RAG 基本概念

Updated 1424 GMT+8 Apr 8, 2025

2025 spring, Complied by Yinan Peng

1 介绍

- 定义：通过检索外部知识库来增强生成式模型的能力。
- RAG的两大核心模块：
  - **检索模块**：从知识库中获取用户输入相关的信息。
    - 这里可检索的知识库是一个广义的概念。
      - 本地资源：构建的各类静态的数据库等。
      - 在线资源：网页内容、第三方检索 API（例如 Arxiv 提供的检索 API）等。
    - 检索模块目标：从知识库中找出有助于满足用户需求的信息、有助于回答用户问题的信息。
  - **生成模块**：基于检索结果生成回答。
- RAG的优势
  - 丰富知识
    - 更细节的专业知识，比如论文中某一段文字。
    - 实时的信息，比如当天的新闻等。
  - 可靠性、可解释性

    例如，在 Deepseek 联网搜索功能下提问“最近一次人大会议的主要内容”

    DeepSeek - 探索未至之境

    - 对于用户的回答，甚至是回答中的每一句话，都可以给出其对应的检索来源，有引用。以下是 Deepseek 联网搜中支持回答文内引用的 prompt 。

```
search_answer_zh_template = \
'''# 以下内容是基于用户发送的消息的搜索结果:
{search_results}
在我给你的搜索结果中，每个结果都是[webpage X begin]...[webpage X end]格式的，X代表每篇文章的数字索引。请在适当的情况下在句子末尾引用上下文。请按照引用编号[citation:X]的格式在答案中对应部分引用上下文。如果一句话源自多个上下文，请列出所有相关的引用编号，例如[citation:3][citation:5]，切记不要将引用集中在最后返回引用编号，而是在答案对应部分列出。
在回答时，请注意以下几点:
- 今天是{cur_date}。
- 并非搜索结果的所有内容都与用户的问题密切相关，你需要结合问题，对搜索结果进行甄别、筛选。
- 对于列举类的问题（如列举所有航班信息），尽量将答案控制在10个要点以内，并告诉用户可以查看搜索来源、获得完整信息。优先提供信息完整、最相关的列举项；如非必要，不要主动告诉用户搜索结果未提供的内容。
```

```
- 对于创作类的问题（如写论文），请务必在正文的段落中引用对应的参考编号，例如[citation:3][citation:5]，不能只在文章末尾引用。你需要解读并概括用户的题目要求，选择合适的格式，充分利用搜索结果并抽取重要信息，生成符合用户要求、极具思想深度、富有创造力与专业性的答案。你的创作篇幅需要尽可能延长，对于每一个要点的论述要推测用户的意图，给出尽可能多角度的回答要点，且务必信息量大、论述详尽。
- 如果回答很长，请尽量结构化、分段落总结。如果需要分点作答，尽量控制在5个点以内，并合并相关的内容。
- 对于客观类的问答，如果问题的答案非常简短，可以适当补充一到两句相关信息，以丰富内容。
- 你需要根据用户要求和回答内容选择合适、美观的回答格式，确保可读性强。
- 你的回答应该综合多个相关网页来回答，不能重复引用一个网页。
- 除非用户要求，否则你回答的语言需要和用户提问的语言保持一致。

# 用户消息为：
{question}'''
```

- 流程
  - 准备阶段：包含知识的文件 → 预处理（解析、切片、向量化等） → 存储到知识库。
  - 运行阶段：用户输入 → 检索模块（知识库匹配） → 生成模块（大模型生成） → 输出。

2 具体技术实现与优化思路（以小北探索为例）

小北探索是我们正在开发的一个基于开源大模型的AI智能助手，主要功能是用户个人知识库的构建、检索和问答。

https://explore.pku.edu.cn/

文件 → 知识库

- 知识来源
  - 非结构化：pdf，doc(x)，ppt(x)，txt，markdown 等文件。
  - 结构化：表格等。
- 主要步骤：
  1. 提取文本
  - 可以使用的工具
    - python库：**PyMuPDF**等。
    - 开源工具：**MinerU** （非常推荐）等，可将多种格式的文件转化为保留结构信息的 Markdown 或已经根据页面结构进行识别和切分的 一个 json list。
  2. 文本预处理
     - 为什么需要预处理?
       - 提取的文本可能包含大量无用信息，所以需要 **清洗** 。
         - 例如 html 格式的文本，可能包含大量标签信息。

           AI赋能育先机，数智驱动开新局——学生工作系统举办专题学习会

```python
from bs4 import BeautifulSoup
soup = BeautifulSoup(table_body, 'html.parser')
# 提取纯文本
table_body_text = soup.get_text(separator=' ', strip=True)
```

- 提取出的文本很长，与用户输入相关的信息可能只是其中的一小部分，直接用原文本有大量无关信息，而且生成模型的**上下文长度有限**，并不适合直接输入整篇文本，所以需要**切片**，每次检索回一些相关的切片即可。
  - 比较经典的方式是按照连续固定 token 数切片，或使用 `llama_index` 等库按句子级别切片。
  - `MinerU` 工具可直接根据页面结构识别段落并切片，返回切好的 list。但其中小标题和段落可能会分开，可以自行进行一些合并。
- 文件中某些信息是干扰信息，如书籍末尾大量参考文献和专业名词解释索引，可以直接识别删除。

```python
cnt=len(re.findall('出版社', text)) + len(re.findall('Publish', text))
if cnt>2:
  # print(f'该文段出现 出版社 或 publish {cnt} 次,大概率参考文献,跳过')
  pass
```

- 直接用传统检索、基于词频的方法效果不好，因为有用的文本可能并不直接包含查询语句中的词，但与查询语句是 **语义相关** 的，所以需要把文本 **向量化**，通过查询语句向量和文本向量的相似度匹配，来支持语义上的检索。（后续会详细介绍）

- 其他可行的预处理步骤
  - 对用户上传的文件整体进行 **摘要**，除文件切片外，把文件摘要也向量化存储，可以提供不同层级的检索。
  - 对切片进行 **上下文压缩**，例如抽取其中的一些关键词，提高文本的语义密度，减少冗余信息。
  - 为文本生成一些 **假设提问句**，增强用提问进行检索时的相关度。

原文：

> HTTPS 是基于 TLS/SSL 的安全通信协议，用于保护数据的机密性和完整性。它通过加密、身份验证和数据完整性检查来确保通信安全。HTTPS 的握手过程包括协商加密算法、交换密钥和生成会话密钥。会话密钥用于对后续通信进行对称加密。

上下文压缩：

- HTTPS：基于 TLS/SSL，保护数据机密性、完整性。
- 握手流程：协商加密算法、交换密钥、生成会话密钥。
- 会话密钥：用于对称加密通信。

假设提问句：

1. HTTPS 是如何保护通信安全的?

2. HTTPS 握手的主要步骤有哪些?
3. 会话密钥在 HTTPS 中的作用是什么?
4. TLS/SSL 在 HTTPS 中的作用是什么?
5. HTTPS 如何实现加密和身份验证?

- **总结**：预处理是一个为了提升检索效率和准确性的过程，不能套公式，需要根据具体应用场景和需求进行调整。

3. 知识存储
   - 主要使用一些支持向量存储的数据库和支持向量检索的搜索引擎，比如 Elasticsearch、Faiss、Milvus 等。
   - 以 **Elasticsearch** 为例
     - 创建索引：ES中的索引相当于数据库中的表，可以根据不同的需求创建不同的索引。
     - 向索引中添加文档：文档是索引的基本单位，在我们的场景中，文档可以是一个切片，也可以是一个文件摘要。里面包含了**切片的元信息**（如切片所属的文件名、切片编号等）、用于生成的文本、向量化后的文本。

```python
# 定义索引映射
mapping = {
    "mappings": {
        "properties": {
            "content_id": {"type": "keyword", "index": True},
            "hash_name": {"type": "keyword", "index": True},
            "page_idx": {"type": "integer", "index": True},
            "chunk_idx": {"type": "integer", "index": True},
            "search_text": {"type": "text", "analyzer": "ik_max_word"},
            "vector": {"type": "dense_vector", "dims": 1024, "index": True, "similarity": "cosine"}
        }
    }
}

# 创建新的索引
es.indices.create(index=index_name, body=mapping)
```

     - 元信息非常重要：用于定位到原文件以及提供有助于回答的额外信息，如文件名、时间、作者、标签等，根据需要可以存储在 ES 中，也可以另外存储在一个关系型数据库中。
     - 在这里注意，**用于后续生成的文本**和**用于向量化的文本**其实是不同的，前者需要包含有助于生成回答的所有信息，后者是为了更高效的检索。以MinerU提取到pdf中的表格为例：

```python
if type=='table':
    table_body=item.get('table_body','')
    soup = BeautifulSoup(table_body, 'html.parser')

```

```
    # 提取纯文本
    table_body_text = soup.get_text(separator=' ', strip=True)
    table_caption=item.get('table_caption','')
    table_footnote=item.get('table_footnote','')

    # 用于向量化检索的文本，不要带表格标题、表格内容、表格脚注这些标识，因为可能会出现在多个表格文本里，稀释这些表格的文本的差异性，降低检索质量
    text_for_vector=f"{table_caption}\n {table_body_text}\n {table_footnote}"

  # 用于后续生成回答的文本，带有清晰标识，且表格内容保留其结构信息的标签。
    text_for_prompt=f"表格标题：{table_caption}\n表格内容：{table_body}\n表格脚注：{table_footnote}"

    processed_content.append({"page_idx": page_idx, "text_for_vector": text_for_vector,"text_for_prompt":text_for_prompt})
```

用户输入 → 查询语句

意图识别

直接把用户输入作为查询语句有什么问题?

1. 用户输入可能不够准确，需要进行 **纠错**。
    - “我想查关于及其学系的内容” -> “我想查关于机器学习的内容”。
2. 用户的自然语言表达有大量冗余信息，尤其是在语义检索中，向量相似度会受影响，需要进行 **关键词提取**。
    - “我想查关于机器学习的内容” -> “机器学习”。
3. 用户的提问是一个简单的句子，但包含其答案可能是一个较长的段落。类比数据入库入库阶段的假设提问，我们可以在这里使用假设回答，叫做 **假设文档嵌入（HyDE）** 方法，把该提问的模拟答案加入查询语句。
    - 原查询：“RAG是什么？” -> 修改后的查询：“RAG是通过检索外部知识库来增强生成式模型的能力。”
4. 多轮对话中，用户提问不完整，利用历史对话信息才能明白提问意图，需要 **上下文理解**。
    - “机器学习的发展历史是什么” “那么深度学习呢？” -> “深度学习发展历史”

以上的方式都可以认为是一种广义的 **意图识别**，目的是从用户输入中提取出有用的信息，以便更好地进行检索。

查询语句 → 检索结果

检索是整个系统的核心，也是最复杂的部分。

检索方法

- **稀疏检索**： 稀疏检索是一种基于词频的传统检索方法，主要依赖于文档中词语的显性表示。在稀疏检索中，文档和查询都被表示为稀疏向量，向量的维度对应词汇表中的单词。

  常用方法

  1. **TF-IDF（Term Frequency-Inverse Document Frequency）**：
     - 衡量单词在文档和整个语料库中的重要性。
     - 适合快速检索与查询相关的文档。
  2. **BM25（Best Matching 25）**：
     - 对 TF-IDF 的改进，考虑了文档长度的归一化处理。
     - 在实际应用中表现更优。

  TF-IDF 为例

  TF-IDF 是一种衡量单词对文档重要性的统计方法，广泛用于文本检索和关键词提取。它结合了两个核心指标：

  1. **TF（Term Frequency，词频）**：
     - 衡量单词在文档中的出现频率。
     - 公式：

$$\mathrm{TF}(t,d) = \frac{\text{单词 } t \text{ 在文档 } d \text{ 中出现的次数}}{\text{文档 } d \text{ 中单词的总数}} \tag{1}$$

     - 作用：反映单词在当前文档中的重要性。
  2. **IDF（Inverse Document Frequency，逆文档频率）**：
     - 衡量单词在整个语料库中的稀有程度。
     - 公式：

$$\mathrm{IDF}(t) = \log \frac{N}{1 + \mathrm{df}(t)} \tag{2}$$

       N：语料库中的文档总数。

       df (t)： 包含单词 t 的文档数。
     - 作用：降低在大部分文档中频繁出现的单词（如“的”、“是”）的权重。
  3. **TF-IDF 的公式**：

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t) \tag{3}$$

     - 结合 TF 和 IDF，既考虑单词在当前文档中的重要性，又考虑其在整个语料库中的区分能力。

  应用步骤

  1. **构建知识库索引**：对知识库中的每个文档计算 TF-IDF 向量。
     - 对知识库文本分词，计算每个词的TF-IDF值
     - 每个文档的 TF-IDF 向量维度等于词表的大小。
       - 如果某个单词未出现在文档中，向量中其对应维度值为 0，出现则值为 TF-IDF 值。
       - 由于文档中只包含词表的一部分单词，TF-IDF 向量通常是稀疏的。

2. **处理用户查询**：同理，将用户查询转化为 TF-IDF 向量。
3. **计算相似度**：使用 **余弦相似度** 计算查询向量与知识库中每个文档向量的相似度。
4. **返回结果**：根据相似度排序，返回与查询最相关的文档。

优点

1. **简单高效**：计算简单，适合快速检索。
2. **易于实现**：广泛应用于信息检索和关键词提取任务。
3. **可解释性强**：可以清晰地看到单词对文档的重要性。

缺点

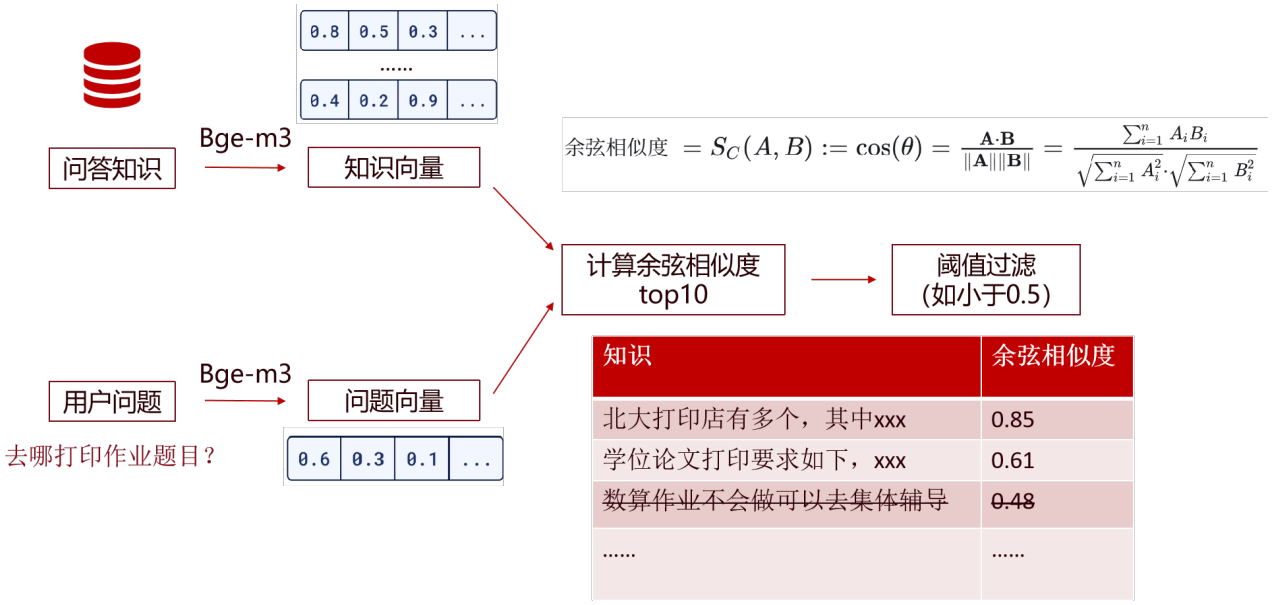
1. **忽略语义信息**：无法理解同义词或上下文。
2. **对长文档效果较差**：长文档中词频可能被稀释。

- **稠密检索**：文本语义向量相似度检索。
  - 原理：通过一个预训练的更强大的嵌入模型，为查询语句和知识库中的文本直接生成一个稠密的向量，计算向量之间的相似度。
    - 例如 BERT 等 预训练 Encoder 模型
  - 优点：可以处理语义相关的匹配，适用于长文本匹配。
  - 缺点：需要额外嵌入模型的支持。
  - 适用场景：较长的文本和复杂的语义匹配。
- 实际应用中，可以结合两种方法，先用传统检索方法进行初步筛选，再用向量检索方法进行精确匹配；或者给两者设置不同的权重，根据具体需求进行调整。

检索流程设计

经典推荐系统和搜索引擎的检索流程是：

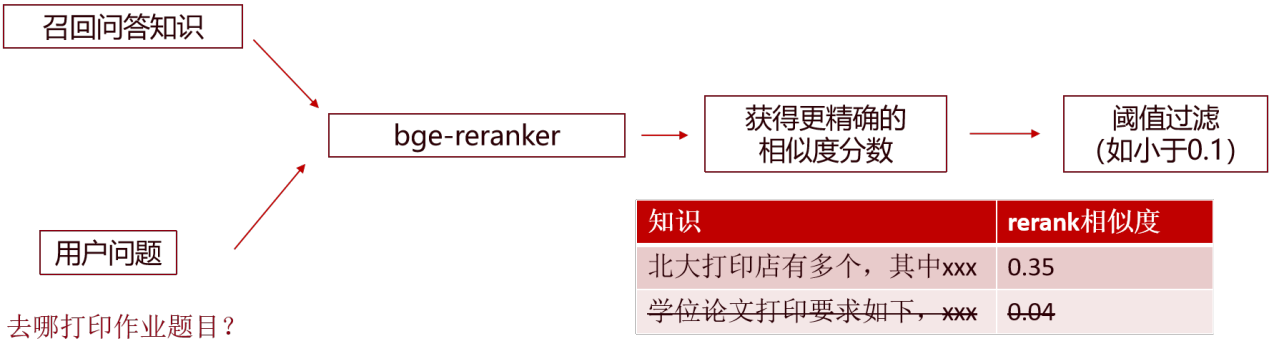
- 用户查询 -> **召回 -> 粗排 -> 精排 -> 重排** -> 得到相关性最高的合适数量的结果。 这里关注的是从**海量上亿级**数据中找到比如与用户**匹配度最高**的商品或者推荐的文章。
- **召回**：双塔模型中，doc 提前向量化存储了，获得用户查询后将其也向量化。然后在众多 doc 中利用 ANN （近似最近邻搜索）找到与 query 向量近邻的 top k 个 doc（直接使用 ES 中的 KNN 插件即可）



知识	余弦相似度
北大打印店有多个，其中xxx	0.85
学位论文打印要求如下，xxx	0.61
~~数算作业不会做可以去集体辅导~~	~~0.48~~
.....	.....

- **重排**：用户 query 和 召回的每个文档拼接，送入具有交叉特性的重排模型，输出一个更精确的相关性分数。依据相关性分数过滤一些更不相关的文档，并将文档以相关性分数排序。（使用开源重排器，如 `bge-reranker`）

知识
北大打印店有多个，其中xxx
学位论文打印要求如下，xxx



知识	rerank相似度
北大打印店有多个，其中xxx	0.35
~~学位论文打印要求如下，xxx~~	~~0.04~~

我们如果要设计实现一个**智能问答应用**（如科研助手、智能客服等），终极目标是**为用户生成高质量的回答**，生成优质回答需要**检索的路径和内容非常多样**，因此检索流程优化的侧重点可能会不同。应对复杂的用户需求，需要仔细考虑，设计多路径或多轮的召回流程，甚至需要在与用户的交互反馈中不断优化检索流程。

案例：
我有大量NLP相关教材和论文，已经通过解析、切片、向量化构建好一个知识库。

- 用户输入1：word2vec 的训练方法是什么?
- 用户输入2：根据知识库的内容形成关于 bert 的简短综述。

思考现有流程是否能回答好这两个问题?

- 输入1是一个**知识性提问**，用 “ word2vec 训练方法” 作为查询语句，直接用检索回的切片就能生成一个比较好的回答了。
- 输入2是一个**复杂的生成任务**，用 “ bert 综述” 作为关键词检索回的切片（论文的一些片段）直接作为模型输入，是否能满足需求?

针对论文知识库场景，对某个主题生成综述的需求有多种解决方案，比如：

- **依据结构特性切片 + 切片检索**：PDF 解析和切片时充分利用论文的结构信息，对每篇论文把 “相关研究” 部分都完美放进一个切片，使用 “ bert 相关研究” 查询，期望能检索回这些论文的 “相关研究” 切片，那么生成模型可以进行组合生成。
- **切片检索 + 论文全文或摘要，父文档检索**
    1. 使用 “ bert ” 作为查询，检索回一些论文切片，包含所属论文的元信息。召回则代表这篇论文讨论了 bert ，大概率应该纳入 bert 综述写作。但是用于生成综述的文本不应该是这次召回的论文片段，而是片段所属论文的整体信息。
    2. 根据元信息将相关论文全文或再生成摘要作为检索模块的最终输出。
- **意图识别 + 摘要检索**：写综述需要的本来就是相关文章的摘要，那么直接用 bert 在知识库的文章摘要中做检索即可，需要实现生成好所有文件的摘要。
    1. 用户上传文件解析时就对文件全文摘要，并向量化存储。
    2. 对用户输入 “根据知识库的内容形成关于 bert 的简短综述” 进行意图识别，判断出需要的是 摘要检索。
    3. 使用 “ bert ” 作为查询，检索回一些文章摘要，直接用这些摘要作为生成模型的输入。
- **多向量检索**：综合切片检索和摘要检索的结果。

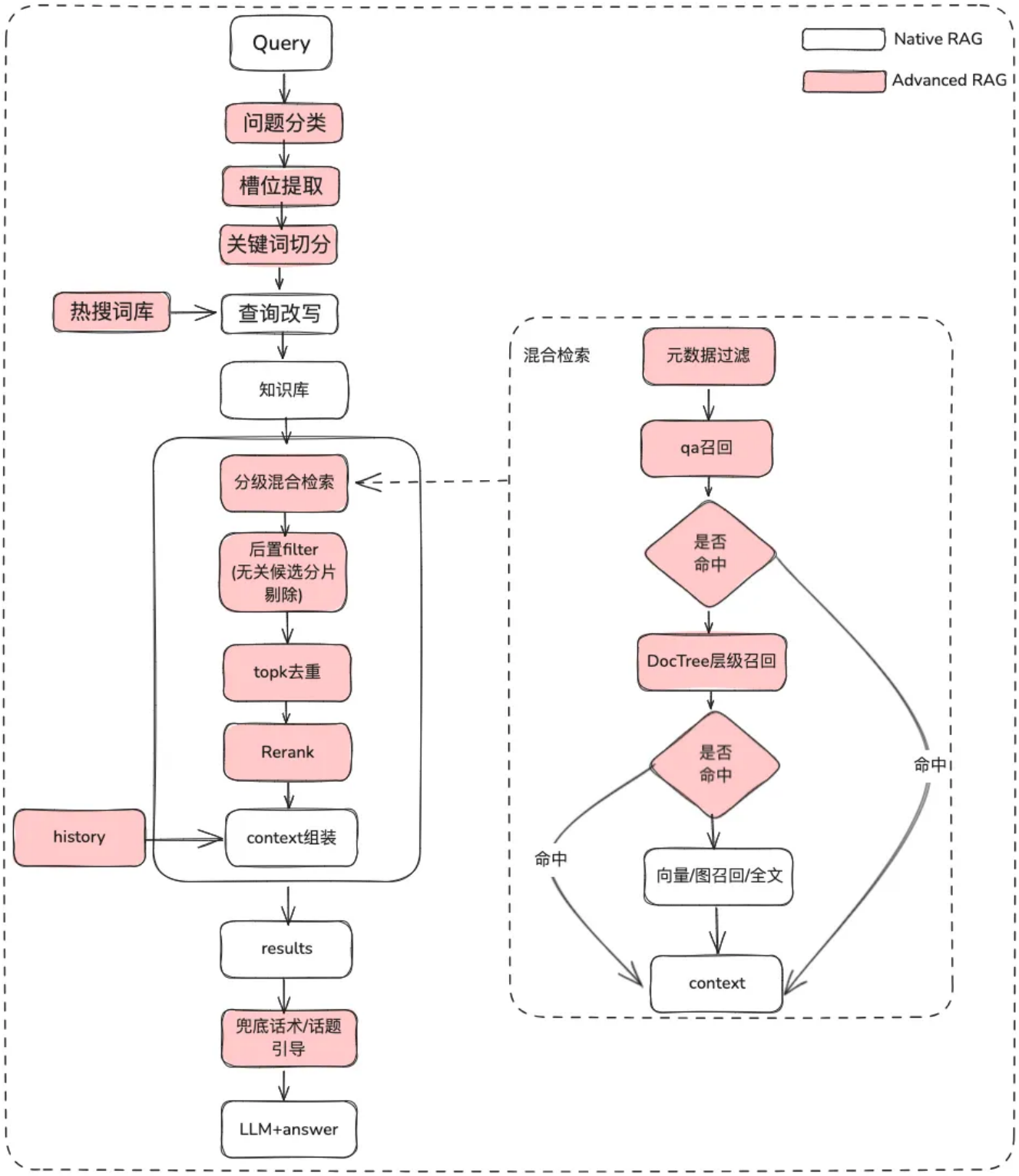
可以看到一个统一的检索流程无法解决类型多样的用户需求，所以现在更火热的概念是 **Agent(智能体)** 。在我们的案例中，需要构建一个检索 Agent ，它能根据用户需求和检索结果自主决策。根据用户输入的不同，选择不同的检索路径，甚至可以在检索过程中根据检索结果不断调整路径。

检索结果 → 生成回答

我个人理解中，检索重要性远大于生成，用户非常关心是哪些材料构成了我的回答。

生成的回答只是一个 **展示形式** ，根据用户意图可以是检索结果的简单拼接、分析、总结、解释、发散等，形式也可以是文本、表格甚至图片。 生成质量主要依赖于

- **检索结果的质量**
- **生成模型的能力**-



应用场景

依托知识库+检索的基本架构，可以开发众多应用。

知识库中的文件	任务或应用场景
简历	受试者推荐
论文	科研写作助手
教材、习题	AI助教
校园生活信息	校园助手
产品信息、售后规定	智能客服
……	……