RAG - Retrieval Augmented Generation

RAG 早在 GPT 等 LLMs 出来之前就有了相关的研究。例如 FaceBook 在 2020 年的研究尝试，将模型知识分为两类 **参数记忆**（内部信息） 和 **非参记忆**（外部信息），基于这些知识来产出内容

首创：Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

RAG方式有其灵活性,即使模型**参数不变**也能适应快速变化的**外部信息**

外部数据可以是文档、数据库、网页信息、个人知识库、日志、图文视频也可以是从API获取的数据等等

这些外部信息的处理方式通常是**切块**然后做**向量化**表征然后在向量数据库中基于用户输入做**检索**

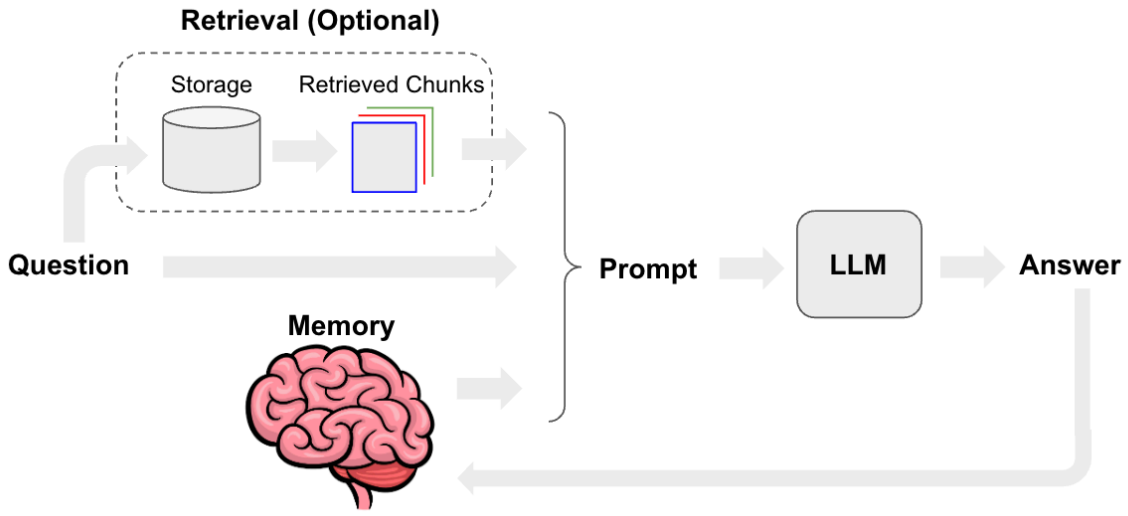
PS：向量数据库只是一种候选，还可以有更多选择，可以泛化为一个信息检索系统

大模型使用外部数据可以提供更准确和及时的回复，同时减少模型猜测输出的可能性即**幻觉**，增强用户的信任度

PS：有研究者指出幻觉是让大模型产出创意的基础，现在大家都在思考如何消除幻觉，其实另一个方向也可做利用

目前看来，RAG 是当前大语言模型场景下最有效的整合最新可验证信息的方式，且无需重新训练和更新以保持较小成本

RAG流程



如图，先把用户的输入文本（包括之前的Memory）转化为Embedding Vector，进向量数据库中查询出来一些文本，将这些文本和用户的输入文本拼在一起作为Prompt给LLM推断。

作为对比，无RAG的情况是用户的输入文本（包括之前的Memory）直接作为Prompt输给LLM。

这里的LLM就是上文所述的参数记忆，之前的Memory和向量数据库中查询出来的文本就是非参记忆。

Retrievers有哪些类型

摘自LangChain官方文档

最基本的Retrievers

Retrievers	存储过程	搜索过程	最佳实践
Vectorstore：最基础的Retrieval方式	将一整段文本Embedding为固定长度的Vector	执行向量搜索返回对应文本	刚入行时快速体验Retrieval
Time-Weighted Vectorstore：带时间戳的Vectorstore	同上，但是每段文本都有个时间戳	同上，但是返回结果除考虑向量相似性还考虑时间，比如时间最近的文本得分高	搜新闻，搜论文等
ParentDocument：Vectorstore + Document Store	将Document切分为chunks，每个chunk分别进行Embedding	执行向量搜索得到chunks，但返回这个chunk所在的完整Document	Document中有很多独立但相互关联的信息，所以这些信息在检索时需要分别对待，但作为一个整体返回

存储或搜索过程需要使用LLM的Retrievers

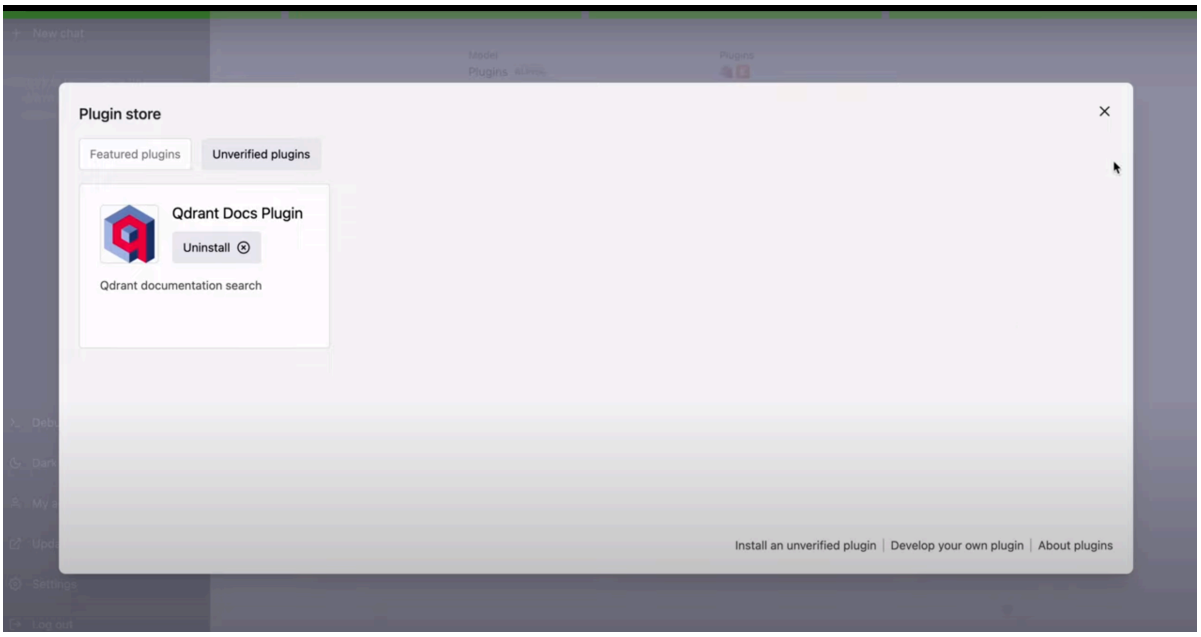
名称	存储过程	搜索过程	最佳实践
Multi Vector：ParentDocument+LLM辅助Embedding	使用多种不同的手段生成Embedding，比如用LLM为Document生成Summary或一些可能会提的问题拿去Embedding	执行向量搜索，返回这些结果所对应的Document	如果有办法提取比Document中的文本本身更有利于搜索的信息，那自然是Multi Vector更好
Multi-Query Retriever：ParentDocument+LLM辅助搜索	不限	先使用LLM将用户的提问转化为多段用于语义查询的文本，再执行搜索	有时用户的提问比较复杂，需要综合多种不同的信息才能准确回复
Self Query：带元数据的Vectorstore	除Embedding之外，还存储Document的元数据	先使用LLM将用户的提问转化为一段用于语义查询的文本和一些元数据匹配规则，而后先用元数据匹配规则筛选Document，再在剩下的内容中执行向量搜索	很多情况下，用户的提问是关于Document的元数据而非文本本身，此时元数据匹配比文本相似性更有利于搜索

强化Retrievers的输出结果

名称	强化过程	最佳实践
Contextual Compression	对Retrieval结果进行进一步提取，比如交给LLM进行Summary之后再输出	返回整个Document太多了？用它
Ensemble	把多个Retrievers的返回内容拼一起	有好几个棒棒的Retrievers？用它
Long-Context Reorder	把Retrievers的返回内容按向量查询时的相似性排序	研究表明一些长序列模型再推断时对于长文本开头和结尾的部分关注度更高，而对中间部分的关注度较低，所以Retrieval结果输入模型之前排序可以提升性能

典型应用：ChatGPT Retriever Plugin

ChatGPT Retriever Plugin本质上是一个基于FastAPI的HTTP Server程序，其API主要有三：Documents的upserting上传, querying查询, 和 deleting删除。
当这个HTTP Server程序运行起来并对外暴露端口后，在ChatGPT的网站中填写你的Server地址和验证方式，就可以让ChatGPT使用你的Plugin了。



目前(commit:b808c100d8baebe832e3fe433358d12e93bba48f)其使用的存储方法同ParentDocument，其将长段的文本内容切分为chunks，而后使用OpenAI的Embedding接口(默认使用`text-embedding-3-large`)将长段的文本内容转化为Embedding Vector存入指定的数据库中(比如向量数据库Qdrant)；
由于ChatGPT目前未开源，所以其搜索过程未知，推测至少应该有Multi-Query Retriever(ChatGPT可以同时使用多种Plugin)和Time-Weighted Vectorstore+Self Query(README里写可以用元数据过滤refine查询结果)。