RAG高级技术与调优

>> 今天的学习目标

RAG高级技术与调优

- 坚实地基：知识库处理

场景1：知识库问题生成与检索优化

场景2：对话知识沉淀

场景3：知识库健康度检查

场景4：知识库版本管理与性能比较

- 精准雷达：高级召回

查询优化: MultiQuery & 改写

混合检索: BM25 + Vector

精细排序: Rerank 模型

- 全局视野：GraphRAG

图谱构建: 实体抽取 & 社区摘要

查询模式: Global vs Local Search

- 智能决策：Qwen Agent中的RAG

NativeRAG

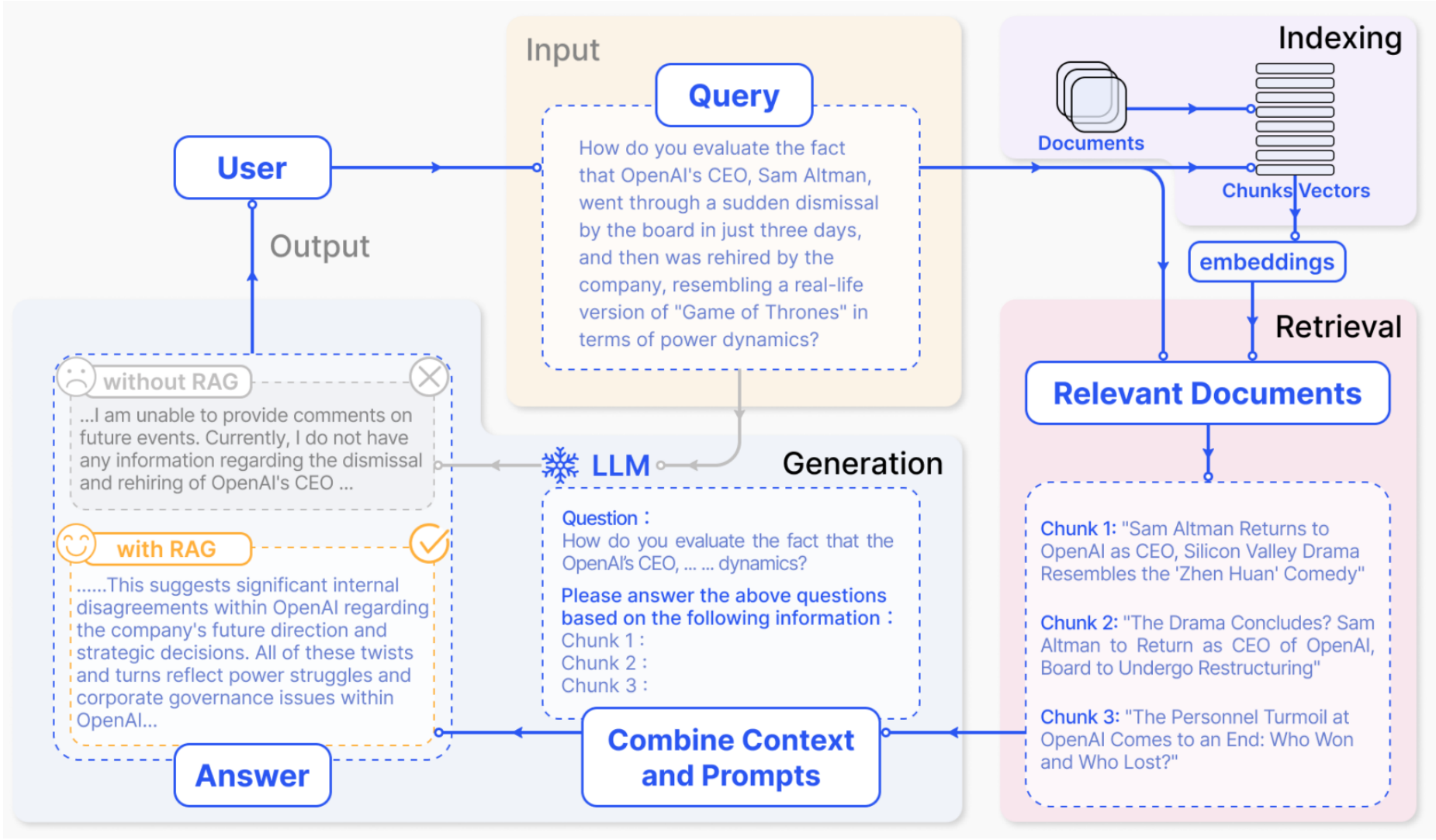
RAG的步骤：

Indexing => 如何更好地把知识存起来。

Retrieval => 如何在大量的知识中，找到一小部分有用的，给到模型参考。

Generation => 如何结合用户的提问和检索到的知识，让模型生成有用的答案。

这三个步骤虽然看似简单，但在 RAG 应用从构建到落地实施的整个过程中，涉及较多复杂的工作内容



RAG高级技术与调优

Thinking：如何从不同的维度对RAG进行调优？

坚实基础：知识库处理

入库前: 问题生成 & 对话沉淀

入库后: 健康度检查 & 版本管理

全局视野：GraphRAG

图谱构建: 实体抽取 & 社区摘要

查询模式: Global vs Local Search

精准雷达：高效召回

查询优化: MultiQuery & 改写

混合检索: BM25 + Vector

精细排序: Rerank 模型

智能决策：Agentic RAG

Level 1: 基础 RAG

Level 2: 并行阅读

Level 3: 多跳推理

坚实地基：知识库处理

场景1：知识库问题生成与检索优化

Thinking：当用户提问与知识切片的相似度不高时，能否通过AI为每个知识切片生成可能的问题，通过问题与问题的匹配来提高检索准确度？

TO DO：对知识库进行问题生成，并进行检索优化

- 自动生成多样化问题：为知识库中的每个知识切片自动生成多种类型、不同难度的问题

generate_questions_for_chunk() : 为单个知识切片生成基础问题

generate_diverse_questions() : 生成更多样化的问题（8个）

- 构建双重检索索引：同时构建基于原文内容和生成问题的BM25检索索引
- 检索评估：针对两种检索方式，进行详细的评估

场景1：知识库问题生成与检索优化

```
def generate_questions_for_chunk(self, knowledge_chunk, num_questions=5):
    """为单个知识切片生成多样化问题"""
    instruction = """
你是一个专业的问答系统专家。给定的知识内容能回答哪些多样化的问题，这些问题可以：
1. 使用不同的问法（直接问、间接问、对比问等）
2. 避免重复和相似的问题
3. 确保问题不超出知识内容范围

请返回JSON格式：
{
  "questions": [
    {
      "question": "问题内容",
      "question_type": "问题类型（直接问/间接问/对比问/条件问等）",
      "difficulty": "难度等级（简单/中等/困难）"
    }
  ]
}
"""
    prompt = f"""
### 指令 ###
{instruction}
```

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

```
### 知识内容 ###
{knowledge_chunk}

### 生成问题数量 ###
{num_questions}

### 生成结果 ###
"""

    response = get_completion(prompt, self.model)

    # 预处理响应，移除markdown代码块格式
    response = preprocess_json_response(response)
    ...
```

知识内容: 上海迪士尼乐园位于上海市浦东新区，是中国大陆首座迪士尼主题乐园，于2016年6月16日开园。乐园占地面积390公顷，包含七大主题园区：米奇大街、奇想花园、探险岛、宝藏湾、明日世界、梦幻世界和迪士尼小镇。

生成的5个问题:

1. 上海迪士尼乐园是什么时候开园的？ (类型: 直接问, 难度: 简单)
2. 中国大陆第一座迪士尼乐园在哪里？ (类型: 间接问, 难度: 简单)
3. 与美国、日本等地的迪士尼相比，上海迪士尼有什么特别之处？ (类型: 对比问, 难度: 中等)
4. 如果想游览上海迪士尼的所有主题园区，需要了解哪些区域？ (类型: 条件问, 难度: 中等)
5. 上海迪士尼乐园占地多少公顷？ (类型: 直接问, 难度: 简单)

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

```
def generate_diverse_questions(self, knowledge_chunk,
num_questions=8):
    """生成更多样化的问题（更丰富）"""
    instruction = """
你是一个专业的问答系统专家。请为给定的知识内容生成高度多样化的问题，确保：
1. 问题类型多样化：直接问、间接问、对比问、条件问、假设问、推理问等
2. 表达方式多样化：使用不同的句式、词汇、语气
3. 难度层次多样化：简单、中等、困难的问题都要有
4. 角度多样化：从不同角度和维度提问
5. 确保问题不超出知识内容范围
请返回JSON格式：
{
```

```
"questions": [
    {
      "question": "问题内容",
      "question_type": "问题类型",
      "difficulty": "难度等级",
      "perspective": "提问角度",
      "is_answerable": "给出的知识能否回答该问题",
      "answer": "基于该知识的回答"
    }
  ]
}
"""
```

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

```
prompt = f"""
### 指令 ###
{instruction}

### 知识内容 ###
{knowledge_chunk}

### 生成问题数量 ###
{num_questions}

### 生成结果 ###
"""
```

生成的8个多样化问题:

1. 上海迪士尼乐园是在哪一年开园的？

   类型: 直接问, 难度: 简单, 角度: 时间信息, 能否回答: True, 回答的答案：2016年

2. 如果一个人想在上海体验迪士尼主题乐园，他应该去哪里？

   类型: 间接问, 难度: 简单, 角度: 地理位置与用途, 能否回答: True, 回答的答案：上海迪士尼乐园，位于上海市浦东新区

3. 上海迪士尼和香港迪士尼相比，哪个是中国大陆首座迪士尼主题乐园？

   类型: 对比问, 难度: 中等, 角度: 地域比较, 能否回答: True, 回答的答案：上海迪士尼是中国大陆首座迪士尼主题乐园

4. 假如你计划去上海迪士尼游玩，你会选择哪个主题园区来体验探险氛围？

   类型: 假设问, 难度: 中等, 角度: 游客体验角度, 能否回答: True, 回答的答案：探险岛

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

5. 为什么说上海迪士尼乐园在中国具有特殊意义？

类型: 推理问, 难度: 困难, 角度: 历史与文化价值, 能否回答: True, 回答的答案：因为它是中国大陆首座迪士尼主题乐园，标志着迪士尼品牌在中国市场的落地与扩展

6. 上海迪士尼乐园占地多少公顷？

类型: 直接问, 难度: 简单, 角度: 面积数据, 能否回答: True, 回答的答案：390公顷

7. 迪士尼小镇属于上海迪士尼乐园的哪个部分？

类型: 间接问, 难度: 中等, 角度: 结构组成, 能否回答: True, 回答的答案：迪士尼小镇是上海迪士尼乐园的一部分，与七大主题园区并列

8. 若要全面游览上海迪士尼乐园的全部主题园区，需要了解哪些区域？

类型: 条件问, 难度: 困难, 角度: 规划与认知, 能否回答: True, 回答的答案：需要了解米奇大街、奇想花园、探险岛、宝藏湾、明日世界、梦幻世界和迪士尼小镇这七大主题园区

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

```
# 示例3: 评估检索方法
test_queries = [
    {
        "query": "如果我想体验最刺激的过山车，应该去哪个区域？",
        "correct_chunk": knowledge_base[4]['content']
    },
    {
        "query": "什么时间去人比较少？",
        "correct_chunk": knowledge_base[2]['content']
    },
    {
        "query": "可以带食物进去吗？",
        "correct_chunk": knowledge_base[5]['content']
    }
]
```

```
# 为知识库生成问题
print('正在为知识库生成问题...')
for chunk in knowledge_base:
    chunk['generated_questions'] =
optimizer.generate_questions_for_chunk(chunk['content'])
print('为知识库生成问题完毕')


# 评估检索方法
results = optimizer.evaluate_retrieval_methods(knowledge_base,
test_queries)
...
```

1-知识库问题生成与检索优化-BM25.py

场景1：知识库问题生成与检索优化

测试查询数量: 3

BM25原文检索准确率: 66.7%

BM25问题检索准确率: 100.0%

问题检索改进的查询数量: 1

=== 详细分析 ===

问题检索方法表现更好的查询（按分数差异排序）:

1. 查询: 如果我想体验最刺激的过山车，应该去哪个区域？

   原文检索分数: 0.155

   问题检索分数: 1.000

   分数差异: +0.845

   原文检索: ✗

   问题检索: ✓

2. 查询: 可以带食物进去吗？

   原文检索分数: 0.153

   问题检索分数: 0.556

   分数差异: +0.402

   原文检索: ✓

   问题检索: ✓

3. 查询: 什么时间去人比较少？

   原文检索分数: 0.168

   问题检索分数: 0.279

   分数差异: +0.112

   原文检索: ✓

   问题检索: ✓

原文检索方法表现更好的查询:

1-知识库问题生成与检索优化-BM25.py

场景2：对话知识沉淀

Thinking：产品上线后每天产生大量对话，如何从这些对话中提取和沉淀有价值的知识，持续丰富知识库？

TO DO：对话知识沉淀

核心功能：

- 使用AI模型（通义千问）从对话中提取结构化知识
- 支持多种知识类型：事实、问题、流程、注意事项等
- 自动识别用户意图和对话摘要

核心函数：

- extract_knowledge_from_conversation() : 从单次对话中提取知识
- batch_extract_knowledge() : 批量提取知识
- merge_similar_knowledge()：使用LLM合并相似知识点

场景2：对话知识沉淀

Thinking：如何对知识进行分类？

使用LLM进行参数提取

```
instruction = """
你是一个专业的知识提取专家。请从给定的对话中提取有价值的知识点，包括：
1. 事实性信息（地点、时间、价格、规则等）
2. 用户需求和偏好
3. 常见问题和解答
4. 操作流程和步骤
5. 注意事项和提醒

请返回JSON格式：
{
  "extracted_knowledge": [
    {
      "knowledge_type": "知识类型（事实/需求/问题/流程/注意）",
      "content": "知识内容",
      "confidence": "置信度(0-1)",
      "source": "来源（用户/AI/对话）",
      "keywords": ["关键词1", "关键词2"],
      "category": "分类"
    }
  ],
  "conversation_summary": "对话摘要",
  "user_intent": "用户意图"
}
"""
```

2-对话知识沉淀.py

场景2：对话知识沉淀

示例1: 从单次对话中提取知识

对话内容:

用户: "我想去上海迪士尼乐园玩，门票多少钱？"

AI: "上海迪士尼乐园的门票价格根据日期有所不同。平日成人票价为399元，周末和节假日为499元。儿童票（1.0-1.4米）平日为299元， 周末为374元。1.0米以下儿童免费。"

用户: "需要提前预订吗？"

AI: "建议提前预订，特别是周末和节假日，可以通过官方网站或第三方平台预订。"

用户: "从浦东机场怎么去？"

AI: "从浦东机场到迪士尼乐园可以乘坐地铁2号线到广兰路站，然后换乘11号线到迪士尼站，全程约1小时。也可以打车，约40分钟。"

2-对话知识沉淀.py

场景2：对话知识沉淀

提取的知识点:

1. 类型: 事实

   内容: 上海迪士尼乐园平日成人票价为399元，周末和节假日为499元。

   置信度: 1.0

   分类: 门票信息

2. 类型: 事实

   内容: 儿童票（1.0-1.4米）平日为299元，周末为374元；1.0米以下儿童免费。

   置信度: 1.0

   分类: 门票信息

3. 类型: 需求

   内容: 用户希望了解上海迪士尼乐园的门票价格，并计划前往游玩。

   置信度: 0.95

   分类: 用户意图

2-对话知识沉淀.py

场景2：对话知识沉淀

4. 类型: 流程

   内容: 从浦东机场到迪士尼乐园可乘坐地铁2号线到广兰路站，换乘11号线到迪士尼站，全程约1小时。

   置信度: 0.9

   分类: 出行指南

5. 类型: 流程

   内容: 从浦东机场打车前往迪士尼乐园约需40分钟。

   置信度: 0.9

   分类: 出行指南

6. 类型: 注意

   内容: 建议提前预订门票，特别是周末和节假日，可通过官方网站或第三方平台预订。

   置信度: 0.95

   分类: 购票提醒

对话摘要: 用户咨询上海迪士尼乐园门票价格、是否需要提前预订以及从浦东机场如何前往。AI提供了详细的票价信息、购票建议和交通方式（地铁与打车）。

用户意图: 获取上海迪士尼乐园的门票价格、购票建议及从浦东机场前往的交通方式，以便规划行程。

2-对话知识沉淀.py

场景2：对话知识沉淀

Thinking：如何过滤掉不是知识的内容，比如客户的需求？

```
# 过滤掉需求和问题类型的知识，因为它们是临时的、个性化的
filtered_knowledge = [
    knowledge for knowledge in knowledge_list
    if knowledge.get('knowledge_type') not in ['需求', '问题']
]
```

2-对话知识沉淀.py

场景2：对话知识沉淀

Thinking：如何对知识点进行合并，形成新的知识？

使用LLM进行智能合并，以下是LLM的合并流程：

```
# 按知识类型分组
knowledge_by_type = {}
for knowledge in filtered_knowledge:
    knowledge_type = knowledge.get('knowledge_type', '其他')
    if knowledge_type not in knowledge_by_type:
        knowledge_by_type[knowledge_type] = []
    knowledge_by_type[knowledge_type].append(knowledge)
```

```
# 对每个知识类型分别进行LLM合并
for knowledge_type, knowledge_group in knowledge_by_type.items():
    if len(knowledge_group) == 1:
        # 只有一个知识点，直接添加
        merged_knowledge.append(knowledge_group[0])
    else:
        # 多个知识点，使用LLM合并
        merged = self.merge_knowledge_with_llm(knowledge_group,
knowledge_type)
        merged_knowledge.append(merged)
```

2-对话知识沉淀.py

场景2：对话知识沉淀

LLM知识合并提示词：

```
prompt = f"""
你是一个专业的知识整理专家。请将以下{knowledge_type}类型的知识点进行智能合并，生成一个更完整、准确的知识点。

### 合并要求：
1. 保留所有重要信息，避免信息丢失
2. 消除重复内容，整合相似表述
3. 提高内容的准确性和完整性
4. 保持逻辑清晰，结构合理
5. 合并后的置信度取所有知识点中的最高值

### 待合并的知识点：
{chr(10).join(knowledge_contents)}

### 请返回JSON格式：
{{
   "knowledge_type": "{knowledge_type}",
   "content": "合并后的知识内容",
   "confidence": 最高置信度值,
   "keywords": ["合并后的关键词列表"],
   "category": "合并后的分类",
   "sources": ["所有来源"],
   "frequency": {len(knowledge_group)}
}}

### 合并结果：
"""
```

2-对话知识沉淀.py

场景2：对话知识沉淀

示例2: 批量提取知识

正在处理对话 1/3...

正在处理对话 2/3...

正在处理对话 3/3...

总共提取了 22 个知识点

所有知识点:

事实:上海迪士尼乐园平日成人票价为399元，周末和节假日为499元。: 1次

事实:儿童票（1.0-1.4米）平日为299元，周末为374元；1.0米以下儿童免费。: 1次

需求:用户希望了解上海迪士尼乐园的门票价格，并计划前往游玩。: 1次

流程:从浦东机场到上海迪士尼乐园可乘坐地铁2号线至广兰路站，换乘11号线至迪士尼站，全程约1小时。: 1次

流程:从浦东机场打车前往迪士尼乐园约需40分钟。: 1次

注意:建议提前预订门票，特别是周末和节假日，可通过官方网站或第三方平台预订。: 1次

事实:上海迪士尼乐园通常每天开放，营业时间为上午8:00至晚上8:00。: 1次

事实:人流量会因日期、季节和活动而变化，周末、节假日和寒暑假通常人较多。: 1次

2-对话知识沉淀.py

场景2：对话知识沉淀

事实:上海迪士尼乐园的必玩项目包括：创极速光轮（明日世界）、七个小矮人矿山车（梦幻世界）、加勒比海盗：战争: 1次

需求:用户想知道当前是否开放、人流量情况以及推荐游玩项目。: 1次

流程:建议用户出发前查看官方网站或APP确认具体营业时间；查看官方APP的实时人流量信息，或关注社交媒体上: 1次

注意:营业时间可能因特殊活动或维护调整，建议出发前再次确认。: 1次

事实:上海迪士尼乐园停车场收费为100元/天。: 1次

事实:部分游乐项目有身高限制。: 1次

事实:可以携带密封包装的零食和水进入迪士尼，但不能带玻璃瓶和酒精饮料。: 1次

流程:建议下载官方APP查看实时排队时间，以优化游玩体验。: 1次

流程:可租用婴儿车，适合带小孩家庭使用。: 1次

注意:建议提前到达停车场，因为可能会满；也可选择地铁出行（11号线迪士尼站）。: 1次

注意:园内餐饮价格相对较高，建议自备适量零食和水。: 1次

需求:用户希望了解带小孩去迪士尼时需要特别注意的事项。: 1次

问题:用户询问停车费如何收取。: 1次

问题:用户询问是否可以携带食物进入园区。: 1次

2-对话知识沉淀.py

场景2：对话知识沉淀

示例3: 合并相似知识

过滤前知识点数量: 22

过滤后知识点数量: 17

过滤掉的'需求'和'问题'类型知识点: 5

合并后剩余 3 个知识点

合并后的知识点:

1. 类型: 事实

内容: 上海迪士尼乐园门票价格为：成人平日票价399元，周末及节假日499元；儿童票（身高1.0-1.4米）平日299元，周末374元，1.0米以下儿童免费。乐园通常每日开放，营业时间为上午8:00至晚上8:00，但人流量会因日期、季节和活动而变化，周末、节假日和寒暑假 期间游客较多。必玩项目包括创极速光轮（明日世界）、七个小矮人矿山车（梦幻世界）、加勒比海盗：战争之潮（宝藏湾）和翱翔·飞越 地平线（探险岛）。园内部分游乐项目设有身高限制，入园时需遵守相关规定，如可携带密封包装的零食和水，但禁止携带玻璃瓶及酒精饮料。此外，园区停车场收费为100元/天。

频率: 8次

置信度: 1.0

2-对话知识沉淀.py

场景2：对话知识沉淀

分类: 综合旅游信息

关键词: ['门票价格', '儿童票', '营业时间', '人流量', '必玩项目', '身高限制', '食物携带', '停车费', '上海迪士尼']

来源: ['AI']

2. 类型: 流程

内容: 从浦东机场前往上海迪士尼乐园，可选择地铁或打车两种交通方式：乘坐地铁2号线至广兰路站，换乘11号线直达迪士尼站，全程约1小时；或直接打车，约需40分钟。出行前建议通过官方APP或官网查询最新营业时间，并查看实时人流量和排队信息以优化行程安排。游玩时可下载官方APP获取实时排队时间，提升体验效率；园区提供婴儿车租赁服务，适合带小孩的家庭使用。

频率: 5次

置信度: 0.9

分类: 出行与游玩准备

关键词: ['交通方式', '浦东机场', '迪士尼', '打车', '官网', 'APP', '人流量', '排队时间', '婴儿车', '租赁']

来源: ['AI', 'AI', 'AI', 'AI', 'AI']

2-对话知识沉淀.py

场景2：对话知识沉淀

3. 类型: 注意

内容: 建议提前预订门票，尤其在周末和节假日期间，可通过官方网站或第三方平台完成；营业时间可能因特殊活动或维护调整，出 行前请再次确认；为避免停车困难，建议提前到达停车场或选择地铁出行（11号线迪士尼站）；园内餐饮价格较高，建议自备适量零食和饮用水以节省开支。

频率: 4次

置信度: 0.95

分类: 综合提醒

关键词: ['提前预订', '门票', '节假日', '营业时间调整', '特殊活动', '维护', '停车', '地铁出行', '餐饮价格', '自备食物']

来源: ['AI']

2-对话知识沉淀.py

场景3：知识库健康度检查

Thinking：如何对整个知识库进行健康度检查，找出缺少的知识、过期的知识、冲突的知识，确保知识库的质量和可靠性。

TO DO：知识库健康度检查

核心功能：

- 完整性检查：评估知识库是否覆盖用户的主要查询需求
- 时效性检查：识别过期或需要更新的知识内容
- 一致性检查：发现知识库中的冲突和矛盾信息
- 综合评分：提供量化的健康度评分和改进建议

场景3：知识库健康度检查

Thinking：如何用LLM检查缺少的知识？

```
instruction = """
你是一个知识库完整性检查专家。请分析给定的测试查询和知识库内容，判断知识库中是否缺少相关的知识。

检查标准：
1. 查询是否能在知识库中找到相关答案
2. 知识是否完整、准确
3. 是否覆盖了用户的主要需求
4. 是否存在知识空白

请返回JSON格式：
{
```

```
"missing_knowledge": [
  {
    "query": "测试查询",
    "missing_aspect": "缺少的知识方面",
    "importance": "重要性（高/中/低）",
    "suggested_content": "建议的知识内容",
    "category": "知识分类"
  }
],
"coverage_score": "覆盖率评分(0-1)",
"completeness_analysis": "完整性分析"
}
"""
```

```
prompt = f"""
### 指令 ###
{instruction}

### 知识库内容 ###
{knowledge_text}

### 测试查询 ###
{queries_text}

### 分析结果 ###
"""
```

3-知识库健康度检查.py

场景3：知识库健康度检查

Thinking：如何用LLM检查过期的知识？

```
instruction = """
你是一个知识时效性检查专家。请分析给定的知识内容，判断是否存在过期或需要更新的信息。
检查标准：
1. 时间相关信息是否过期（年份、日期、时间范围）
2. 价格信息是否最新（价格、费用、票价等）
3. 政策规则是否更新（政策、规定、规则等）
4. 活动信息是否有效（活动、节日、特殊安排等）
5. 联系方式是否准确（电话、地址、网址等）
6. 技术信息是否过时（版本、技术标准等）
请返回JSON格式：
{
```

```
"outdated_knowledge": [
    {
        "chunk_id": "知识切片ID",
        "content": "知识内容",
        "outdated_aspect": "过期方面",
        "severity": "严重程度（高/中/低）",
        "suggested_update": "建议更新内容",
        "last_verified": "最后验证时间"
    }
  ],
  "freshness_score": "新鲜度评分(0-1)",
  "update_recommendations": "更新建议"
} """
```

```
prompt = f"""
### 指令 ###
{instruction}

### 知识库内容 ###
{knowledge_text}

### 当前时间 ###
{datetime.now().strftime('%Y年%m月%d日')}

### 分析结果 ###
"""
```

3-知识库健康度检查.py

场景3：知识库健康度检查

Thinking：如何用LLM检查冲突的知识？

```
instruction = """
你是一个知识一致性检查专家。请分析给定的知识库，找出可能存在冲突或矛盾的信息。

检查标准：
1. 同一主题的不同说法（地点、名称、描述等）
2. 价格信息的差异（价格、费用、收费标准等）
3. 时间信息的不一致（营业时间、开放时间、活动时间等）
4. 规则政策的冲突（规定、政策、要求等）
5. 操作流程的差异（步骤、方法、流程等）
6. 联系方式的差异（地址、电话、网址等）
请返回JSON格式：
```

```
{
  "conflicting_knowledge": [
    {
      "conflict_type": "冲突类型",
      "chunk_ids": ["相关切片ID"],
      "conflicting_content": ["冲突内容"],
      "severity": "严重程度（高/中/低）",
      "resolution_suggestion": "解决建议"
    }
  ],
  "consistency_score": "一致性评分(0-1)",
  "conflict_analysis": "冲突分析"
} """
```

```
prompt = f"""
### 指令 ###
{instruction}

### 知识库内容 ###
{knowledge_text}

### 分析结果 ###
"""
```

3-知识库健康度检查.py

场景3：知识库健康度检查

正在检查知识库健康度...

1. 检查缺少的知识...

2. 检查过期的知识...

3. 检查冲突的知识...

=== 知识库健康度报告 ===

整体健康度评分: 0.60

健康等级: 良好

检查时间: 2025-08-03T08:57:36.164103

=== 详细分析 ===

1. 缺少的知识分析:

覆盖率: 60.0%

缺少知识点数量: 2

1. 查询: 有什么特别活动？

缺少方面: 乐园当前或定期举办的特别活动信息（如节日庆典、演出、限时活动等）

重要性: 高

2. 查询: 停车费是多少？

缺少方面: 停车场收费标准（包括小时计费、全天封顶价、是否提供免费停车等）

重要性: 中

场景3：知识库健康度检查

2. 过期的知识分析:

新鲜度评分: 0.60

过期知识点数量: 2

1. 切片ID: kb_002

过期方面: 价格信息

严重程度: 高

2. 切片ID: kb_004

过期方面: 活动信息

严重程度: 中

3. 冲突的知识分析:

一致性评分: 0.60

冲突数量: 2

1. 冲突类型: 价格信息的差异

相关切片: ['kb_002', 'kb_003']

严重程度: 高

2. 冲突类型: 时间信息的不一致

相关切片: ['kb_004']

严重程度: 中

=== 改进建议 ===

1. 补充2个缺少的知识点，提高覆盖率

2. 更新2个过期知识点，确保信息时效性

3. 解决2个知识冲突，提高一致性

场景4：知识库版本管理与性能比较

Thinking：如何对知识库进行版本管理，实现回归测试、上线前验收，并比较不同版本的知识库性能，选择最优版本。

TO DO：知识库版本管理与性能比较

核心功能：

- 版本创建：为知识库创建带描述和统计信息的版本
- 哈希值计算：使用MD5计算版本的唯一标识
- 统计信息：记录知识切片数量、内容长度、分类分布等
- 版本比较：比较两个版本的差异和变化

场景4：知识库版本管理与性能比较

1. 文本向量化模块

```
def get_text_embedding(text):
    # 调用通义千问embedding API
    response = client.embeddings.create(
        model="text-embedding-v4",
        input=text,
        dimensions=1024
    )
    return response.data[0].embedding
```

将文本转换为1024维向量表示。相似语义的文本向量距离更近。

调用embedding API → 获取向量数据

2. 向量索引构建模块

```
def build_vector_index(self, knowledge_base):
    # 遍历知识库，生成向量和元数据
    for chunk in knowledge_base:
        vector = get_text_embedding(chunk['content'])
        metadata = {"id": i, "content": content, "chunk_id": chunk_id}
    # 创建FAISS索引并添加向量
    text_index = faiss.IndexFlatL2(1024)
    text_index.add_with_ids(vectors, ids)
    return metadata_store, text_index
```

构建FAISS向量索引支持高效检索。

遍历知识库 → 生成向量 → 创建索引 → 添加向量数据。

场景4：知识库版本管理与性能比较

3.版本差异检测模块

```
def detect_changes(self, kb1, kb2):
    # 创建ID映射字典
    kb1_dict = {chunk['id']: chunk for chunk in kb1}
    kb2_dict = {chunk['id']: chunk for chunk in kb2}
    # 使用集合运算检测变化
    added_ids = set(kb2_dict.keys()) - set(kb1_dict.keys())
    removed_ids = set(kb1_dict.keys()) - set(kb2_dict.keys())
    common_ids = set(kb1_dict.keys()) & set(kb2_dict.keys())

    # 检测内容修改
    for chunk_id in common_ids:
        if kb1_dict[chunk_id]['content'] != kb2_dict[chunk_id]['content']:
            modified_chunks.append(...)
```

识别版本间的增删改变化。

创建ID映射 → 集合运算检测 → 内容对比 → 生成变更报告。

基于集合论，通过差集、交集运算快速识别新增、删除和修改的知识切片。

场景4：知识库版本管理与性能比较

4.向量检索模块

```
def retrieve_relevant_chunks(self, query, version_name, k=3):
    # 获取查询向量
    query_vector = get_text_embedding(query)
    # FAISS相似度搜索
    distances, indices = text_index.search(query_vector, k)

    # 构造返回结果
    for i, doc_id in enumerate(indices[0]):
        chunk = {
            "id": match["chunk_id"],
            "content": match["content"],
            "similarity_score": 1.0 / (1.0 + distances[0][i])
        }
```

基于向量相似度检索相关知识。

查询向量化 → FAISS搜索 → 距离转换 → 返回Top-K结果。

计算查询向量与知识库向量的欧几里得距离，距离越小相似度越高，返回最相似的K个结果。

场景4：知识库版本管理与性能比较

5. 性能评估模块

```
def evaluate_version_performance(self, version_name, test_queries):
    # 遍历测试查询
    for query_info in test_queries:
        # 记录响应时间
        start_time = datetime.now()
        retrieved_chunks = self.retrieve_relevant_chunks(query, version_name)
        response_time = (end_time - start_time).total_seconds()
        # 评估检索质量
        is_correct = self.evaluate_retrieval_quality(query, retrieved_chunks,
expected_answer)
    # 计算整体指标
    accuracy = correct_answers / total_queries
    avg_response_time = sum(response_times) / len(response_times)
```

量化评估知识库检索性能。

执行测试查询 → 记录响应时间 → 评估准确性 → 计算综合指标。

通过标准测试集评估检索准确率和响应时间，准确率=正确检索次数/总查询次数。

embedding用于检索：将查询转换为向量，在知识库中检索top-3最相似的知识切片

简单字符串匹配评估：检查期望答案是否在检索到的任何一个知识切片中

场景4：知识库版本管理与性能比较

6. 性能比较模块

```
def compare_version_performance(self, version1_name, version2_name,
test_queries):
    # 分别评估两个版本
    perf1 = self.evaluate_version_performance(version1_name, test_queries)
    perf2 = self.evaluate_version_performance(version2_name, test_queries)

    # 计算改进幅度
    accuracy_improvement = perf2['accuracy'] - perf1['accuracy']
    time_improvement = perf1['avg_response_time'] -
perf2['avg_response_time']
    # 生成建议
    recommendation = self.generate_performance_recommendation(perf1,
perf2)
```

对比版本性能并生成建议。

分别评估 → 计算差异 → 权衡分析 → 生成推荐建议。

基于A/B测试思想，在相同测试集上比较两个版本的性能指标，计算改进幅度并权衡准确率和速度。

场景4：知识库版本管理与性能比较

7. 回归测试模块

```
def generate_regression_test(self, version_name, test_queries):
    # 执行测试用例
    for query_info in test_queries:
        retrieved_chunks = self.retrieve_relevant_chunks(query, version_name)
        is_passed = self.evaluate_retrieval_quality(query, retrieved_chunks,
expected_answer)

        if is_passed:
            passed_tests += 1

    # 计算通过率
    pass_rate = passed_tests / total_tests
```

确保版本更新不破坏原有功能。

执行测试用例 → 验证检索质量 → 统计通过情况 → 计算通过率。

基于回归测试理论，使用历史测试用例验证新版本是否保持原有功能

通过率=通过测试数/总测试数。

场景4：知识库版本管理与性能比较

```
# 创建版本1（基础版本）
  knowledge_base_v1 = [
    {"id": "kb_001", "content": "上海迪士尼乐园位于上海市浦东新区，是中国大陆首座迪士尼主题乐园，于2016年6月16日开园。"},
    {"id": "kb_002", "content": "上海迪士尼乐园的门票价格：平日成人票价为399元，周末和节假日为499元。"},
    {"id": "kb_003", "content": "上海迪士尼乐园营业时间为上午8:00至晚上8:00。"}
  ]
```

```
# 创建版本2（增强版本）
  knowledge_base_v2 = [
    {"id": "kb_001", "content": "上海迪士尼乐园位于上海市浦东新区，是中国大陆首座迪士尼主题乐园，于2016年6月16日开园。乐园占地面积390公顷，包含七大主题园区。"},
    {"id": "kb_002", "content": "上海迪士尼乐园的门票价格：平日成人票价为399元，周末和节假日为499元。儿童票（1.0-1.4米）平日为299元，周末为374元。1.0米以下儿童免费。"},
    {"id": "kb_003", "content": "上海迪士尼乐园营业时间为上午8:00至晚上8:00，全年无休。建议出发前查看官方网站确认具体时间。"},
    {"id": "kb_004", "content": "从上海市区到迪士尼乐园可以乘坐地铁11号线到迪士尼站，或乘坐迪士尼专线巴士。"},
    {"id": "kb_005", "content": "上海迪士尼乐园的特色项目包括：创极速光轮、七个小矮人矿山车、加勒比海盗等。"}
  ]
```

场景4：知识库版本管理与性能比较

功能1: 创建知识库版本

版本1信息:

版本名: v1.0

描述: 基础版本

知识切片数量: 3

平均切片长度: 36字符

版本2信息:

版本名: v2.0

描述: 增强版本

知识切片数量: 5

平均切片长度: 54字符

功能2: 版本差异比较

版本比较结果:

新增知识切片: 2个

删除知识切片: 0个

修改知识切片: 3个

新增的知识切片:

1. ID: kb_004

内容: 从上海市区到迪士尼乐园可以乘坐地铁11号线到迪士尼站，或乘坐迪士尼专线巴士。

2. ID: kb_005

内容: 上海迪士尼乐园的特色项目包括：创极速光轮、七个小矮人矿山车、加勒比海盗等。

场景4：知识库版本管理与性能比较

修改的知识切片:

1. ID: kb_003

旧内容: 上海迪士尼乐园营业时间为上午8:00至晚上8:00。

新内容: 上海迪士尼乐园营业时间为上午8:00至晚上8:00，全年无休。建议出发前查看官方网站确认具体时间。

2. ID: kb_002

旧内容: 上海迪士尼乐园的门票价格：平日成人票价为399元，周末和节假日为499元。

新内容: 上海迪士尼乐园的门票价格：平日成人票价为399元，周末和节假日为499元。儿童票（1.0-1.4米）平日为299元，周末为374元。1.0米以下儿童免费。

3. ID: kb_001

旧内容: 上海迪士尼乐园位于上海市浦东新区，是中国大陆首座迪士尼主题乐园，于2016年6月16日开园。

新内容: 上海迪士尼乐园位于上海市浦东新区，是中国大陆首座迪士尼主题乐园，于2016年6月16日开园。乐园占地面积390公顷，包 含七大主题园区。

场景4：知识库版本管理与性能比较

功能3: 版本性能评估

版本1性能:

准确率: 60.0%

平均响应时间: 115.8ms

版本2性能:

准确率: 100.0%

平均响应时间: 120.2ms

功能4: 性能比较与建议

性能比较结果:

准确率提升: 40.0%

响应时间变化: 21.6ms

建议: 推荐使用版本2，准确率提升40.0%，响应时间提升

功能5: 回归测试

回归测试结果:

测试通过率: 100.0%

测试用例数量: 5

详细测试结果:

1. 上海迪士尼乐园在哪里？ ✓
2. 门票多少钱？ ✓
3. 营业时间是什么？ ✓
4. 怎么去迪士尼？ ✓
5. 有什么好玩的项目？ ✓

Summary

- 知识库问题生成与检索优化

为知识库内容生成多样化问题，使用BM25算法进行文本检索优化

LLM作用：生成多样化问题，评估检索质量，优化知识库结构

- 对话知识沉淀

从用户对话中提取有价值的知识点，进行智能合并和分类

LLM作用：提取对话中的知识点，智能合并相似知识，生成结构化知识内容

- 知识库健康度检查

检查知识库的完整性、时效性和一致性，识别知识空白和冲突

LLM作用：分析知识缺失、检查过时内容、识别知识冲突，生成健康度报告

- 知识库版本管理与性能比较

Embedding作用：构建向量索引，进行语义检索，支持版本间的性能评估

版本差异检测使用精确文本匹配（简化版，没有使用到LLM）

打卡：知识库处理

在你的RAG系统中，如何打造更强大的知识库：

- 知识库问题生成与检索优化
- 对话知识沉淀
- 知识库健康度检查
- 知识库版本管理与性能比较

结合你的业务场景，或者以迪士尼RAG助手为例，选择其中一个场景进行实践

精准雷达：高效召回

CASE：DeepSeek + Faiss搭建本地知识库检索

上海浦东发展银行西安分行

个金客户经理管理考核暂行办法

第一章 总 则

第一条 为保证我分行个金客户经理制的顺利实施，有效调动个金客户经理的积极性，促进个金业务快速、稳定地发展，根据总行《上海浦东发展银行个人金融营销体系建设方案（试行）》要求，特制定《上海浦东发展银行西安分行个金客户经理管理考核暂行办法（试行）》（以下简称本办法）。

第二条 个金客户经理系指各支行（营业部）从事个人金融产品营销与市场开拓，为我行个人客户提供综合银行服务的我行市场人员。

第三条 考核内容分为二大类，即个人业绩考核、工作质量考核。个人业绩包括个人资产业务、负债业务、卡业务。工作质量指个人业务的资产质量。

第四条 为规范激励规则，客户经理的技术职务和薪资实行每年考核浮动。客户经理的奖金实行每季度考核浮动，即客户经理按其考核内容得分与行员等级结合，享受对应的行员等级待遇。

客户经理被投诉了，投诉一次扣多少分

根据文件内容，客户经理被投诉一次扣2分。具体规定如下：

（二）服务质量考核：

2. 客户服务效率低，态度生硬或不及时为客户提供维护服务，有客户投诉的，每投诉一次扣2分。

客户经理每年评聘申报时间是怎样的?

根据文件内容，客户经理每年评聘申报时间如下：

第十一条 每年一月份为客户经理评聘的申报时间，由分行人力资源部、个人业务部每年二月份组织统一的资格考试。考试合格者由分行颁发个金客户经理资格证书，其有效期为一年。

因此，客户经理每年评聘的申报时间是一月份。

CASE：DeepSeek + Faiss搭建本地知识库检索

```
from langchain_community.embeddings import DashScopeEmbeddings
 # 创建嵌入模型
    embeddings = DashScopeEmbeddings(
        model="text-embedding-v1",
        dashscope_api_key=DASHSCOPE_API_KEY,
    )

 # 从文本块创建知识库
knowledgeBase = FAISS.from_texts(chunks, embeddings)
```

```
from langchain_community.llms import Tongyi
llm = Tongyi(model_name="deepseek-v3", dashscope_api_key=DASHSCOPE_API_KEY)
```

RAG高效召回方法

Thinking：都有哪些RAG召回的策略，提升召回的质量？

1. 如果要召回更多的片段，如何设置？

```
docs = knowledgeBase.similarity_search(query, k=10)
```

RAG高效召回方法（优化查询扩展）

2. 优化查询扩展

相似语义改写：使用大模型将用户查询改写成多个语义相近的查询，提升召回多样性。在 LangChain旧版本中提供了MultiQueryRetriever支持多查询召回，新版本需要自己编写

```
def generate_multi_queries(query: str, llm, num_queries: int = 3) -> List[str]:
    """
    使用LLM生成多个查询变体
    """
    prompt = f"""你是一个AI助手，负责生成多个不同视角的搜索查询。
给定一个用户问题，生成{num_queries}个不同但相关的查询，以帮助检索更全面的信息。
每个查询应该从不同角度表达相同的信息需求。
原始问题: {query}
请直接输出{num_queries}个查询，每行一个，不要编号和其他内容:"""
    ...
```

参考：2-chatpdf-faiss-MultiQueryRetriever.py

RAG高效召回方法（优化查询扩展）

TO DO：编写 2-chatpdf-faiss-MultiQueryRetriever.py

LangChain新版需要自己实现 multi_query_search 支持多查询召回，再进行回答问题

查询: 客户经理被投诉了，投诉一次扣多少分

生成的查询变体: ['客户经理被投诉了，投诉一次扣多少分', '客户经理投诉扣分标准是什么', '银行客户经理被投诉一次会扣除多少绩效分', '金融机构客户经理投诉处罚机制']

找到 4 个相关文档

回答:

根据提供的《上海浦东发展银行西安分行个金客户经理管理考核暂行办法》内容，客户经理被投诉的扣分标准如下：

在**第五章 工作质量考核标准**的第九条中，“（一）服务质量考核”第2点明确规定：

> **2、客户服务效率低，态度生硬或不及时为客户提供维护服务，有客户投诉的，每投诉一次扣 2分。**

因此，客户经理被投诉一次扣2分。

来源页码:

- 第 1 页
- 第 6 页
- 第 8 页

RAG高效召回方法（索引扩展）

3. 索引扩展

1）离散索引扩展：使用关键词抽取、实体识别等技术生成离散索引，与向量检索互补，提升召回准确性。

2）连续索引扩展：结合多种向量模型（如OpenAI的Ada、智源的BGE）进行多路召回，取长补短。

3）混合索引召回：将BM25等离散索引与向量索引结合，通过Ensemble Retriever实现混合召回，提升召回多样性

BM25是一种经典的文本检索算法，它是TF-IDF（词频-逆文档频率）的改进版本，通过更精细的词频饱和度和文档长度归一化，提升了检索的相关性排序效果。

$$\text{BM25}(Q, D) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \text{TF}_{\text{BM25}}(q_i, D)$$

$$\text{TF}_{\text{BM25}} = \frac{(k_1 + 1) \cdot \text{tf}}{k_1 \cdot (1 - b + b \cdot \frac{\text{dl}}{\text{avgdl}}) + \text{tf}}$$

$$\text{IDF} = \log\left(\frac{N - n + 0.5}{n + 0.5} + 1\right)$$

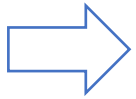
RAG高效召回方法（索引扩展）

1）离散索引扩展：使用关键词抽取、实体识别等技术生成离散索引，与向量检索互补，提升召回准确性。

关键词抽取：从文档中提取出重要的关键词，作为离散索引的一部分，用于补充向量检索的不足。

文档内容

本文介绍了深度学习模型训练中的优化技巧，包括：
1. 使用 AdamW 优化器替代传统的 SGD。
2. 采用混合精度训练，减少显存占用。
3. 使用分布式训练技术加速大规模模型的训练。



通过关键词抽取技术（如 TF-IDF、TextRank）提取出以下关键词：

["深度学习", "模型训练", "优化技巧", "AdamW", "混合精度训练", "分布式训练"]

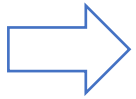
当用户查询“如何优化深度学习模型训练？”时，离散索引中的关键词能够快速匹配到相关文档。

RAG高效召回方法（索引扩展）

实体识别：从文档中识别出命名实体（如人名、地点、组织等），作为离散索引的一部分，增强检索的精确性。

文档内容

2023年诺贝尔物理学奖授予了三位科学家，以表彰他们在量子纠缠领域的研究成果。



通过实体识别技术（如 SpaCy、BERT-based NER）提取出以下实体：

["2023年", "诺贝尔物理学奖", "量子纠缠"]

当用户查询“2023年诺贝尔物理学奖的获奖者是谁？”时，离散索引中的实体能够快速匹配到相关文档。

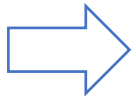
RAG高效召回方法（索引扩展）

混合索引召回：将离散索引（如关键词、实体）与向量索引结合，通过混合召回策略提升检索效果。

文档内容

本文介绍了人工智能在医疗领域的应用，包括：

1. 使用深度学习技术进行医学影像分析。
2. 利用自然语言处理技术提取电子病历中的关键信息。
3. 开发智能诊断系统辅助医生决策。



关键词抽取：["人工智能", "医疗领域", "深度学习", "医学影像分析", "自然语言处理", "电子病历", "智能诊断系统"]

实体识别：["人工智能", "医疗领域", "深度学习", "自然语言处理"]

当用户查询“人工智能在医疗领域的应用有哪些？”时：

离散索引通过关键词和实体匹配到相关文档。

向量索引通过语义相似度匹配到相关文档。

综合两种召回结果，提升检索的准确性和覆盖率。

RAG高效召回方法（索引扩展）

TO DO：编写 3-chatpdf-faiss-HybridSearch.py

实现 BM25检索 + 向量检索

BM25 检索

- 原理: 词频/逆文档频率
- 优势: 精确匹配专有名词
- 劣势: 无法理解语义
- 适合: "投诉扣分" 精确查找

向量检索

- 原理: Embedding 语义相似度
- 优势: 理解同义词、语义
- 劣势: 可能漏掉精确关键词
- 适合: "处罚规定" = "扣分标准"

RAG高效召回方法（索引扩展）

Step1 输入: 用户查询进入系统

Step2 并行检索:

同时执行 BM25（分词 -> 词频匹配）和 向量检索（Embedding -> 相似度计算）

Step3 归一化:

将两种分数统一缩放到 [0, 1] 区间，便于融合

Step4 加权融合:

按 alpha 权重合并分数，Score = α × Vector + (1-α) × BM25

Step5 输出:

按融合分数排序，返回 Top-K 结果

3-chatpdf-faiss-HybridSearch.py

RAG高效召回方法（索引扩展）

```
# BM25检索

# 中文分词

tokenized_query = tokenize_chinese(query)

# ["客户", "经理", "投诉", "扣", "分"]



# 计算 BM25 分数

bm25_scores = self.bm25.get_scores(tokenized_query)



# 归一化到 [0, 1]

max_bm25 = max(bm25_scores) if max(bm25_scores) > 0 else 1

bm25_scores_normalized = [s / max_bm25 for s in bm25_scores]
```

```
# 向量检索

# 获取所有文档的相似度距离

vector_results = self.vectorstore.similarity_search_with_score(

    query, k=len(self.chunks)

)



# 距离转分数 (距离越小 -> 分数越高)

for doc, distance in vector_results:

    vector_scores[idx] = 1 - (distance / max_distance)
```

3-chatpdf-faiss-HybridSearch.py

RAG高效召回方法（索引扩展）

```
# 分数融合

# alpha 控制两种方法的权重

# alpha = 0.5 表示各占 50%

combined = alpha * vector_score + (1 - alpha) * bm25_score
```

$$\text{Score}_{\text{hybrid}} = \alpha \times \text{Score}_{\text{vector}} + (1 - \alpha) \times \text{Score}_{\text{BM25}}$$

调参建议:

- 专业术语多的文档 -> 降低 alpha (偏 BM25)
- 用户问题口语化 -> 提高 alpha (偏向量)
- 不确定时 -> 从 α = 0.5 开始，根据效果微调

Alpha 值	检索倾向	适用场景
α = 0.0	纯 BM25，只看关键词	专业术语查询、精确匹配需求
α = 0.3	偏向关键词，兼顾语义	技术文档、法规条文检索
α = 0.5	平衡，各占一半	通用场景（推荐默认值）
α = 0.7	偏向语义，兼顾关键词	口语化问答、模糊查询
α = 1.0	纯向量，只看语义	同义词丰富、表述多样的场景

3-chatpdf-faiss-HybridSearch.py

Rerank模型使用

Thinking：什么是重排序 Rerank？

重排序Rerank主要用于优化初步检索结果的排序，提高最终输出的相关性或准确性。BGE-Rerank和Cohere Rerank是两种广泛使用的重排序模型，它们在检索增强生成（RAG）系统、搜索引擎优化和问答系统中表现优异。

1. BGE-Rerank

由北京智源人工智能研究院（BAAI）开源发布，属于FlagEmbedding项目的一部分。

基于Transformer的Cross-Encoder结构，直接计算查询（Query）与文档（Document）的交互相关性得分。

训练数据：支持多语言（中、英等），训练数据包括T2Ranking、MSMARCO、NLI等数据集。

提供bge-reranker-base和bge-reranker-large两个版本，后者在精度上更优。

部署方式：可本地部署

开源免费，适合本地化部署，保护数据隐私。

在中文任务中表现优秀，适用于垂直领域优化

Rerank模型使用

```
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer


tokenizer = AutoTokenizer.from_pretrained('BAAI/bge-reranker-large')
model = AutoModelForSequenceClassification.from_pretrained('BAAI/bge-reranker-large')
model.eval()


pairs = [['what is panda?', 'The giant panda is a bear species endemic to China.']]
inputs = tokenizer(pairs, padding=True, truncation=True, return_tensors='pt')
scores = model(**inputs).logits.view(-1).float()
print(scores)  # 输出相关性分数 4.9538
```

在BGE-Rerank模型中，相关性分数scores是一个未归一化的对数几率（logits）值，范围没有固定的上限或下限（不像某些模型限制在0-1）。不过BGE-Rerank的分数通常落在以下范围：
高相关性： 3.0~10.0
中等相关性：0.0~3.0
低相关性/不相关：负数（如-5.0以下）

bge-reranker.py

Rerank模型使用

2. Cohere Rerank

由Cohere公司提供的商业API服务。

基于专有的深度学习模型，支持多语言（如rerank-multilingual-v3.0）。

训练数据：优化了语义匹配，特别适用于混合检索（如结合BM25和向量检索）后的结果优化。

使用方式：通过API调用，集成到LangChain、LlamaIndex等框架中。

优势：

- 简单易用，适合快速集成到现有系统。
- 在英文和多语言任务中表现优异，如提升Hit Rate（命中率）和MRR（平均倒数排名）。

Rerank模型使用

```
import cohere
co = cohere.Client(api_key="YOUR_API_KEY")
query = "What is the capital of France?"
docs = ["Paris is the capital of France.", "Berlin is the capital of Germany."]
results = co.rerank(query=query, documents=docs, top_n=2, model='rerank-multilingual-v3.0')
print(results)  #Cohere Rerank的API返回的是归一化后的相关性分数（如0-1），更易解释。
```

特性	BGE-Rerank	Cohere Rerank
开源/商业	开源	商业API
部署方式	可本地部署	云端调用
多语言支持	中英优化	多语言（v3.0）
适用场景	数据敏感、垂直领域	快速集成、多语言优化

cohere-reranker.py

Rerank模型使用

TO DO：编写 4-chatpdf-faiss-HybridSearch-Rerank.py

通过modelscope获取以下rerank模型：

BAAI/bge-reranker-base (轻量级，推荐)

BAAI/bge-reranker-large (效果更好，但更慢)

在原来 3-chatpdf-faiss-HybridSearch.py 的基础上，添加rerank

Rerank模型使用

```
class Reranker:
    def __init__(self, model_name="BAAI/bge-reranker-base", cache_dir="./models"):
        # 从 ModelScope 下载模型到指定目录
        model_dir = snapshot_download(model_name, cache_dir=cache_dir)

        # 加载 tokenizer 和模型
        self.tokenizer = AutoTokenizer.from_pretrained(model_dir)
        self.model = AutoModelForSequenceClassification.from_pretrained(model_dir)
        self.model.eval()

        # 自动选择 GPU/CPU
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.model.to(self.device)
```

4-chatpdf-faiss-HybridSearch-Rerank

Rerank模型使用

```
def rerank(self, query: str, documents: List[Document], top_k: int = None):
    # 1. 构建 [Query, Doc] 对
    pairs = [[query, doc.page_content] for doc in documents]
    # 2. Tokenize 并送入模型
    with torch.no_grad():
        inputs = self.tokenizer(
            pairs,
            padding=True,
            truncation=True,
            max_length=512,       # 限制长度
            return_tensors="pt"
        ).to(self.device)

        # 3. 模型输出相关性分数
        scores = self.model(**inputs).logits.squeeze(-1).cpu().tolist()
    # 4. 按分数排序， 返回 Top-K
    scored_docs = sorted(zip(documents, scores), key=lambda x: x[1], reverse=True)
    return [doc for doc, _ in scored_docs[:top_k]]
```

4-chatpdf-faiss-HybridSearch-Rerank

Rerank模型使用

```
def hybrid_multi_query_search_with_rerank(query, hybrid_retriever,
reranker, llm,  initial_k=10, final_k=4):
    # Stage 1: 粗排召回
    queries = generate_multi_queries(query, llm)  # 生成查询变体

    candidate_docs = []
    for q in queries:
        docs = hybrid_retriever.search(q, k=initial_k)  # 混合检索
        candidate_docs.extend(docs)

    candidate_docs = deduplicate(candidate_docs)  # 去重
    print(f"初步召回 {len(candidate_docs)} 个候选")
```

```
 # Stage 2: 精排重排序
    reranked_docs = reranker.rerank(query, candidate_docs,
top_k=final_k)
    print(f"Rerank 后保留 {len(reranked_docs)} 个")


    return reranked_docs
```

4-chatpdf-faiss-HybridSearch-Rerank

Summary（Rerank模型使用）

- Rerank 模型对每个 (Query, Doc) 对都要过一次，候选数量不宜过多
- 首次运行会从 ModelScope 下载模型，建议设置 cache_dir 到本地目录
- GPU 加速效果明显，建议在有 GPU 的环境运行
- 长文档会被截断到 max_length，可考虑对长文档分段处理
- 参数设置建议

参数	说明	建议值
initial_k	粗排召回数量	10-20 (召回越多，Rerank 精度越高，但速度越慢)
final_k	最终返回数量	3-5 (太多会引入噪声，太少可能漏掉)
max_length	Tokenizer 最大长度	512 (默认值，过长会截断)
alpha	混合检索权重	0.5 (平衡 BM25 和向量)

RAG高效召回方法（双向改写）

双向改写：将查询改写成文档（Query2Doc）或为文档生成查询（Doc2Query），缓解短文本向量化效果差的问题

Query2Doc：将查询改写成文档

用户查询：“如何提高深度学习模型的训练效率？”

Query2Doc 改写：

原始查询较短，可能无法充分表达用户意图。

通过 Query2Doc 生成一段扩展文档：

提高深度学习模型的训练效率可以从以下几个方面入手：

1. 使用更高效的优化算法，如 AdamW 或 LAMB。
2. 采用混合精度训练（Mixed Precision Training），减少显存占用并加速计算。
3. 使用分布式训练技术，如数据并行或模型并行。
4. 对数据进行预处理和增强，减少训练时的冗余计算。
5. 调整学习率调度策略，避免训练过程中的震荡。

Doc2Query：为文档生成关联查询

文档内容：

本文介绍了深度学习模型训练中的优化技巧，包括：

1. 使用 AdamW 优化器替代传统的 SGD。
2. 采用混合精度训练，减少显存占用。
3. 使用分布式训练技术加速大规模模型的训练……

通过 Doc2Query 生成一组可能的查询：

1. 如何选择深度学习模型的优化器？
2. 混合精度训练有哪些优势？
3. 分布式训练技术如何加速深度学习？
4. 如何减少深度学习训练中的显存占用？
5. 深度学习模型训练的最佳实践是什么？

RAG高效召回方法（Small-to-Big）

Small-to-Big 索引策略：

一种高效的检索方法，特别适用于处理长文档或多文档场景。核心思想是通过小规模内容（如摘要、关键句或段落）建立索引，并链接到大规模内容主体中。这种策略的优势在于能够快速定位相关的小规模内容，并通过链接获取更详细的上下文信息，从而提高检索效率和答案的逻辑连贯性。

小规模内容（索引部分）：

摘要：从每篇论文中提取摘要作为索引内容。

> 摘要1：本文介绍了 Transformer 模型在机器翻译任务中的应用，并提出了改进的注意力机制。
>
> 摘要2：本文探讨了 Transformer 模型在文本生成任务中的性能，并与 RNN 模型进行了对比。

关键句：从论文中提取与查询相关的关键句。

> 关键句1：Transformer 模型通过自注意力机制实现了高效的并行计算。
>
> 关键句2：BERT 是基于 Transformer 的预训练模型，在多项 NLP 任务中取得了显著效果。

RAG高效召回方法（Small-to-Big）

大规模内容（链接部分）：

每篇论文的完整内容作为大规模内容，通过链接与小规模内容关联。

论文1：链接到完整的 PDF 文档，包含详细的实验和结果。
论文2：链接到完整的 PDF 文档，包含模型架构和性能分析。

Small-to-Big机制：

小规模内容检索：用户输入查询后，系统首先在小规模内容（如摘要、关键句或段落）中检索匹配的内容。小规模内容通常是通过摘要生成、关键句提取等技术从大规模内容中提取的，并建立索引。

链接到大规模内容：当小规模内容匹配到用户的查询后，系统会通过预定义的链接（如文档 ID、URL 或指针）找到对应的大规模内容（如完整的文档、文章）。大规模内容包含更详细的上下文信息，为 RAG 提供丰富的背景知识。

上下文补充：将大规模内容作为 RAG 系统的上下文输入，结合用户查询和小规模内容，生成更准确和连贯的答案。

Summary（RAG高效召回方法）

1. 优化查询扩展（相似语义改写）：使用大模型将用户查询改写成多个语义相近的查询，提升召回多样性

2. 混合检索：结合向量检索和关键词检索的优势，通过重排序模型对结果进行归一化处理，提升召回质量

3. 引入重排序（Reranking）

重排序模型：对召回结果进行重排，提升问题和文档的相关性。常见的重排序模型有BGE-Rerank和Cohere Rerank。

> 场景：用户查询“如何提高深度学习模型的训练效率？”
> 召回结果：初步召回10篇文档，其中包含与“深度学习”、“训练效率”相关的文章。
> 重排序：BGE-Rerank对召回的10篇文档进行重新排序，将与“训练效率”最相关的文档（如“优化深度学习训练的技巧”）排在最前面，而将相关性较低的文档（如“深度学习基础理论”）排在后面。

4. 改进检索算法

知识图谱：利用知识图谱中的语义信息和实体关系，增强对查询和文档的理解，提升召回的相关性

打卡：高效召回

在你的RAG系统中，如何打造高效召回的系统：

- 优化查询扩展（相似语义改写）： multi_query_search
- 混合检索：Hybrid Search: BM25 + Vector
- 重排序：使用 bge-reranker

结合你的业务场景，或者以《个金客户经理考核办法》为例，选择其中一个高效召回的方法进行实践

全局视野：GraphRAG

GraphRAG

GraphRAG：

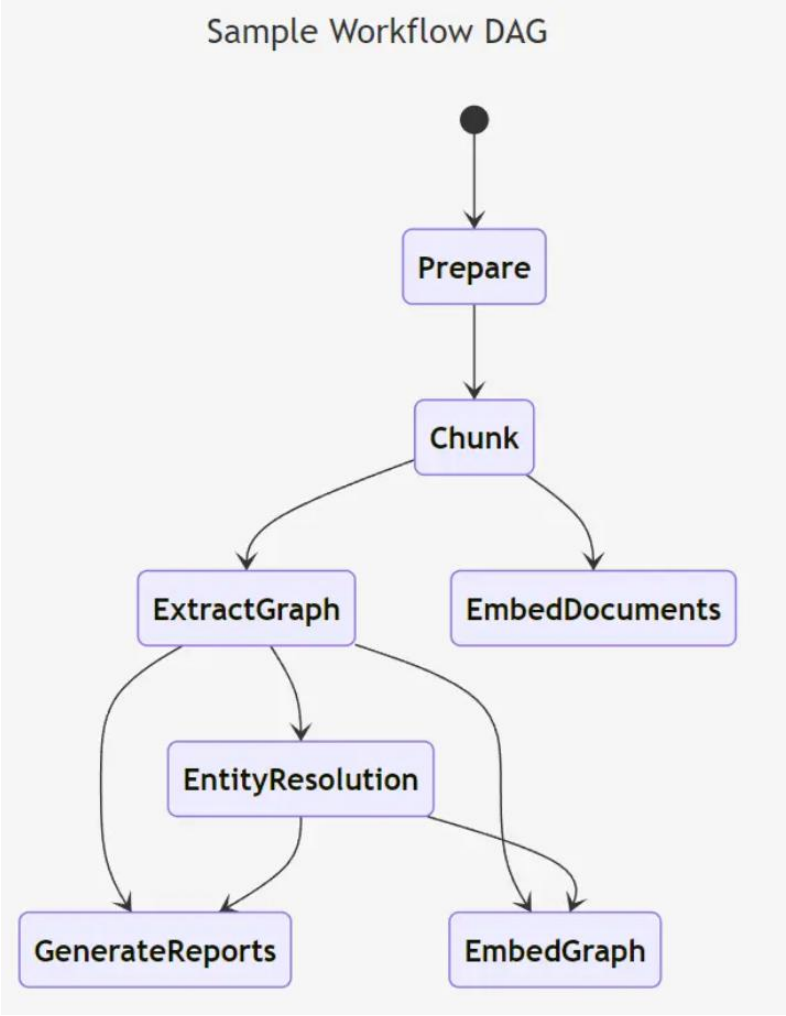
- 是一种结构化的、分层的检索增强生成（RAG）方法，而不是使用纯文本片段的语义搜索方法。
- GraphRAG 过程包括：

原始文本中提取出知识图谱

构建社区层级(这种结构通常用来描述个体、群体及它们之间的关系，帮助理解信息如何在社区内部传播、知识如何共享以及权力和影响力如何分布)

为这些社区层级生成摘要。

然后在执行基于 RAG 的任务时，会利用这些结构。



GraphRAG工作流，DAG（有向无环）

GraphRAG

GraphRAG 与 基线RAG：

大多数 RAG 使用矢量相似性作为搜索技术，称之为 基线 RAG

GraphRAG 使用知识图谱来在处理复杂信息时提供问题和回答性能的显著改进。

在某些情况下，基线 RAG 的性能非常差：

- 基线 RAG 难以连接各个要点。这种情况发生在回答问题需要通过共享属性遍历不同的信息片段，以提供新的综合见解。
- 基线 RAG 在被要求全面理解大量的数据（跨文档）或甚至单个大文档的的语义概念时表现不佳。

GraphRAG方法：

- 使用 LLMs 来创建基于输入语料库的知识图谱。

这个知识图谱、社区层级摘要、以及知识图谱机器学习输出会在用户查询时用于增强提示。

- GraphRAG 在回答上述两类问题时可以显著改进回答能力，远超基线RAG

传统 RAG的局限性:

连接点缺失,问题的答案分散在文档的不同位置，且没有直接的语义重叠，传统 RAG 很难把它们串联起来

宏观理解缺失。 如果用户问“这篇文章的主旨是什么？” 传统 RAG 很难给出一个概括性的答案，

GraphRAG

Query：19世纪的艺术运动是如何影响20世纪现代艺术的发展的？

LLM：19世纪的艺术运动通过鼓励对色彩、形式和主题的实验影响了20世纪的现代艺术。这些运动为抽象主义、表现主义和其他创新艺术铺平了道路。

RAG检索：1. 像克劳德·莫奈这样的印象派艺术家引入了新技术，彻底改变了对光和颜色的描绘。2. 印象派的技法影响了后来的艺术运动。3. 巴勃罗·毕加索开创了立体主义，从根本上改变了视觉表现的方式。4. 立体主义出现在20世纪初，挑战了传统的艺术观点。

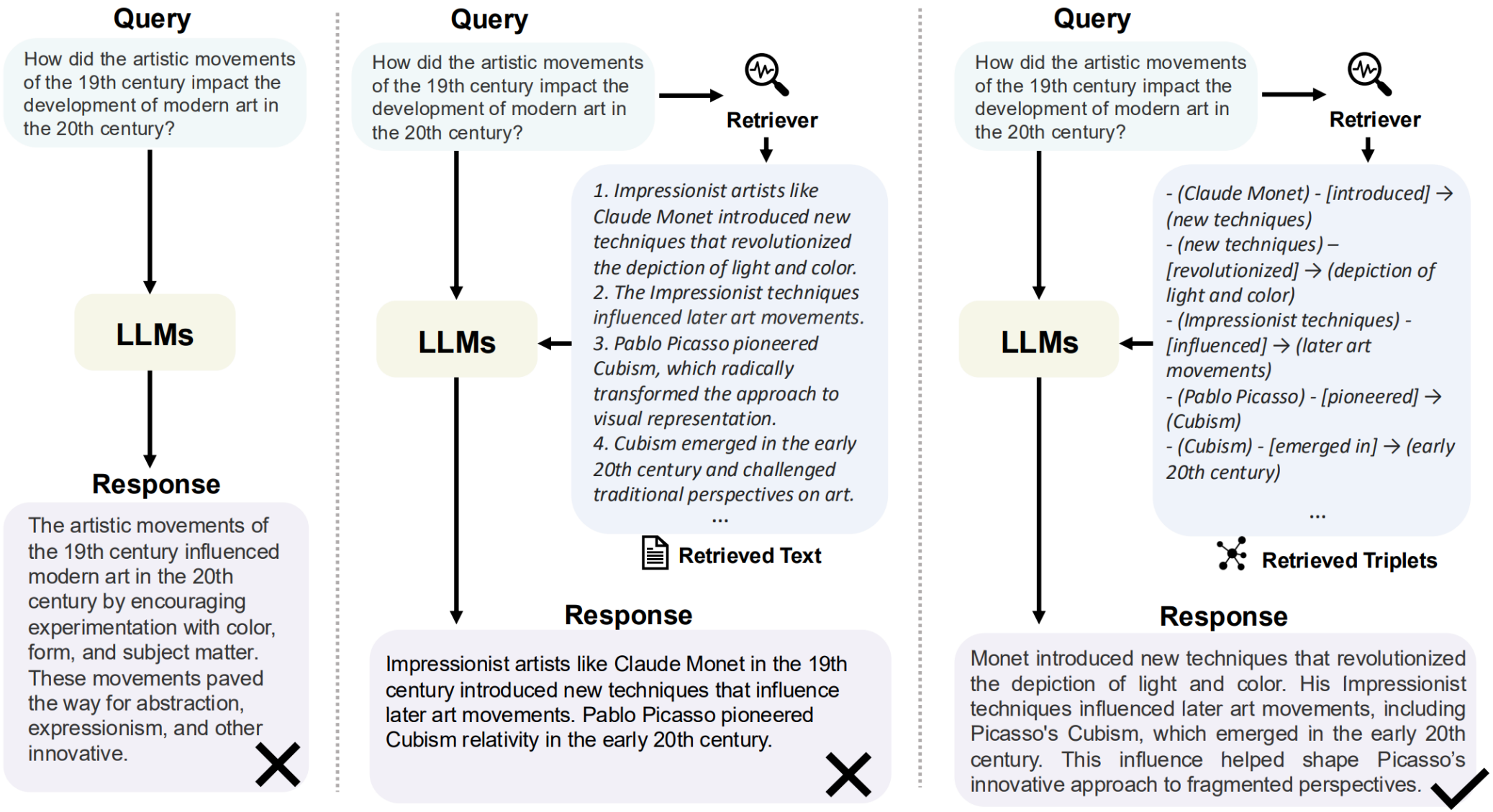
RAG回答：像19世纪的克劳德·莫奈这样的印象派艺术家引入了影响后来艺术运动的新技术。巴勃罗·毕加索在20世纪初开创了立体主义相对论。

GraphRAG检索：（莫奈）-[引进]→（新技术）-
（新技术）-[革新]→（光和颜色的描绘）
（印象派技术）-[影响]→（后来的艺术运动）
（毕加索）-[开创]→（立体主义）
（立体主义）-[出现]→（20世纪初）

GraphRAG回答：

莫奈引进的新技术彻底改变了对光和色彩的描绘。他的印象派技巧影响了后来的艺术运动，包括20世纪初出现的毕加索的立体主义。这种影响有助于塑造毕加索对碎片化视角的创新方法。

GraphRAG

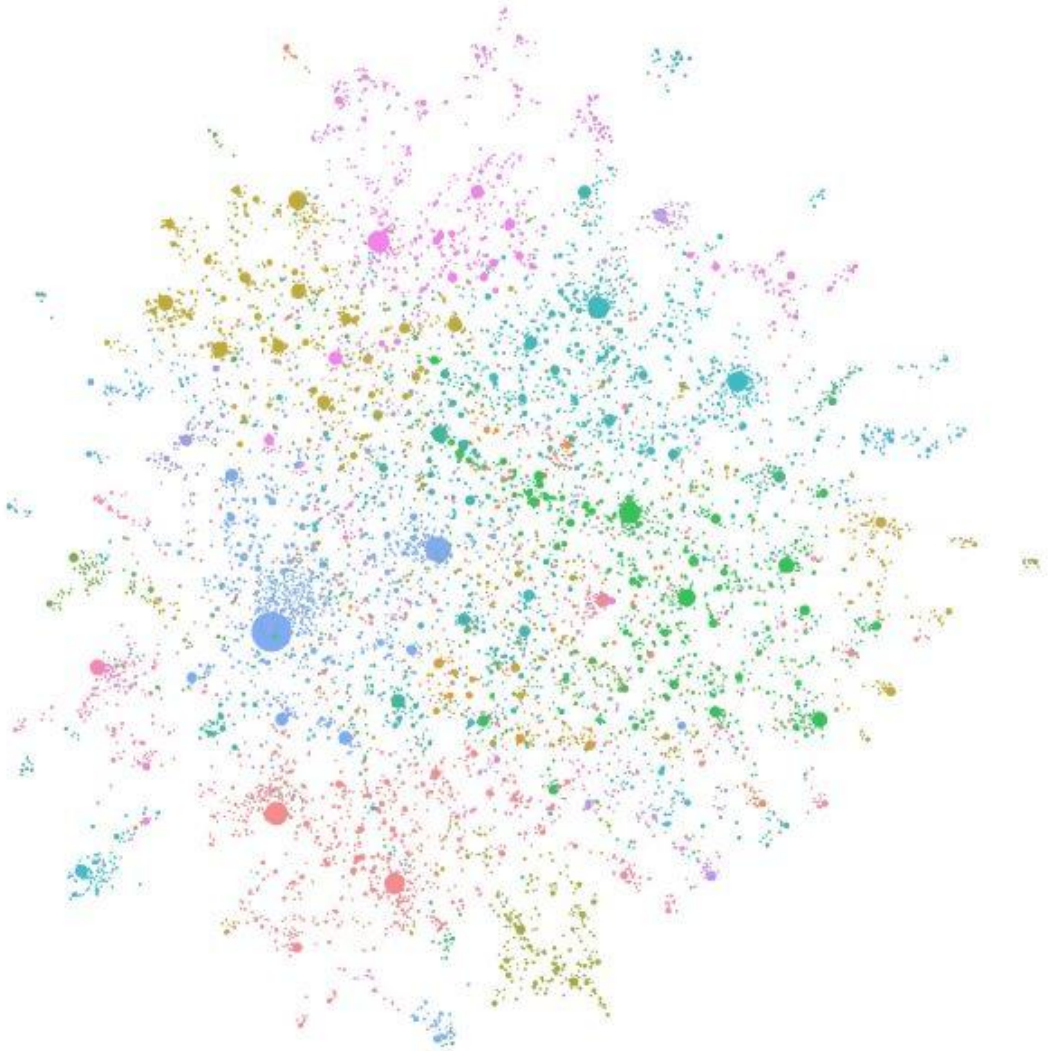

Query
How did the artistic movements of the 19th century impact the development of modern art in the 20th century?
LLMs
Response
The artistic movements of the 19th century influenced modern art in the 20th century by encouraging experimentation with color, form, and subject matter. These movements paved the way for abstraction, expressionism, and other innovative.
Query
How did the artistic movements of the 19th century impact the development of modern art in the 20th century?
Retriever
1. Impressionist artists like Claude Monet introduced new techniques that revolutionized the depiction of light and color.
2. The Impressionist techniques influenced later art movements.
3. Pablo Picasso pioneered Cubism, which radically transformed the approach to visual representation.
4. Cubism emerged in the early 20th century and challenged traditional perspectives on art.
...
Retrieved Text
LLMs
Response
Impressionist artists like Claude Monet in the 19th century introduced new techniques that influence later art movements. Pablo Picasso pioneered Cubism relativity in the early 20th century.
Query
How did the artistic movements of the 19th century impact the development of modern art in the 20th century?
Retriever
- (Claude Monet) - [introduced] → (new techniques)
- (new techniques) – [revolutionized] → (depiction of light and color)
- (Impressionist techniques) - [influenced] → (later art movements)
- (Pablo Picasso) - [pioneered] → (Cubism)
- (Cubism) - [emerged in] → (early 20th century)
...
Retrieved Triplets
LLMs
Response
Monet introduced new techniques that revolutionized the depiction of light and color. His Impressionist techniques influenced later art movements, including Picasso's Cubism, which emerged in the early 20th century. This influence helped shape Picasso's innovative approach to fragmented perspectives.

GraphRAG

GraphRAG 的基本步骤如下：

索引

- 将输入语料库分割为一系列的文本单元（TextUnits），这些单元作为处理以下步骤的可分析单元，并在我们的输出中提供细粒度的引用。
- 使用 LLM 从文本单元中提取所有实体、关系和关键声明。
- 使用 Leiden 技术对知识图谱进行层次聚类。每个圆圈都是一个实体（例如人、地点或组织），大小表示实体的度，颜色表示其社区层级。
- 自下而上地生成每个社区层级及其组成部分的摘要 => 有助于对数据集的整体理解。



使用 Leiden 技术对知识图谱进行层次聚类

GraphRAG

索引构建（Indexing）—— 知识的结构化

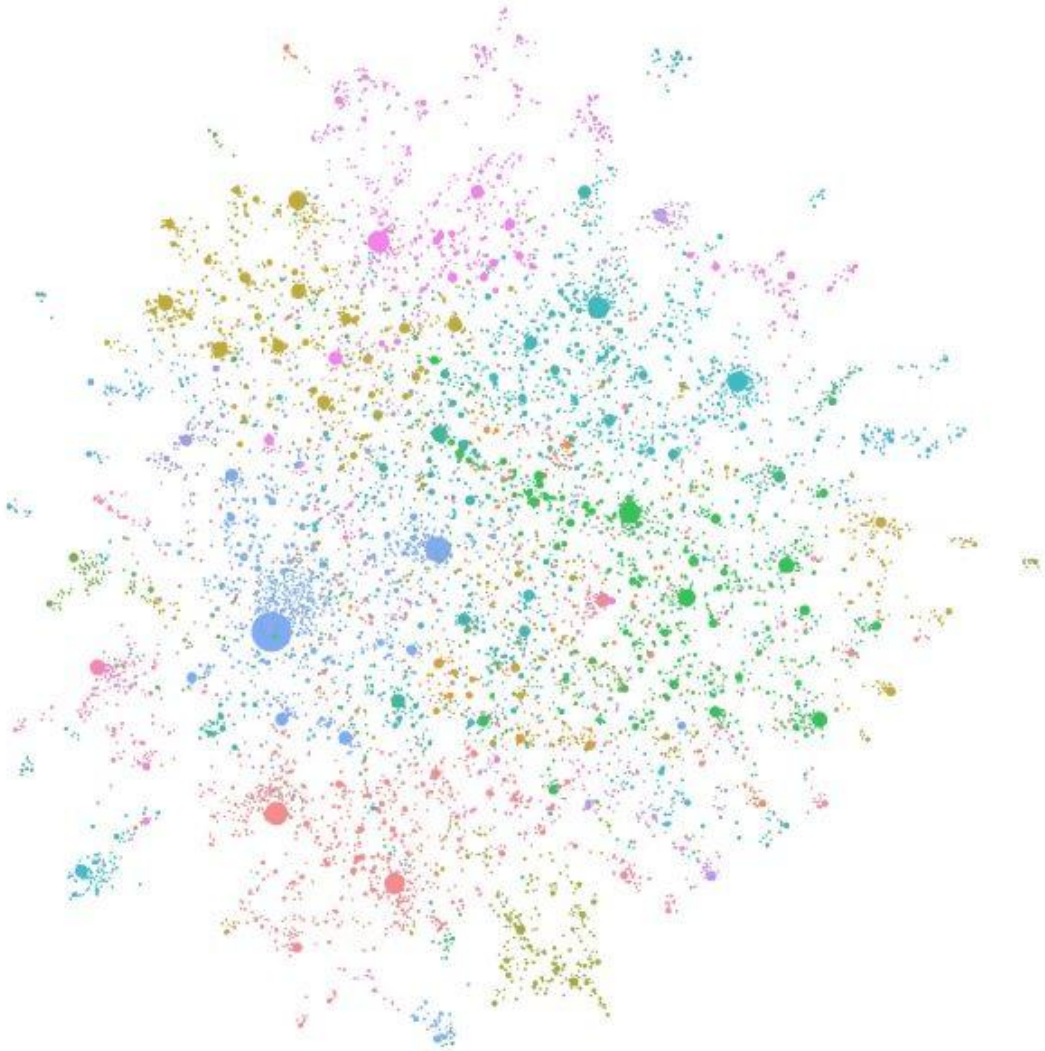
这是 GraphRAG 最“重”的一步，把非结构化文本 => 结构化

1. 切片（Source Text to TextUnits）： 把文档切分成文本块 。

2. 抽取（Extract Graph）： 使用 LLM 从文本块中提取实体（人、地、物）、关系（谁做了什么）和主张（Claim） 。

3. 聚类与摘要（Community Detection & Summarization）：

- 使用 Leiden 算法对图谱进行层级聚类，形成不同的社区。
- 自下而上地为每个社区生成摘要。这就像给原本零散的数据建立了一个从“村委会”到“市政府”再到“中央”的层级汇报体系 。



使用 Leiden 技术对知识图谱进行层次聚类

GraphRAG

查询：

- 在查询时，使用这些结构为 LLM 上下文窗口提供材料来回答问题。主要查询模式有：

全局搜索，通过社区层级摘要来推理有关语料库的整体问题。

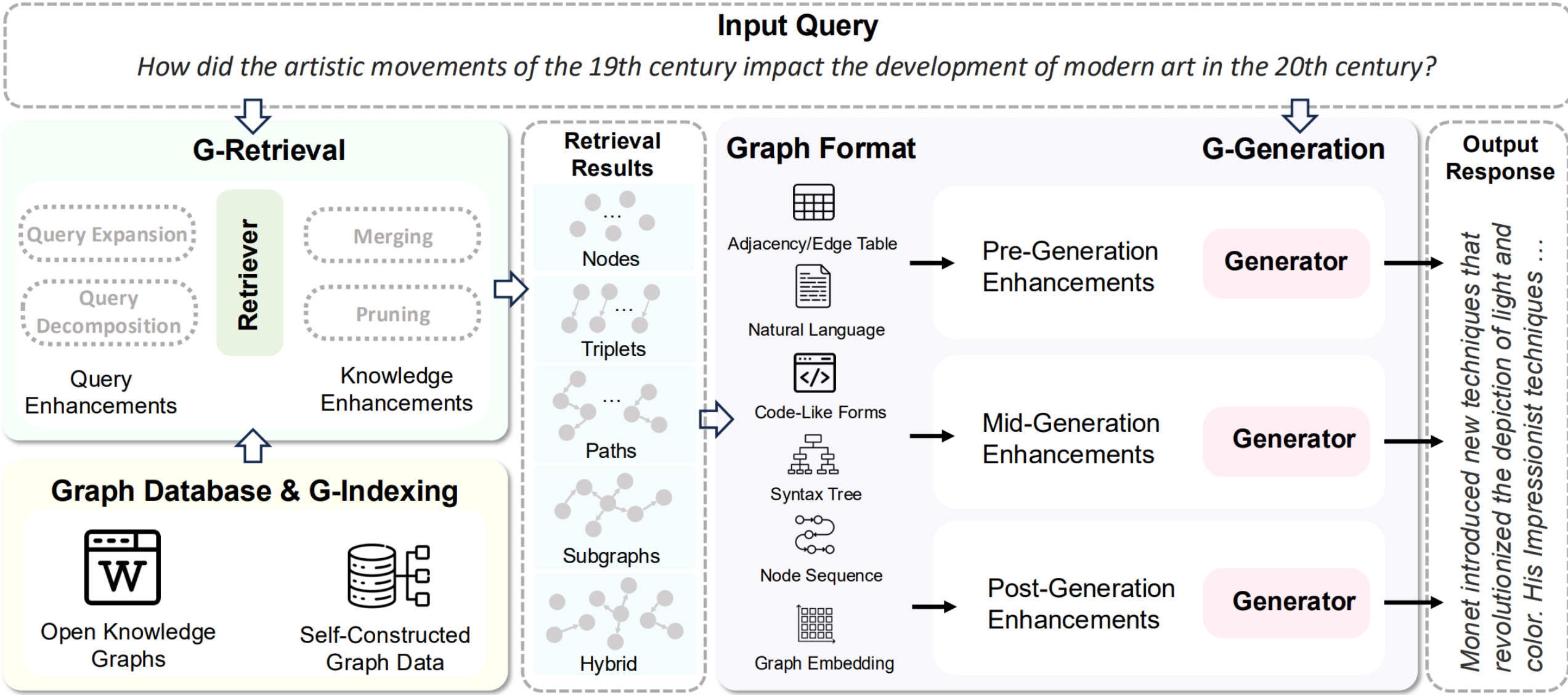
局部搜索，通过扩展到其邻居和相关概念来推理特定实体的情况。

GraphRAG方法是使用LLM构建基于图的文本索引，分两个阶段：

首先从源文档中派生出实体知识图谱

然后为所有密切相关的实体组预生成社区摘要。

GraphRAG


Input Query
How did the artistic movements of the 19th century impact the development of modern art in the 20th century?
G-Retrieval
Query Expansion
Query Decomposition
Retriever
Merging
Pruning
Query Enhancements
Knowledge Enhancements
Graph Database & G-Indexing
Open Knowledge Graphs
Self-Constructed Graph Data
Retrieval Results
Nodes
Triplets
Paths
Subgraphs
Hybrid
Graph Format
Adjacency/Edge Table
Natural Language
Code-Like Forms
Syntax Tree
Node Sequence
Graph Embedding
G-Generation
Pre-Generation Enhancements
Mid-Generation Enhancements
Post-Generation Enhancements
Generator
Generator
Generator
Output Response
Monet introduced new techniques that revolutionized the depiction of light and color. His Impressionist techniques ...

GraphRAG索引数据流

知识模型：

在GraphRAG的存储库中，包括实体类型如Document、TextUnit、Entity、Relationship、Covariate、Community Report和Node。

默认配置工作流程：将文本文档转换为知识图谱模型，主要步骤包括：

第一阶段：组合 TextUnits

将输入文档转换为TextUnits，用于知识图谱提取的文本块。

用户可以配置块大小和分组方式。

第二阶段：知识图谱提取

分析每个TextUnit，用来提取实体、关系和主张。

实体和关系在entity_extract动词中提取，而主张在claim_extract动词中提取。

实体和关系提取：

使用LLM从原始文本中提取实体和关系。

合并具有相同名称和类型的实体，以及具有相同源和目标的关系。

实体和关系概述：

通过询问LLM获取每个实体和关系的简要概述。

实体解析（默认未启用）：

解析表示相同现实世界实体，但具有不同名称的实体。

主张提取和发射：

从源TextUnits中提取主张，这些主张是正面事实陈述，并作为Covariates发射。

GraphRAG索引数据流

第三阶段：知识图谱增强

了解实体的社区结构，并增强知识图谱。

使用层次Leiden算法进行社区检测，使用Node2Vec算法进行知识图谱嵌入。

第四阶段：社区总结

生成社区报告，了解知识图谱在各个粒度级别上的高层次情况。

使用LLM生成每个社区的摘要。

第五阶段：文档处理

为知识模型创建“文档”表。

如果工作流在CSV数据上运行，可以配置工作流，用于向文档输出添加其他字段。

第六阶段：网络可视化

执行UMAP降维，用于在2D空间中可视化知识图谱。

UMAP嵌入作为“节点”表格发出。

GraphRAG工作流程是将文本数据转换为结构化的知识图谱，以便理解和分析数据。通过这个流程，用户可以提取关键信息，如实体、关系和主张，并在知识图谱中进行进一步的分析和可视化。

GraphRAG

GraphRAG方法：

https://github.com/microsoft/graphrag

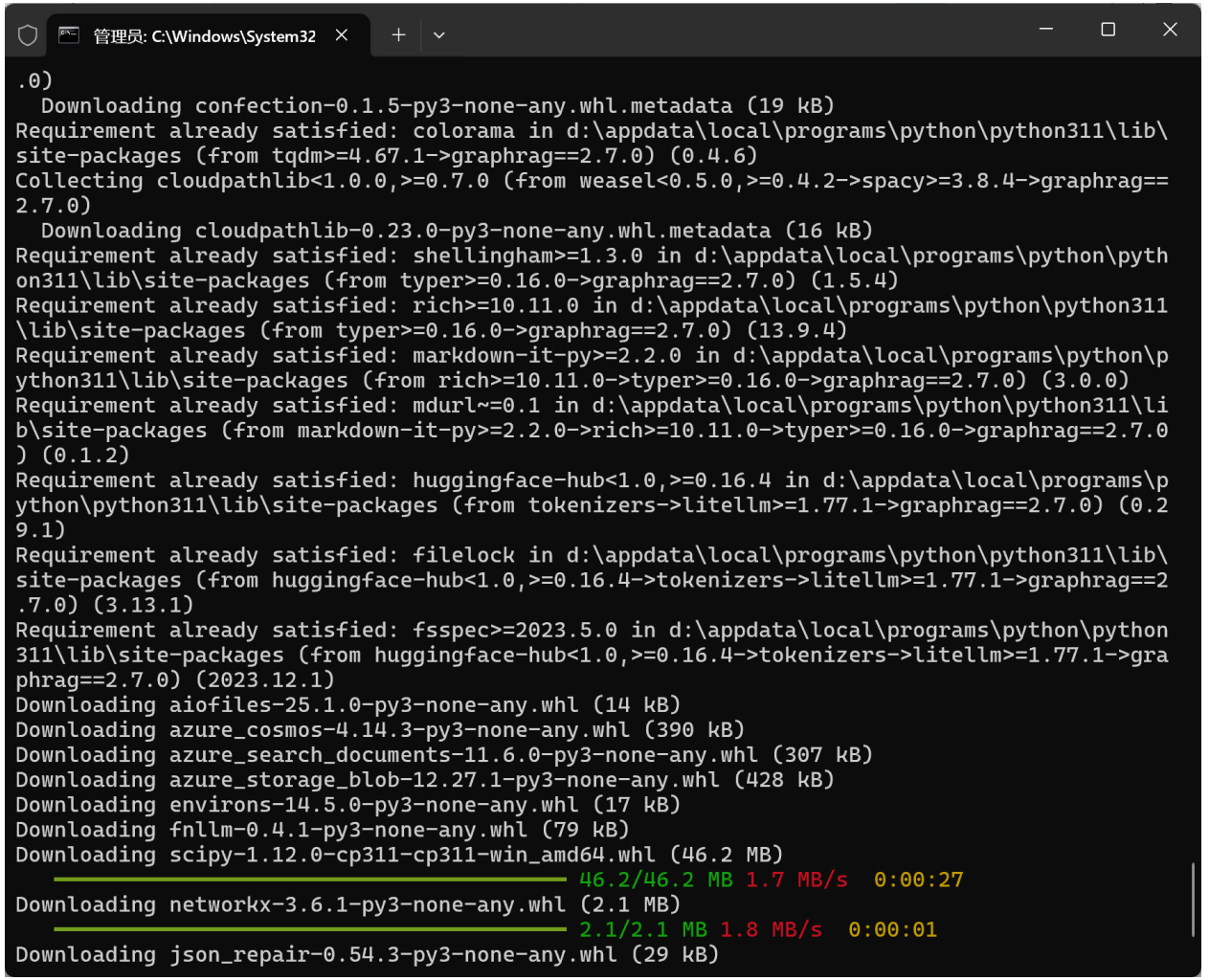
Step1，下载源代码

git clone https://github.com/microsoft/graphrag.git

Step2，下载依赖并初始化项目

pip install -e .

这里会自动基于 graphrag-main/pyproject.toml 进行python包的安装



GraphRAG

Step3，直接运行 graphrag 命令

graphrag init --root .

正确运行后，此处会在graphrag目录下生成prompts、.env、settings.yaml文件

settings.yaml - 主配置文件，包含：

- LLM 模型配置：默认使用 OpenAI 的 gpt-4-turbo-preview 和 text-embedding-3-small
- 输入设置：从 input 文件夹读取文本文件
- 输出设置：结果存到 output，缓存到 cache，日志到 logs
- 向量存储：使用 LanceDB
- 图抽取/社区报告等工作流配置
- 查询设置：local_search, global_search, drift_search 等

.env - 环境变量文件：

GRAPHRAG_API_KEY=<API_KEY>

你需要把 <API_KEY> 替换成你的 OpenAI API Key或者 DASHSCOPE_API_KEY。

prompts/ 文件夹：

生成了一些 prompt 模板文件

GraphRAG

Step4，参数文件配置

配置 settings.yaml

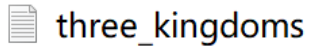
```yaml
default_chat_model:
  type: chat
  model_provider: openai
  api_key: ${GRAPHRAG_API_KEY}
  model: qwen-flash
  api_base: https://dashscope.aliyuncs.com/compatible-mode/v1
  encoding_model: cl100k_base

default_embedding_model:
  type: embedding
  model_provider: openai
  api_key: ${GRAPHRAG_API_KEY}
  model: text-embedding-v3
  api_base: https://dashscope.aliyuncs.com/compatible-mode/v1
```

对 .env文件配置 api_key

```
GRAPHRAG_API_KEY=s
```

Step5，将待检索的文档放到 ./input 目录下



说明：更新最新的 graphrag，运行时可能报错，需要修改 litellm源码，即：

```
4513
4514            if encoding_format is not None:
4515                optional_params["encoding_format"] = encoding_format
4516 -          else:
4517 -              # Omiting causes openai sdk to add default value of "float"
4518 -              optional_params["encoding_format"] = None
4516
4517            api_version = None
4518
```

GraphRAG

Step6，创建GraphRAG索引（耗时较长，取决于文本的大小）

graphrag index --root .

```
Starting pipeline with workflows: load_input_documents, create_base_text_units, cr
eate_final_documents, extract_graph, finalize_graph, extract_covariates, create_co
mmunities, create_final_text_units, create_community_reports, generate_text_embedd
ings
Starting workflow: load_input_documents

Workflow complete: load_input_documents
Starting workflow: create_base_text_units
  1 / 1 ..................................................................
..................
Workflow complete: create_base_text_units
Starting workflow: create_final_documents

Workflow complete: create_final_documents
Starting workflow: extract_graph
  630 / 763 ..............................................................
  631 / 763 ..............................................................
  632 / 763 ..............................................................
  633 / 763 ..............................................................
  634 / 763 ..............................................................
  635 / 763 ..............................................................
  636 / 763 ..............................................................
  637 / 763 ..............................................................
  638 / 763 ..............................................................
  639 / 763 ..............................................................
  640 / 763 ..............................................................
  641 / 763 ..............................................................
  642 / 763 ..............................................................
```

```
  738 / 763 ..............................................................
  739 / 763 ..............................................................
  740 / 763 ..............................................................
  741 / 763 ..............................................................
  742 / 763 ..............................................................
  743 / 763 ..............................................................
  744 / 763 ..............................................................
  745 / 763 ..............................................................
  746 / 763 ..............................................................
  747 / 763 ..............................................................
  748 / 763 ..............................................................
  749 / 763 ..............................................................
  750 / 763 ..............................................................
  751 / 763 ..............................................................
  752 / 763 ..............................................................
  753 / 763 ..............................................................
  754 / 763 ..............................................................
  755 / 763 ..............................................................
  756 / 763 ..............................................................
  757 / 763 ..............................................................
  758 / 763 ..............................................................
  759 / 763 ..............................................................
  760 / 763 ..............................................................
  761 / 763 ..............................................................
  762 / 763 ..............................................................
  763 / 763 ..............................................................
  763 / 763 ..............................................................
Unclosed client session
client_session: <aiohttp.client.ClientSession object at 0x0000018DC4AD3350>
Pipeline complete.....................
```

GraphRAG

Step6，创建GraphRAG索引（耗时较长，取决于文本的大小）

graphrag index --root .

```
🚀 create_base_text_units
                                 id  ... n_tokens
0    801b6814471d697381794bb6956fd0c6  ...     1200
1    e3dedbef878b93e8bf31d89efe569d2d  ...     1200
2    7e6a07a57a5bce1b1c7062b23dbb74b7  ...     1200
3    cf129ba1766b69b7320e56a265103fb6  ...     1200
4    1f13f6ff579a1698e4f38c4d6f50c9e0  ...     1200
..                                ...  ...      ...
185  5840b8ca61c8d142db5b0ef438d61084  ...     1200
186  64748e530daca0dfe4ccb72b53653af6  ...     1200
187  8f4a5f417780c51f68f784c98208af22  ...     1200
188  de9746b3e28d9102b781bb4d5537be1b  ...     1200
189  14ce6192f23892049905b4fb03431ac3  ...      603

[763 rows x 5 columns]
🚀 create_base_extracted_entities
                                        entity_graph
0  <graphml xmlns="http://graphml.graphdrawing.or...
🚀 create_summarized_entities
                                        entity_graph
0  <graphml xmlns="http://graphml.graphdrawing.or...
🚀 create_base_entity_graph
   level                                clustered_graph
0      0  <graphml xmlns="http://graphml.graphdrawing.or...
1      1  <graphml xmlns="http://graphml.graphdrawing.or...
2      2  <graphml xmlns="http://graphml.graphdrawing.or...
```

```
D:\AppData\Local\Programs\Python\Python311\Lib\site-packages\numpy\core\fromnumeric.py:59: Fut
'DataFrame.swapaxes' is deprecated and will be removed in a future version. Please use 'DataFr
  return bound(*args, **kwds)
🚀 create_final_entities
                                 id  ...                        description_embedding
0    e06f3f6665aa42ac821ff4d01d98ceba  ...  [0.010473140515387058, 0.022974025458097458, -...
1    257cdee6da194ec6bbec93c2f9941c38  ...  [-0.0069566573947668076, 0.010649037547409534,...
2    ad019054a71a430a8a41e0e3d91b2c5c  ...  [-0.013226160779595375, -0.0005299110780470073...
3    ca823d5f25194f68b39ece117097d195  ...  [-0.03153751790523529, -0.021626872941851616, ...
4    ac81cfd5ca844d7f8cff43ff78f1af4c  ...  [-0.020771170035004616, 0.010091422125697136, ...
..                                ...  ...                                              ...
234  25fffea45b424b3c9d1e581c6fd681b8  ...  [-0.05620903894305229, -0.010554901324212551, ...
235  c1c178fa53f3483f8664633a085ffa18  ...  [-0.013504572212696075, -0.03294695168733597, ...
236  85182bda61ef45cb9a38e49318730879  ...  [-0.021599022671580315, -0.06492338329553604, ...
237  e0a0f05a3a4f47d5b156e375903748b9  ...  [-0.019512852653861046, -0.013822405599057674,...
238  f2cb3f0d39364f77924ceefe904f3eb1  ...  [-0.014387020841240883, -0.03131725639104843, ...

[3593 rows x 8 columns]
```

```
      level   title          type  ...                top_level_node_id  x  y
0         0    三国演义         EVENT  ...  e06f3f6665aa42ac821ff4d01d98ceba  0  0
1         0     罗贯中        PERSON  ...  257cdee6da194ec6bbec93c2f9941c38  0  0
2         0      长江           GEO  ...  ad019054a71a430a8a41e0e3d91b2c5c  0  0
3         0      汉朝  ORGANIZATION  ...  ca823d5f25194f68b39ece117097d195  0  0
4         0      桓帝        PERSON  ...  ac81cfd5ca844d7f8cff43ff78f1af4c  0  0
...     ...     ...           ...  ...                               ... .. ..
21553     5  三百一十三县         EVENT  ...  25fffea45b424b3c9d1e581c6fd681b8  0  0
21554     5      高祖        PERSON  ...  c1c178fa53f3483f8664633a085ffa18  0  0
21555     5     光武帝        PERSON  ...  85182bda61ef45cb9a38e49318730879  0  0
21556     5      东汉  ORGANIZATION  ...  e0a0f05a3a4f47d5b156e375903748b9  0  0
21557     5      天府           GEO  ...  f2cb3f0d39364f77924ceefe904f3eb1  0  0

[21558 rows x 15 columns]
🚀 create_final_communities
      id  ...                                       text_unit_ids
0     13  ...  [298a3982b5caca8d7469b70dfa25dde9,31b86a6ef947...
1     12  ...  [0520611682d791b2b24230b97e678086,34e26f0d3d0f...
2     16  ...  [0155f78da1a3c002f786eb383427f425,2b6e6bb6274d...
3      1  ...  [3b9ff1bfd4325c5ef3b1a04afe05d324,e3dedbef878b...
4      9  ...  [2231c35ad19351487db2e3d6bbbc8c97,30da0f99935d...
..   ...  ...                                              ...
```

GraphRAG

```
🚀 create_final_relationships
     source     target  weight  ... source_degree target_degree rank
0      三国演义        罗贯中     9.0  ...             1             1    2
1        长江        十万箭     5.0  ...             3             1    4
2        长江         严颜     3.0  ...             3            11   14
3        长江         东吴     1.0  ...             3            74   77
4        汉朝         桓帝     8.0  ...            15             3   18
...       ...        ...     ...  ...           ...           ...  ...
6771  DU YU  SHEN YING     7.0  ...             8             2   10
6772     晋国         陶濬     7.0  ...             7             3   10
6773     陶濬         北兵     6.0  ...             3             2    5
6774     晋帝        归命侯     1.0  ...             3             1    4
6775     高祖        光武帝     7.0  ...             1             1    2

[6776 rows x 10 columns]
🚀 create_final_text_units
                                 id  ...                                relationship_ids
0    801b6814471d697381794bb6956fd0c6  ...  [9080aded4c9b4f44952bc3a02454eb92, 19f6dcd5317...
1    e3dedbef878b93e8bf31d89efe569d2d  ...  [11792907413a46ad870b4ea6b64e8bd2, e50fbb6b703...
2    7e6a07a57a5bce1b1c7062b23dbb74b7  ...  [fcec411d9ebf42709d61ff5f40645782, 61cc1b4f6bd...
3    cf129ba1766b69b7320e56a265103fb6  ...  [9a4db82ca62748c0896e732acf1c839a, b014e174e90...
4    1f13f6ff579a1698e4f38c4d6f50c9e0  ...  [9e3bd9746ee445bbb4767a14c3fb559f, 90893fd8103...
..                                ...  ...                                              ...
758  5840b8ca61c8d142db5b0ef438d61084  ...  [3640377f34c3454a86546f88b50e746a, ca095dd8ae5...
759  64748e530daca0dfe4ccb72b53653af6  ...  [6f85ba6a3ec44575a62417e76673f0b2, 2a28f93c5e6...
760  8f4a5f417780c51f68f784c98208af22  ...  [9eea4da941e44a149f809a52a8fbf9f1, f6e3780d8f6...
761  de9746b3e28d9102b781bb4d5537be1b  ...  [b020dc312c9b49788c71fb7dca87a97c, 939b0747da3...
762  14ce6192f23892049905b4fb03431ac3  ...  [a9af6efea0954150837cc6c7ff3be099, e5fc4e28e61...
```

```
🚀 create_final_community_reports
     community  ...                                    id
0          671  ...  f7a276e5-8706-4700-9950-688fc35106dc
1          672  ...  11af1b98-4065-44e2-abec-71770b644424
2          628  ...  02663567-b02e-4fd9-ada8-8cfc3c7954d4
3          629  ...  9edcbda5-82ca-4eed-85fd-47fd879a4a26
4          630  ...  53fdb3ea-1042-4125-b09c-d968b0dd2429
..         ...  ...                                   ...
668         12  ...  cacdeb2c-122a-45f4-b00b-8b26a365d99f
669         13  ...  4b1b70ec-3d55-4b4c-af69-285d3a83bdd9
670         14  ...  4a401a7b-c48d-49f0-89be-5321d444fb04
671         16  ...  47cd804d-1c34-42b2-8694-8214873c92df
672          9  ...  392f2225-07b2-4a3e-bbf2-9cbbf0b2640e

[673 rows x 10 columns]
```

```
  datetime_column = pd.to_datetime(column, errors="ignore")
🚀 create_base_documents
                                 id  ...                title
0  203662678a5410290a61489e3a6b6e1f  ...  three_kingdoms.txt

[1 rows x 4 columns]
🚀 create_final_documents
                                 id  ...                title
0  203662678a5410290a61489e3a6b6e1f  ...  three_kingdoms.txt

[1 rows x 4 columns]
i. GraphRAG Indexer
├── Loading Input (InputFileType.text) - 1 files loaded (0 filtered)
├── create_base_text_units
├── create_base_extracted_entities
├── create_summarized_entities
├── create_base_entity_graph
├── create_final_entities
├── create_final_nodes
├── create_final_communities
├── create_final_relationships
├── create_final_text_units
├── create_final_community_reports
├── create_base_documents
└── create_final_documents
🚀 All workflows completed successfully.
```

- community_reporting
- entity_extraction
- summarize_descriptions
- text_embedding

创建GraphRAG索引后，会在./cache文件夹下面生成4个文件夹，方便后续进行提问

GraphRAG

Step7，进行查询

graphrag query --root . --method global --query "和曹操相关的人物都有哪些?"

```
creating llm client with {'api_key': 'REDACTED,len=51', 'type': "openai_chat", 'model': 'gpt-4o-mini', 'max_tokens': 4000, 'temperature': 0.0, 'top_p': 1.0, 'n': 1, 'request_timeout': 180.0, 'api_base': None, 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_endpoint': None, 'deployment_name': None, 'model_supports_json': True, 'tokens_per_minute': 0, 'requests_per_minute': 0, 'max_retries': 10, 'max_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 25}

SUCCESS: Global Search Response:
## 曹操相关人物概述

曹操是三国时期魏国的主要领导者，他与许多重要人物之间的关系复杂且深远。这些人物不仅在军事和政治上与曹操有着密切的联系，还在当时的权力斗争中扮演了关键角色。以下是与曹操相关的一些重要人物及其关系的概述。

### 主要对手

1. **刘备**: 刘备是曹操的主要对手之一，二者之间的竞争和冲突在三国历史中占据了重要地位。刘备的崛起对曹操的统治构成了威胁 ，双方之间的多次冲突和战略对抗影响了整个三国的政治格局 [Data: Reports (181, 123, 299, +more)]。

2. **孙权**: 作为东吴的领导者，孙权与曹操之间存在激烈的竞争关系，尤其是在赤壁之战等关键战役中，两者的对抗塑造了当时的政 治格局 [Data: Reports (125, 288, 202, +more)]。

3. **袁绍**: 袁绍是曹操的重要对手之一，二者在权力斗争中形成了明显的对抗关系。袁绍的失败在很大程度上为曹操的崛起铺平了道 路 [Data: Reports (62, 205)]。

4. **董卓**: 董卓是曹操的敌对人物，曹操曾领导军队反对董卓的暴政，显示了他在当时政治斗争中的重要性 [Data: Reports (377, 165)]。

### 重要盟友与将领

1. **曹丕**: 曹操的儿子，继承了父亲的权力，并在其父去世后成为魏国的第一位皇帝。父子关系在魏国的建立和发展中起到了关键作 用 [Data: Reports (42, 119)]。

2. **夏侯惇**: 曹操的重要将领，以其忠诚和军事才能著称，参与了多次重要战役，支持曹操的军事策略 [Data: Reports (212, 79, 336, +more)]。

3. **张辽**: 另一位著名的将领，他在曹操的指挥下参与了多次重要战役，展示了曹操在军事指挥中的影响力 [Data: Reports (178, 153)]。

4. **荀彧**: 作为曹操的军事顾问，荀彧以其战略眼光和智谋著称，对曹操的决策产生了深远影响 [Data: Reports (195, 42)]。

5. **关羽**: 虽然关羽以忠诚著称，但他与曹操的关系复杂，涉及尊重和内心冲突 [Data: Reports (299, 180)]。

### 其他相关人物

- **吕布**: 吕布曾与曹操结盟，但最终成为敌人，导致了激烈的权力斗争 [Data: Reports (370, 160)]。
- **诸葛亮**: 蜀汉的主要战略家，与曹操的对抗在三国历史中占据了重要地位 [Data: Reports (240, 53)]。
- **华佗**: 著名的医师，曾为曹操治疗，显示了医疗专业在政治环境中的重要性 [Data: Reports (51)]。

## 结论

曹操的历史地位与他周围的众多人物密切相关。这些人物的关系和互动不仅影响了曹操的军事和政治生涯，也在三国时期的权力格局中发挥了重要作用。通过对这些人物的分析，我们可以更好地理解曹操在历史上的复杂角色及其影响力。
```

GraphRAG

python -m graphrag.query --root ./cases --method local "和曹操相关的人物都有哪些?"

```
creating llm client with {'api_key': 'REDACTED,len=51', 'type': "openai_chat", 'model'
: 'gpt-4o-mini', 'max_tokens': 4000, 'temperature': 0.0, 'top_p': 1.0, 'n': 1, 'reques
t_timeout': 180.0, 'api_base': None, 'api_version': None, 'organization': None, 'proxy
': None, 'cognitive_services_endpoint': None, 'deployment_name': None, 'model_supports
_json': True, 'tokens_per_minute': 0, 'requests_per_minute': 0, 'max_retries': 10, 'ma
x_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests'
: 25}
creating embedding llm client with {'api_key': 'REDACTED,len=51', 'type': "openai_embe
dding", 'model': 'text-embedding-3-small', 'max_tokens': 4000, 'temperature': 0, 'top_
p': 1, 'n': 1, 'request_timeout': 180.0, 'api_base': None, 'api_version': None, 'organ
ization': None, 'proxy': None, 'cognitive_services_endpoint': None, 'deployment_name':
 None, 'model_supports_json': None, 'tokens_per_minute': 0, 'requests_per_minute': 0,
'max_retries': 10, 'max_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True,
 'concurrent_requests': 25}

SUCCESS: Local Search Response:
## 曹操相关人物概述

曹操是中国历史上著名的政治家和军事统帅，他的周围有许多重要人物，这些人物在他的军事和政治生涯中扮演了关键角色。以下是与曹操相关的一些主要人物及其简要介绍。

### 1. 曹洪

曹洪是曹操的主要将领之一，以其在对抗袁绍的军事行动中表现出的战略才能而闻名。他在曹操的军队中担任重要职务，参与了多次关键战役，展现了对曹操的忠诚和支持 [Data: Entities (291); Relationships (723, 2223, 2226, +more)]。

### 2. 许褚

许褚是曹操的另一位重要将领，以其勇敢和忠诚著称。他不仅在战斗中表现出色，还担任曹操的护卫，确保其安全。许褚在多次战役中与曹操并肩作战，展现了深厚的信任关系 [Data: Entities (530); Relationships (735, 2953, 258, +more)]。

### 3. 荀彧

荀彧是曹操的重要谋士，以其卓越的智谋和战略眼光而闻名。他在曹操的军事和政治决策中提供了重要的建议，帮助曹操在复杂的局势中做出明智的选择 [Data: Entities (903); Relationships (未提供)]。

### 4. 刘备

刘备是曹操的主要对手之一，最初与曹操有过合作，但后来因权力斗争而成为敌人。刘备的崛起与曹操的扩张形成鲜明对比，二者之间的冲突深刻影响了三国时期的历史进程 [Data: Entities (未提供); Relationships (497, 258, +more)]。

### 5. 吕布

吕布是另一位与曹操有过直接冲突的著名武将。他以勇猛著称，但因其反复无常的性格而受到各方的忌惮。吕布与曹操之间的敌对关系在历史上留下了深刻的印记 [Data: Entities (未提供); Relationships (1789, +more)]。

## 总结

曹操的周围有许多重要人物，他们在不同的历史阶段对曹操的军事和政治生涯产生了深远的影响。这些人物不仅包括他的将领和谋士，还包括他的对手，他们的互动和冲突共同塑造了三国时期的历史格局。这些关系的复杂性和多样性使得曹操的故事更加引人入胜，成为后世研究的重要课题。
```

GraphRAG查询模式

GraphRAG 提供了两种查询模式：

- Global Query（全局查询）
- Local Query（本地查询）

Global Query（全局查询）：

用于回答全局性的问题，例如“《三国演义》的主题是什么”。

它通过利用社区摘要，对整个语料库进行整体问题的推理，利用LLM生成的知识图谱来组织和聚合信息。

在具体实现上，Global Query 方法使用从社区层次结构指定层级中收集的报告作为上下文数据，以类似Map-Reduce的方式生成响应。

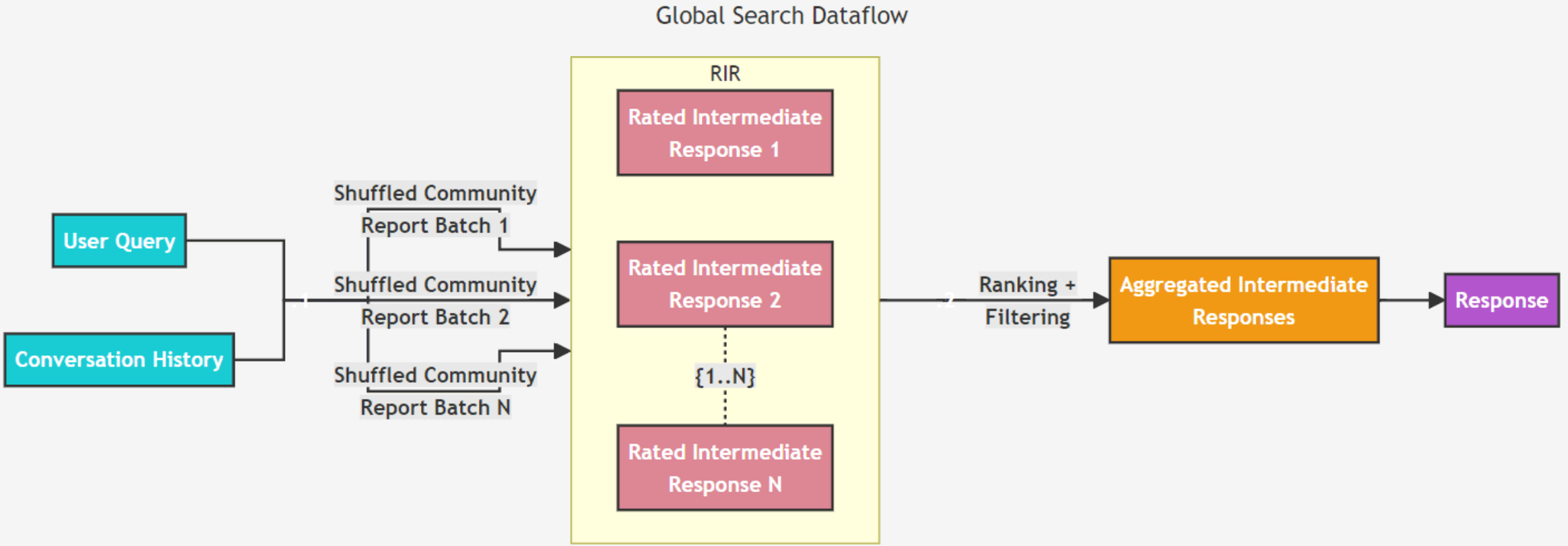
在Map步骤中，社区报告被分割成文本块，每个文本块用于生成中间响应，其中每个点都有一个数值评级。

在Reduce步骤中，从中间响应中挑选出最重要的点并进行聚合，最终形成用于生成最终响应的上下文。这种方法的直观理解是：越宏观的问题需要越宏观的视角和信息来回答。

这种查询方式是资源密集型的，但通常能够很好地回答那些需要对数据集整体有全面理解的问题。

GraphRAG查询模式

全局搜索数据流



GraphRAG查询模式

Local Query（本地查询）：

用于回答更加具体的问题，例如询问“洋甘菊有哪些治疗特性？”。

本地查询则基于更加微观的视角，结合知识图谱中的结构化数据与原始文档中的非结构化数据，来增强检索和生成过程中的上下文。

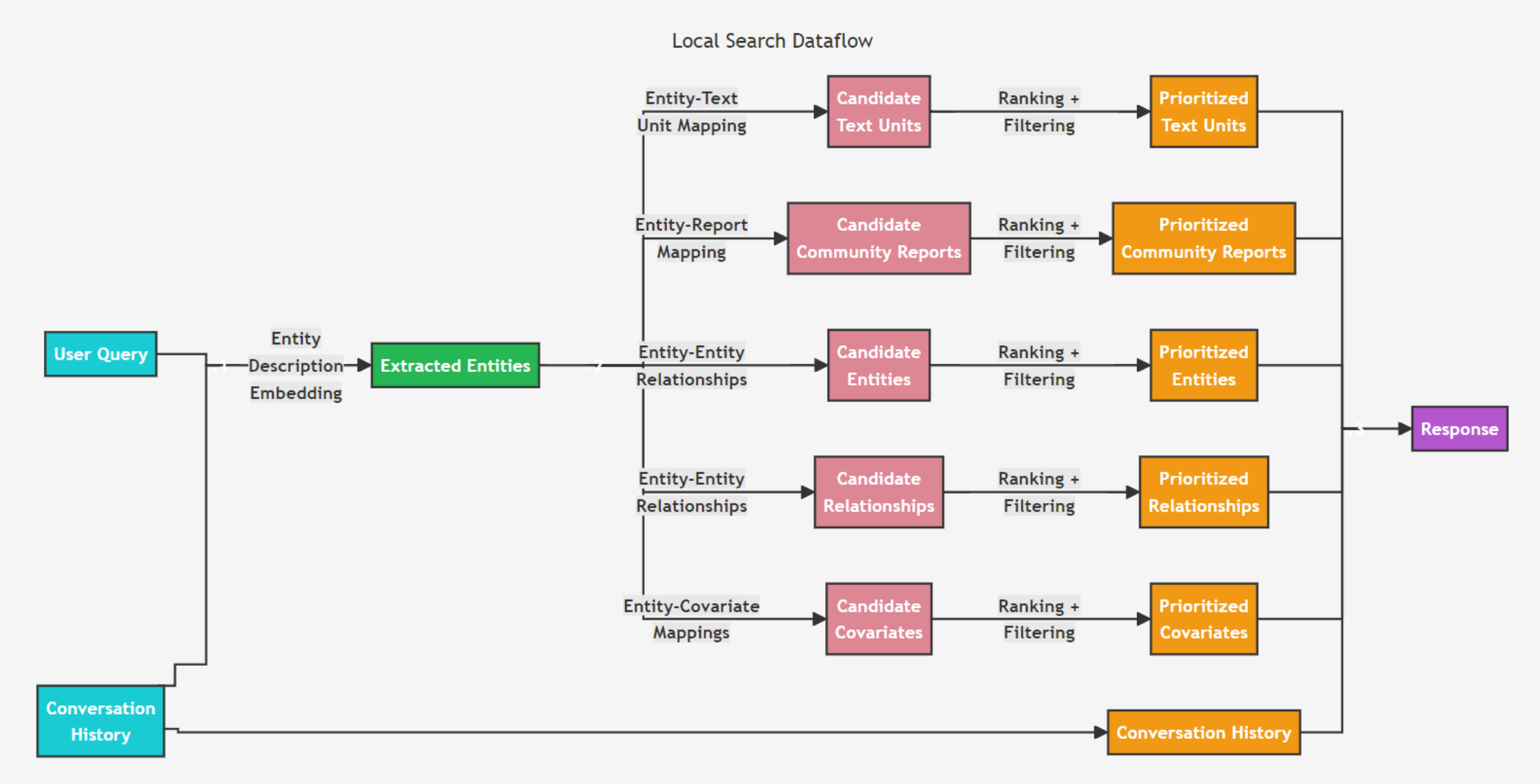
在具体实现上，系统将依据原始提问，从知识图谱中识别出一组与用户输入语义相关的实体。然后，利用这些实体作为查询条件，在知识图谱或相关数据库中进行检索，找到与这些实体直接相关的内容，包含：TextUnit、社区报告、实体、关系或协变量（如主张）。检索的结果经过过滤和重排序后，选择高质量的数据源，并将其整合进一个预定义大小的上下文窗口。

这种方法适用于需要理解输入文档中特定实体的问题，通过结合AI提取的知识图谱和原始文档的文本块生成答案。

Global Query 适合处理需要跨数据集汇总信息的宏观问题，而 Local Query 适合处理需要理解文档中特定实体的微观问题。

GraphRAG查询模式

局部搜索数据流



GraphRAG查询

python -m graphrag.query --root ./cases --method local "关羽战胜过哪些武将?"

```
SUCCESS: Local Search Response:
## 关羽的战斗成就

关羽，作为三国时期著名的将领，以其卓越的武艺和忠诚的精神而闻名。他在多场战斗中展现了非凡的战斗能力，战胜了多位敌将。以下是关羽战胜的一些重要武将：

### 1. 吕布
关羽在与吕布的战斗中表现出色，尽管吕布是当时公认的顶尖武将，但关羽凭借其高超的战斗技巧和勇气，成功击败了吕布。这场战斗不仅展示了关羽的武力，也突显了他在战场上的战略思维 [Data: Relationships (363)]。

### 2. 颜良
在著名的白马之战中，关羽面对敌将颜良，展现了他的英勇和战斗技巧。关羽在这场战斗中成功击败了颜良，进一步巩固了他在蜀汉的声望 [Data: Entities (1022)]。

### 3. 阮籍
关羽在与阮籍的交战中也取得了胜利，显示了他在战斗中的灵活应变能力和战术运用 [Data: Relationships (2036)]。

### 4. 黄承儿
在与黄承儿的战斗中，关羽同样表现出色，成功将其击败，进一步证明了他的战斗实力 [Data: Relationships (2036)]。

### 5. 其他敌将
关羽在多次战斗中还战胜了其他敌将，如管亥等，展现了他在战场上的统治力和影响力 [Data: Entities (32)]。

## 总结
关羽的战斗成就不仅体现在他战胜的敌将上，更在于他在三国历史中的重要地位和影响力。他的忠诚、勇气和战斗技巧使他成为了三国时期最受尊敬的将领之一。关羽的传奇故事至今仍在中国文化中广为流传，成为了忠义的象征。
```

python -m graphrag.query --root ./cases --method global "关羽战胜过哪些武将?"

```
SUCCESS: Global Search Response:
## 关羽的战胜记录

关羽，亦称为关云长，是三国时期著名的将领，以其卓越的武艺和战斗能力而闻名。他在与多位敌将的交战中取得了显著的胜利，这些胜利不仅巩固了他在蜀汉的地位，也为他赢得了传奇般的声誉。

### 主要战胜的敌将

1. **吕布**：关羽在与吕布的战斗中取得了显著的胜利，吕布被视为当时最强的武将之一。这场胜利展示了关羽的武力和战略能力 [Data: Reports (299, 141, +more)]。

2. **曹操的将领**：关羽曾战胜曹操的多位将领，包括曹仁、颜良和文丑等。这些胜利不仅展示了他的个人实力，也在三国时期的权力斗争中产生了深远的影响 [Data: Reports (132, 154, 277, 342, 394, +more)]。

3. **黄忠**：关羽在与黄忠的交战中也表现出色，进一步巩固了他的声誉 [Data: Reports (298, 181, +more)]。

4. **张辽**：关羽在与张辽的战斗中取得了胜利，展示了他在战斗中的卓越才能 [Data: Reports (323, 255, +more)]。

5. **周瑜**：虽然与周瑜的对抗中未能完全战胜，但关羽在多次交锋中展现了强大的战斗力，给敌方造成了显著的压力 [Data: Reports (79, 212, 249, 400, +more)]。

### 战斗的影响

关羽的胜利不仅提升了他的个人声望，也为蜀汉的军事战略做出了重要贡献。他在战斗中展现出的勇气和智慧，使他成为三国历史上不可或缺的军事领导者。这些战斗的胜利为他赢得了“武圣”的称号，成为后世崇拜的对象 [Data: Reports (323, 255, +more)]。

总的来说，关羽在三国时期的战斗中战胜了多位著名武将，这些胜利不仅巩固了他在蜀汉的地位，也为他在中国历史上留下了深远的影响。
```

GraphRAG查询

Thinking：如何让匹配上的entities和关系更多一些？

```
local_search:
  # text_unit_prop: 0.5
  # community_prop: 0.1
  # conversation_history_max_turns: 5
  # top_k_mapped_entities: 10
  # top_k_relationships: 10
  # llm_temperature: 0 # temperature for sampling
  # llm_top_p: 1 # top-p sampling
  # llm_n: 1 # Number of completions to generate
  # max_tokens: 12000
```

GRAPHRAG_LLM_API_BASE	API基本URL。默认值：None
GRAPHRAG_LLM_TYPE	LLM操作类型。可以是openai_chat或azure_openai_chat。默认值：openai_chat
GRAPHRAG_LLM_MAX_RETRIES	请求失败时尝试的最大重试次数。默认值：20
GRAPHRAG_EMBEDDING_API_BASE	API基本URL。默认值：None
GRAPHRAG_EMBEDDING_TYPE	要使用的嵌入客户端。可以是openai_embedding或azure_openai_embedding。默认值：openai_embedding
GRAPHRAG_EMBEDDING_MAX_RETRIES	请求失败时尝试的最大重试次数。默认值：20
GRAPHRAG_LOCAL_SEARCH_TEXT_UNIT_PROP	上下文窗口用于相关文本单位的比例。默认值：0.5
GRAPHRAG_LOCAL_SEARCH_COMMUNITY_PROP	上下文窗口用于社区报告的比例。默认值：0.1
GRAPHRAG_LOCAL_SEARCH_CONVERSATION_HISTORY_MAX_TURNS	包括在对话历史中的最大轮次数。默认值：5
GRAPHRAG_LOCAL_SEARCH_TOP_K_ENTITIES	从实体描述嵌入存储中检索的相关实体数。默认值：10
GRAPHRAG_LOCAL_SEARCH_TOP_K_RELATIONSHIPS	控制将多少个网络关系引入上下文窗口。默认值：10
GRAPHRAG_LOCAL_SEARCH_MAX_TOKENS	根据你的模型的标记限制进行更改（如果你使用的模型具有8k限制，则好的设置可能是5000）。默认值：12000
GRAPHRAG_LOCAL_SEARCH_LLM_MAX_TOKENS	根据你的模型的标记限制进行更改（如果你使用的模型具有8k限制，则好的设置可能为1000=1500）。默认值：2000
GRAPHRAG_GLOBAL_SEARCH_MAX_TOKENS	根据你的模型的标记限制进行更改（如果你使用的模型具有8k限制，则好的设置可能为5000）。默认值：12000
GRAPHRAG_GLOBAL_SEARCH_DATA_MAX_TOKENS	根据你的模型的标记限制进行更改（如果你使用的模型具有8k限制，则好的设置可能为5000）。默认值：12000
GRAPHRAG_GLOBAL_SEARCH_MAP_MAX_TOKENS	默认值：500
GRAPHRAG_GLOBAL_SEARCH_REDUCE_MAX_TOKENS	根据你的模型的标记限制进行更改（如果你使用的模型具有8k限制，则好的设置可能为1000-1500）。默认值：2000
GRAPHRAG_GLOBAL_SEARCH_CONCURRENCY	默认值：32

GraphRAG查询

对比维度	Global模式	Local模式
适用场景	用于支持全局型的查询任务，回答基于高层语义理解的概要性问题，如“数据集中的主要主题是什么？”	用于针对具体事实的提问，回答关于特定实体的信息与关系等
查询架构	采用分布式计算中的Map-Reduce架构	结合知识图谱结构化信息与原始文档的非结构化数据构建上下文
查询过程	1. MAP过程：根据用户输入问题与对话历史，查询指定层次结构上的所有社区报告，对这些社区报告分成多个批次生成带有评分的中间响应（RIR），评分用来表示这个观点的重要 2. Reduce过程：对中间响应进行排序，选择最重要的观点汇总并作为参考的上下文，最后交给LLM生成最终响应结果	1. 根据输入的查询问题与对话历史，从知识图谱中识别出最相关的实体 2. 从这些实体开始，提取更多的相关信息，包括关联的原始文本块、关联的社区、关联的实体和关联的关系，并对这些提取的信息进行排序与筛选，最终形成参考的上下文 3. 借助LLM与提示模板，输入上下文与原始问题，生成最终响应
数据来源	主要基于社区报告	包括知识图谱中的实体、关系、社区报告以及原始文本块
对LLM调用的特点	需要多次调用LLM，先为每个社区的summary生成答案，再汇总所有答案生成最终结果	一次调用LLM，将构建好的上下文与原始问题一起输入
上下文规模	由于使用社区报告，且需要进行Map-Reduce处理，上下文规模较大	相比RAG会产生更大的上下文
查询成本	非常高，因为需要处理大量的上下文和多次调用LLM	高，需要多种信息构建上下文

Local答案生成：针对具体问题，GraphRAG通过结合元素和元素摘要生成初步答案，这些答案来源于GraphRAG中的特定社区，

Global答案生成：对于需要涵盖整个数据集的全局性问题，GraphRAG采用Map-Reduce机制，将所有社区的初步答案组合起来。


Thank You
Using data to solve problems