

基于LangChain的本地知识库
智能检索与问答增强

北京邮电大学



NLP课程设计开题报告

班 级： 2022219107

姓 名： 王博远、池瀚

学 号： 2022211738、2022211740

指导老师： 袁彩霞

2025 年 5 月 16 日

1. 选题概述

1.1 选题来源

本项目源于对大语言模型在特定领域知识应用的思考。传统大模型在回答专业领域问题时常因**知识有限或更新不及时**而表现不佳，而联网检索又可能引入**隐私和延迟问题**。因此，我们提出构建一个**基于LangChain的本地知识库智能检索与增强问答系统**，通过整合本地部署的大模型与高效检索技术，实现对特定领域文档的精准问答。该系统旨在解决**知识时效性、回答准确性和部署隐私性**三大问题，为教育、研究等场景提供高质量的智能问答服务。

1.2 预期目标

- 结合指定题目1(大模型检索增强生成)和题目2(基于LangChain的本地化智能问答系统)，将RAG技术与本地部署模型相结合，解决专业领域问答中的知识更新、准确性与隐私保护问题
- 构建基于Ollama+LangChain+RAG的本地化智能问答系统，实现对特定文档的精准问答，支持PDF等多格式文档处理，提供可溯源的高质量回答
- 探索混合检索策略在提升回答准确性方面的效果，实现可配置的多路召回机制和重排序优化

1.3 涉及到的知识点

- 检索增强生成 (RAG)
- 文档分块 (Chunking)
- 混合信息检索与重排序优化
- 大语言模型部署与微调

2. 技术方案

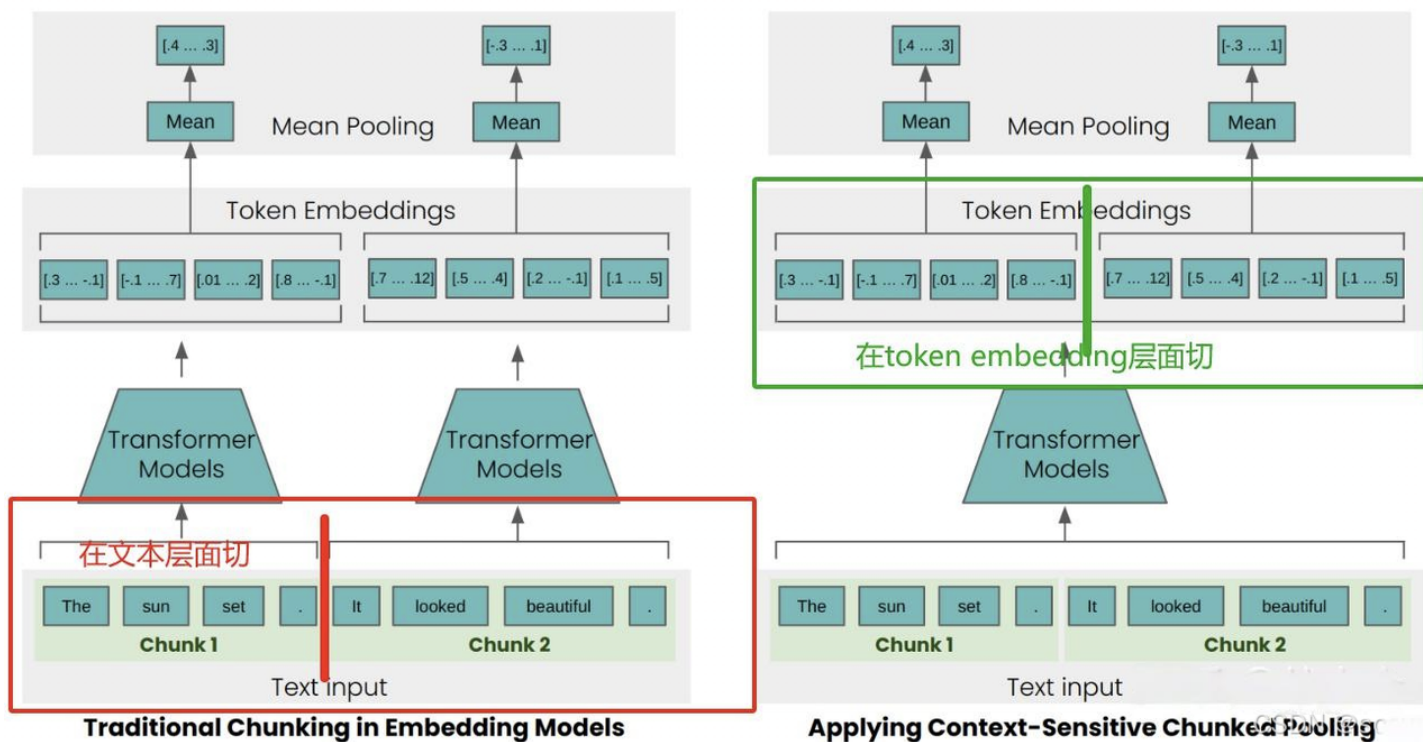
本项目的技术方案围绕“检索增强生成”(Retrieval-Augmented Generation, RAG)这一核心理念展开。RAG技术弥补了大模型知识有限的缺陷，通过从外部知识库检索相关信息，增强模型回答的准确性和可靠性。系统核心技术大致由四个关键模块组成：文档分块、混合检索、重排序优化以及大模型增强。

2.1 整体实现架构

- 前端**：Streamlit构建交互界面
- 后端**：LangChain构建RAG流程
- 模型**：Ollama部署deepseek-r1系列问答模型(1.5b→7b/8b)+词嵌入模型all-MiniLM-L6-v2/bge-large-zh-v1.5
- 知识库**：FAISS向量数据库+BM25的混合检索

2.2 核心技术模块

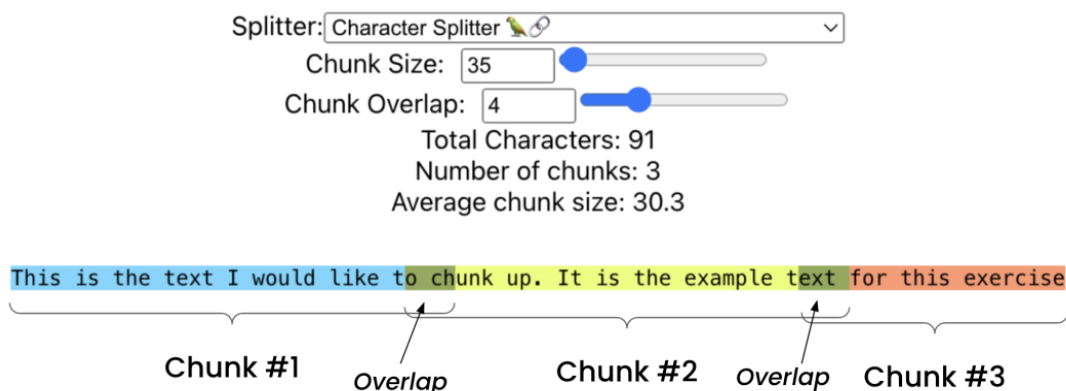
2.2.1 文档分块 (Chunking)



文档分块是RAG系统的基础，负责将不同格式的文档转换为结构化的知识块，为后续检索提供基础。

1. 递归分块算法 (基础实现)

- **块大小 Chunk Size:** 希望在块中包含的字符数，如50、100、100,000等。
- **块重叠 Chunk Overlap:** 希望连续块之间重叠的量。这是为了避免将单个上下文切割成多个部分。这将在块之间创建重复的数据。



- **实现原理:** 可以采用RecursiveCharacterTextSplitter实现自适应分块，该算法会递归地尝试在不同分隔符（如换行符、句号、空格等）处分割文本
- **优势:** 能够尽量在自然段落边界处分割，保持语义完整性
- **缺点:** 忽略块与块之间的语义信息，可以考虑结合语义分块
- **参数优化:**

```

1  text_splitter = RecursiveCharacterTextSplitter(
2      chunk_size=chunk_size,          # 可配置块大小，默认512
3      chunk_overlap=chunk_overlap,    # 块间重叠，默认50
4      length_function=len,
5      separators=["\n\n", "\n", "。 ", "! ", "? ", ". ", " ", ""] # 分隔符优先级
6  )

```

2. 结构感知分块（优化策略）

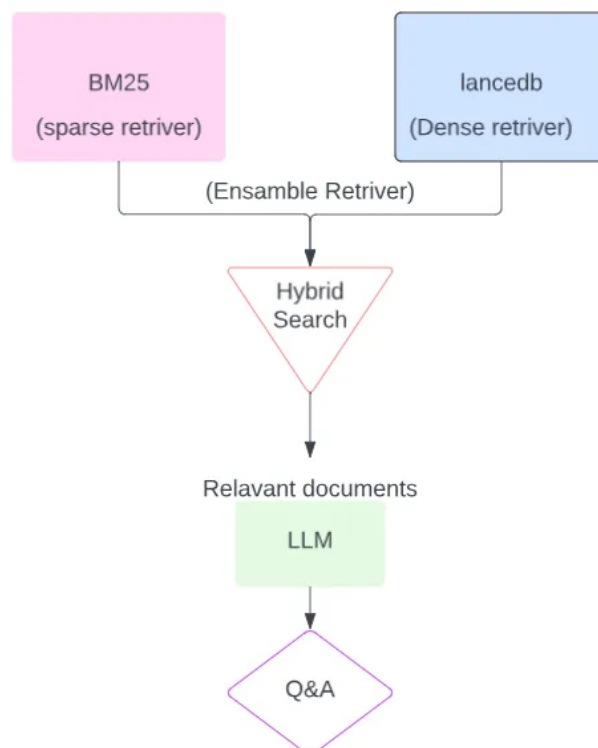
- **实现方法：**针对PDF文档，结合PDFPlumberLoader提取的布局信息，保留文档的层次结构
- **标题识别：**基于字体大小、加粗等特征识别标题，构建文档层次
- **表格处理：**识别表格结构，将表格作为整体进行处理，避免破坏表格语义
- **图文关联：**将图片说明与相应图片关联，保持图文一致性

3. 元数据增强（增强方案）

- **页码追踪：**为每个文本块添加原始页码信息，便于引用溯源
- **章节标记：**记录文本块所属章节，增强上下文理解
- **位置信息：**保存文本块在原文中的相对位置，有助于重建文档结构

2.2.2 混合检索

混合检索模块是系统的核心创新点之一，通过结合多种检索策略提高相关内容的召回率和精确度。



1. BM25检索原理

BM25: an intuitive view

Repetitions of query words → good

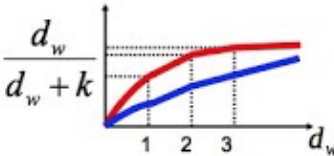
Common words less important

$$\log \frac{P(D | R=1)}{P(D | R=0)} \approx \sum_w \left(\frac{d_w (1+k)}{d_w + k((1-b) + \frac{b \cdot dl}{\text{avg. dl}})} \cdot \log \frac{N - N_w + \frac{1}{2}}{N_w + \frac{1}{2}} \right)$$

More words in common with the query → good

Repetitions less important than different query words

But more important if document is relatively long (wrt. average)

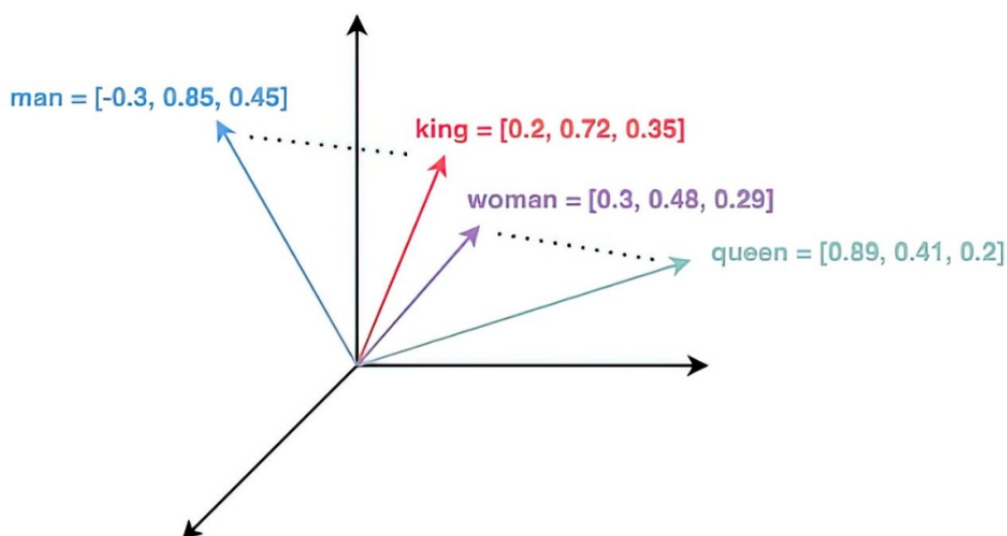


- **算法基础：**BM25(Best Matching 25)是一种基于词频-逆文档频率(TF-IDF)的经典检索算法
- **优势：**擅长精确关键词匹配，对专有名词、术语等检索效果好
- **基础实现：**

代码块

```
1 bm25_retriever = BM25Retriever.from_documents(documents)
2 bm25_retriever.k = retriever_k # 配置返回文档数量
```

2. 向量检索原理



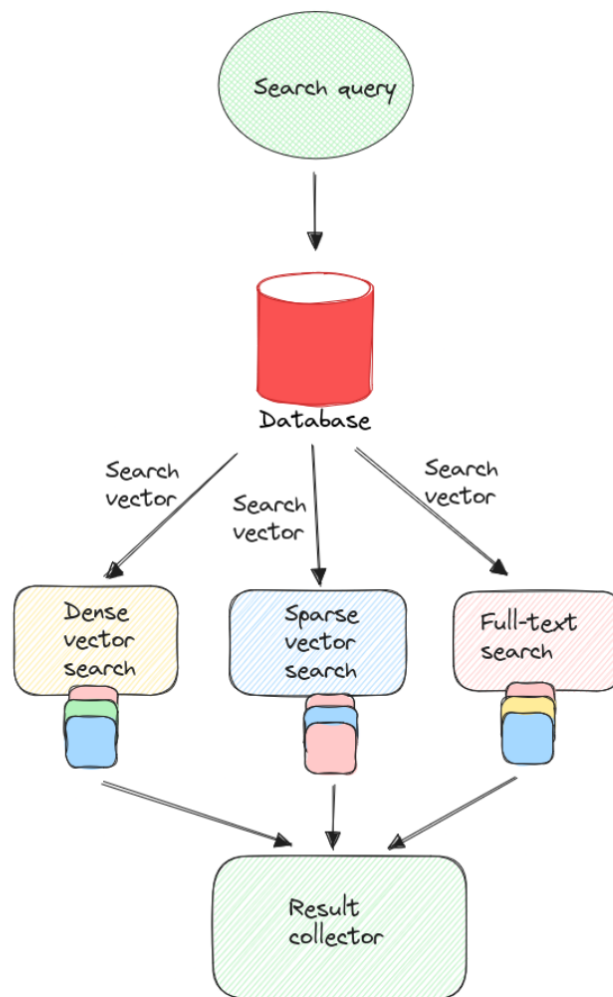
- **算法基础：**将文本转换为高维向量表示，通过计算向量余弦相似度进行语义匹配
- **优势：**能捕捉语义相似性，对同义表达、概念关联等有良好效果

- 基础实现：

代码块

```
1 embedder = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
2 vector_store = FAISS.from_documents(documents, embedder)
3 vector_retriever = vector_store.as_retriever(
4     search_type="similarity",
5     search_kwargs={"k": retriever_k}
6 )
```

3. 可配置权重的多路召回



- 原理：**根据不同问题类型，动态调整BM25（Full-text Search）和向量检索（包括 Dense Vector Search和Sparse Vector Search）的权重比例
- 基础实现：

代码块

```
1 retriever = EnsembleRetriever(
2     retrievers=[bm25_retriever, vector_retriever],
```

```
3     weights=[bm25_weight, vector_weight] # 可配置权重
4 )
```

- **优化策略：**

- 问题类型识别：分析用户问题类型，自动调整检索权重
- 反馈学习：基于用户反馈，优化检索权重配置
- 动态阈值：根据检索结果相似度分布，自适应调整阈值

2.2.3 重排序优化

重排序是对初步检索结果进行二次评估和排序的过程，目的是提高最终结果的相关性。

BGE Reranker是一种基于BERT架构的跨语言重排序模型，专为中英文语义匹配优化。它的目标是对候选检索结果进行精细排序，提升检索的最终精度。它在第一阶段检索（如BGE）基础上，对 Top-K 结果进行更精确的相关性计算。

1. 两阶段检索架构

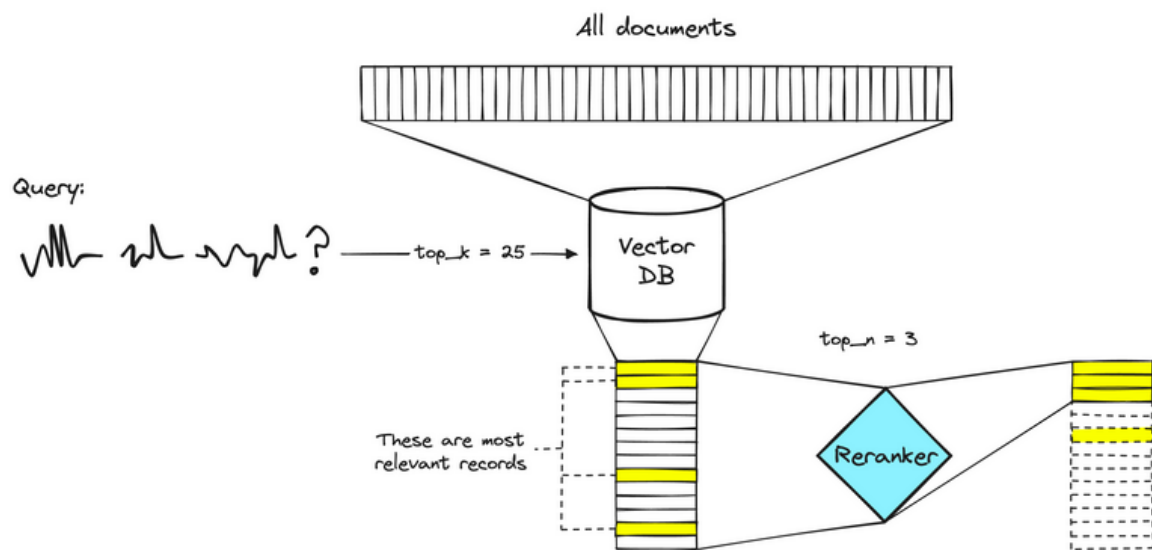
如图所示，重排序采用两阶段检索架构（Two-stage Retrieval System）来提升检索质量：

第一阶段（初始检索）：

- 用户查询（Query）首先通过向量数据库（Vector DB）进行初步检索
- 系统从全部文档中检索出top_k（例如25个）候选文档
- 这一阶段使用双向编码器（bi-encoder）或稀疏嵌入模型高效检索大量文档
- 图中黄色标记的文档代表相关性较高的记录，但它们在初始检索结果中可能排序不够理想

第二阶段（重排序）：

- 初步检索的候选文档被送入重排序器（Reranker）
- 重排序器对每个（**查询，文档**）对进行精确评分
- 最终输出相关性更高的top_n（例如3个）文档
- 图示右侧显示重排序后，黄色高相关文档被集中排在前列



A two-stage retrieval system. The vector DB step will typically include a bi-encoder or sparse embedding model.

2. 交叉编码器与双向编码器的区别

重排序阶段使用的交叉编码器（Cross-encoder）与初始检索阶段使用的双向编码器（Bi-encoder）有本质区别：

- **双向编码器（初始检索）**： $\text{相似度} = \text{Similarity}(\text{Encoder}(\text{查询}), \text{Encoder}(\text{文档}))$
 - 独立编码查询和文档，计算向量相似度
 - 优势：可预计算文档向量，检索速度快
 - 劣势：无法捕捉查询与文档间的交互信息
- **交叉编码器（重排序）**： $\text{相关性分数} = \text{CrossEncoder}([\text{查询}, \text{文档}])$
 - 将查询和文档作为一个整体输入模型
 - 优势：能捕捉查询与文档间的细微交互，评估更准确
 - 劣势：计算成本高，无法预计算，不适用于大规模初始检索

3. BGE Reranker实现与优化

基于两阶段检索架构，我们计划采用BGE Reranker，实现方案如下：

- **候选集大小优化**：
 - 图中显示初始检索的 $\text{top}_k=25$ ，重排序后保留 $\text{top}_n=3$
 - 通过实验确定最佳的 $\text{top}_k/\text{top}_n$ 比例，平衡检索质量和计算效率
 - 实验表明， top_k 通常设为最终所需文档数的5-10倍效果较好
- **分段重排序**：
 - 对于长文档，将其分割为多个段落
 - 分别对每个段落进行重排序评分
 - 选择得分最高的段落作为最终结果

- 这种方法避免了长文档中相关信息被不相关内容稀释的问题
- **动态阈值筛选：**
 - 设置相关性得分阈值（如0.7）
 - 只保留得分超过阈值的文档
 - 当所有文档得分都较低时，可动态降低阈值或扩大检索范围
- **相关性分数归一化：**
 - 实现min-max或z-score归一化
 - 使不同查询间的相关性分数具有可比性
 - 便于设置统一的筛选阈值

代码块

```

1  from langchain_community.retrievers import BM25Retriever
2  from langchain_community.vectorstores import FAISS
3  from langchain_huggingface import HuggingFaceEmbeddings
4  from langchain.retrievers import EnsembleRetriever
5  from langchain_community.retrievers.bge_reranker import BGEReranker
6  # 1. 初始检索阶段 - 混合检索获取候选文档
7  embedder = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-
  L6-v2")
8  vector_store = FAISS.from_documents(documents, embedder)
9  vector_retriever = vector_store.as_retriever(search_type="similarity",
  search_kwargs={"k": 25})
10 bm25_retriever = BM25Retriever.from_documents(documents)
11 bm25_retriever.k = 25
12 # 混合检索器获取初始候选文档集
13 ensemble_retriever = EnsembleRetriever(
14     retrievers=[bm25_retriever, vector_retriever],
15     weights=[bm25_weight, vector_weight]
16 )
17 # 2. 重排序阶段 - 使用BGE Reranker精确评分并筛选
18 reranker = BGEReranker(
19     model_name="BAAI/bge-reranker-v2",
20     top_n=3 # 保留排序后的前3个文档
21 )
22 # 构建完整检索流程
23 retriever_pipeline = ensemble_retriever | reranker
  
```

2.2.4 大模型增强

大模型增强模块负责将检索到的相关内容与用户问题结合，生成准确、连贯的回答。

1. 提示工程原则

- 明确角色定位：将模型定位为“专业文档问答助手”，引导其生成符合预期的回答
- 详细指令说明：提供清晰的回答规则，限制模型只使用检索内容回答
- 结构化输出：指导模型生成结构良好的回答，包括引用标记

2. 核心提示模板

代码块

```
1  prompt = ""
2  你是一个专业的文档问答助手。根据以下提供的上下文信息，回答用户的问题。
3
4  规则：
5  1. 只使用提供的上下文来回答问题，不要添加自己的知识
6  2. 如果上下文中没有足够信息回答问题，直接说"抱歉，我在文档中找不到相关信息"
7  3. 保持回答简洁但信息完整
8  4. 回答中引用出处时，标明来源页码
9
10 上下文信息：
11  {context}
12
13  用户问题：{question}
14
15  回答:""
```

3. 上下文组织优化

- 相关性排序：按相关性排序检索结果，确保最相关内容优先呈现
- 元数据增强：在上下文中包含文档元数据，如来源、页码等
- 格式标准化：统一上下文格式，便于模型处理

4. 引用追踪机制

实现方法：

- 源文档关联：在检索阶段保留文档元数据，包括文件名、页码等
- 引用标记生成：指导模型在回答中明确标记信息来源
- 后处理验证：系统自动检查回答中的引用是否与检索文档匹配

可视化展示：

- 高亮引用段落：在用户界面中高亮显示回答中引用的原文
- 来源链接：提供快速查看原文上下文的链接
- 置信度指示：显示每个引用部分的相关性得分

3. 技术难点与解决方案

3.1 本地模型性能受限

难点分析：本地部署的小型模型(如DeepSeek-r1-1.5b)在推理能力上不如大型云端模型，可能影响回答质量。

解决方案：

- **优化分块策略：**精细调整分块大小和重叠度，提供更精准的上下文
- **渐进式模型升级：**从小模型起步(1.5b)，根据硬件条件逐步升级至更大模型(7b/8b)
- **模型量化技术：**采用GGUF格式进行4-bit或8-bit量化，减少内存占用
- **上下文压缩：**使用句子嵌入相似度筛选最相关段落，减少无关信息

3.2 检索质量与相关性不平衡

难点分析：检索系统需要在召回率与精确率之间取得平衡，单一检索方法难以满足多样化的查询需求。

解决方案：

- **混合检索策略：**结合BM25关键词匹配和向量语义检索的优势
- **重排序机制：**使用BGE Reranker对初步检索结果进行精确排序
- **动态权重调整：**根据查询特征自适应调整检索权重
- **查询改写：**使用模型对原始查询进行扩展或改写，提高检索效果

3.3 生成内容及上下文可靠性不稳定

难点分析：大模型可能产生幻觉或超出检索内容的回答，降低系统可靠性。

解决方案：

- **引用机制：**要求模型在回答中标明信息来源，实现可溯源性
- **不确定性表达：**在缺乏足够信息时，引导模型表达不确定性而非猜测
- **后处理验证：**自动检查回答中的事实是否在检索文档中有依据
- **反馈循环：**收集用户反馈，持续优化系统

4. 实施计划

4.1 阶段划分

1. 基础框架构建（第11-12周，5.9-5.16）

- **搭建基本RAG流程：**实现文档加载、向量化和基础检索
- **实现PDF文档处理：**开发PDF解析和结构化提取功能
- **部署本地模型：**使用Ollama部署DeepSeek-r1模型

2. 核心功能开发（第12-14周，5.16-5.30）

- 实现混合检索策略：集成BM25和向量检索，开发权重配置机制
- 优化文档分块：实现结构感知分块和元数据增强
- 设计提示模板：优化提示工程，提升回答质量

3. 系统优化与评估（第14-15周，5.30-6.6）

- 添加重排序机制：集成BGE Reranker进行结果重排序
- 实现多轮对话：支持上下文相关的连续问答
- 构建评估框架：开发自动化测试和性能评估工具

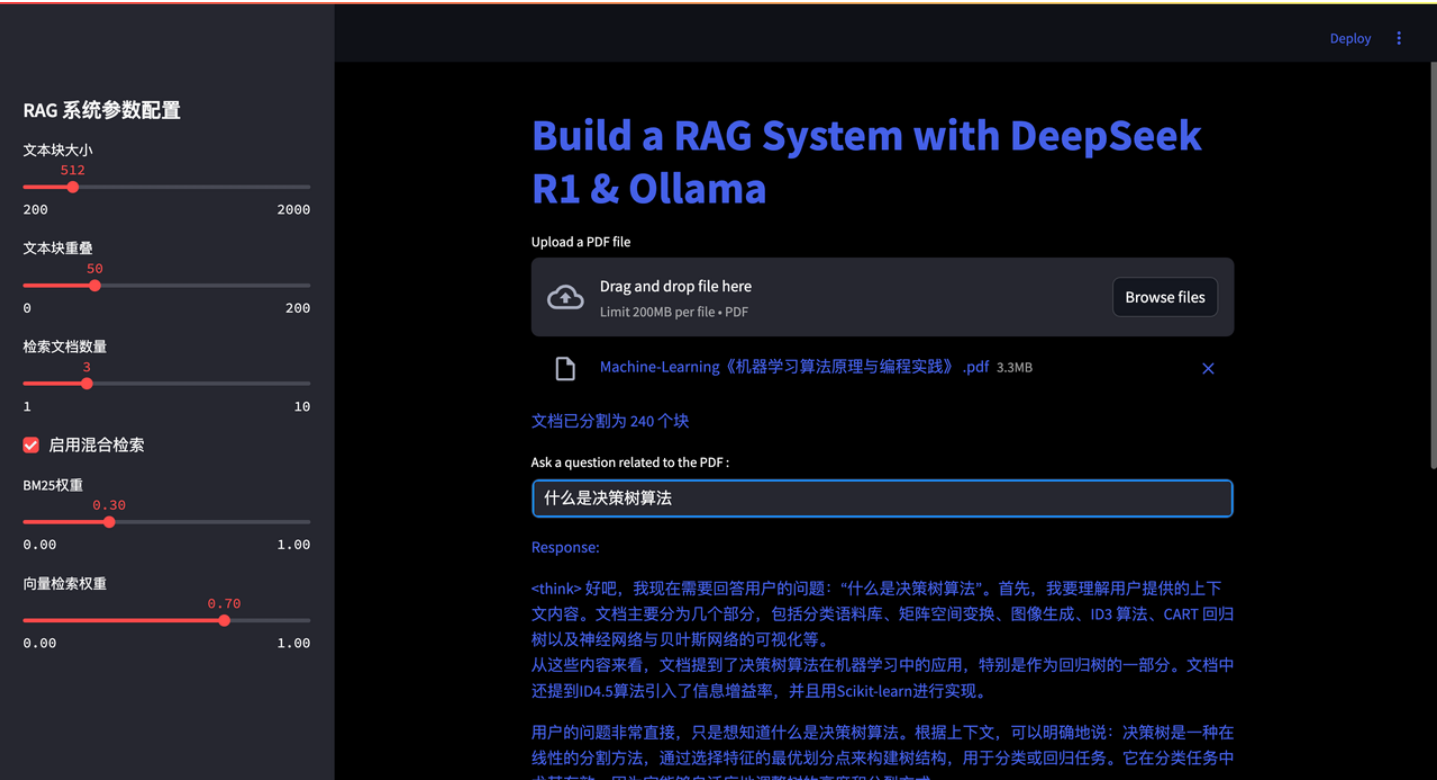
4. 功能扩展与完善（第15-16周，6.6-6.13）

- 支持更多文档格式：扩展对Markdown、Word等格式的支持
- 优化用户界面：改进Streamlit界面，提升用户体验
- 系统性能调优：优化内存使用和响应速度

4.2 里程碑与交付物

里程碑1：可运行的MVP（Minimum Viable Product）版本

- 时间点：第12周周五（5.16）
- 核心功能：基本PDF处理、向量检索、简单问答
- 交付物：开题报告、基础系统代码（如需要）





里程碑2：完整RAG系统

- 时间点：第14周五（5.30）
- 核心功能：混合检索、优化分块、引用追踪
- 交付物：系统中期报告

里程碑3：优化后的最终系统

- 时间点：第16周截止（6.20）
- 核心功能：重排序机制、多轮对话、多格式支持
- 交付物：完整系统代码、技术文档、演示PPT

5. 预期成果与评估方法

5.1 预期成果

- 一个完整可用的本地化智能问答系统：支持PDF文档处理、混合检索和精准问答
- 针对RAG技术的实验分析报告：包括不同检索策略、分块方法的对比分析
- 系统性能与用户体验评估数据：回答质量、检索精度和系统性能数据

5.2 评估方法

检索质量评估

- **Precision@k**：前k个检索结果中相关文档的比例
- **Recall@k**：成功检索到的相关文档占所有相关文档的比例

- **MRR (Mean Reciprocal Rank)**: 首个相关文档排名的倒数平均值
- **nDCG (Normalized Discounted Cumulative Gain)**: 考虑排序质量的评估指标

回答质量评估

- **BLEU/ROUGE**: 与参考答案的文本相似度
- **人工评分**: 基于准确性、完整性、相关性的人工评分
- **引用准确率**: 回答中引用内容与原文的匹配度
- **幻觉检测率**: 检测模型生成的不在原文中的内容比例

系统性能评估

- **响应时间**: 从提问到获得回答的平均时间
- **资源占用**: CPU、GPU、内存使用情况
- **可扩展性测试**: 随文档量增加的性能变化
- **用户体验评分**: 基于交互流畅度、界面友好性的用户评价