片段，然后在将这些上下文信息提供给LLM进行推理之前，先扩展到父块的上下文窗口。



在检索过程中，首先获取较小的数据块。如果在前k个检索到的块中，有超过n个块指向同一个父节点（即更大的块），那么我们就用这个父节点来替换原先提供给大语言模型（LLM）的上下文内容。这个过程类似于自动将几个小块合并成一个更大的父块，因此得名。

融合检索与混合检索

结合传统的关键词搜索与现代的语义向量搜素，产生综合的检测结果。



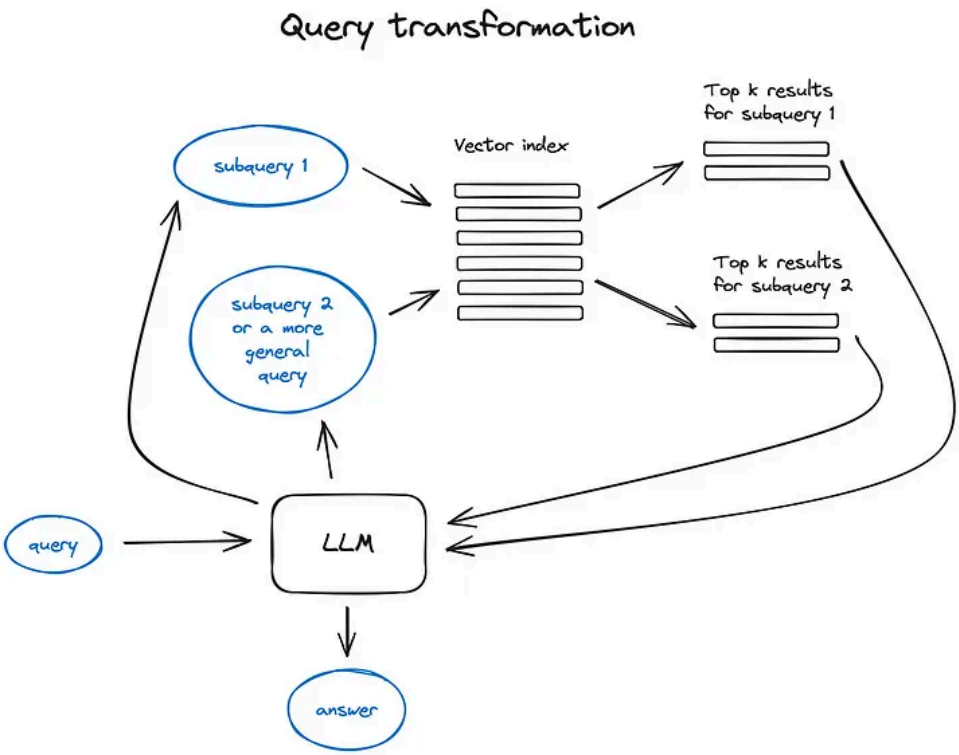
重新排名与过滤

根据上述搜索索引方法得到了检索的结果，还需要通过过滤、重新排名以及一些转换操作来进一步优化这些结果。LlamaIndex提供了多种后处理器，可以基于相似度评分、关键词、元数据等过滤结果，或者使用其他模型进行重新排名，如LLM、句子-转换器交叉编码器[18]、Cohere重新排名端点[19]，或者基于元数据如日期的新近性来进行——几乎涵盖了你能想到的所有情况。

更高级的RAG技术

查询转换(query transform)

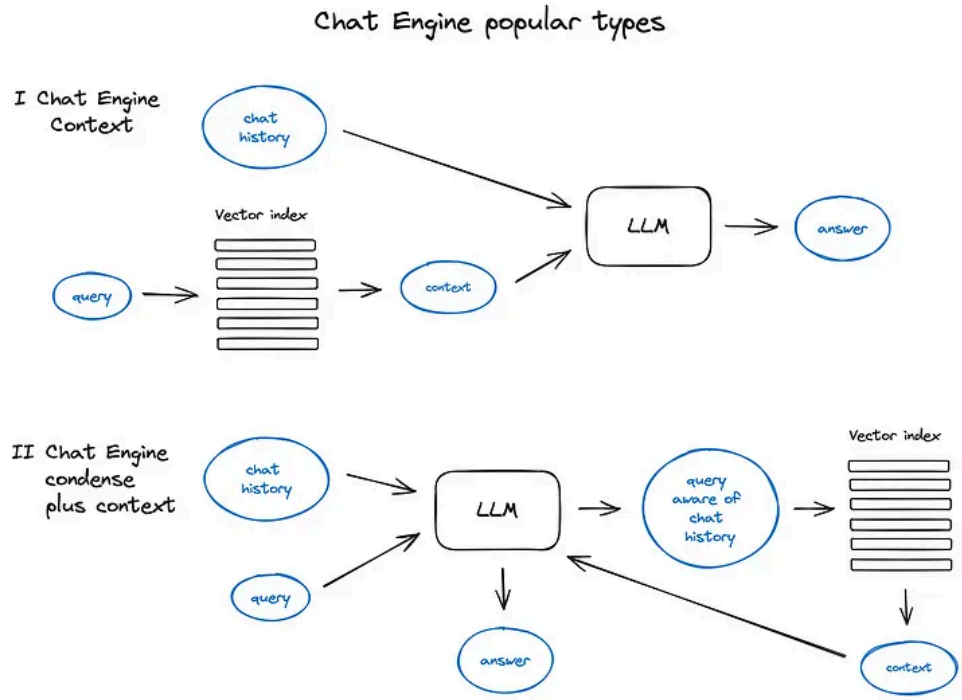
属于Prompt Engineering，使用LLM对用户的query进行优化，进而提高检索质量。有多种实现方法。例如可以将query分步（思维链方法），也可分为多个子查询进行并行查询，随后组合为一个输入。



对话引擎

在构建一个能够对单个搜索查询反复有效工作的高效 RAG 系统中，下一个关键进展是聊天逻辑的开发。这种逻辑的开发是为了支持用户提出后续问题、处理上下文中的指代或与先前对话上下文相关的任意命令，为了解决这一问题，研究者们采用了一种**查询压缩技术**，这种技术在处理用户的查询时同时考虑了聊天的上下文。

- 一种受欢迎且相对简单的方法是使用 ContextChatEngine。这种引擎首先检索与用户查询相关的上下文，然后将其连同聊天历史从内存缓冲区一起发送给 LLM，确保 LLM 在生成下一条回答时能够理解之前的上下文。
- 更为复杂的一个例子是 CondensePlusContextMode。在这种模式下，每次互动中的聊天历史和最后一条消息会被压缩成一个新的查询。然后，这个查询被发送到索引系统，检索到的上下文连同原始用户消息一起回传。



查询路由(query routing)

查询路由是一种基于LLM的决策步骤，用于确定针对用户的查询接下来应采取的行动。通常的选项包括概括回答、针对某个数据索引执行搜索，或尝试多条不同的途径，然后将它们的结果综合成一个答案。

查询路由器还用于选择合适的索引或更广泛的数据存储位置来处理用户查询。这可能涉及多个数据来源，比如传统的向量存储、图数据库或关系型数据库，或者是一个索引层级结构。在多文档存储情况下，一个常见的设置是一个概要索引和另一个文档块向量的索引。

响应合成器

这是 RAG 架构中的最后一步 —— 基于我们精心收集的所有上下文和用户的初步查询来生成答案。

最简单的方法是直接将所有相关度较高的上下文和查询串联起来，一次性输入到LLM中。

然而，还有一些更复杂的方法，这些方法涉及多次使用LLM来细化检索到的上下文，并据此生成更加准确的答案。响应合成的几种主要方法包括：

1. 将检索到的上下文分块后逐次发送给大语言模型（LLM），以此迭代地精炼答案。
2. 总结检索到的上下文，使其适应输入提示。
3. 基于不同上下文块生成多个答案，然后将这些答案连接或总结起来。

参考资料

最全的检索增强生成（RAG）技术概览-CSDN博客

检索增强生成 (RAG) | Prompt Engineering Guide (promptingguide.ai)

关于开源项目：一些背景知识

在基于大语言模型的管道和应用领域，两个最著名的开源库分别是LangChain和LlamaIndex

Introduction | 🦜🔗 LangChain

Langchain简介

Langchain是一个开源框架，允许开发者将LLM与外部计算和数据结合起来（很适合用来做RAG）。

主要思想

①Components：提供LLM、搜索相关信息、Prompt模板等组件。

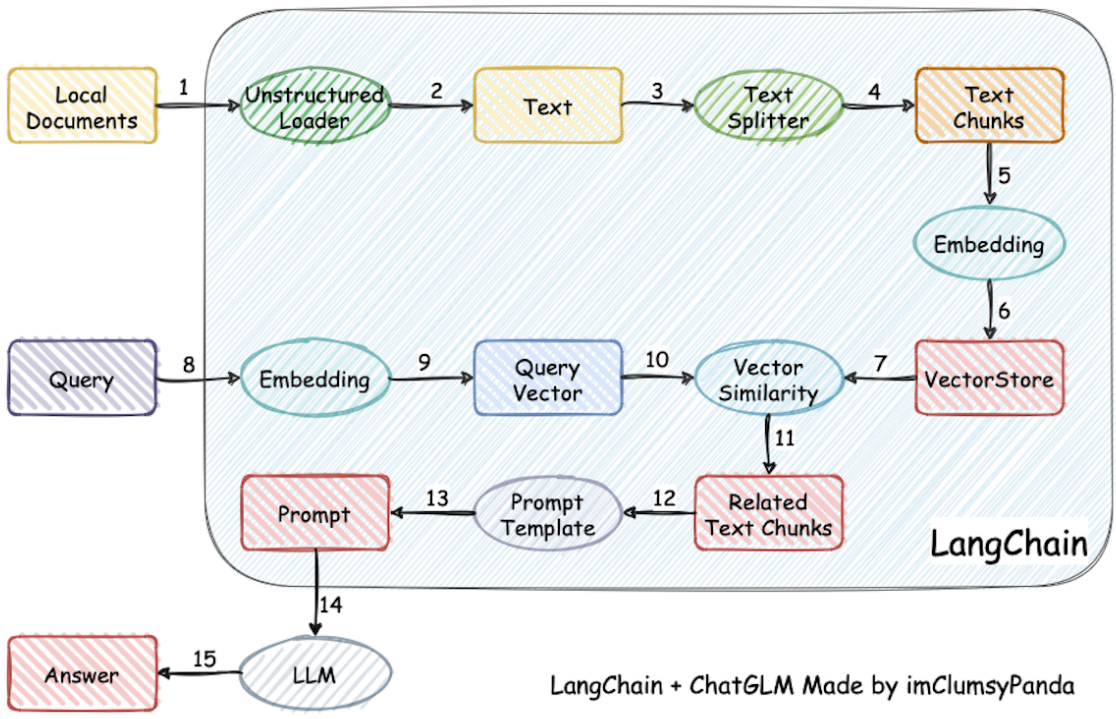
②Chains：允许开发者将多个组件结合在一起完成一个特定任务，建立完整的LLM应用程序。

③Agents：允许LLM与外部API互动

https://www.bilibili.com/video/BV14o4y1K7y3/?spm_id_from=333.337.search-card.all.click

Langchain-Chatchat简介

工作流程



过程包括加载文件 -> 读取文本 -> 文本分割 -> 文本向量化 -> 问句向量化 -> 在文本向量中匹配出与问句向量最相似的 `top k` 个 -> 匹配出的文本作为上下文和问题一起添加到 `prompt` 中 -> 提交给 `LLM` 生成回答。

开源项目：逐级递进

- langchain-ai/langchain: 🦜🔗 Build context-aware reasoning applications (github.com)
- chatchat-space/Langchain-Chatchat: Langchain-Chatchat（原Langchain-ChatGLM）基于 Langchain 与 ChatGLM 等语言模型的本地知识库问答 | Langchain-Chatchat (formerly

langchain-ChatGLM), local knowledge based LLM (like ChatGLM) QA app with langchain (github.com)

- X-D-Lab/LangChain-ChatGLM-Webui: 基于LangChain和ChatGLM-6B等系列LLM的针对本地知识库的自动问答 (github.com)

开源项目

- infiniflow/ragflow：infiniflow/ragflow: RAGFlow is an open-source RAG (Retrieval-Augmented Generation) engine based on deep document understanding. (github.com)
- explodinggradients/ragas：explodinggradients/ragas: Evaluation framework for your Retrieval Augmented Generation (RAG) pipelines (github.com)
- run-llama/rags：run-llama/rags: Build ChatGPT over your data, all with natural language (github.com)
- langgenius/dify：langgenius/dify: Dify is an open-source LLM app development platform. Dify's intuitive interface combines AI workflow, RAG pipeline, agent capabilities, model management, observability features and more, letting you quickly go from prototype to production. (github.com)
- langflow-ai/langflow：langflow-ai/langflow: Langflow is a visual framework for building multi-agent and RAG applications. It's open-source, Python-powered, fully customizable, model and vector store agnostic. (github.com)
- vanna-ai/vanna：vanna-ai/vanna: Chat with your SQL database. Accurate Text-to-SQL Generation via LLMs using RAG. (github.com)
- weaviate/Verba：weaviate/Verba: Retrieval Augmented Generation (RAG) chatbot powered by Weaviate (github.com)
- llmware-ai/llmware(小模型)：llmware-ai/llmware: Unified framework for building enterprise RAG pipelines with small, specialized models (github.com)
- pathwaycom/llm-app：pathwaycom/llm-app: LLM App templates for RAG, knowledge mining, and stream analytics. Ready to run with Docker, in sync with your data sources. (github.com)
- truefoundry/cognita：truefoundry/cognita: RAG (Retrieval Augmented Generation) Framework for building modular, open source applications for production by TrueFoundry (github.com)

上面的大部分开源框架/应用都是自带前端的，部分是基于提供者云端的服务，部分可以自己通过Docker部署。