Experiment 4

基于Streamlit和Milvus构建极简RAG系统

为什么学习RAG系统?

背景:

- 数据挖掘的核心是提取有用信息。
- 大语言模型（LLM）在问答任务中常面临“幻觉”问题（生成错误信息）。
- RAG（Retrieval-Augmented Generation）结合检索与生成，提升回答准确性。

实验目标:

- 理解RAG的工作原理。
- 学习Milvus向量数据库的向量存储与检索。
- 使用Streamlit快速构建交互式应用。

RAG定义

检索增强生成（Retrieval-Augmented Generation）。

工作流程：

1. 将文档切分为小块（chunks）。
2. 使用嵌入模型（例如Sentence-BERT）将文本转为向量。
3. 存储向量到向量数据库（Milvus）。
4. 用户查询 → 向量检索 → 召回相关文档。
5. 将检索结果与查询输入LLM，生成回答。

优势：

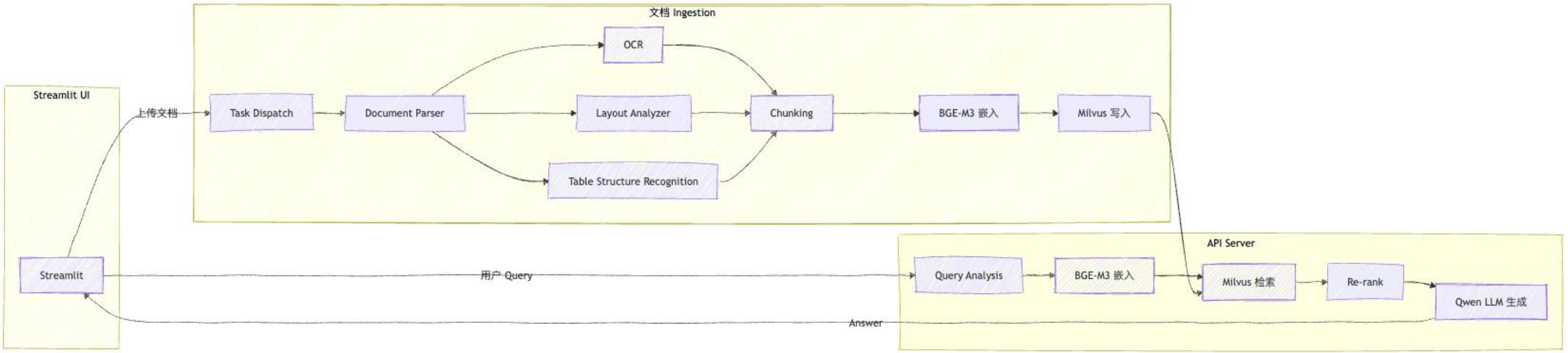
- 减少LLM幻觉。
- 支持动态知识库更新。
- 适合处理私有数据。

实验工具介绍

使用的工具

- Milvus：
  - 开源向量数据库，专为高维向量检索设计。
  - 特点：高性能、支持多种索引（HNSW、IVF等）。
- Streamlit：
  - Python库，用于快速构建Web应用。
  - 特点：纯Python、无需前端知识、交互性强。
- BGE-m3：
  - 向量嵌入模型

系统架构


文档 Ingestion
OCR
Streamlit UI
上传文档
Task Dispatch
Document Parser
Layout Analyzer
Chunking
BGE-M3 嵌入
Milvus 写入
Table Structure Recognition
API Server
Streamlit
用户 Query
Query Analysis
BGE-M3 嵌入
Milvus 检索
Re-rank
Qwen LLM 生成
Answer

数据来源

微信公众号数据集（私有，请勿上传到外部公共平台）

采用爬虫获得的微信公众号数据集，html格式，包含相信的文章信息和内容

主要是血液疾病相关，但也存在大量噪音，需要预处理

实验要求

目标：利用以上工具构建完整的RAG系统。

1. 文档过滤：

- 如何实现良好的数据预处理

2. 文档切分：

- 如何对文档切分，获得比较好的知识召回效果。

3. Re-ranking：

- 是否可以有更好的reranking步骤来优化搜索结果。

4. 迭代过程：

- 是否可以用迭代过程来优化搜索？ 添加多轮对话功能，根据用户需求迭代搜索结果进行优化最终回答

5. 存储结构优化（可选）： GraphRag/NodeRAG

- 是否可以用图结构来优化搜索?