NLP课程设计中期报告

> 题目：基于LangChain的本地知识库智能检索与问答增强
>
> 组长：王博远（2022211738）组员：池瀚（2022211740）

1. 详细设计

1.1 整体架构设计

- **前端层**：Streamlit Web应用，提供文档上传、问答交互、结果展示
- **业务逻辑层**：LangChain编排的RAG流程，包含文档处理、检索、生成三大模块
- **模型层**：基础问答语言模型Qwen1.5-7B-Chat+中文词嵌入模型bge-large-zh-v1.5
- **存储层**：FAISS向量数据库 + 原始文档存储

1.2 核心模块设计细化

1. **文档处理模块**

文档处理模块采用流水线式处理，通过PDFPlumber提取文本内容，使用递归字符分割算法（RecursiveCharacterTextSplitter）将长文档切分为适合检索的语义块。

> RecursiveCharacterTextSplitter 是一个Python类，用于将文本递归地分割成指定大小的块。它的核心思想是根据一组分隔符（separators）逐步分割文本，直到每个块的大小都符合预设的chunk_size。如果某个块仍然过大，它会继续递归地分割，确保每个块既不会太长影响检索精度，也不会太短损失语境信息。

实现方式：

- 使用PDFPlumber读取PDF并提取结构化信息（文本、页码、字体等）
- 通过LangChain的文本分割器进行智能分块，设置chunk_size=512，overlap=50
- 利用BGE嵌入模型将文本块转换为768维向量表示
- 将向量与元数据一起存入FAISS索引，支持高效相似度搜索

2. **混合检索模块**

混合检索结合了基于词频的稀疏检索（BM25）和基于语义的稠密检索（向量检索）。BM25擅长精确匹配关键词，而向量检索能理解语义相似性。通过并行执行两种检索并融合结果，可以同时捕获词汇匹配和语义关联。

> BM25算法是常见的用来计算query和文章相关度的一种算法。算法的原理就是将需要计算的query分词成w1，w2，…，wn，然后求出每一个词和文章的相关度，最后将这些相关度进行累加，得到文本相似度计算结果。
>
> $$Score(Q,d)=\sum_{i}^{n}W_i\cdot R(q_i,d)$$
>
> 公式中第一项Wi表示第i个词的权重，一般会使用TF-IDF算法来计算词语的权重。公式第二项R(qi,d)表示查询query中的每一个词和文章d的相关度。

实现方式：

- BM25分支：构建文档索引，计算查询词与文档的TF-IDF相关性得分
- 向量分支：将查询向量化，使用FAISS进行最近邻搜索
- 支持动态调整两种检索方法的权重比例（默认BM25:向量=0.4:0.6）

3. **答案生成模块**

基于检索到的相关文档片段，构建包含上下文的提示词，引导大语言模型生成准确且有依据的回答。通过精心设计的提示词模板，限制模型只基于提供的上下文回答，避免幻觉。

实现方式：

- 对检索结果按相关性得分排序，选取top-k个片段
- 构建结构化提示词：系统角色定义 + 上下文 + 用户问题 + 回答要求
- 调用Qwen模型进行推理，合理设置temperature保证回答稳定性
- 后处理：提取引用信息，标注来源页码

2. 已完成工作

2.1 基础框架搭建（已完成）

在开题报告前后，我们基本完成了项目框架的搭建，通过部署本地轻量的1.5b问答模型+轻量的词嵌入模型all-MiniLM-L6-v2，基本上实现了检索+问答的功能：

- **文档加载**：实现PDF文档解析，支持中文编码
- **文档分块**：RecursiveCharacterTextSplitter集成完成
- **向量化存储**：FAISS向量数据库构建，支持持久化
- **基础检索**：实现相似度检索，返回top-k结果
- **简单问答**：通过LangChain构建基础问答链

2.2 核心功能开发（部分完成）

但是我们在中期实验过程中也发现，轻量模型的性能和效果的确不是很好，并且all-MiniLM-L6-v2在中文知识库的嵌入上经常会出现乱码、重复和英文的错配。因此我们考虑尝试更适合中文语料BGE-large-zh-v1.5模型+Qwen1.5-7B-Chat。

> **BGE-large-zh-v1.5**是由智源研究院开发的中文语义表示模型，基于BERT架构优化，专门针对中文文本检索任务训练。该模型在中文语料上进行了大规模预训练，能够更好地理解中文语义、处理中文特有的语言现象（如成语、文言文等），在中文检索任务上显著优于通用的多语言模型。
>
> **Qwen1.5-7B-Chat**是阿里云开发的对话模型，具有70亿参数，在中文理解和生成任务上表现优异。相比轻量级模型，它具有更强的上下文理解能力、更准确的中文表达能力，以及更好的指令遵循能力，特别适合需要深度理解和精准回答的知识问答场景。

在此调整的基础上，我们进一步完成一些调试和优化：

2.2.1 文档处理优化

- **完成部分**：
  - 基础分块功能实现（基于字符长度的递归分割）
  - 中文分词初步支持（采用jieba分词工具）
  - PDF页码元数据信息的提取
- **进行中**：
  - 结构感知分块（包括章节识别、标题识别等等）
  - 表格、公式等一些复杂内容分块处理
  - 文档、页码和内容的精确溯源

2.2.2 混合检索实现

- **完成部分**：
  - BM25Retriever集成
  - 向量检索器实现
  - EnsembleRetriever基础配置
- **进行中**：
  - 权重动态调整机制
  - 检索结果重排优化（BGE Reranker）

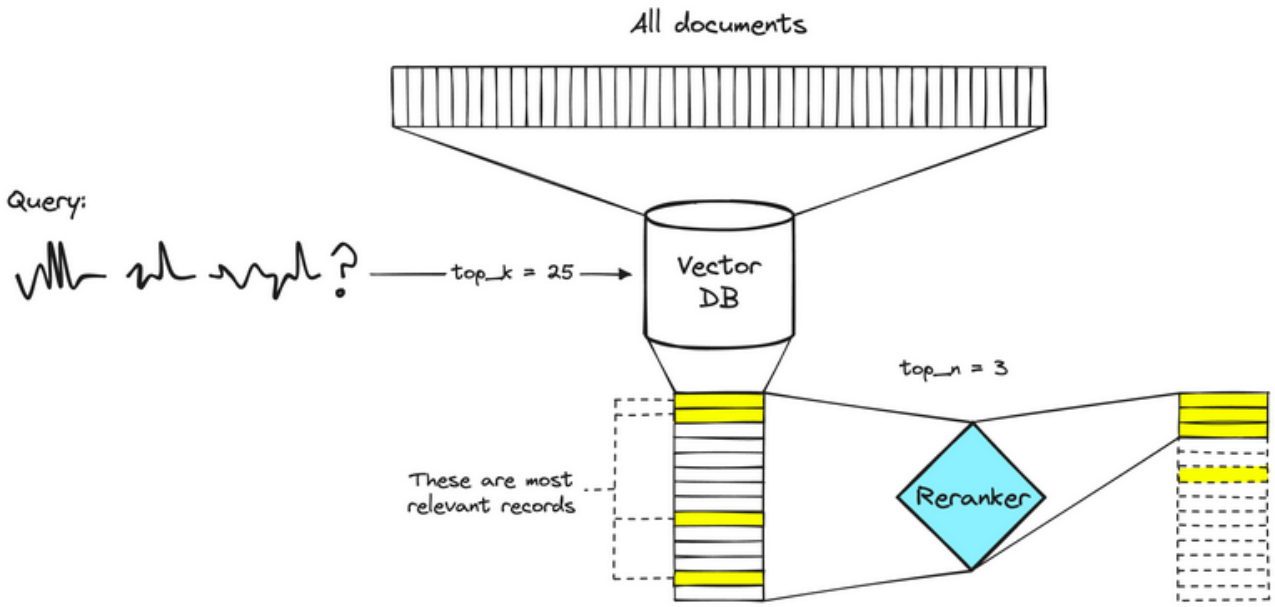
这里需要解释和明晰一下三个问题：

> **① Reranker与Embedding模型的区别**
>
> 尽管Reranker和Embedding模型都用于信息检索系统，它们的定位和功能却是不同的。
>
> Embedding模型（向量模型）主要用于初步筛选文档。它将文本转换为向量表示，并计算这些向量之间的相似度，从而筛选出一组可能相关的候选文档。Embedding模型的优势在于它的计算效率高，适合处理大规模数据集。
>
> Reranker（即重排序器）则用于对Embedding模型筛选出的候选文档进行精细排序。Reranker通常基于复杂的深度神经网络，能够更好地理解文本的上下文和细微差异，从而提供更高的排序精度。
>
> 简单来说，Embedding模型负责“找出一批可能相关的候选文档”，而Reranker负责“在这些候选文档中找出最相关的，并将它们按相关性排序”。
>
> **② 为什么要额外引入重排序器Reranker进行第二次排序？**

既然嵌入模型已经能够提供初步筛选结果，为什么还需要额外的重排序器呢？答案在于准确性。

向量模型需要将整个文档压缩成一个固定长度的数字序列，这个过程就像把一本书概括成一句话，虽然能抓住大意，但很多细节信息会在压缩过程中丢失。更重要的是，文档的向量表示是预先计算好的，它代表的是文档的"平均含义"，无法针对每个具体查询做出调整。

相比之下，Reranker可以避免这种信息丢失。它在收到用户查询后，会将查询和每个候选文档作为一个整体进行分析，直接处理原始文本信息，不存在压缩损失的问题。这种方式能够捕捉到查询与文档之间的细微关联，比如识别出同一个词在不同语境下的不同含义。正是因为Reranker能够进行这种精细的交互分析，所以能够将真正相关的文档准确地排到前面。



③ 那么，为什么不直接使用Reranker进行检索呢？

理论上，Reranker可以直接用于检索，但在实际应用中并不常见。这是因为Reranker的计算复杂度非常高，因为它是针对所有的（查询，文档）二元组进行遍历，直接使用它对整个数据集进行排序会导致极高的计算成本，难以满足实时性要求。因此，通常我们会先用Embedding模型进行初步筛选，将文档集合缩小到一个合理的范围，然后再使用Reranker进行精细排序。

2.2.3 提示工程初步实现

- 设计并测试了多个提示词模板，目前采用的提示词模板如下

代码块

```
self.prompt_template = PromptTemplate(
template="""基于以下上下文信息，请回答用户的问题。如果上下文中没有相关信息，请说明无法从提供的文档中找到答案。
上下文信息：
{context}
问题：{question}
请提供详细、准确的回答："""，
```

```
    input_variables=["context", "question"])
```

- 实现了基础的引用标注功能
- 初步的答案质量控制

3. 初步结论

3.1 技术可行性验证

- **本地部署可行**：DeepSeek-r1-1.5b在普通硬件上运行流畅
- **RAG效果初显**：检索增强明显提升了回答的准确性和相关性
- **混合检索优势**：BM25+向量检索的组合优于单一方法

3.2 系统使用反馈

- 界面简洁直观，易于使用，但仍需改进优化核心功能
- 检索结果展示有助于理解答案来源，但是还不够准确和精准
- 中文支持良好，但仍有优化空间，需要剔除一些英文和特殊符号等不必要信息

4. 问题及可能的解决方案

问题1：文档处理方面的问题

在文档加载和分块的过程中，处理纯文字的内容很容易，但是像机器学习方面的电子文档，原书中常常会在字里行间嵌入一些公式或图表来辅助说明，那么怎样处理像公式、图片和表格这些复杂格式的语料呢？进一步地，又如何让模型回答的过程中“检索”到这些混合公式和文字的分块呢？

可能的解决办法：

1. 公式/图表的处理：
    - 使用PDFPlumber的高级功能识别LaTeX公式区域
    - 将公式转换为LaTeX源码保存，并创建文本描述（如"高斯分布公式"）
    - 在向量化时，同时编码公式/图表的文本描述和上下文
    - 识别图表标题，为图表创建描述性文本索引，支持基于描述的检索
2. 混合内容分块：
    - 考虑设计一个特殊内容类型的识别器，区分纯文本、公式、图表等
    - 实现“语义单元”的概念，将相关的文字和公式/图表作为整体分块
    - 在检索时同时返回完整的语义单元，确保内容完整性

问题2：检索效果方面的问题

如何保证系统不会生成检索内容之外的信息？也就是说如何确保检索的结果的确来自原始知识库的分块，而不是来自问答模型自己“生成”出来的答案呢？

可能的解决办法：

- 基于关键词匹配算法，确保答案中的关键信息在检索文档中存在
- 设置置信度阈值，低于阈值时明确告知用户"文档中未找到相关信息"
- 在提示词中明确要求"仅基于上下文内容回答，不得添加额外信息"
- 进行多轮验证，让模型自我检查答案是否超出给定内容
- 要求模型在回答时标注出每个观点的来源

5. 后续工作计划

第14-15周：系统继续调试和优化

- **性能优化**
  - 实现查询和文档缓存
  - 优化向量检索速度
  - 减少模型推理延迟
- **功能增强**
  - 处理复杂内容分块
  - 优化检索结果重排
  - 完善引用追踪机制

第15-16周：核心功能集成和和测试

- **功能完善**
  - 调优分块策略和检索方法
  - 确保检索+生成的基本功能
- **系统测试**
  - 构建测试文档和知识库
  - 测试系统运行和反馈结果
- **文档完善**
  - 技术文档、用户手册编写
  - 演示材料准备

时间节点

- 5月30日：提交中期进度报告
- 6月6日：完善核心功能，开始系统集成测试
- 6月13日：完成最终优化，准备项目答辩