### 复试附加问题

1. **知识库文档中出现文字、图、表等结合的情况，如何做好数据预处理？**

在本次复试题目要求中所使用的外部数据格式为pdf，所以就需要选取相应的pdf解析器。本demo中使用的是langchain社区开发包langchain\_community中的文件解析器document\_loaders中的**PyPDFLoader**，其可以很好地解析pdf文件中的文字部分。但是往往外部知识库中还会提供很多的图或表，如本demo中提供的LLM.pdf文件，仅仅使用文字解析并不能很好地工作。

所以对于其中的**图片部分**我选择使用**OCR**技术，来光学字符识别其图片中的内容。在本demo中使用的PyPDFLoader可以通过传入extract\_images=True参数，来开启OCR功能，其依赖的rapidocr-onnxruntime软件包可以通过pip进行安装（README.md中已给出）。

而对于**表格部分**，其实有专门的pdf解析器可以提取表格，比如**pdfplumber**，但方便起见我最终还是选择把pdf中的图表看做图片，也使用**OCR**技术进行提取。

1. **如何优化检索模块的效率？**

首先我们可以通过**选择合适的向量数据库**，如 FAISS、Pinecone、Milvus 等。这些引擎专门为高维向量检索设计，能够在大规模数据下实现快速检索。在本demo中使用的便是Faiss向量数据库的cpu版本。

另一种方法便是可以通过**Re-Rank重排序方法**，其就是把向量数据库中检索到的结果（本demo中是相似度前5的结果），再进行一次排序，并筛选出更小一部分结果，比如前2条。从而使RAG提供给LLM的contents范围更小。但这里其实涉及到一个问题就是为什么不直接选择前2条，还要再进行排序？这是因为向量数据库检索结果，一般会使用向量索引，向量索引的算法（常见的为HNSW算法）都有一定的随机性，会导致除了第一条以外，其他的结果不太准确，因此需要再进行排序。

1. **如果知识库非常大，如何处理性能问题？**

**其一**是针对PDF这种格式的知识库，我们可以在处理时通过**lazy\_load()**方法来**逐页地处理知识库**，而不是一次性一股脑全部丢给解析器。

**其二**是可以使用**分布式向量库**，通过将**知识库分段存储到分布式系统中的多个节点**上，使得检索过程可以在多个节点上**并行执行**，不但解决了单节点放不下的问题，也通过并行提高了检索效率。

**其三**便是使用**缓存机制**，通过对高频出现的查询结果应用缓存（如Redis、Memcached），来将常见查询的检索结果缓存起来，避免重复计算；亦或是使用向量缓存技术，将经常使用的文档向量缓存到内存中（例如使用LRU缓存策略），减少向量化计算的时间（空间换时间）。

1. **谈谈你对 RAG 技术的理解，及其在实际应用中的优势和局限性。**

RAG是一种将**检索与生成相结合**的技术。其核心思想是：**在生成之前先检索**。首先从外部知识库或文档中检索相关信息，然后将检索到的内容输入到生成式模型中（类似于使用ChatGPT时，我们直接提供给大模型的那一段“特定输入”，比如说我想让大模型总结提炼一段文本，那这段给定的文本就是“检索到的内容”），生成更具上下文相关性的答案。

RAG的**优势**包括：**增强了**生成模型的**知识范围**，使模型在**专业领域能力更强**；**提高**生成结果的**准确性**来**减轻“幻觉”**；能**动态适应新信息**保证向量数据库可以随时更新；其能够显著**减少训练（微调）的成本**。尤其是在**企业**中需要的**L1/L2垂直领域大模型**，往往**需要特定领域的知识**（比如贵公司的光伏发电、风力发电等），但这无可避免的需要**大量的GPU资源**去进行再训练（微调）来让大模型“掌握”这种专业领域的知识。而RAG技术就**有效的缓解了这点**，**无需微调**即可接入专业知识库。

RAG的**局限性**包括：其**LLM**整体的**生成质量**取决于**知识库的质量**并与其**强相关**，只有知识库做得好，LLM才会回答得好；**知识库**会**不断膨胀**，虽然有上述提到缓解知识库非常大的方法，但这仍是一个显著问题；RAG的**检索**会**带来额外计算量和开销**，且每进行一次对话都需要去计算相似度然后检索，在一些需要**高实时性的场景**，可能还**需要**做**优化**才能落地。