ClickHouse与大模型

生产级RAG架构

滴普科技-陈峰

CATALOG
目录

01

RAG架构概述

...

大模型特点——训练难

成本高

需要大量的计算资源、大量的存储空间、消耗大量的电力

时间长

大语言模型的训练过程通常需要数周甚至数月的时间。在这个过程中，需要一队专业的研究人员来监督训练过程，调整模型参数，以及处理可能出现的问题。

高质量数据

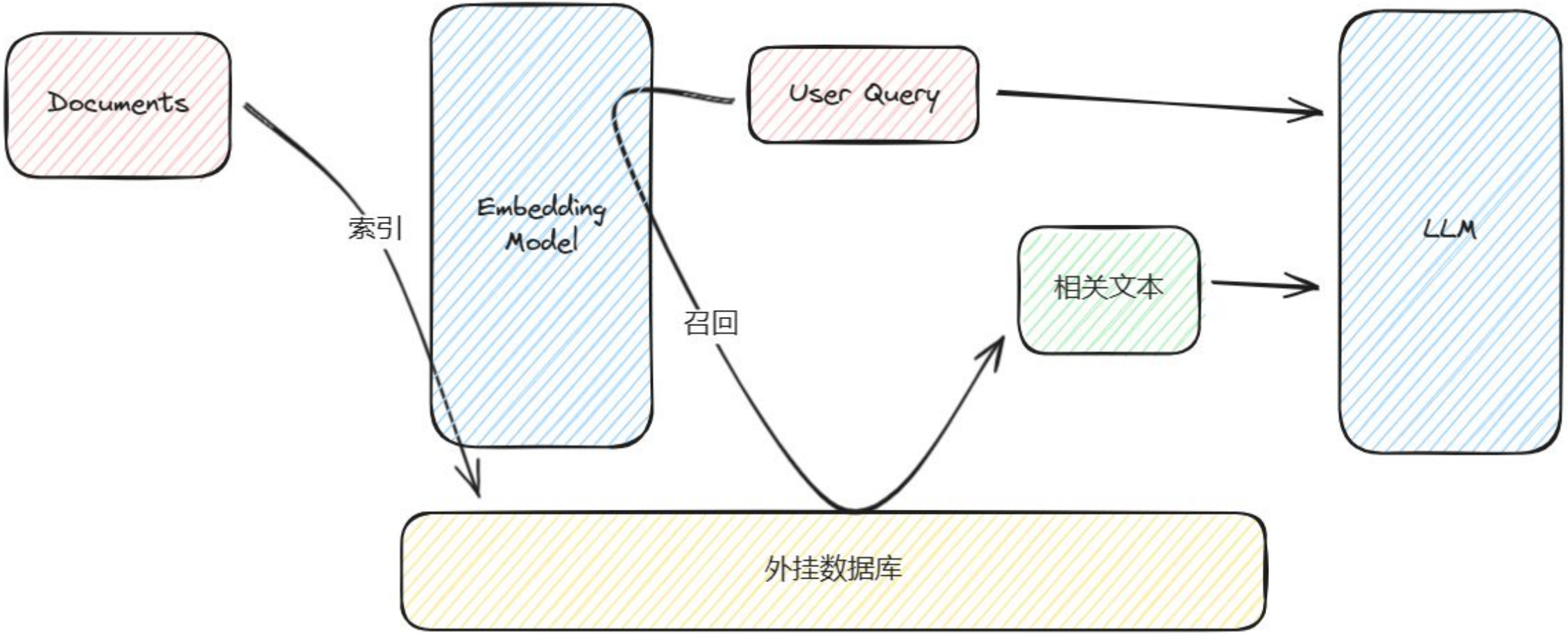
大语言模型的训练需要大量的标注数据。对于一些任务，如机器翻译或问答，获取高质量的标注数据可能需要大量的人力和 时间。

大模型的缺点-知识库滞后

大语言模型的训练涉及到海量的文本数据，并需要大量的计算资源和时间。这意味着，一旦模型训练完成，它的知识库就相当于在训练结束时“冻结”了。换句话说，模型的知识库无法实时更新或接收新的信息。

这种情况在处理涉及到最新事件、新兴技术或正在发展的情况时可能会成为一个问题。例如，如果一个新的科学发现在模型训练结束后发表，那么模型就无法提供这个新发现的相关信息。这就限制了大语言模型在需要最新、实时信息的应用场景中的效用。

典型的RAG架构

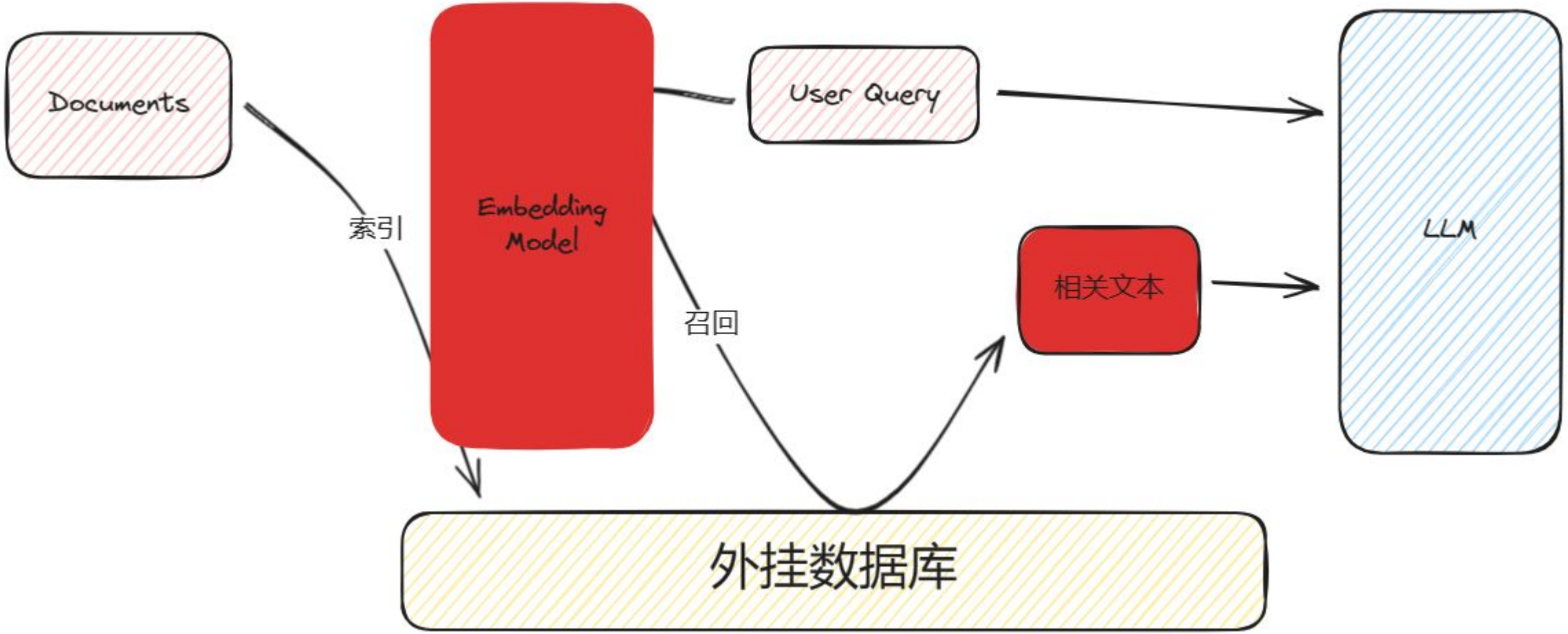


RAG的优点

不需要改动模型，只是外挂一个知识库和检索工具即可。

增加知识只需要在知识库中操作即可

RAG的缺点

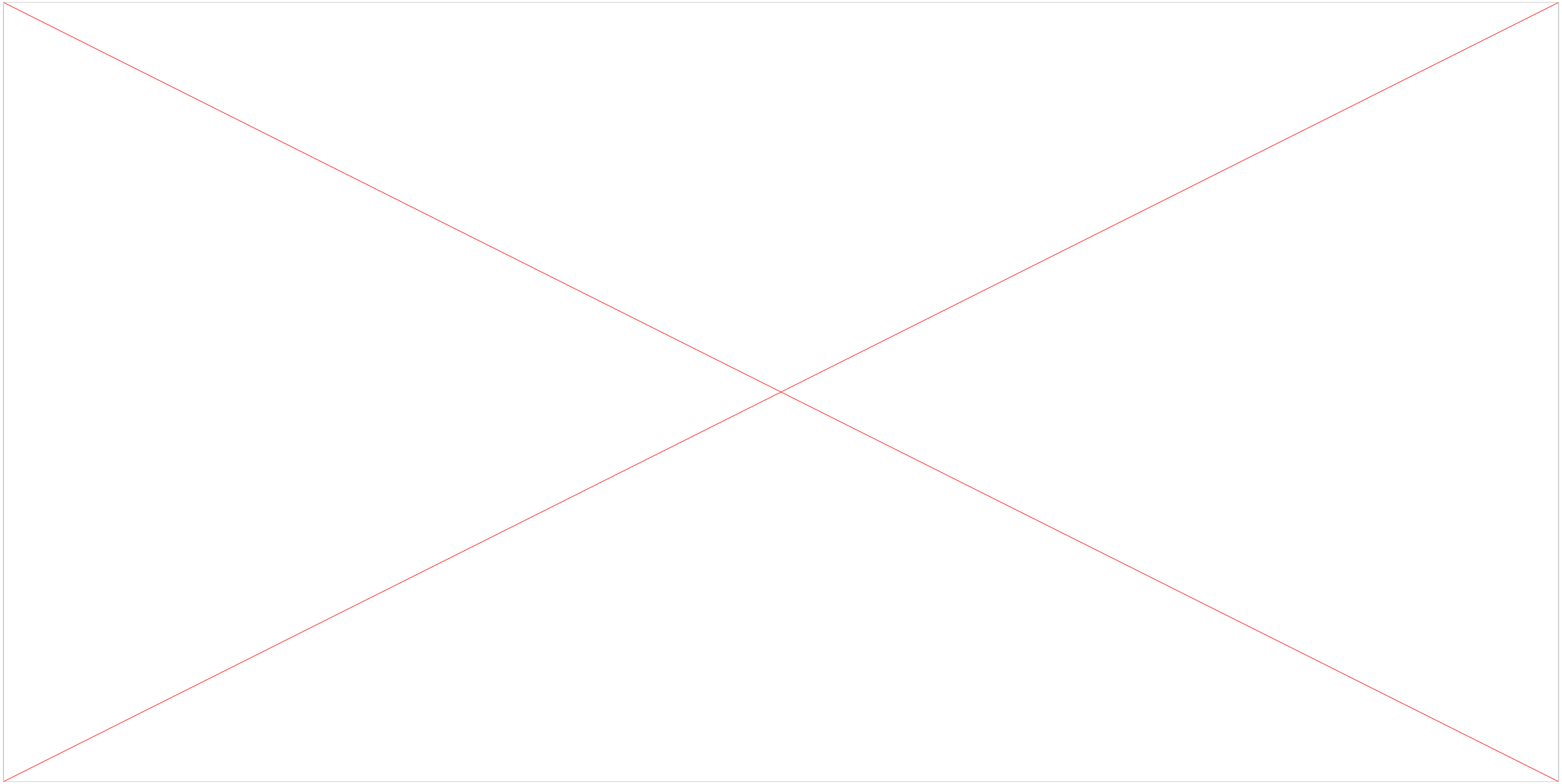


02

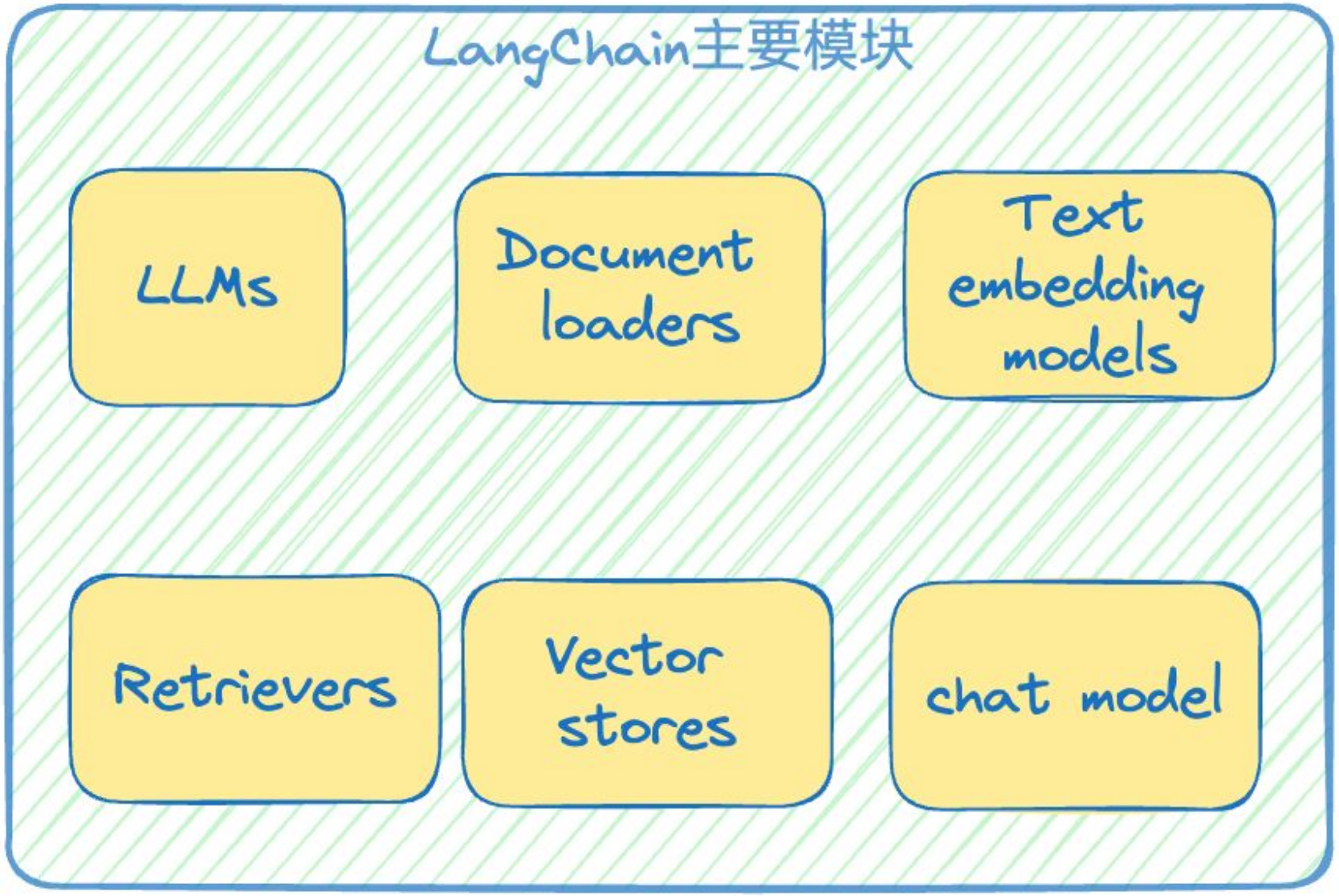
LangChain

...

Langchain简介



LangChain主要模块



llms代码

```
from langchain.embeddings import AzureOpenAIEmbeddings

os.environ["AZURE_OPENAI_API_KEY"] = AZURE_OPENAI_API_KEY
os.environ["AZURE_OPENAI_ENDPOINT"] = AZURE_OPENAI_ENDPOINT

embeddings = AzureOpenAIEmbeddings(
    azure_deployment='text-embedding-ada-002',
    openai_api_version="2023-05-15",
)
```

```
from langchain.chat_models import AzureChatOpenAI
from langchain.schema import HumanMessage,SystemMessage

model = AzureChatOpenAI(
    openai_api_version="2023-05-15",
    azure_deployment="gpt-35-turbo-16k",
)
```

1. 使用AzureOpenAI提供的服务
2. 使用text-embedding-ada-002嵌入模型
3. 使用gpt-3.5-16k作为生成模型

保存到向量数据库

```
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter

book_path="book1.txt"
loader = TextLoader(book_path)
documents = loader.load()

# 将文档按照章节切块
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
docs = text_splitter.split_documents(documents)

# 保存到向量数据库
from langchain.vectorstores import Clickhouse
db = Clickhouse.from_documents(docs, embeddings)
```

检索并生成回复

```
query = "激励相容是什么"
# 检索相关内容
some_query_result = db.similarity_search(query)

# 构造prompt
prompt_message = SystemMessage(
    content="你是一个文档总结机器人，请根据文档内容回答用户问题。文档内容为："+recall_result
)
human_message=HumanMessage(content=query)
# 调用LLM获得回复
resp = model([prompt_message, human_message])


print(f"AI的回答是：{resp.content}")
```

LangChain内置的向量数据库

01

chromadb

本地数据库

02

Pinecone

云服务

03

ClickHouse

单击此处添加文本具体内容，简明扼要的阐述您的观点。

langchain对ClickHouse的支持

```
CREATE TABLE IF NOT EXISTS default.langchain(
    id Nullable(String),
    document Nullable(String),
    embedding Array(Float32),
    metadata JSON,
    uuid UUID DEFAULT generateUUIDv4(),
    CONSTRAINT cons_vec_len CHECK length(embedding) = 1536,
    INDEX vec_idx embedding TYPE annoy('L2Distance',100)
    GRANULARITY 1000
) ENGINE = MergeTree ORDER BY uuid SETTINGS index_granularity = 8192
```

03

性能测试

...

测试集基本概括

- 选择了22本图书
- 按照chunk_size=1000切割
- 共17889条chunk
- 源文件约44M
- 使用Azure OpenAI的text-ada-embedding-002作为向量化模型，输出维度为1536
- 保存到ck中后，表大小为268M。相同体积的chroma是465M

性能概览

22本书，17889条chunk，hunk_size=1000（约1788万字），268M

	问题	来源	ClickHouse时间(秒)
1	四大公害的诉讼是哪四个？	白夜行	0.11623215675354004
2	什么是激励相容？	置身事内	0.1509778118133545
3	柯克霍夫原则（Kerckhoff's rinciple）是什么	Java加密与解密的艺术	0.0968079566955566 4
4	sdsclear有什么优点	Redis设计与源码分析	0.11394429206848145

平均值：0.11秒

性能概览

2本书，179条chunk，hunk_size=1000（约17万字），2.5M

	问题	来源	ClickHouse时间(秒)
1	四大公害的诉讼是哪四个？	白夜行	0.025397539138793945
2	什么是激励相容？	置身事内	0.02760815620422363
3	柯克霍夫原则（Kerckhoff's rinciple）是什么	Java加密与解密的艺术	0.029270172119140625
4	sdsclear有什么优点	Redis设计与源码分析	0.023684024810791016

平均值：0.026秒

性能概览

220本书，198892条chunk，hunk_size=1000（约2亿字），2.9G

	问题	来源	ClickHouse时间(秒)
1	四大公害的诉讼是哪四个？	白夜行	0.6702959537506104
2	什么是激励相容？	置身事内	0.727811336517334
3	柯克霍夫原则（Kerckhoff's rinciple）是什么	Java加密与解密的艺术	0.7485396862030029
4	sdsclear有什么优点	Redis设计与源码分析	0.731569766998291

平均值：0.7195秒

性能概览

3000本书，3375141条chunk，hunk_size=1000（约33亿字），73G

	问题	来源	ClickHouse时间(秒)
1	四大公害的诉讼是哪四个？	白夜行	3.337501049041748
2	什么是激励相容？	置身事内	3.2200841903686523
3	柯克霍夫原则（Kerckhoff's rinciple）是什么	Java加密与解密的艺术	3.6809117794036865
4	sdsclear有什么优点	Redis设计与源码分析	3.592667579650879

平均值：3.45秒

总结

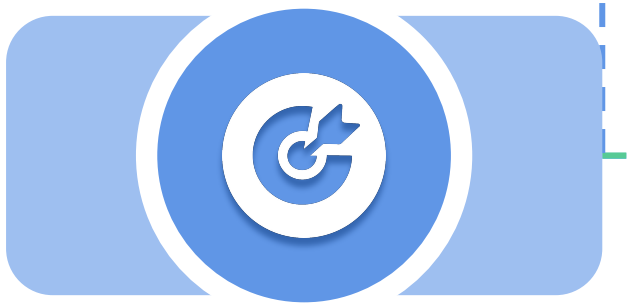
数据量（chunk）	查询时间（秒）	
179	0.026	1
17889	0.11	99.94/4.23
198892	0.7195	11.81/6.5
3375141	3.45	16.97/4.85

04

生产环境中的RAG

...

三大问题



并发

Python并发性能并不高

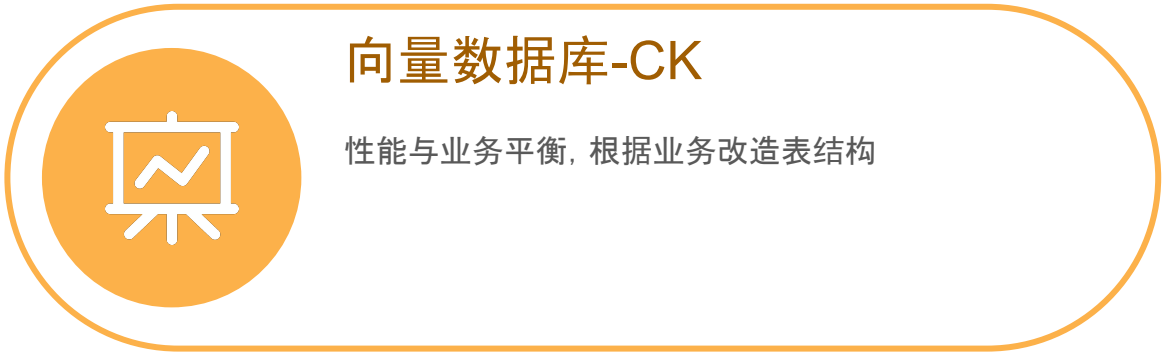
业务

Langchain的向量数据库无法支撑复杂业务

集成

权限、用户等多个系统需要集成

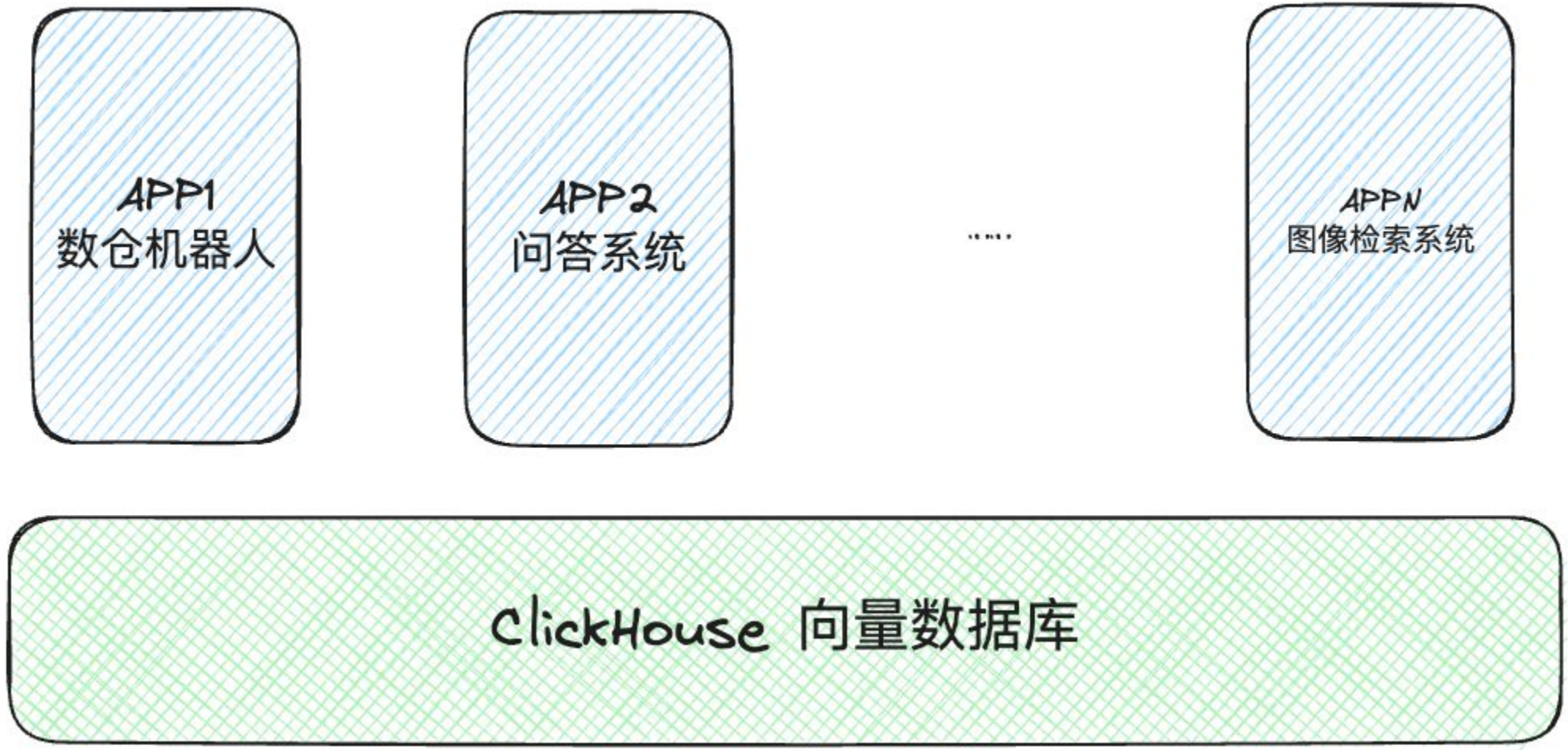
技术选型推荐



示例

```
CREATE TABLE IF NOT EXISTS default.embedding
(
    `id` Nullable(String),
    `document` Nullable(String),
    `tanantid` UInt64,
    `embedding` Array(Float32),
    `metadata` JSON,
    `uuid` UUID DEFAULT generateUUIDv4(),
    INDEX vec_idx embedding TYPE annoy('L2Distance', 100) GRANULARITY 1000,
    CONSTRAINT cons_vec_len CHECK length(embedding) = 1536
)
ENGINE = MergeTree
ORDER BY (tanantid, embedding)
SETTINGS index_granularity = 8192
```

示意图



感谢观看