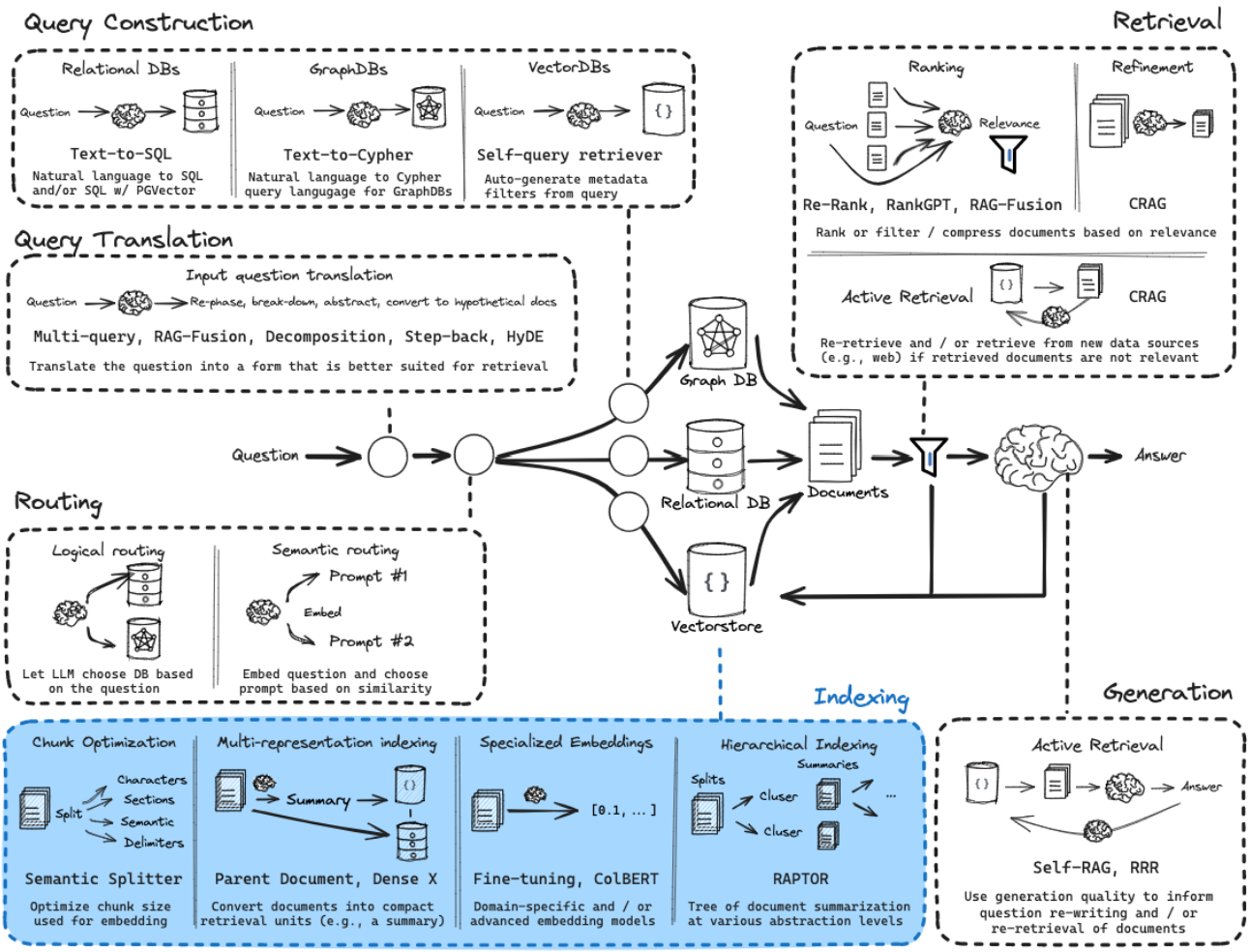
Rag for Scratch——langchain

Indexing

indexing是rag pipeline的第四阶段，就是将文档压缩为数值表示，以建立索引，便于后续的检索和生成。

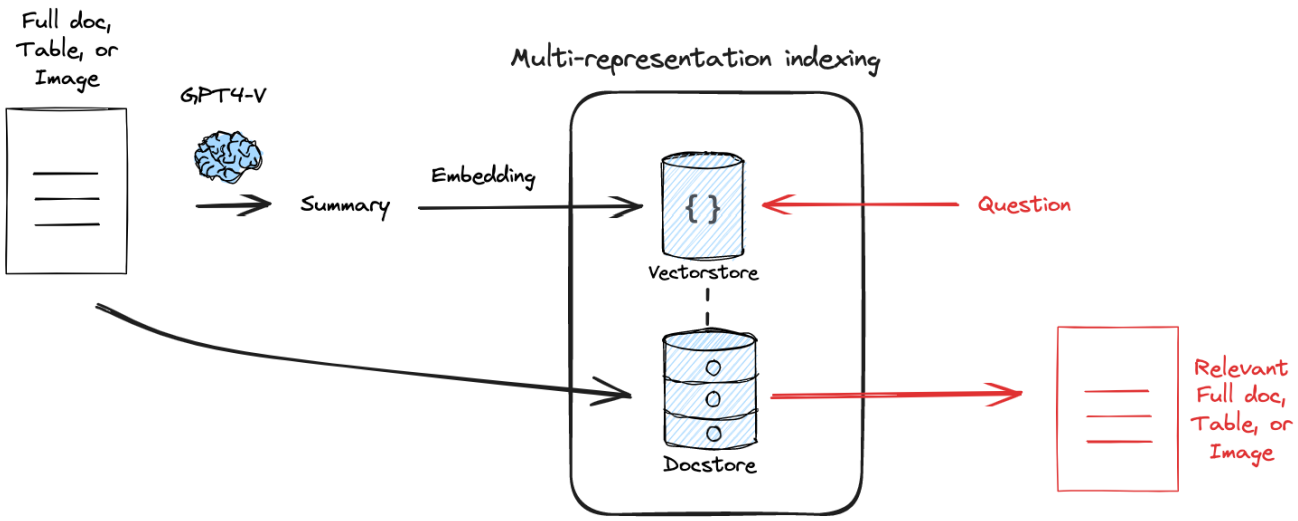


1 Chunking

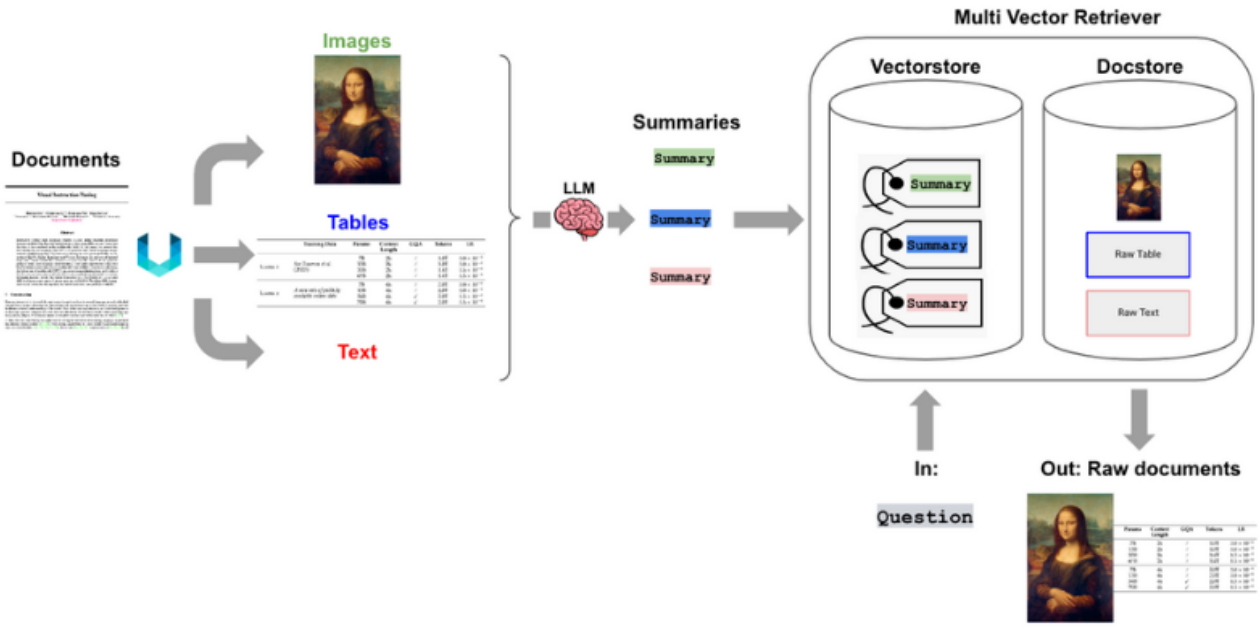
五个等级：

- Character：按固定字符数分割。
- Recursive：通过递归遍历文档，根据不同的分隔符进行分块。
- Document：使用特定的分隔符或规则进行分割，如markdown中的标题符号、Python代码中的类和函数等。
- Semantic：通过计算句子间embedding距离，把具有相似主题或内容的句子分为一块。
- Agentic：最高级别的chunking方法，通过LLM做决策，将文本分块为独立的命题。

2 Multi-representation indexing



- Multi-Vector Retriever使用一个简单而强大的RAG理念：将用于答案生成的文档与用于检索的参考文档解耦。
- 例如，可以创建一个优化的文档摘要，用于基于向量相似性搜索，但仍然将完整文档传递给LLM以确保在答案生成期间不丢失上下文。



论文：Dense X Retrieval: What Retrieval Granularity Should We Use?

使用两个检索器：（1）multi-vector retriever嵌入摘要，但向LLM返回完整文档。（2）parent-doc retriever嵌入块，但向LLM返回完整文档。

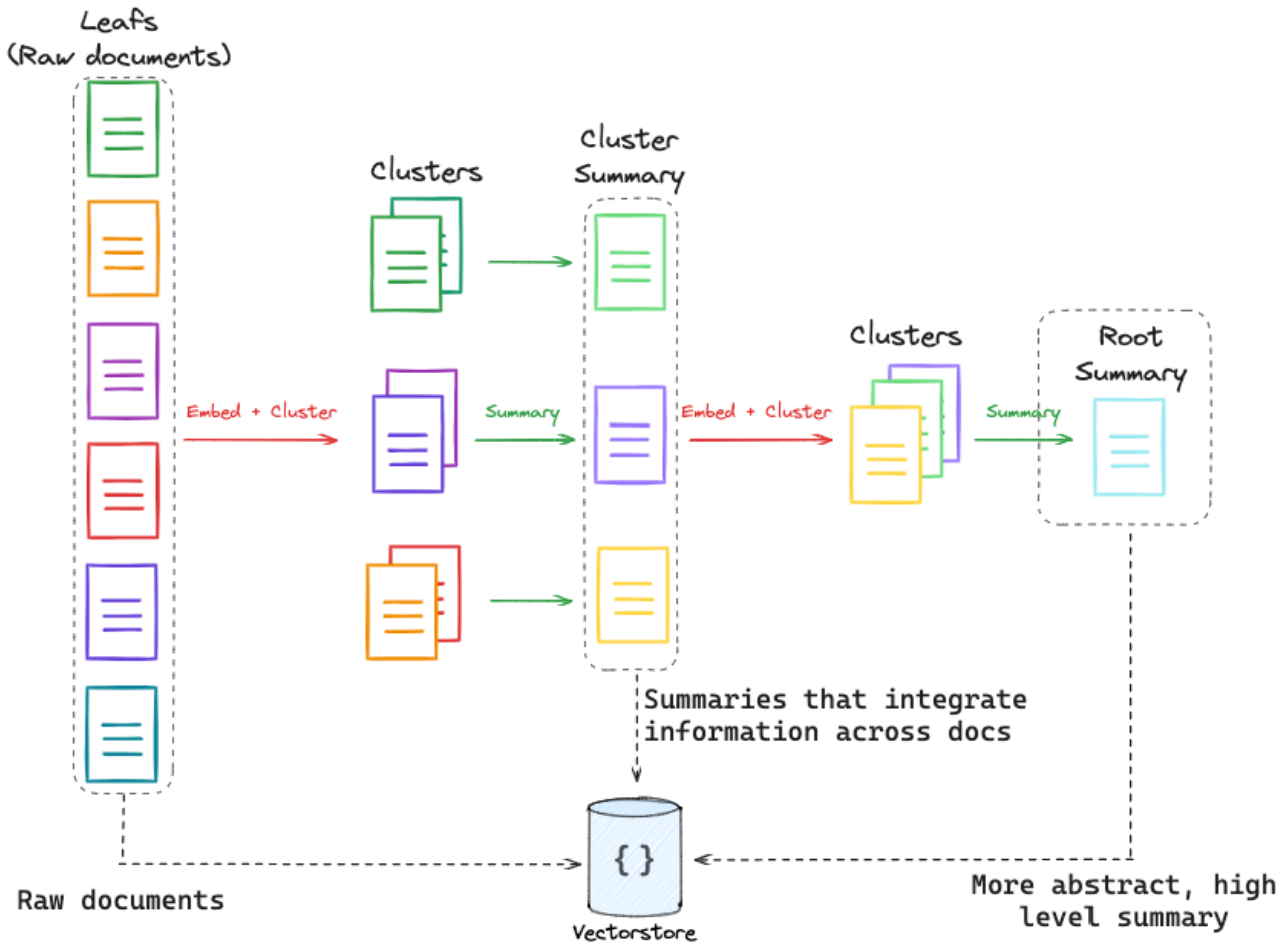
基本思想：使用更小/简洁的表示（摘要或块）进行检索，但将它们链接到完整文档/上下文以进行生成。

这种方法非常通用，可以应用于表格或图像：在这两种情况下，索引摘要但返回原始表格或图像进行推理。这解决了直接嵌入表格或图像（多模态嵌入）的挑战，使用摘要作为基于文本的相似性搜索的表示。

3 RAPTOR

问题：“低层次"的query，指那些只需要单一文档就可以回答的query；"高层次"的问题，指那些需要多个文档结合才会回答的query。此时，典型的kNN检索可能并不适用，因为kNN检索只能检索有限数量的文档块。

想法：通过创建捕捉更高层次概念的文档摘要来解决这个问题。嵌入并聚类文档，然后总结每个聚类。以**递归**的方式这样做，产生一个包含越来越高层次概念的摘要树。摘要和起始文档一起被索引，覆盖用户query的范围。



4 ColBERT

DPR（dense passage retrieval）存在的问题：

1、现成的模型在训练过程中会对一些不常见的词的不熟悉，比如人名、术语

2、DPR对chunking strategy敏感，如果一个相关的passage被很多不相关的冗余信息包围的话，很容易检索不到。

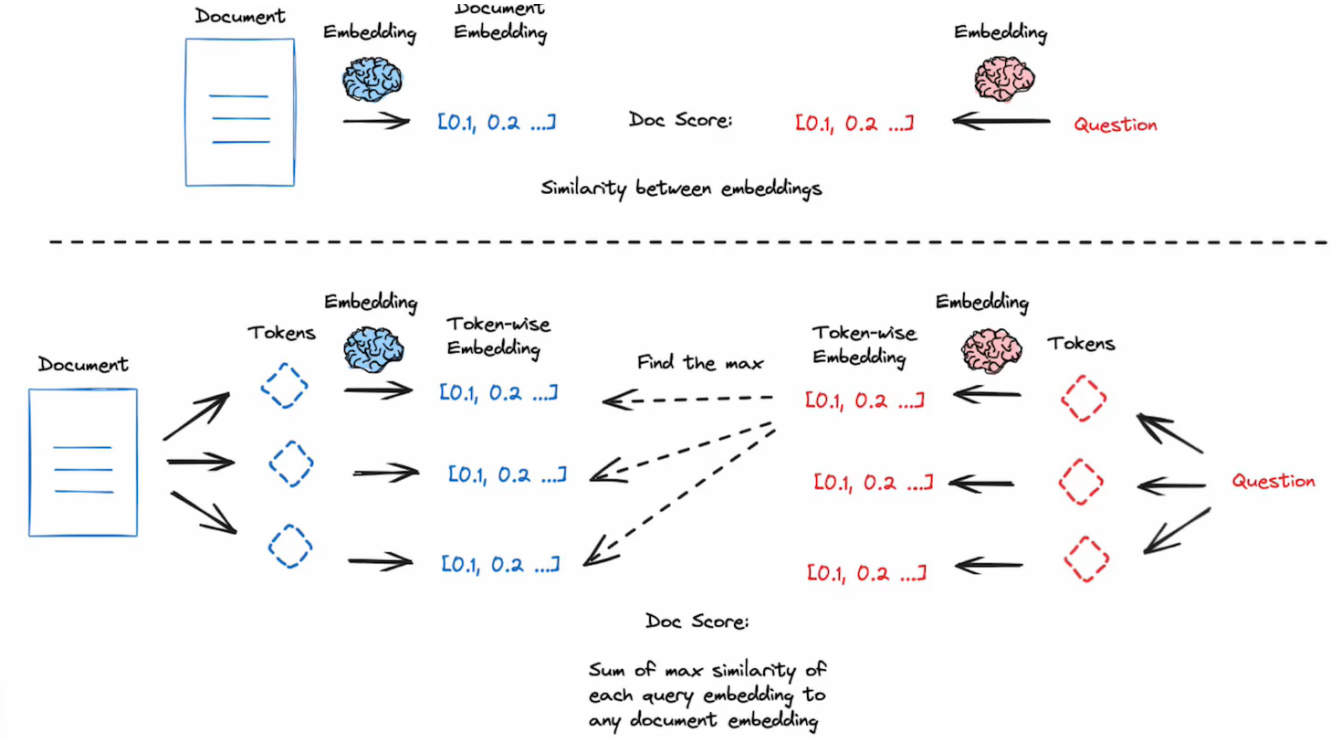
想法：ColBERT是一种使用BERT语言模型对段落相关性进行评分的新方法，通过更细粒度的嵌入来解决这个问题。它与single-vector-based DPR不同，它不是将段落转换成单个“embedding”向量，而是为段落中的每个token生成受上下文影响的向量。同样，ColBERT也为query中的每个token生成向量。

然后，每个文档的得分是query嵌入与文档嵌入中任何一个最大相似度的总和：

```
def maxsim(qv, document_embeddings):
    return max(qv @ dv for dv in document_embeddings)

def score(query_embeddings, document_embeddings):
    return sum(maxsim(qv, document_embeddings) for qv in query_embeddings)
```

论文：ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT



如何限制需要计算分数的文档集合的大小?因为计算每个文档的分数可能代价较高。

使用像DataStax Astra DB这样的向量数据库可以很容易地解决这个问题。两个步骤：

1、摄取（Ingestion） 使用点积作为相似度函数

2、检索（Retrieval） 执行ANN搜索