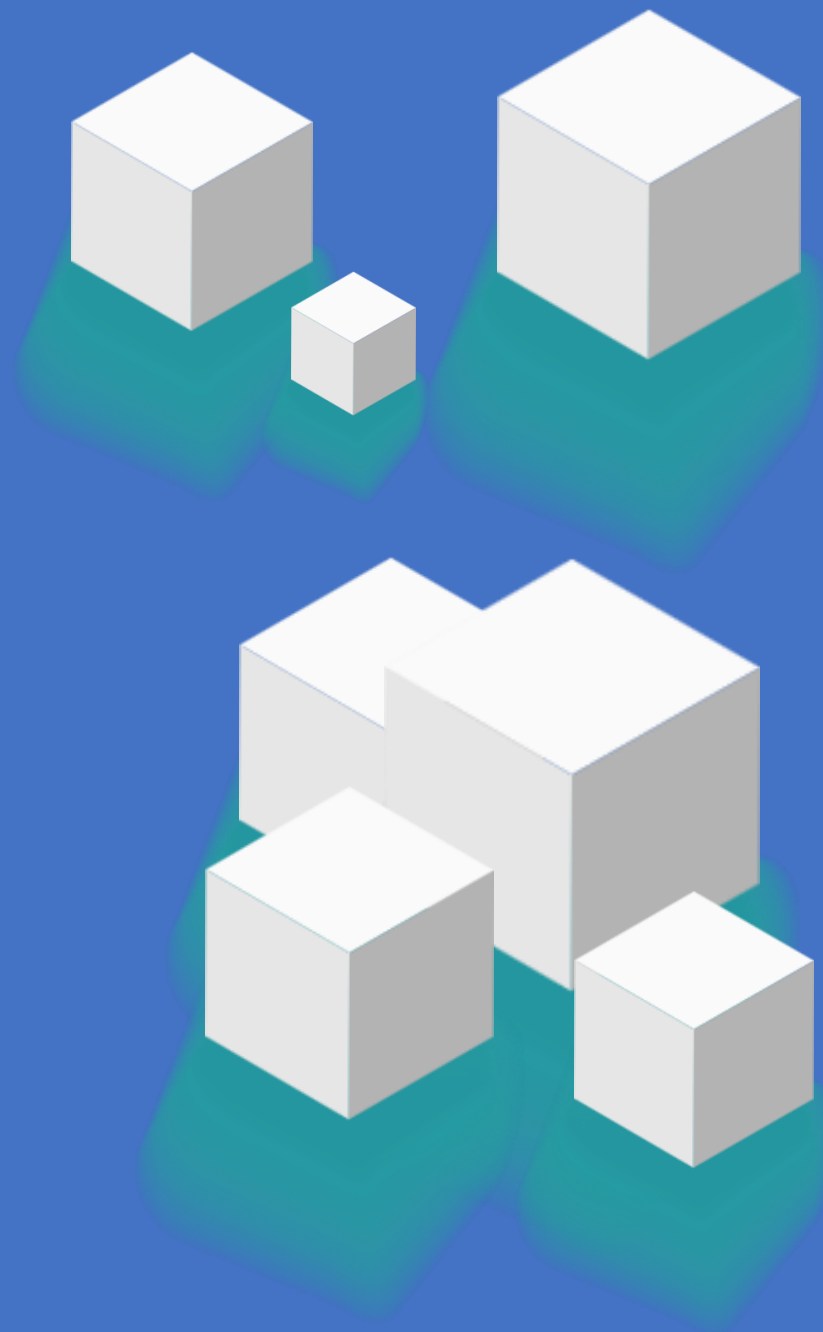
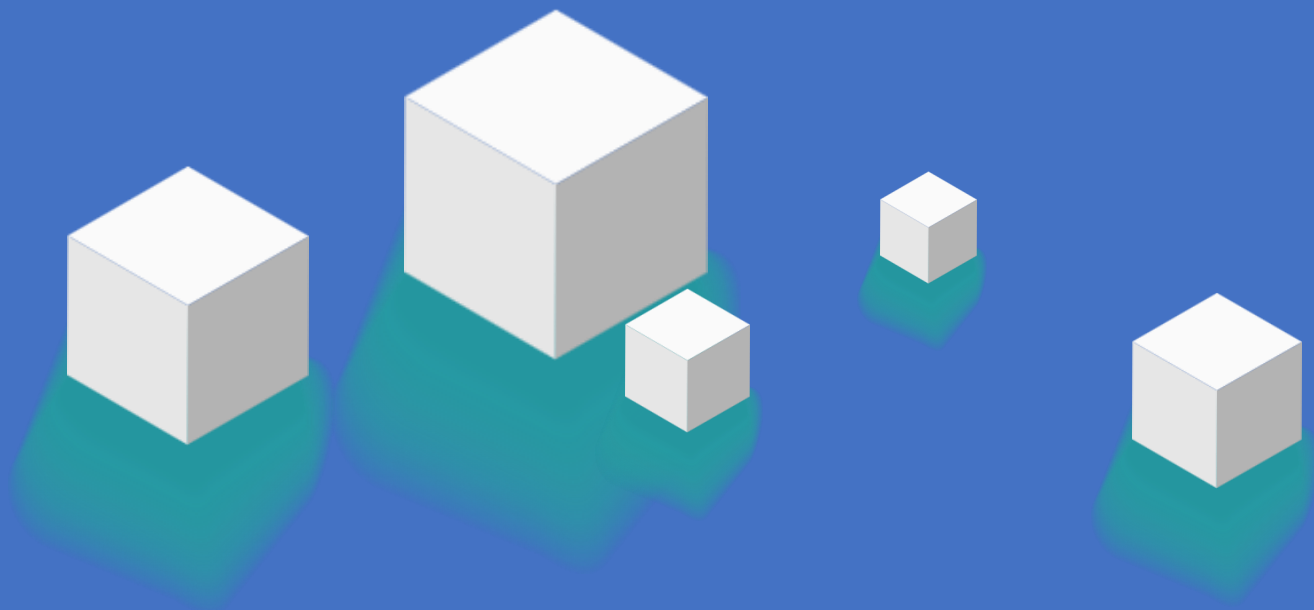


# RAG技术与应用



# >> 今天的学习目标

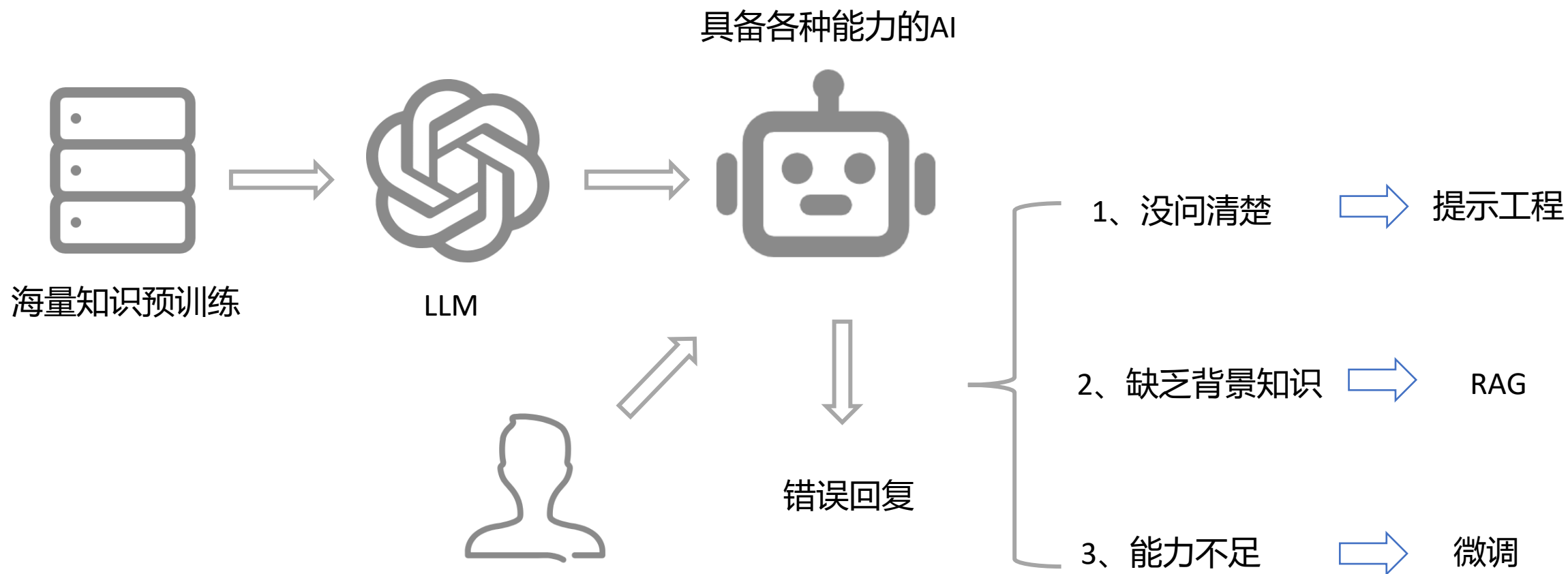
---

## RAG技术与应用

- 大模型应用开发的三种模式
- RAG的核心原理与流程
- NativeRAG
- NoteBookLM使用
- Embedding模型选择
- CASE：DeepSeek + Faiss搭建本地知识库检索
- RAG常见问题——如何提升RAG质量

# 大模型应用开发

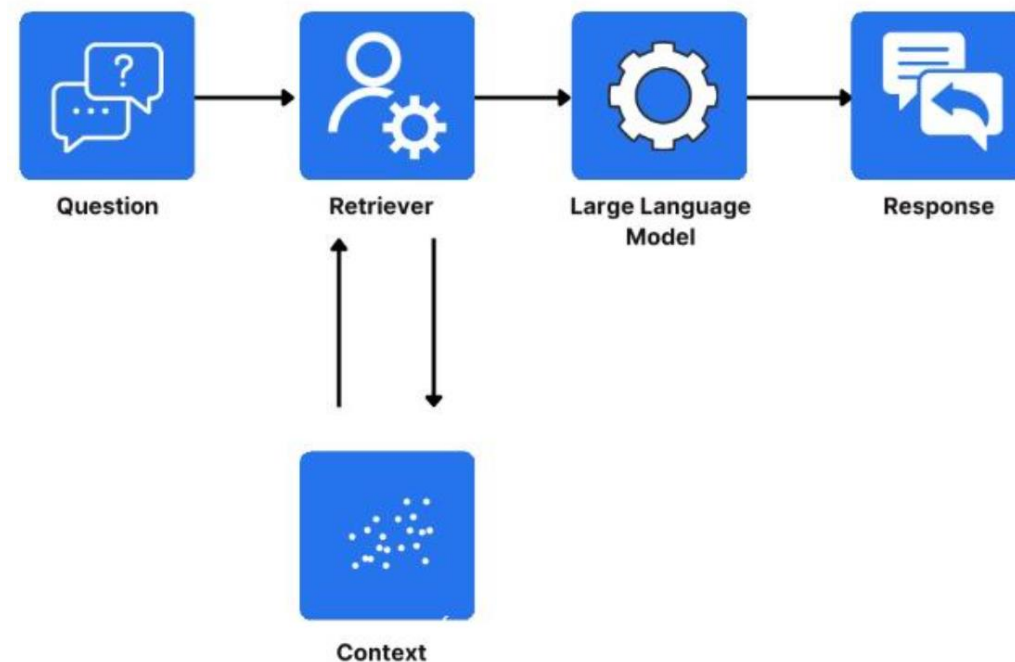
Thinking: 提示工程 VS RAG VS 微调, 什么时候使用?



# 什么是RAG

RAG（Retrieval-Augmented Generation）：

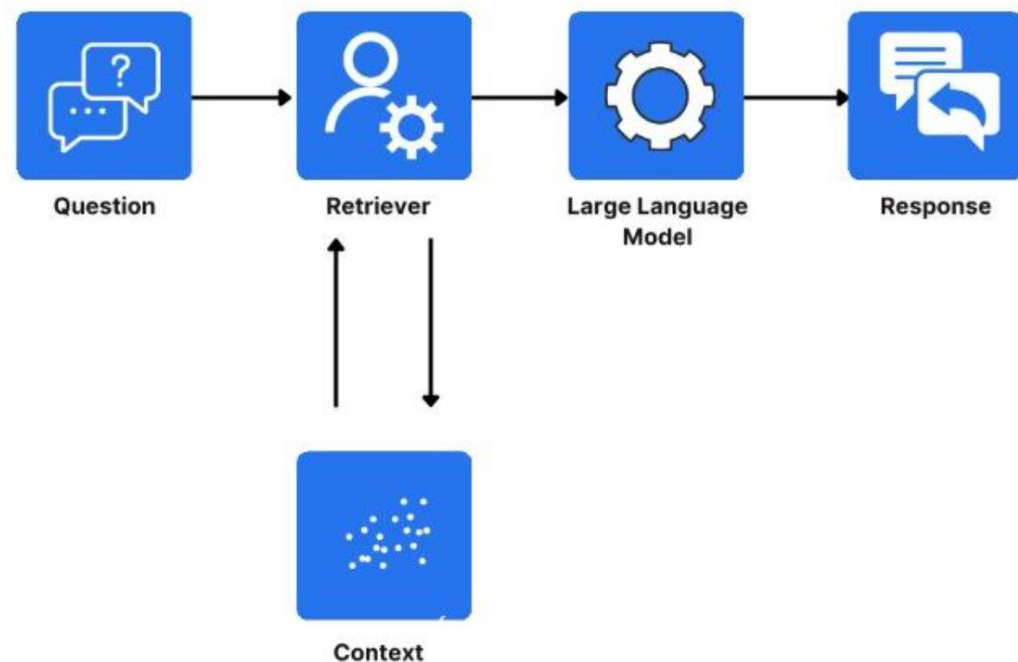
- 检索增强生成，是一种结合信息检索（Retrieval）和文本生成（Generation）的技术
- RAG技术通过实时检索相关文档或信息，并将其作为上下文输入到生成模型中，从而提高生成结果的时效性和准确性。



# RAG的优势

Thinking: RAG的优势是什么？

- **解决知识时效性问题：**大模型的训练数据通常是静态的，无法涵盖最新信息，而RAG可以检索外部知识库实时更新信息。
- **减少模型幻觉：**通过引入外部知识，RAG能够减少模型生成虚假或不准确内容的可能性。
- **提升专业领域回答质量：**RAG能够结合垂直领域的专业知识库，生成更具专业深度的回答



# RAG的核心原理与流程

---

## Step1，数据预处理

- **知识库构建**：收集并整理文档、网页、数据库等多源数据，构建外部知识库。
- **文档分块**：将文档切分为适当大小的片段（**chunks**），以便后续检索。分块策略需要在语义完整性与检索效率之间取得平衡。
- **向量化处理**：使用嵌入模型（如BGE、M3E、Chinese-Alpaca-2等）将文本块转换为向量，并存储在向量数据库中

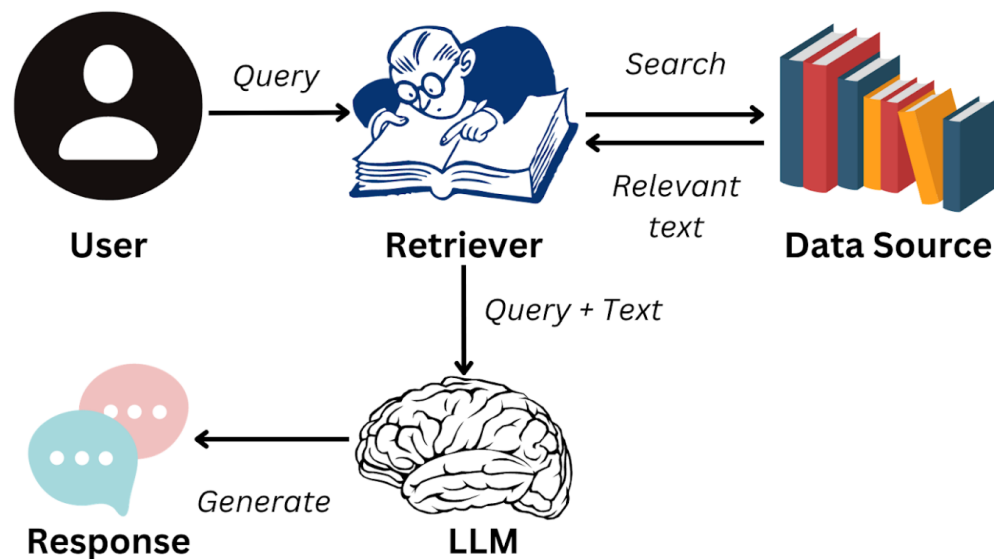
# RAG的核心原理与流程

## Step2, 检索阶段

- **查询处理**: 将用户输入的问题转换为向量, 并在向量数据库中进行相似度检索, 找到最相关的文本片段。
- **重排序**: 对检索结果进行相关性排序, 选择最相关的片段作为生成阶段的输入

## Step3, 生成阶段

- **上下文组装**: 将检索到的文本片段与用户问题结合, 形成增强的上下文输入。
- **生成回答**: 大语言模型基于增强的上下文生成最终回答。



# NativeRAG

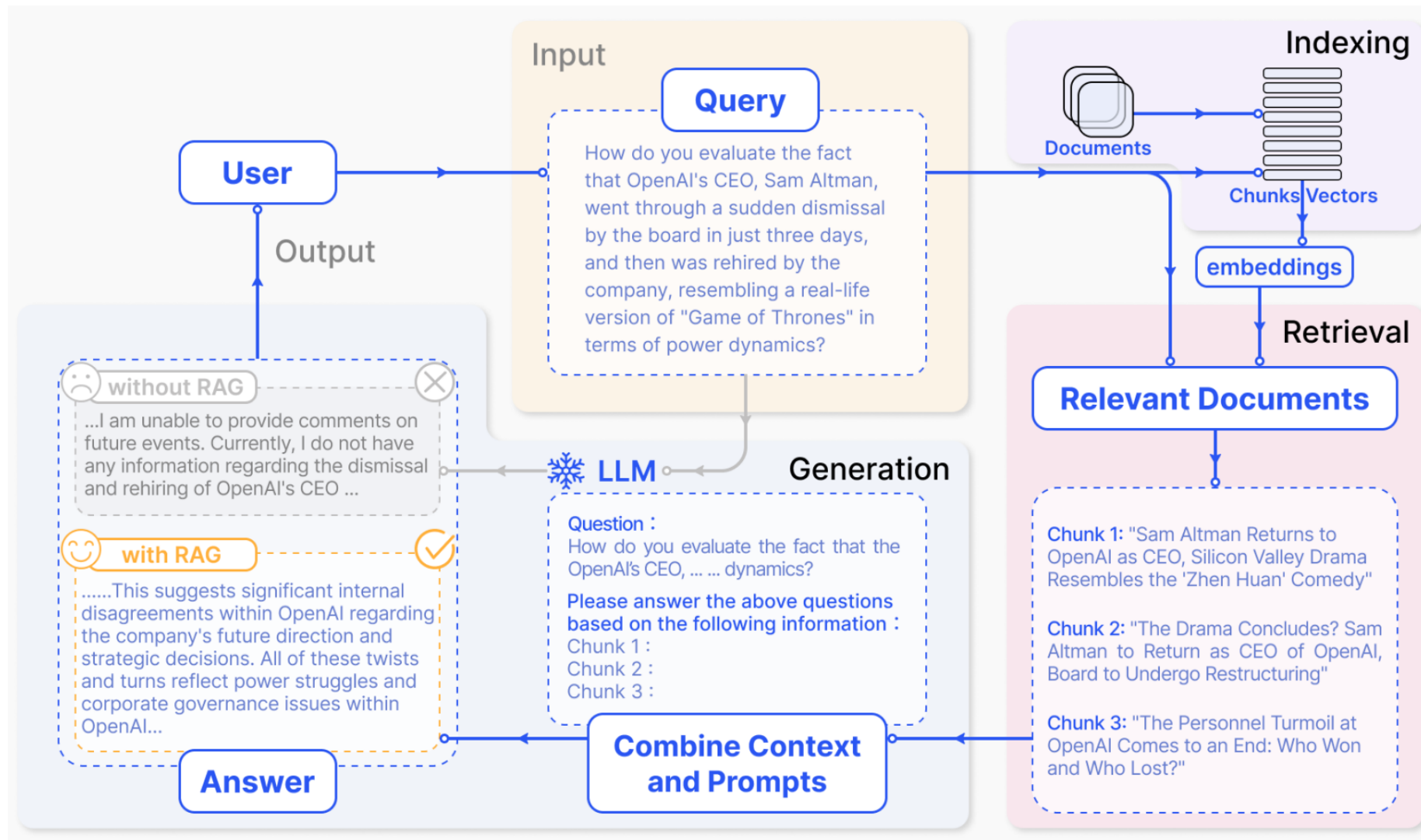
RAG的步骤:

Indexing => 如何更好地把知识存起来。

Retrieval => 如何在大量的知识中，找到一小部分有用的，给到模型参考。

Generation => 如何结合用户的提问和检索到的知识，让模型生成有用的答案。

这三个步骤虽然看似简单，但在 RAG 应用从构建到落地实施的整个过程中，涉及较多复杂的工作内容





# NotebookLM使用

# Embedding模型选择

# Embedding模型选择

Summary Performance per task Task information

| Rank<br>(Borda) | Model  | Zero-shot | Clustering | Instruction Retrieval | Multilabel Classification | Pair Classification | Reranking | Retrieval | STS   |
|-----------------|--|-----------|------------|-----------------------|---------------------------|---------------------|-----------|-----------|-------|
| 1               | <a href="#">Linq-Embed-Mistral</a>             | 99%       | 0.27       | 0.94                  | 24.77                     | 80.43               | 64.37     | 58.69     | 74.86 |
| 2               | <a href="#">gte-Qwen2-7B-instruct</a>          | ⚠ NA      | 0.36       | 4.94                  | 25.48                     | 85.13               | 65.55     | 60.08     | 73.98 |
| 3               | <a href="#">multilingual-e5-large-instruct</a> | 99%       | 0.54       | -0.40                 | 22.91                     | 80.86               | 62.61     | 57.12     | 76.81 |
| 4               | <a href="#">SFR-Embedding-Mistral</a>          | 96%       | 0.57       | 0.16                  | 24.55                     | 80.29               | 64.19     | 59.44     | 74.79 |
| 5               | <a href="#">GritLM-7B</a>                      | 99%       | 0.48       | 3.45                  | 22.77                     | 79.94               | 63.78     | 58.31     | 73.33 |
| 6               | <a href="#">GritLM-8x7B</a>                    | 99%       | 0.88       | 2.44                  | 24.43                     | 79.73               | 62.61     | 57.54     | 73.16 |
| 7               | <a href="#">e5-mistral-7b-instruct</a>         | 99%       | 0.39       | -0.62                 | 22.20                     | 81.12               | 63.82     | 55.75     | 74.02 |
| 8               | <a href="#">Cohere-embed-multilingual-v3.0</a> | ⚠ NA      | 0.61       | -1.89                 | 22.74                     | 79.88               | 64.07     | 59.16     | 74.80 |
| 9               | <a href="#">gte-Qwen2-1.5B-instruct</a>        | ⚠ NA      | 0.59       | 0.74                  | 24.02                     | 81.58               | 62.58     | 60.78     | 71.61 |
| 10              | <a href="#">bilingual-embedding-large</a>      | 98%       | 0.24       | -3.04                 | 22.36                     | 79.83               | 61.42     | 55.10     | 77.81 |
| 11              | <a href="#">text-embedding-3-large</a>         | ⚠ NA      | 0.49       | -2.68                 | 22.03                     | 79.17               | 63.89     | 59.27     | 71.68 |

<https://huggingface.co/spaces/mteb/leaderboard> 比较了1000多种语言中的100多种文本嵌入模型

完整榜单下载见 Embedding榜单-202503.csv

# Embedding模型选择

---

Thinking: 有哪些常见的Embedding模型？

## 1、通用文本嵌入模型

**BGE-M3**（智源研究院）

- 特点：支持100+语言，输入长度达8192 tokens，融合密集、稀疏、多向量混合检索，适合跨语言长文档检索。
- 适用场景：跨语言长文档检索、高精度RAG应用。

**text-embedding-3-large**（OpenAI）

- 特点：向量维度3072，长文本语义捕捉能力强，英文表现优秀。
- 适用场景：英文内容优先的全球化应用。

**Jina-embeddings-v2**（Jina AI）

- 特点：参数量仅35M，支持实时推理（RT<50ms），适合轻量化部署。
- 适用场景：轻量级文本处理、实时推理任务。

## 2、中文嵌入模型

**xiaobu-embedding-v2**

- 特点：针对中文语义优化，语义理解能力强。
- 适用场景：中文文本分类、语义检索。

**M3E-Turbo**

- 特点：针对中文优化的轻量模型，适合本地私有化部署。
- 适用场景：中文法律、医疗领域检索任务。

**stella-mrl-large-zh-v3.5-1792**

- 特点：处理大规模中文数据能力强，捕捉细微语义关系。
- 适用场景：中文文本高级语义分析、自然语言处理任务。

# Embedding模型选择

---

## 3、指令驱动与复杂任务模型

### gte-Qwen2-7B-instruct（阿里巴巴）

- 特点：基于Qwen大模型微调，支持代码与文本跨模态检索。
- 适用场景：复杂指令驱动任务、智能问答系统。

### E5-mistral-7B（Microsoft）

- 特点：基于Mistral架构，Zero-shot任务表现优异。
- 适用场景：动态调整语义密度的复杂系统。

## 4、企业级与复杂系统

### BGE-M3（智源研究院）

- 特点：适合企业级部署，支持混合检索。
- 适用场景：企业级语义检索、复杂RAG应用。

### E5-mistral-7B（Microsoft）

- 特点：适合企业级部署，支持指令微调。
- 适用场景：需要动态调整语义密度的复杂系统。

# CASE: bge-m3 使用

---

```
from FlagEmbedding import BGEM3FlagModel

model = BGEM3FlagModel('BAAI/bge-m3',

                        use_fp16=True) # Setting use_fp16 to True speeds up
computation with a slight performance degradation

sentences_1 = ["What is BGE M3?", "Defination of BM25"]

sentences_2 = ["BGE M3 is an embedding model supporting dense
retrieval, lexical matching and multi-vector interaction.",

               "BM25 is a bag-of-words retrieval function that ranks a set of
documents based on the query terms appearing in each document"]
```

```
embeddings_1 = model.encode(sentences_1,

                             batch_size=12,

                             max_length=8192, # If you don't need such a
long length, you can set a smaller value to speed up the
encoding process.
```

```
)['dense_vecs']

embeddings_2 = model.encode(sentences_2)['dense_vecs']

similarity = embeddings_1 @ embeddings_2.T

print(similarity)
```

```
[[0.626  0.3477]
```

```
[0.3499 0.678 ]]
```

# CASE: bge-m3 使用

`similarity = embeddings_1 @ embeddings_2.T`

在计算两组嵌入向量（`embeddings`）之间的相似度矩阵。

`embeddings_1` 包含了第一组句子 (`sentences_1`) 的嵌入向量，形状为 [sentences\_1的数量, 嵌入维度]

`embeddings_2` 包含了第二组句子 (`sentences_2`) 的嵌入向量，形状为 [sentences\_2的数量, 嵌入维度]

`embeddings_2.T` 是对 `embeddings_2` 进行转置操作，形状变为 [嵌入维度, sentences\_2的数量]

`@` 符号在Python中表示矩阵乘法运算

=> 通过矩阵乘法计算了两组句子之间的余弦相似度矩阵。结果 `similarity` 的形状是 [sentences\_1的数量, sentences\_2的数量]。

[[0.626 0.3477]

[0.3499 0.678 ]]

可以看出：

"What is BGE M3?" 与 "BGE M3 is an embedding model..." 的相似度为 0.6265（较高）

"What is BGE M3?" 与 "BM25 is a bag-of-words retrieval function..." 的相似度为 0.3477（较低）

"Defination of BM25" 与 "BGE M3 is an embedding model..." 的相似度为 0.3499（较低）

"Defination of BM25" 与 "BM25 is a bag-of-words retrieval function..." 的相似度为 0.678（较高）

# CASE: gte-qwen2 使用

```
from sentence_transformers import SentenceTransformer
model_dir = "/root/autodl-tmp/models/iic/gte_Qwen2-1___5B-instruct"
model = SentenceTransformer(model_dir, trust_remote_code=True)
# In case you want to reduce the maximum length:
model.max_seq_length = 8192
queries = [
    "how much protein should a female eat",
    "summit define",
]
documents = [
    "As a general guideline, the CDC's average requirement of protein for women
ages 19 to 70 is 46 grams per day. But, as you can see from this chart, you'll need
to increase that if you're expecting or training for a marathon. Check out the
chart below to see how much protein you should be eating each day.",
```

```
"Definition of summit for English Language Learners. : 1 the highest
point of a mountain : the top of a mountain. : 2 the highest level. : 3 a
meeting or series of meetings between the leaders of two or more
governments.",
]
```

```
query_embeddings = model.encode(queries, prompt_name="query")
document_embeddings = model.encode(documents)
```

```
scores = (query_embeddings @ document_embeddings.T) * 100
print(scores.tolist())
```

```
[[78.49691772460938, 17.04286003112793],
 [14.924489974975586, 75.37960815429688]]
```



# CASE: gte-qwen2 使用

```
import torch
import torch.nn.functional as F
from torch import Tensor
from modelscope import AutoTokenizer, AutoModel

# 定义最后一个token池化函数
# 该函数从最后的隐藏状态中提取每个序列的最后一个有效token的表示
def last_token_pool(last_hidden_states: Tensor,
                    attention_mask: Tensor) -> Tensor:
    left_padding = (attention_mask[:, -1].sum() == attention_mask.shape[0])
    if left_padding:
        return last_hidden_states[:, -1]
    else:
        sequence_lengths = attention_mask.sum(dim=1) - 1
        batch_size = last_hidden_states.shape[0]
        return last_hidden_states[torch.arange(batch_size,
                                                device=last_hidden_states.device),
                                sequence_lengths]
```

```
# 将任务描述和查询组合成特定格式的指令
def get_detailed_instruct(task_description: str, query: str) -> str:
    return f'Instruct: {task_description}\nQuery: {query}'

task = 'Given a web search query, retrieve relevant passages that answer the query'

queries = [
    get_detailed_instruct(task, 'how much protein should a female eat'), # 女性应该摄入多少蛋白质
    get_detailed_instruct(task, 'summit define') # summit（顶峰）的定义
]

# 检索文档
documents = [
    "As a general guideline, the CDC's average requirement of protein for women ages 19 to 70 is 46 grams per day. But, as you can see from this chart, you'll need to increase that if you're expecting or training for a marathon. Check out the chart below to see how much protein you should be eating"
```

# CASE: gte-qwen2 使用

```
each day.", # 关于女性蛋白质摄入量的文档
    "Definition of summit for English Language Learners. : 1 the highest point
of a mountain : the top of a mountain. : 2 the highest level. : 3 a meeting or
series of meetings between the leaders of two or more governments." # 关
于summit定义的文档
]
# 将查询和文档合并为一个输入文本列表
input_texts = queries + documents

# 设置模型路径
model_dir = "/root/autodl-tmp/models/iic/gte_Qwen2-1___5B-instruct"
# 加载分词器, trust_remote_code=True允许使用远程代码
tokenizer = AutoTokenizer.from_pretrained(model_dir,
trust_remote_code=True)
# 加载模型
model = AutoModel.from_pretrained(model_dir, trust_remote_code=True)
```

```
max_length = 8192
batch_dict = tokenizer(input_texts, max_length=max_length, padding=True,
truncation=True, return_tensors='pt')
outputs = model(**batch_dict)
# 使用last_token_pool函数从最后的隐藏状态中提取每个序列的表示
embeddings = last_token_pool(outputs.last_hidden_state,
batch_dict['attention_mask'])

embeddings = F.normalize(embeddings, p=2, dim=1)
scores = (embeddings[:2] @ embeddings[2:].T) * 100
print(scores.tolist())
```

```
[[78.49689483642578, 17.042858123779297],
 [14.924483299255371, 75.37962341308594]]
```

# Summary

---

gte-Qwen2-7B-instruct 是基于 Qwen2 的指令优化型嵌入模型

**指令优化：** 经过大量指令-响应对的训练，特别擅长理解和生成高质量的文本。

**性能表现：** 在文本生成、问答系统、文本分类、情感分析、命名实体识别和语义匹配等任务中表现优异。

**适合场景：** 适合复杂问答系统，处理复杂的多步推理问题，能够生成准确且自然的答案。

**优势：**

- 指令理解和执行能力强，适合复杂的指令驱动任务。
- 多语言支持，能够处理多种语言的文本。
- 在文本生成和语义理解任务中表现优异。

**局限：**

- 计算资源需求较高，适合资源充足的环境。

# CASE: DeepSeek + Faiss 搭建本地知识库检索

# CASE: DeepSeek + Faiss搭建本地知识库检索

百度文库 - 好好学习，天天向上

## 上海浦东发展银行西安分行 个金客户经理管理考核暂行办法

### 第一章 总 则

第一条 为保证我分行个金客户经理制的顺利实施，有效调动个金客户经理的积极性，促进个金业务快速、稳定地发展，根据总行《上海浦东发展银行个人金融营销体系建设方案（试行）》要求，特制定《上海浦东发展银行西安分行个金客户经理管理考核暂行办法（试行）》（以下简称本办法）。

第二条 个金客户经理系指各支行（营业部）从事个人金融产品营销与市场开拓，为我行个人客户提供综合银行服务的我行市场人员。

第三条 考核内容分为两大类，即个人业绩考核、工作质量考核。个人业绩包括个人资产业务、负债业务、卡业务。工作质量指个人业务的资产质量。

第四条 为规范激励规则，客户经理的技术职务和薪资实行每年考核浮动。客户经理的奖金实行每季度考核浮动，即客户经理按其考核内容得分与行员等级结合，享受对应的行员等级待遇。



客户经理被投诉了，投诉一次扣多少分

根据文件内容，客户经理被投诉一次扣2分。具体规定如下：

（二）服务质量考核：

2. 客户服务效率低，态度生硬或不及时为客户提供维护服务，有客户投诉的，每投诉一次扣2分。

客户经理每年评聘申报时间是怎样的？

根据文件内容，客户经理每年评聘申报时间如下：

第十一条 每年一月份为客户经理评聘的申报时间，由分行人力资源部、个人业务部每年二月份组织统一的资格考试。考试合格者由分行颁发个金客户经理资格证书，其有效期为一年。

因此，客户经理每年评聘的申报时间是一月份。

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
from PyPDF2 import PdfReader
from langchain.chains.question_answering import load_qa_chain
from langchain_openai import OpenAI
from langchain_community.callbacks.manager import get_openai_callback
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import DashScopeEmbeddings
from langchain_community.vectorstores import FAISS
from typing import List, Tuple
```

```
def extract_text_with_page_numbers(pdf) -> Tuple[str, List[int]]:
```

```
    """
```

从PDF中提取文本并记录每行文本对应的页码

参数:

pdf: PDF文件对象

返回:

text: 提取的文本内容

page\_numbers: 每行文本对应的页码列表

```
    """
```

```
    text = ""
```

```
    page_numbers = []
```

```
    for page_number, page in enumerate(pdf.pages, start=1):
```

```
        extracted_text = page.extract_text()
```

```
        if extracted_text:
```

```
            text += extracted_text
```

```
            page_numbers.extend([page_number]
```

```
*)
```

```
            len(extracted_text.split("\n")))
```

```
        else:
```

```
            Logger.warning(f"No text found on page {page_number}.")
```

```
    return text, page_numbers
```

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
def process_text_with_splitter(text: str, page_numbers: List[int]) -> FAISS:
```

```
    """
```

处理文本并创建向量存储

参数:

text: 提取的文本内容

page\_numbers: 每行文本对应的页码列表

返回:

knowledgeBase: 基于FAISS的向量存储对象

```
    """
```

# 创建文本分割器, 用于将长文本分割成小块

```
text_splitter = RecursiveCharacterTextSplitter(
```

```
    separators=["\n\n", "\n", ".", " ", ""],
```

```
    chunk_size=1000,
```

```
    chunk_overlap=200,
```

```
    length_function=len,
```

```
)
```

```
# 分割文本
```

```
chunks = text_splitter.split_text(text)
```

```
print(f"文本被分割成 {len(chunks)} 个块。")
```

```
# 创建嵌入模型
```

```
embeddings = DashScopeEmbeddings(
```

```
    model="text-embedding-v1",
```

```
    dashscope_api_key=DASHSCOPE_API_KEY,
```

```
)
```

```
# 从文本块创建知识库
```

```
knowledgeBase = FAISS.from_texts(chunks, embeddings)
```

```
#Logger.info("Knowledge base created from text chunks.")
```

```
print("已从文本块创建知识库。")
```

```
# 存储每个文本块对应的页码信息
```

```
knowledgeBase.page_info = {chunk: page_numbers[i] for i, chunk in enumerate(chunks)}
```

```
return knowledgeBase
```

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
# 读取PDF文件
pdf_reader = PdfReader('./浦发上海浦东发展银行西安分行个金客户经理考核
办法.pdf')
# 提取文本和页码信息
text, page_numbers = extract_text_with_page_numbers(pdf_reader)
text
```

'百度文库 - 好好学习, 天天向上 \n-1 上海浦东发展银行西安分行 \n个金客户经理管理考核暂行办法 \n \n \n第一章 总 则 \n第一条 为保证我分行个金客户经理制的顺利实施, 有效调动个\n金客户经理的积极性, 促进个金业务快速、稳定地发展, 根据总行《上\n海浦东发展银行个人金融营销体系建设方案(试行)》要求, 特制定\n《上海浦东发展银行西安分行个金客户经理管理考核暂行办法(试\n行)》(以下简称本办法)。 \n第二条 个金客户经理系指各支行(营业部)从事个人金融产品\n营销与市场开拓, 为我行个人客户提供综合银行服务的我行市场人\n员。 \n第三条 考核内容分为两大类, 即个人业绩考核、 工作质量考核。 \n个人业绩包括个人资产业务、负债业务、卡业务。工作质量指个人业\n务的资产质量。 \n第四条 为规范激励规则, 客户经理的技术职务和薪资实行每年\n考核浮动。客户经理的奖金实行每季度考核浮动, 即客户经理按其考\n核内容得分与行员等级结合, 享受对应的行员等级待遇。 \n 百度文库 - 好好学习, 天天向上 \n-2 第二章 职位设置与职责 \n第五条 个金客户经理职位设置为: 客户经理助理、客户经理、 \n高级客户经理、资深客户经理。 \n第六条 个金客户经理的基本职责: \n (一) 客户开发。研究客户信息、联系与选择客户、与客户建.....



# CASE: DeepSeek + Faiss搭建本地知识库检索

---

```
print(f"提取的文本长度: {len(text)} 个字符。")
```

```
# 处理文本并创建知识库
```

```
knowledgeBase = process_text_with_splitter(text, page_numbers)
```

```
knowledgeBase
```

提取的文本长度: 3881 个字符。

文本被分割成 5 个块。

已从文本块创建知识库。

```
<langchain_community.vectorstores.faiss.FAISS at 0x170ab59f7d0>
```

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
from langchain_community.llms import Tongyi
llm = Tongyi(model_name="deepseek-v3",
dashscope_api_key=DASHSCOPE_API_KEY)
# 设置查询问题
query = "客户经理被投诉了，投诉一次扣多少分"
query = "客户经理每年评聘申报时间是怎样的？"
if query:
    # 执行相似度搜索，找到与查询相关的文档
    docs = knowledgeBase.similarity_search(query)

# 加载问答链
chain = load_qa_chain(llm, chain_type="stuff")

# 准备输入数据
input_data = {"input_documents": docs, "question": query}
```

```
# 使用回调函数跟踪API调用成本
with get_openai_callback() as cost:
    # 执行问答链
    response = chain.invoke(input=input_data)
    print(f"查询已处理。成本: {cost}")
    print(response["output_text"])
    print("来源:")

# 记录唯一的页码
unique_pages = set()
# 显示每个文档块的来源页码
for doc in docs:
    text_content = getattr(doc, "page_content", "")
    source_page = knowledgeBase.page_info.get(
        text_content.strip(), "未知"
    )
    if source_page not in unique_pages:
        unique_pages.add(source_page)
        print(f"文本块页码: {source_page}")
```

# Summary

---

## 1. PDF文本提取与处理

使用PyPDF2库的PdfReader从PDF文件中提取文本

在提取过程中记录每行文本对应的页码，便于后续溯源

使用RecursiveCharacterTextSplitter将长文本分割成小块，便于向量化处理

## 2. 向量数据库构建

使用OpenAIEmbeddings将文本块转换为向量表示

使用FAISS向量数据库存储文本向量，支持高效的相似度搜索

为每个文本块保存对应的页码信息，实现查询结果溯源

## 3. 语义搜索与问答链

基于用户查询，使用similarity\_search在向量数据库中检索相关文本块

使用OpenAI语言模型和load\_qa\_chain构建问答链

将检索到的文档和用户问题作为输入，生成回答

## 4. 成本跟踪与结果展示

使用get\_openai\_callback跟踪API调用成本

展示问答结果和来源页码，方便用户验证信息

# 打卡：创建你的RAG问答



结合你的业务场景，创建你的RAG问答（LangChain+DeepSeek+Faiss）

Step1, 收集整理知识库（客户经理考核办法.pdf只是示例，用你的PDF进行检索）

Step2, 从PDF中提取文本并记录每行文本对应的页码

Step3, 处理文本并创建向量存储

Step4, 执行相似度搜索，找到与查询相关的文档

Step5, 使用问到链对用户问题进行回答（使用你的DASHSCOPE\_API\_KEY）

Step6, 显示每个文档块的来源页码（当前页码来源有问题，可以用Cursor完善）

# CASE: DeepSeek + Faiss搭建本地知识库检索

Thinking: 如何调整文本块页码, 当前文本块页码计算有问题

这个文本块页码不对, 帮我查看 文本块页码的计算逻辑, 是否有问题

Thought for 5 seconds

Searched codebase for "文本块页码的计算逻辑"

Read chatpdf-faiss.py

问题找到了。当前页码计算逻辑有以下问题:

1. 在 `extract_text_with_page_numbers` 函数中, 页码是按行文本进行记录的:

python

▷ Apply to chatpdf-fais...

```
page_numbers.extend([page_number] * len(extracted_text.split("\n")))
```

这里对每个PDF页面提取的文本按换行符分割, 然后为每一行都分配当前页码。

2. 但在 `process_text_with_splitter` 函数中, 文本被分成块时使用的是:

python

▷ Apply to chatpdf-fais...

```
page_info = {chunk: page_numbers[i] for i, chunk in enumerate(chunks)}
```

这段代码假设分割后的chunks数量与原始按行记录的page\_numbers长度相同, 直接使用索引i来获取页码。

但实际上, `RecursiveCharacterTextSplitter` 分割器会按照多种分隔符 (`\n\n`, `\n`, `.`, `,`, ```) 将文本分割成chunk\_size大小的块, 与原始按行分割的对应关系已经不存在了。

解决方案是修改页码记录逻辑, 使页码能够正确对应到分割后的文本块:

解决方案是修改页码记录逻辑, 使页码能够正确对应到分割后的文本块:

chatpdf-faiss.py +32 -1 ✓

# 修正页码逻辑: 为每个文本块找到最频繁出现的页码

```
page_info = {}
```

```
for chunk in chunks:
```

```
    # 找到文本块在原始文本中的起始位置
```

```
    chunk_start = text.find(chunk)
```

```
    if chunk_start == -1:
```

```
        # 如果找不到精确匹配, 可能是因为文本分割的边界问题
```

```
        # 使用一个默认页码 (例如第一页) 或标记为未知
```

```
        page_info[chunk] = "未知"
```

```
        continue
```

```
    # 计算文本块结束位置
```

```
    chunk_end = chunk_start + len(chunk)
```

# LangChain中的问答链

LangChain问答链中的4种chain\_type:

1) stuff

直接把文档作为prompt输入给OpenAI

2) map\_reduce

对于每个chunk做一个prompt（回答或者摘要），然后再做合并

3) refine

在第一个chunk上做prompt得到结果，然后合并下一个文件再输出结果

4) map\_rerank

对每个chunk做prompt，然后打个分，然后根据分数返回最好的文档中的结果

```
query = "客户经理每年评聘申报时间是怎样的？"
```

```
if query:
```

```
    # 执行相似度搜索，找到与查询相关的文档
```

```
    docs = knowledgeBase.similarity_search(query,k=10)
```

```
    # 加载问答链
```

```
    chain = load_qa_chain(llm, chain_type="stuff")
```

```
    # 准备输入数据
```

```
    input_data = {"input_documents": docs, "question": query}
```

```
    # 使用回调函数跟踪API调用成本
```

```
    with get_openai_callback() as cost:
```

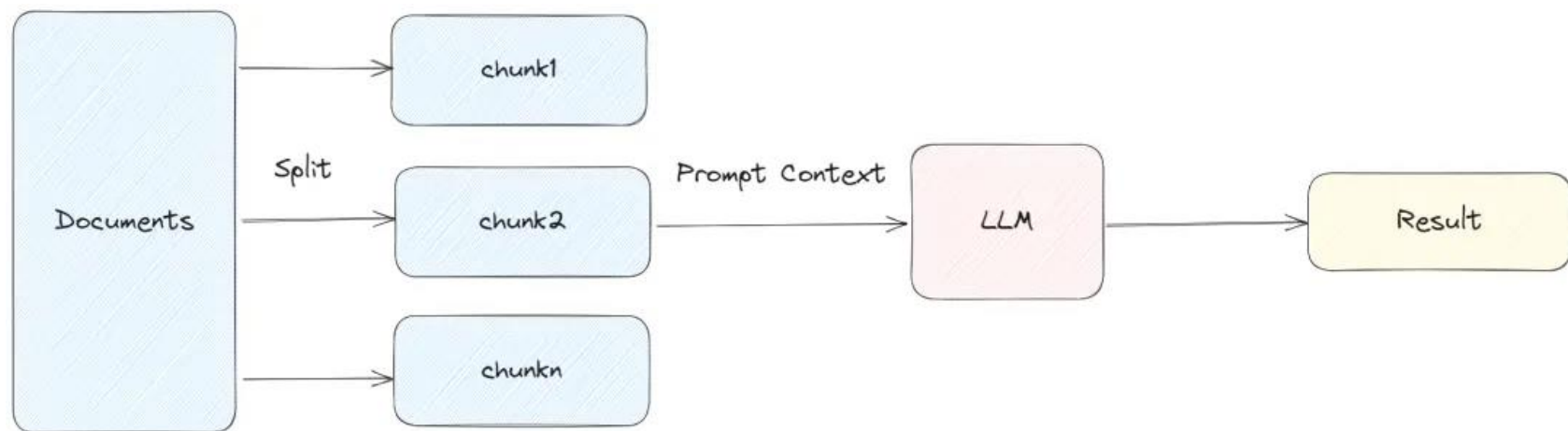
```
        # 执行问答链
```

```
        response = chain.invoke(input=input_data)
```

```
        print(f"查询已处理。成本: {cost}")
```

```
        print(response["output_text"])
```

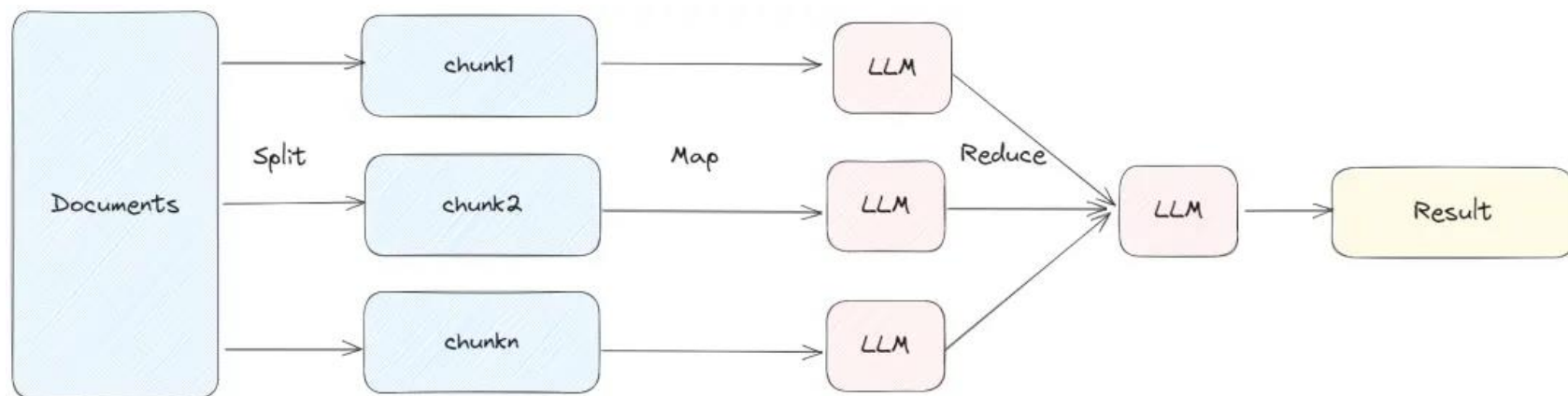
# LangChain中的问答链



## 1) stuff

适合文档拆分的比较小，一次获取文档比较少的情況

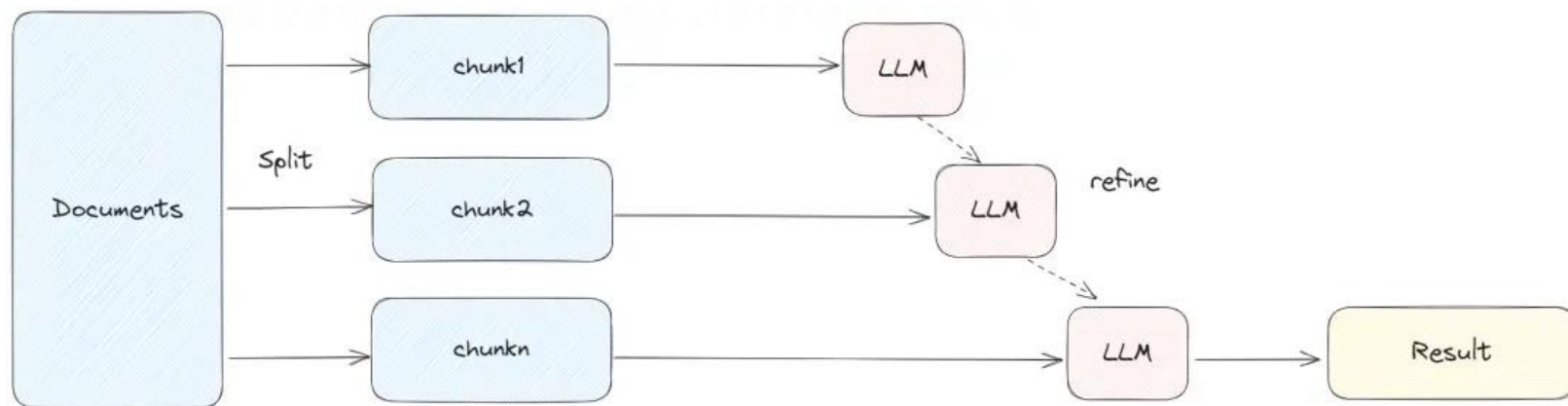
调用 LLM 的次数也比较少，能使用 stuff 的就使用这种方式。



## 2) map\_reduce

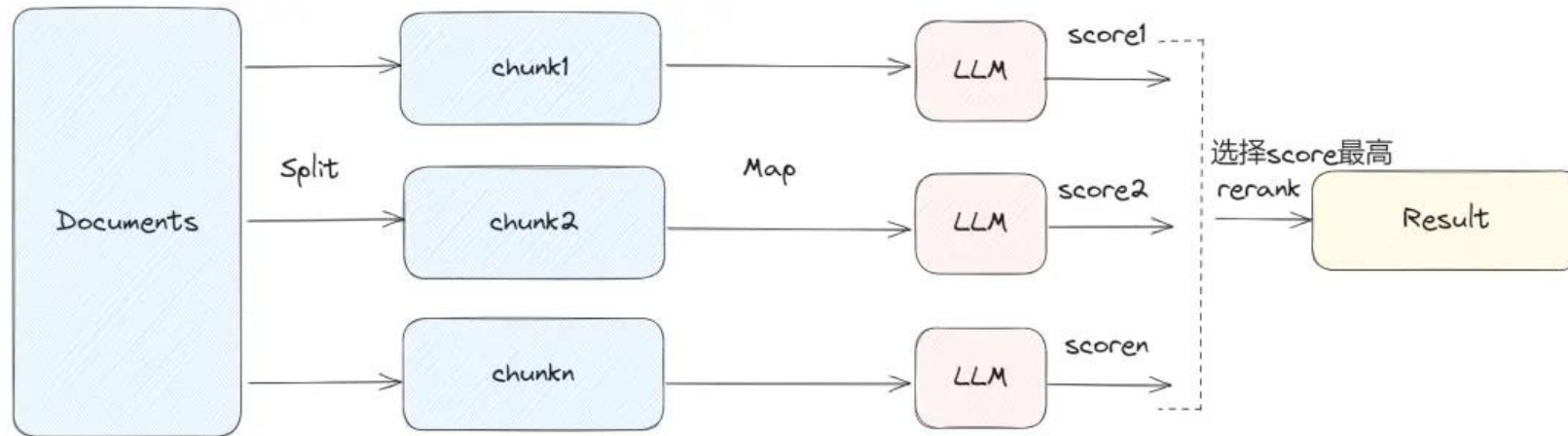
将每个 document 单独处理，可以并发进行调用。但是每个文档之间缺少上下文。

# LangChain中的问答链



## 3) refine

Refine 这种方式能部分保留上下文，以及 token 的使用能控制在一定范围。



## 4) map\_rerank

会大量地调用 LLM，每个 document 之间是独立处理



# Q&A

---

**Thinking:** 如果LLM可以处理无限上下文了，RAG还有意义吗？

**效率与成本:** LLM处理长上下文时计算资源消耗大，响应时间增加。RAG通过检索相关片段，减少输入长度。

**知识更新:** LLM的知识截止于训练数据，无法实时更新。RAG可以连接外部知识库，增强时效性。

**可解释性:** RAG的检索过程透明，用户可查看来源，增强信任。LLM的生成过程则较难追溯。

**定制化:** RAG可针对特定领域定制检索系统，提供更精准的结果，而LLM的通用性可能无法满足特定需求。

**数据隐私:** RAG允许在本地或私有数据源上检索，避免敏感数据上传云端，适合隐私要求高的场景。

=> 结合LLM的生成能力和RAG的检索能力，可以提升整体性能，提供更全面、准确的回答。

# RAG常见问题—— 如何提升RAG质量

# RAG常见问题：数据准备阶段

---

数据准备阶段：

- **数据质量差：**企业大部分数据（尤其是非结构化数据）缺乏良好的数据治理，未经标记/评估的非结构化数据可能包含敏感、过时、矛盾或不正确的信息。
- **多模态信息：**提取、定义和理解文档中的不同内容元素，如标题、配色方案、图像和标签等存在挑战。
- **复杂的PDF提取：**PDF是为人类阅读而设计的，机器解析起来非常复杂。

**Thinking：** 如何提升数据准备阶段的质量？

- 构建完整的数据准备流程
- 智能文档技术

# RAG准备：构建完整的数据准备流程

Thinking：如何构建完整的数据准备流程？

## 1. 数据评估与分类

数据审计：全面审查现有数据，识别敏感、过时、矛盾或不准确的信息。

数据分类：按类型、来源、敏感性和重要性对数据进行分类，便于后续处理。

识别敏感信息，比如：

客户姓名、身份证号、手机号、银行账号、交易记录等个人身份信息（PII）。

信用卡号、CVV码、有效期等支付信息。

发现问题：

这些信息可能未经加密存储，存在泄露风险。

部分数据可能被未经授权人员访问。

识别过时信息，比如：

客户地址、联系方式未及时更新。

已结清的贷款或信用卡账户仍被标记为“活跃”。

发现问题：

过时信息可能导致客户沟通失败或决策错误。

# RAG准备：构建完整的数据准备流程

## 2. 数据清洗

去重：删除重复数据。

纠错：修正格式错误、拼写错误等。

更新：替换过时信息，确保数据时效性。

一致性检查：解决数据矛盾，确保逻辑一致。

## 3. 敏感信息处理

识别敏感数据：使用工具或正则表达式识别敏感信息，如个人身份信息（PII）。

脱敏或加密：对敏感数据进行脱敏处理，确保合规。

## 4. 数据标记与标注

元数据标记：为数据添加元数据，如来源、创建时间等。

内容标注：对非结构化数据进行标注，便于后续检索和分析。

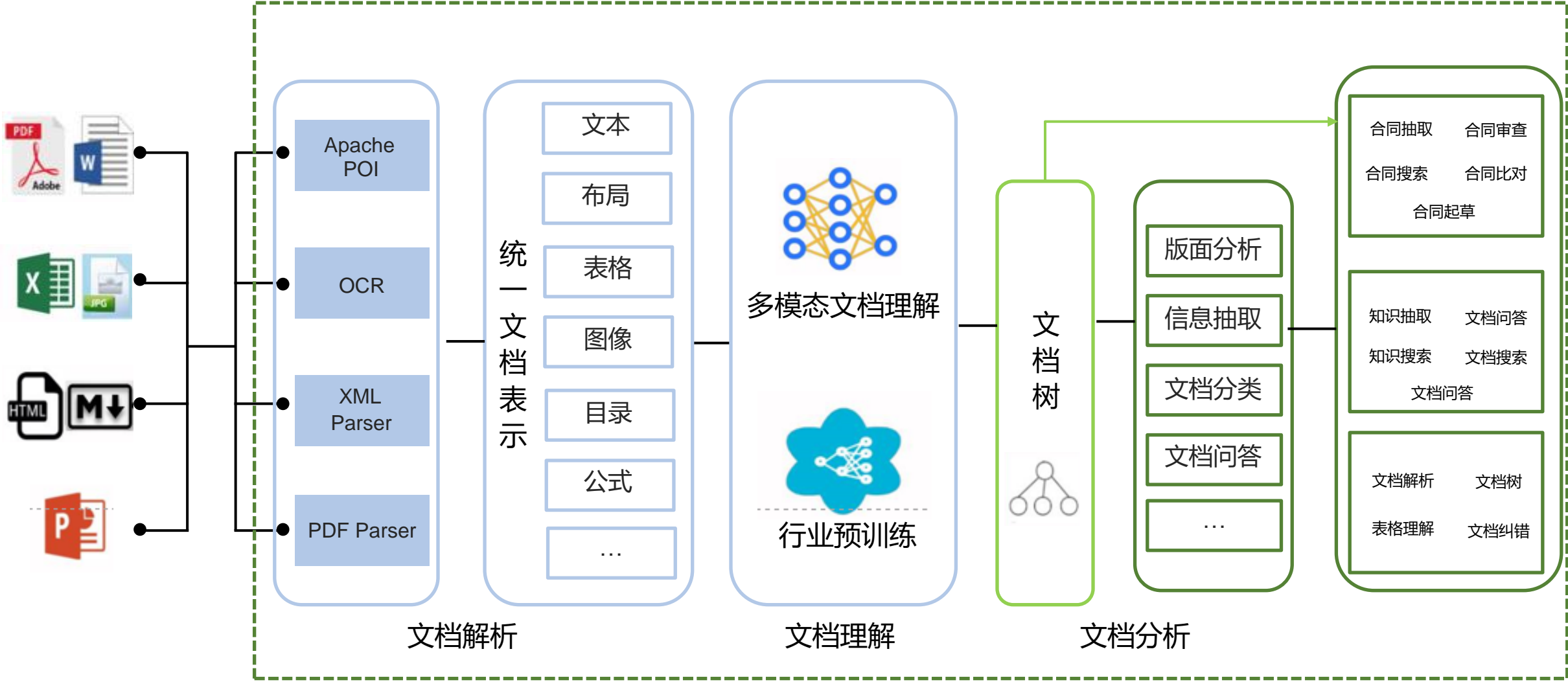
## 5. 数据治理框架

制定政策：明确数据管理、访问控制和更新流程。

责任分配：指定数据治理负责人，确保政策执行。

监控与审计：定期监控数据质量，进行审计。

# RAG准备：智能文档技术



# RAG常见问题：知识检索阶段

---

知识检索阶段：

- **内容缺失**：当检索过程缺少关键内容时，系统会提供不完整的答案 => 降低RAG的质量
- **错过排名靠前的文档**：用户查询相关的文档时被检索到，但相关性极低。因为在检索过程中，用户通过主观判断决定检索“文档数量”。理论上所有文档都要被排序并考虑进一步处理，但在实践中，通常只有排名top k的文档才会被召回，而k值需要根据经验确定。

**Thinking**：如何提升知识检索阶段的质量？

- 通过查询转换澄清用户意图。
- 采用混合检索和重排策略。

# RAG检索：通过查询转换澄清用户意图

## 1. 查询转换澄清用户意图

**场景：**用户询问“如何申请信用卡？”

**问题：**用户意图可能模糊，例如不清楚是申请流程、所需材料还是资格条件。

**解决方法：**通过查询转换明确用户意图。

**步骤：**

- 意图识别：使用自然语言处理技术识别用户意图。例如，识别用户是想了解流程、材料还是资格。
- 查询扩展：根据识别结果扩展查询。例如：

如果用户想了解流程，查询扩展为“**信用卡申请的具体步骤**”。

如果用户想了解材料，查询扩展为“**申请信用卡需要哪些材料**”。

如果用户想了解资格，查询扩展为“**申请信用卡的资格条件**”。

- 检索：使用扩展后的查询检索相关文档。

**示例：**

- 用户输入：“如何申请信用卡？”
- 系统识别意图为“流程”，扩展查询为“信用卡申请的具体步骤”。
- 检索结果包含详细的申请步骤文档，系统生成准确答案。



# RAG检索：混合检索和重排策略

## 2. 混合检索和重排策略

**场景：**用户询问“信用卡年费是多少？”

**问题：**直接检索可能返回大量文档，部分相关但排名低，导致答案不准确。

**解决方法：**采用混合检索+重排策略。

**步骤：**

- **混合检索：**结合关键词检索和语义检索。比如：

关键词检索：“信用卡 年费”。

语义检索：使用嵌入模型检索与“信用卡年费”语义相近的文档。

- **重排：**对检索结果进行重排
- **生成答案：**从重排后的文档中生成答案。

示例：

- 用户输入：“信用卡年费是多少？”
- 系统进行混合检索，结合关键词和语义检索。
- 重排后，最相关的文档（如“信用卡年费政策”）排名靠前。
- 系统生成准确答案：“信用卡年费根据卡类型不同，普通卡年费为100元，金卡为300元，白金卡为1000元。”

RAG检索阶段的质量提升：

- **查询转换：**明确用户意图，提高检索准确性。
- **混合检索与重排：**确保最相关的文档被优先处理，生成更准确的答案。

# RAG常见问题：答案生成阶段

---

## 答案生成阶段：

- **未提取：**正确答案出现在所提供的上下文中，但LLM却没有准确提取。这种情况通常发生在上下文中存在过多噪音或存在冲突的信息。
- **不完整：**尽管能够利用上下文生成答案，但存在信息缺失，最终导致LLM回答不完整。
- **格式错误：**当prompt中的附加指令格式不正确时，LLM 可能误解这些指令，从而导致错误的回答。
- **幻觉：**大模型可能会产生虚假信息。

## Thinking：如何提升答案生成阶段的质量？

- 改进提示词模板。
- 实施动态防护栏。

# RAG答案生成：改进提示词模板

**场景：**用户询问“如何申请信用卡？”

**原始提示词：**“根据以下上下文回答问题：如何申请信用卡？”

**改进后的提示词：**“根据以下上下文，提取与申请信用卡相关的具体步骤和所需材料：如何申请信用卡？”

**场景：**用户询问“信用卡的年费是多少？”

**原始提示词：**“根据以下上下文回答问题：信用卡的年费是多少？”

**改进后的提示词：**“根据以下上下文，详细列出不同信用卡的年费信息，并说明是否有减免政策：信用卡的年费是多少？”

**场景：**用户询问“什么是零存整取？”

**原始提示词：**“根据以下上下文回答问题：什么是零存整取？”

**改进后的提示词：**“根据以下上下文，准确解释零存整取的定义、特点和适用人群，确保信息真实可靠：什么是零存整取？”

**Thinking：**如何对原有的提示词进行优化？

可以通过DeepSeek-R1的推理链，对提示词进行优化：

- **信息提取：**从原始提示词中提取关键信息。
- ☆ **需求分析：**分析用户的需求，明确用户希望获取的具体信息。
- **提示词优化：**根据需求分析的结果，优化提示词，使其更具体、更符合用户的需求。

# RAG答案生成：实施动态防护栏

---

实施动态防护栏（Dynamic Guardrails）是一种在生成式AI系统中用于实时监控和调整模型输出的机制，目的是确保生成的内容符合预期、准确且安全。它通过设置规则、约束和反馈机制，动态地干预模型的生成过程，避免生成错误、不完整、不符合格式要求或有幻觉的内容。

在RAG系统中，动态防护栏的作用尤为重要，因为它可以帮助解决以下问题：

- **未提取**：确保模型从上下文中提取了正确的信息。
- **不完整**：确保生成的答案覆盖了所有必要的信息。
- **格式错误**：确保生成的答案符合指定的格式要求。
- **幻觉**：防止模型生成与上下文无关或虚假的信息。

# RAG答案生成：实施动态防护栏

## 场景1：防止未提取

- 用户问题：“如何申请信用卡？”
- 上下文：包含申请信用卡的步骤和所需材料。
- 动态防护栏规则：

检查生成的答案是否包含“步骤”和“材料”。  
如果缺失，提示模型重新生成。

- 示例：
- 错误输出：“申请信用卡需要提供一些材料。”
- 防护栏触发：检测到未提取具体步骤，提示模型补充。

## 场景2：防止不完整

- 用户问题：“信用卡的年费是多少？”
- 上下文：包含不同信用卡的年费信息。
- 动态防护栏规则：

检查生成的答案是否列出所有信用卡的年费。  
如果缺失，提示模型补充。

- 示例：
- 错误输出：“信用卡A的年费是100元。”
- 防护栏触发：检测到未列出所有信用卡的年费，提示模型补充。

## 场景4：防止幻觉

- 用户问题：“什么是零存整取？”
- 上下文：包含零存整取的定义和特点。
- 动态防护栏规则：

检查生成的答案是否与上下文一致。  
如果不一致，提示模型重新生成。

- 示例：
- 错误输出：“零存整取是一种贷款产品。”
- 防护栏触发：检测到与上下文不一致，提示模型重新生成。

# RAG答案生成：实施动态防护栏

---

Thinking: 如何实现动态防护栏技术？

**事实性校验规则**，在生成阶段，设置规则，验证生成内容是否与检索到的知识片段一致。例如，可以使用参考文献验证机制，确保生成内容有可靠来源支持，避免输出不合理的回答。

Thinking: 如何制定事实性校验规则？

当业务逻辑明确且规则较为固定时，可以人为定义一组规则，比如：

- 规则1：生成的答案必须包含检索到的知识片段中的关键实体（如“年费”、“利率”）。
- 规则2：生成的答案必须符合指定的格式（如步骤列表、表格等）。

实施方法：

使用正则表达式或关键词匹配来检查生成内容是否符合规则。

例如，检查生成内容是否包含“年费”这一关键词，或者是否符合步骤格式（如“1. 登录；2. 设置”）。




# RAG在不同阶段提升质量的实践

---

在数据准备环节，阿里云考虑到文档具有多层标题属性且不同标题之间存在关联性，提出多粒度知识提取方案，按照不同标题级别对文档进行拆分，然后基于Qwen14b模型和RefGPT训练了一个面向知识提取任务的专属模型，对各个粒度的chunk进行知识提取和组合，并通过去重和降噪，保证知识不丢失、不冗余。最终将文档知识提取成多个事实型对话，提升检索效果；

在知识检索环节，哈啰出行采用多路召回的方式，主要是向量召回和搜索召回。其中，向量召回使用了两类，一类是大模型的向量、另一类是传统深度模型向量；搜索召回也是多链路的，包括关键词、ngram等。通过多路召回的方式，可以达到较高的召回查全率。

在答案生成环节，中国移动为了解决事实性不足或逻辑缺失，采用FoRAG两阶段生成策略，首先生成大纲，然后基于大纲扩展生成最终答案。



Thank You  
Using data to solve problems