初见RAG

于 2025-07-08 18:26:37 发布

编辑

GitCode-AI社区 文章已被社区收录

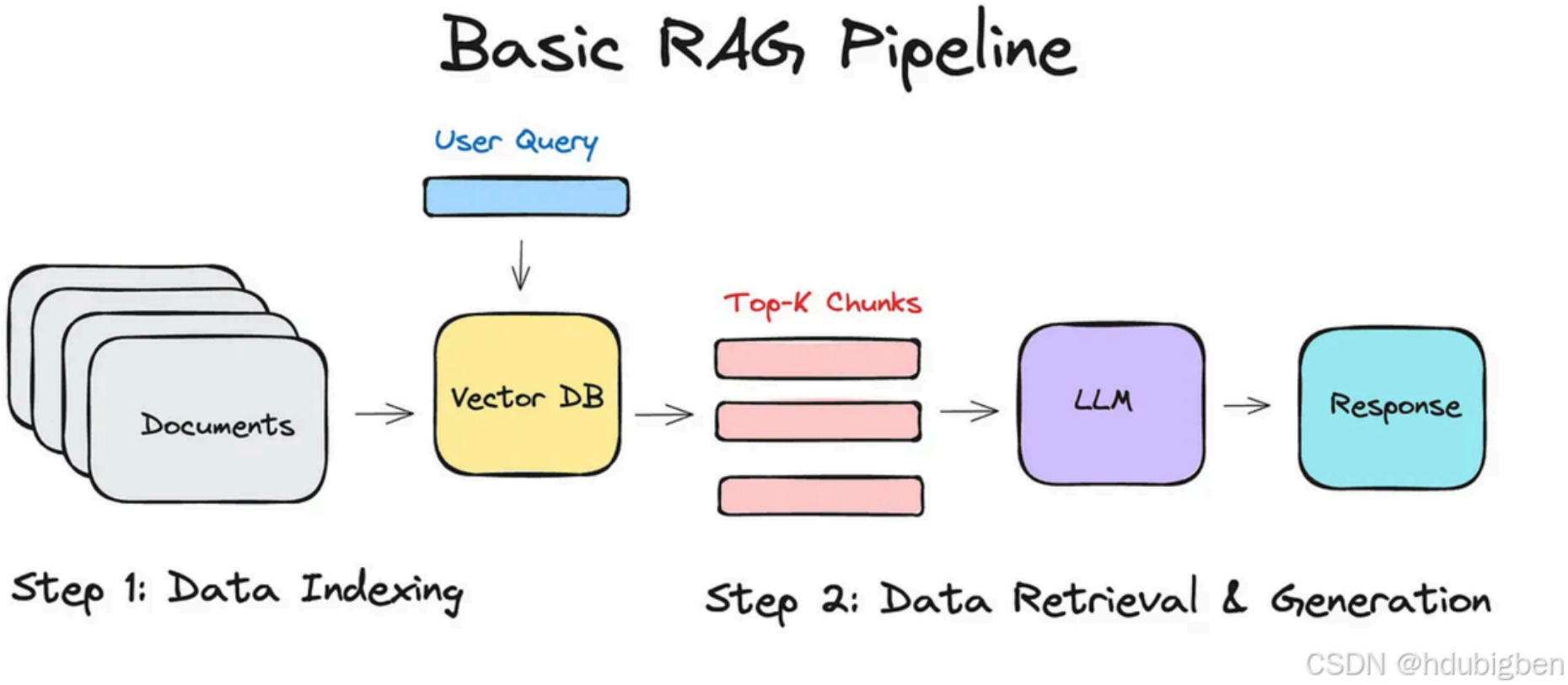
加入社区

AI+大模型 专栏收录该内容

3 篇文章

部署运行你感兴趣的模型镜像 一键部署

RAG，全称 Retrieval-Augmented Generation，是一种结合“检索”与“大 语言模型 生成”的技术架构。它的核心思想是：“在生成答案之前，先检索相关文档，再把这些文档作为上下文喂给语言模型进行生成。”通过引入外部知识源，RAG 弥补了语言模型的“知识过时”、“上下文长度受限”等问题，实现“可更新、可控”的问答能力。



构建知识库

在 RAG 项目中，构建知识库 是实现高质量问答的基础环节，它的目标是：将结构化或非结构化原始资料，转化为可被检索与理解的“向量化知识片段”，供大语言模型（LLM）在生成前参考。

一句话理解：构建知识库 = 收集知识 → 清洗预处理 → 切分 → Embedding → 存入向量数据库。其中最重要也是影响最大的一步就是文本切块（Chunking）。

为什么切块?

- Embedding 模型输入有限（如 512 tokens）
- 长文检索粒度太粗，难以精准定位答案

- chunking 能提升检索粒度，使模型更容易检索到相关内容

常见策略：

方法	描述
固定长度	每 300-500 字符切一块
滑动窗口	重叠切块（如 500 长度 + 50 重叠）
按结构切分	按段落、标题、章节进行（语义分段）

知识库设计的关键建议

项目	建议
切块大小	300–500 tokens，中文建议按字数切
重叠设置	保证语义连贯，设置 overlap=50–100
切割方式	尽量按自然段、句号、标题进行语义切块
Embedding 模型选择	中文用 bge-*, qwen-*, 英文可用 text-embedding-3
数据存储方式	向量 + metadata（包含原文/页码等）
多文档组织	metadata 记录文档来源、章节等信息
上下文拼接	控制拼接后的总 token ≤ 模型最大长度（如 4K）

Embedding → Rerank → LLM推理

过程梳理

这三者在整个系统中的先后顺序和职责如下：

	作用	特点	工具	主要指标
Embedding 检索（粗排）	**快速找到大概相关的候选文档** 把用户的 query（问题） 和 文档库中的所有文档 变成向量。 通过向量之间的**相似度（如余弦相似度）**进行快速召回。 通常使用向量数据库（如 FAISS, Milvus, Weaviate）进行 ANN（近似最近邻）搜索。	高效，但不是最精确（粗排） 通常返回 Top 10 ~ Top 50 个候选片段	text-embedding-ada-002, gte-base, Qwen-Embedding	覆盖率、Recall
Rerank 重排序（精排）	**对粗排出的候选文档进行更精准的相关性排序** • 输入是 query 和前面召回的文档片段对。 • 模型是一个文本对分类器，给每一对 query + doc 打一个相关性分数。 • 根据这个分数重新排列文档，保留 top-k（例如 top-5）。	更准确，但更慢（通常几十毫秒/条） 类似于“看了一眼全文再判断哪篇文章更相关”	gte-rerank, bge-reranker, colbert, cross-encoder 等	精度、NDCG

LLM 推理（阅读 + 回答）	**结合用户 query 和排序靠前的文档，生成回答** 将 rerank 后的 top 文档作为上下文拼接，连同问题输入 LLM。 LLM 再输出最终回答、摘要、分析等。	属于“阅读理解”+“问答生成” 最慢但最智能的部分	GPT-4, Qwen2.5, Claude, Gemini, ChatGLM, 等等	答案质量、上下文利用度

🔁 三者之间的关系图示：

AI写代码 | 复制

```
        [ 用户问题 Query ]
                ↓
          ┌──────────────┐
          │  Embedding 模型 │ → 向量化
          └──────────────┘
                ↓
        (向量数据库召回 Top-N)
                ↓
          ┌──────────────┐
          │  Rerank 模型   │ → 文本对打分排序
```

展开

常用模型

系统推理模型

		代表	能力	用途
OpenAI	glm-4	glm-4 是 GPT-4 架构的多模态模型，也被称为 GPT-4o（其中 “o” 代表 Omni，多模态）	• 能同时理解文本、图像（包括照片、截图、图表、文档等）。	最推荐使用的通用模型。

	• 是目前（截至 2025 年）OpenAI 最先进、最通用的模型。 • 支持更快响应时间和更低成本，相比 GPT-4-turbo。	在 ChatGPT 中默认启用（Plus 用户）。 可以用于图像识别、数据可视化、截图理解、PDF解析等任务。
glm-z1	这是一个内部测试版或变体，OpenAI 没有公开详细文档，可能是： • GPT-4o 的一个实验性分支。 • 或者一个专门为视频/音频/感知等多模态能力优化的模型。	（推测）： 名称类似于内部测试或特定优化（比如用于处理 Sora 生成的视频的辅助模型）。 “z1” 可能代表第一个 zero-shot 融合模型。	目前 不对外公开，除非你在参与某个特殊的 OpenAI 测试项目。

	glm-3-turbo	GPT-3.5 架构下的多模态变体。	• 支持图像输入，但理解能力弱于 GPT-4o（glm-4）。 • 响应速度快，成本低。 • 多用于轻量图像任务（比如简单的图片 OCR、图表识别）。 • 上线时间：2023 年末–2024 年初测试使用过，但后续逐步被 GPT-4o 替代。	• 适合预算敏感、对图像处理要求不高的场景。 • 在图像理解复杂度中明显弱于 GPT-4o。
		所属代际	专长方向	用途场景
通义千问 如何使用 Function Calling 功能_大模型服务平台百炼(Model Studio)-阿里云帮助中心	DeepSeek			
	qwen-coder	Qwen 1	通用大模型	代码生成、调试、解释任务
	qwen-math	Qwen 1	数学推理	奥数、复杂数学公式理解与解题
	qwen-max	Qwen 1	通用大模型	最强通用 LLM，适合问答、写作、代码等通用任务
	qwen-plus	Qwen 1	通用大模型	介于 max 与 turbo 之间
	qwen-turbo	Qwen 1	通用大模型	更快更轻量，适合部署和成本敏感场景

	qwen-vl-plus	Qwen 1	图文多模态	图片理解、图文问答、多模态场景
	qwen2-math	Qwen 2	数学推理	数学能力增强版，适合解题
	qwen2.5-vl	Qwen 2	图文多模态	多模态场景升级版，强于 Qwen-vl-plus
	qwen2.5	Qwen 2	通用大模型	更强推理能力和对齐能力，通用任务优选
	qwen3	Qwen 3	通用大模型	最新千问旗舰，能力对标 GPT-4/Claude 3

Embedding 模型

设置知识库文档嵌入处理的默认模型，检索和导入知识库均使用该Embedding模型进行向量化处理，切换后将导致已导入的知识库与问题之间的向量维度不一致，从而导致检索失败。

		特点	使用场景
OpenAI	embedding-2（默认）	快、稳定、成本低（便宜很多） 输出维度为 1536	如果不指定模型，OpenAI embedding API 默认使用它 适用于 RAG、语义检索、FAQ 匹配 相比新模型（如 embedding-3），在部分多语言、长文本表现略差
	embedding-3	更高质量的语义编码，更适合复杂语义搜索 支持可选维度输出（256 / 1024） 多语言支持更好 兼容 OpenAI 最新 API 格式	推荐用于对精度要求高的场景，比如复杂问题召回、长文档 RAG、聊天记忆检索

通义千问	text_embedding	中文语义理解和对齐效果优于 OpenAI 模型（尤其在中文召回） 维度通常为 1024（也可能有 768/1536 版本） 免费开源，适合本地部署	如果你的主任务是中文语义搜索、中文RAG，推荐使用它 也适合和 Qwen 系列 LLM（如Qwen2.5/Qwen3）组合使用
	text-embedding-v1		

Rerank 模型

重排序模型将根据候选文档列表与用户问题语义匹配度进行重新排序，从而改进语义排序的结果

百度（Baidu）	gte-rerank-v2	
	gte-rerank	

常见后缀含义

后缀词（关键词）	含义/全称	主要作用或区别点
Instruct	Instruction-tuned	已进行“指令微调”，适合问答、对话、指令任务
Chat	Chat-tuned	与 Instruct 类似，多用于对话风格优化
Plus	Pro / 企业增强版	比基础版本更强大，可能包含更大训练数据、更长上下文
Turbo	高速版（如 GPT-4-turbo）	性能优化版本，速度更快，成本更低
Max	大容量能力最大化（如 Qwen-max）	上下文最大、精度最高，可能用于多模态或复杂推理
VL	Vision-Language（多模态）	支持图文输入（图像 + 文本）
Flash	轻量极速版	小模型中做极致推理速度优化（适合移动端/边缘部署）

Distill	蒸馏版	从大模型压缩得到的小模型，保留能力、体积更小
Air	轻量级部署优化	强调小、快、便携，适合设备部署（类似 Flash）
MoE	Mixture of Experts	混合专家架构，推理时只激活部分专家子网络

思考

1. 如何提升 RAG 系统的回答准确性?

常见优化手段：

- ✅ 更换高质量的 Embedding 模型（如 bge-m3, Qwen-Embedding, E5）
- ✅ 使用 Reranker 模型对召回结果重新打分（如 gte-rerank-v2）
- ✅ 文档切块策略更合理（如语义分段）
- ✅ 使用较大的 LLM 或 Finetuned-Instruct LLM（如 Qwen2.5-Instruct）
- ✅ 对 LLM 使用提示词工程（Prompt Engineering）提升引导效果
- ✅ 多轮交互中保留对话上下文（Memory）
- ✅ 引入工具调用，如搜索、数据库查询（即 RAG+Tool-augmented）

2. 多轮问答场景下，RAG 如何保持上下文一致性?

- 使用 对话历史摘要：将前几轮对话浓缩后作为系统提示加入 prompt
- 保持短期上下文窗口：保留前几轮 QA，限制 token 长度
- 引入 向量记忆存储（vector memory）：将用户历史问答做向量化匹配上下文
- 使用 RAG + Memory 架构（如 LangChain、LlamaIndex Memory 模块）

3. 向量数据库在 RAG 中扮演什么角色？推荐哪些？

向量数据库用于存储文档的向量表示，并在检索阶段进行相似度搜索（ANN）。

- FAISS（轻量、快速、本地部署）
- Milvus（支持分布式、企业级）
- Weaviate（有 REST API、内置 Rerank 支持）
- Qdrant（轻量支持 Docker + RAG Friendly）

4. 如何评估一个 RAG 系统的效果？

可从以下维度进行评估：

- 回答质量（准确率）：人工或自动打分（BLEU, Rouge, GPT judge）
- 文档命中率（Recall@K）：query 是否检索到正确文档
- 排序效果（NDCG, MAP）：Rerank 模型对文档的排序质量
- 响应延迟（Latency）：整体问答系统的响应速度
- 上下文利用率：LLM 是否有效利用传入文档内容