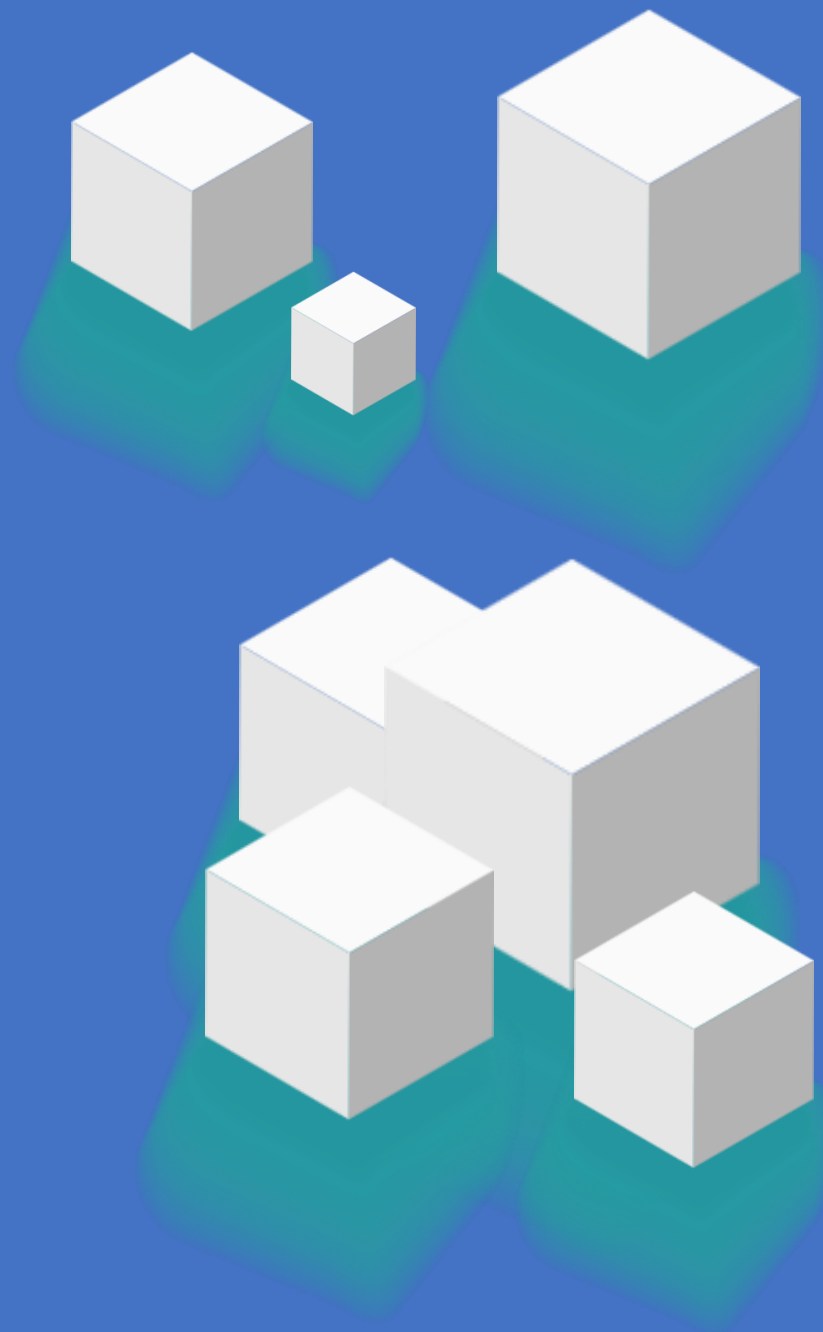
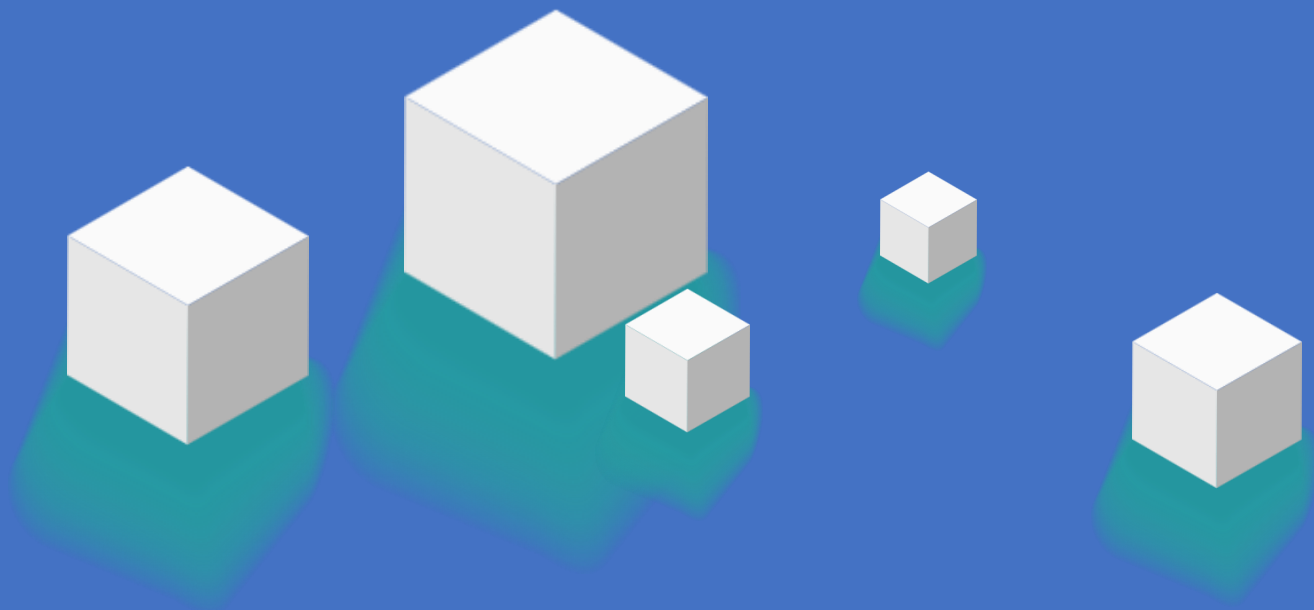


RAG技术与应用



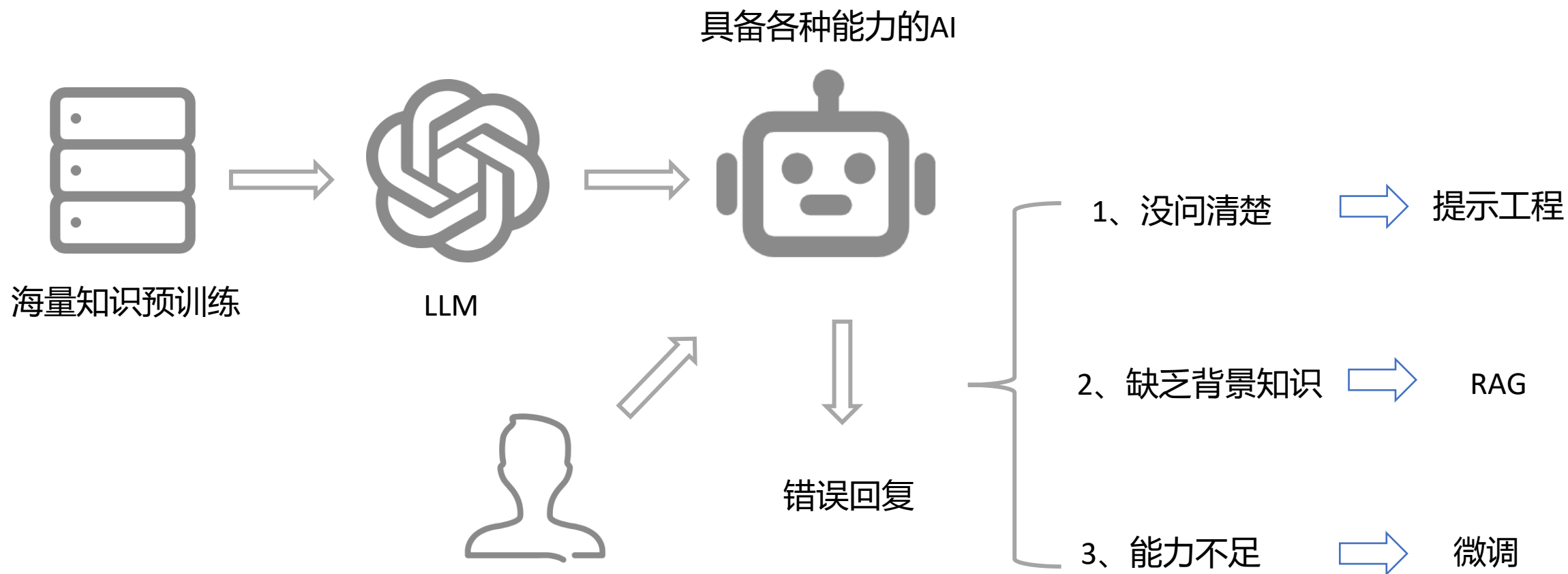
>> 今天的学习目标

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
- Embedding模型选择
- CASE：DeepSeek + Faiss搭建本地知识库检索
- CASE：迪士尼RAG助手
- RAG常见问题——如何提升RAG质量

大模型应用开发

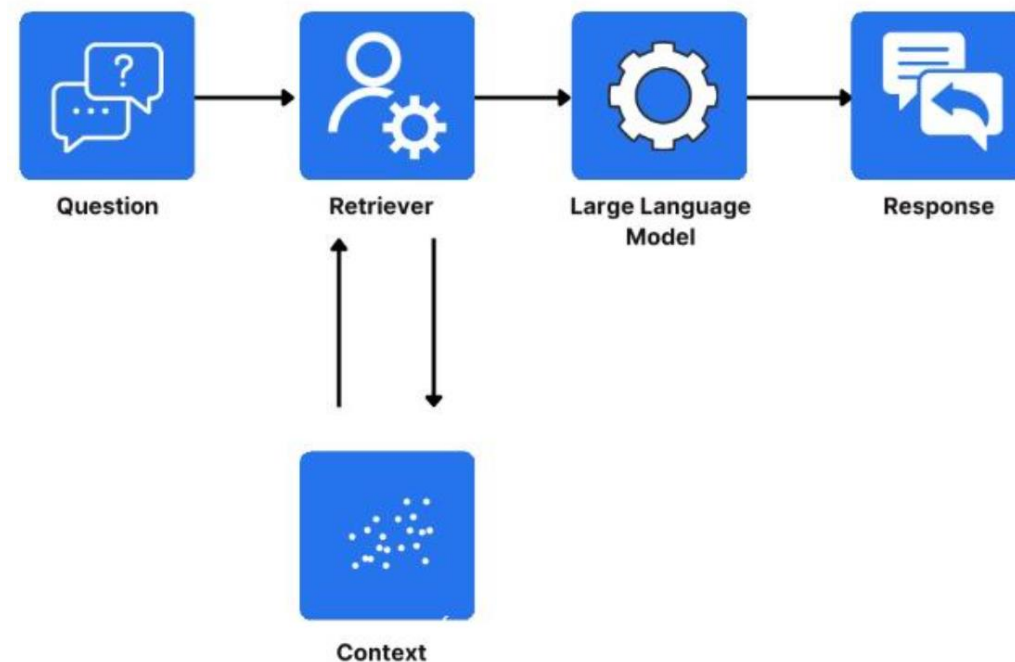
Thinking: 提示工程 VS RAG VS 微调, 什么时候使用?



什么是RAG

RAG（Retrieval-Augmented Generation）：

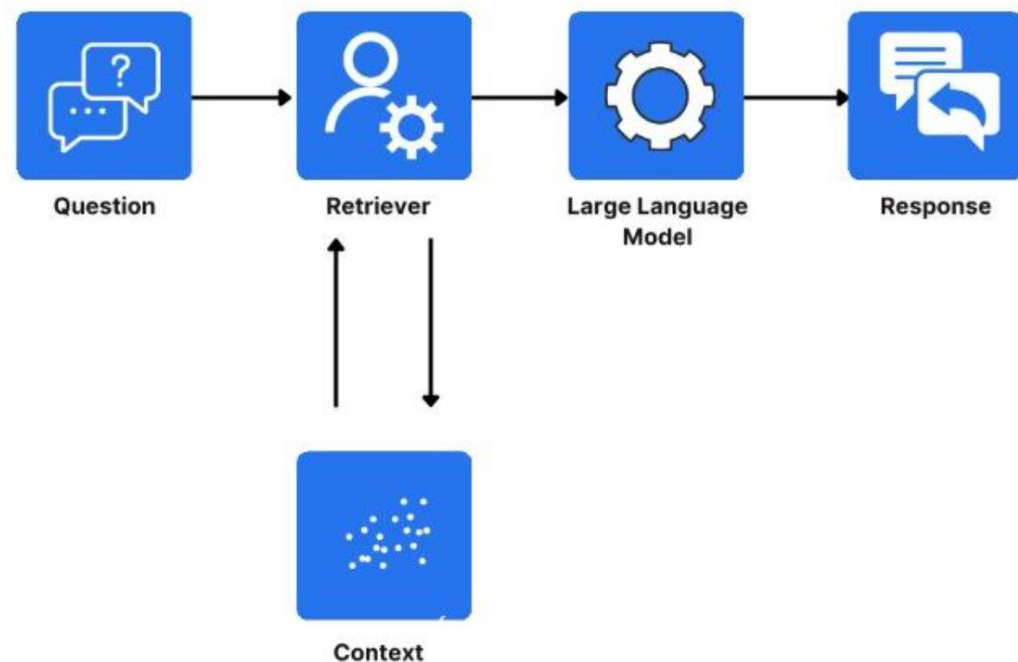
- 检索增强生成，是一种结合信息检索（Retrieval）和文本生成（Generation）的技术
- RAG技术通过实时检索相关文档或信息，并将其作为上下文输入到生成模型中，从而提高生成结果的时效性和准确性。



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

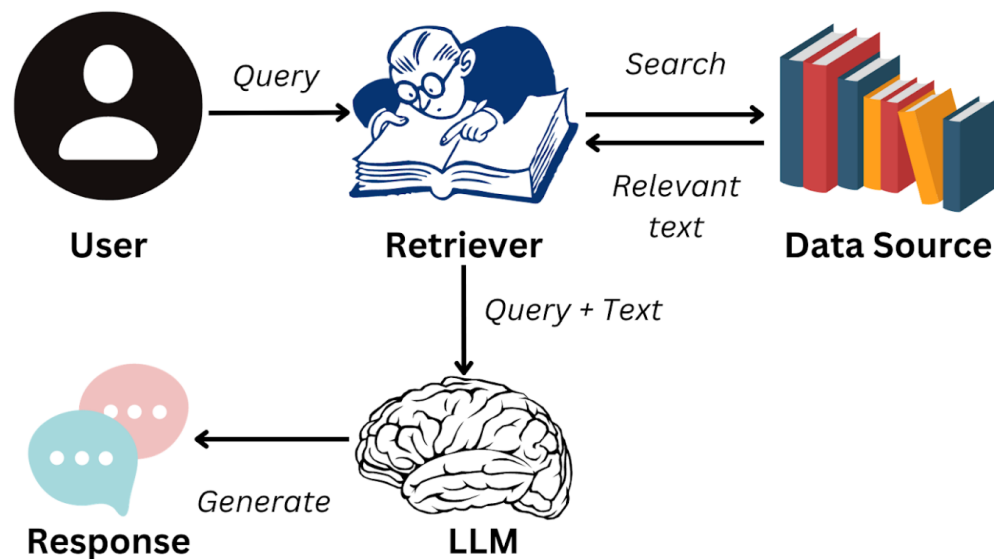
RAG的核心原理与流程

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

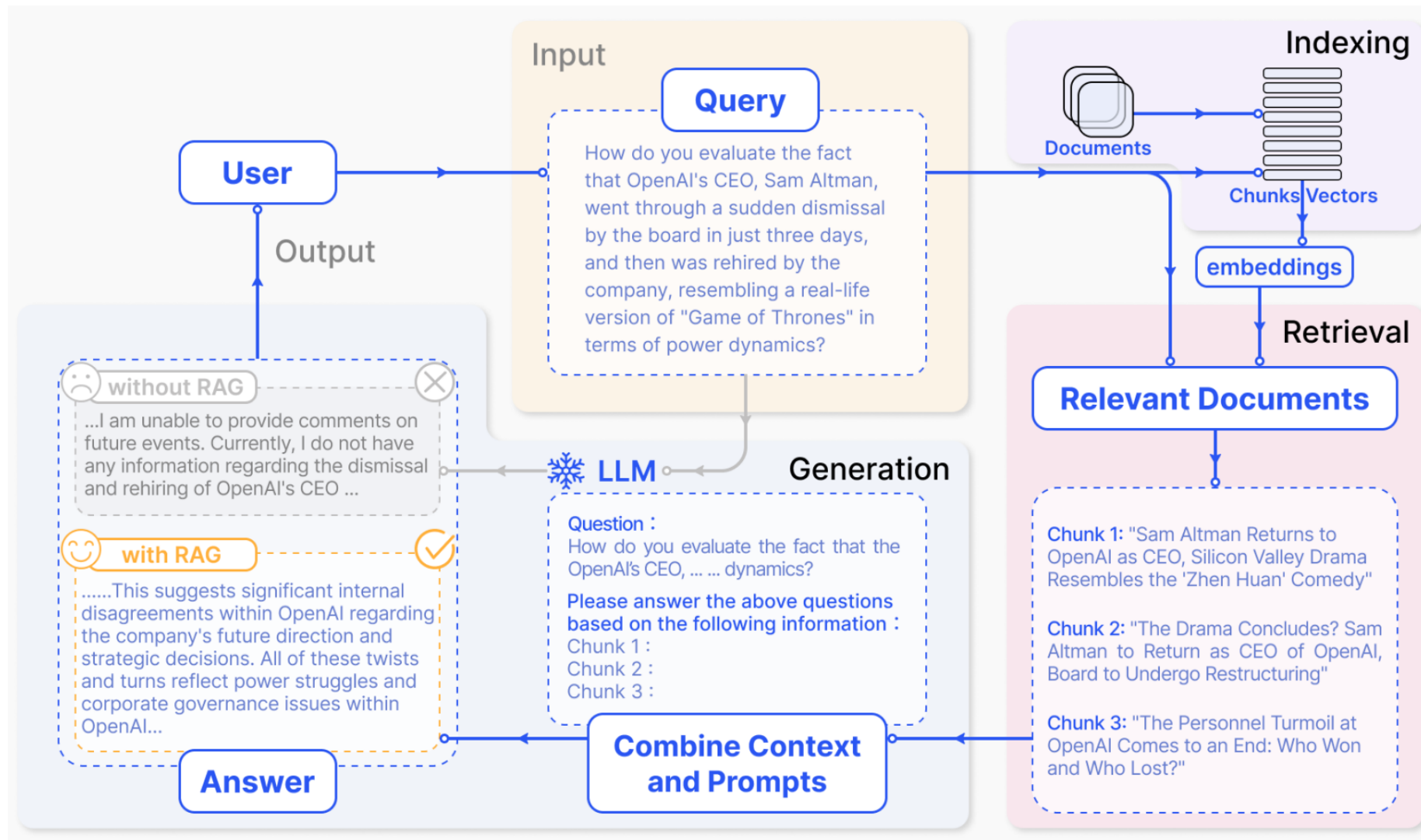
RAG的步骤:

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 (Borda)	Model	Zero-shot	Clustering	Instruction Retrieval	Multilabel Classification	Pair Classification	Reranking	Retrieval	STS
1	Linq-Embed-Mistral	99%	0.27	0.94	24.77	80.43	64.37	58.69	74.86
2	gte-Qwen2-7B-instruct	⚠ NA	0.36	4.94	25.48	85.13	65.55	60.08	73.98
3	multilingual-e5-large-instruct	99%	0.54	-0.40	22.91	80.86	62.61	57.12	76.81
4	SFR-Embedding-Mistral	96%	0.57	0.16	24.55	80.29	64.19	59.44	74.79
5	GritLM-7B	99%	0.48	3.45	22.77	79.94	63.78	58.31	73.33
6	GritLM-8x7B	99%	0.88	2.44	24.43	79.73	62.61	57.54	73.16
7	e5-mistral-7b-instruct	99%	0.39	-0.62	22.20	81.12	63.82	55.75	74.02
8	Cohere-embed-multilingual-v3.0	⚠ NA	0.61	-1.89	22.74	79.88	64.07	59.16	74.80
9	gte-Qwen2-1.5B-instruct	⚠ NA	0.59	0.74	24.02	81.58	62.58	60.78	71.61
10	bilingual-embedding-large	98%	0.24	-3.04	22.36	79.83	61.42	55.10	77.81
11	text-embedding-3-large	⚠ 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。
- 文件大小：2.3G

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-small（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-Base

- 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。
- 文件大小：0.4G（m3e-base）

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

Thinking: 项目的架构与技术选型是什么?

1. RAG架构

检索（Retrieval）：使用向量相似度搜索从PDF文档中检索相关内容

增强（Augmentation）：将检索到的文档片段作为上下文

生成（Generation）：基于上下文和用户问题生成答案

2. 技术栈选择

- 向量数据库：Faiss作为 高效的向量检索
- 嵌入模型：阿里云DashScope的text-embedding-v1
- 大语言模型：deepseek-v3
- 文档处理：PyPDF2用于PDF文本提取

CASE: DeepSeek + Faiss搭建本地知识库检索

Thinking: 程序的逻辑结构是什么？

Step1: 文档预处理

PDF文件 → 文本提取 → 文本分割 → 页码映射

1) PDF文本提取

- 逐页提取文本内容
- 记录每行文本对应的页码信息
- 处理空页和异常情况

2) 文本分割策略

- 使用递归字符分割器
- 分割参数: `chunk_size=1000, chunk_overlap=200`
- 分割符优先级: 段落 → 句子 → 空格 → 字符

3) 页码映射处理

- 基于字符位置计算每个文本块的页码
- 使用众数统计确定文本块的主要来源页码
- 建立文本块与页码的映射关系

CASE: DeepSeek + Faiss搭建本地知识库检索

Step2: 知识库构建

文本块 → 嵌入向量 → Faiss索引 → 本地持久化

1) 向量数据库构建

- 使用DashScope嵌入模型生成向量
- 将向量存储到Faiss索引结构

2) 数据持久化**

- 保存Faiss索引文件（.faiss）
- 保存元数据信息（.pkl）
- 保存页码映射关系（page_info.pkl）

Step3: 问答查询

用户问题 → 向量检索 → 文档组合 → LLM生成 → 答案输出

1) 相似度检索

- 将用户问题转换为向量
- 在Faiss中搜索最相似的文档块，返回Top-K相关文档

2) 问答链处理

- 使用LangChain的load_qa_chain
- 采用 stuff 策略组合文档
- 将组合后的上下文和问题发送给LLM

3) 答案生成与展示

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_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)
```

```
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",
dashescope_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DashScopeEmbeddings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Qwen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

如果是openai模型，可以用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 (LangChain)



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: 如果得到的文本块的页码不对，如何调整？

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:
```

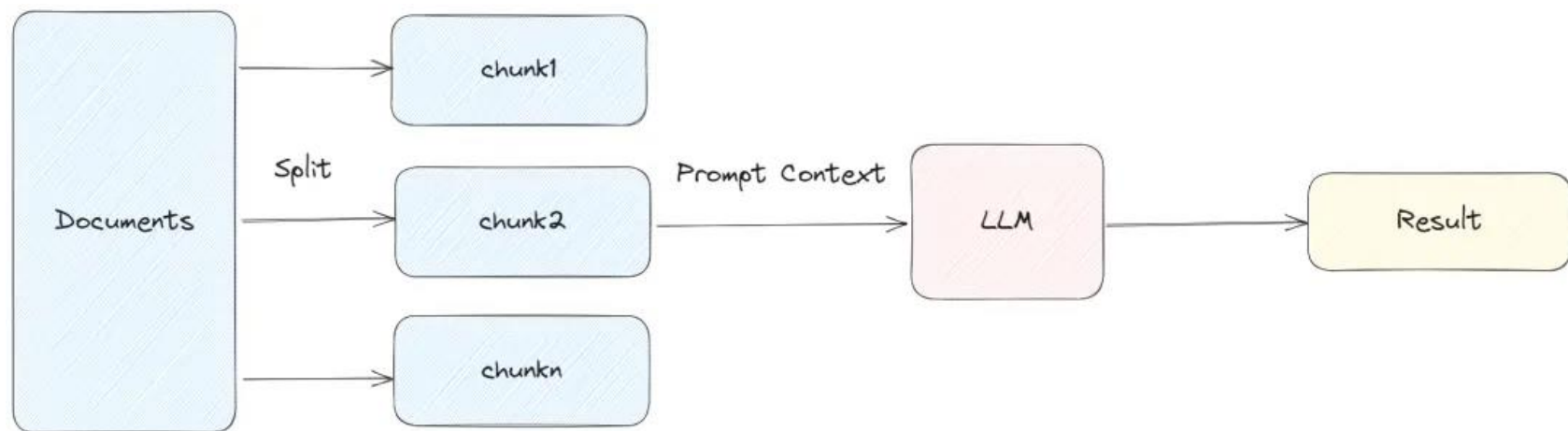
```
        # 执行问答链
```

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

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

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

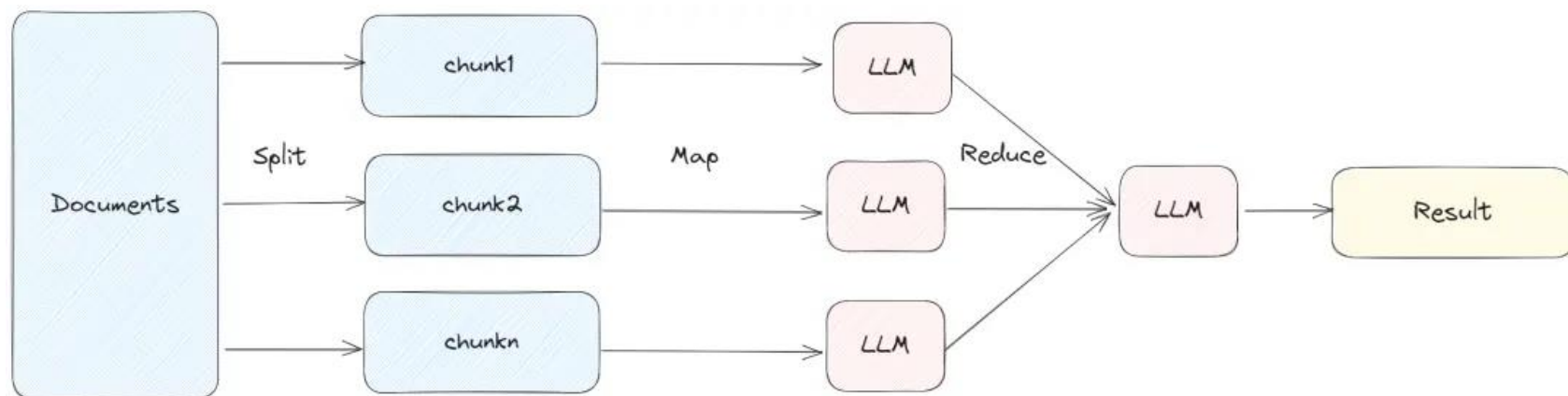
LangChain中的问答链



1) stuff

适合文档拆分的比较小，一次获取文档比较少的情況

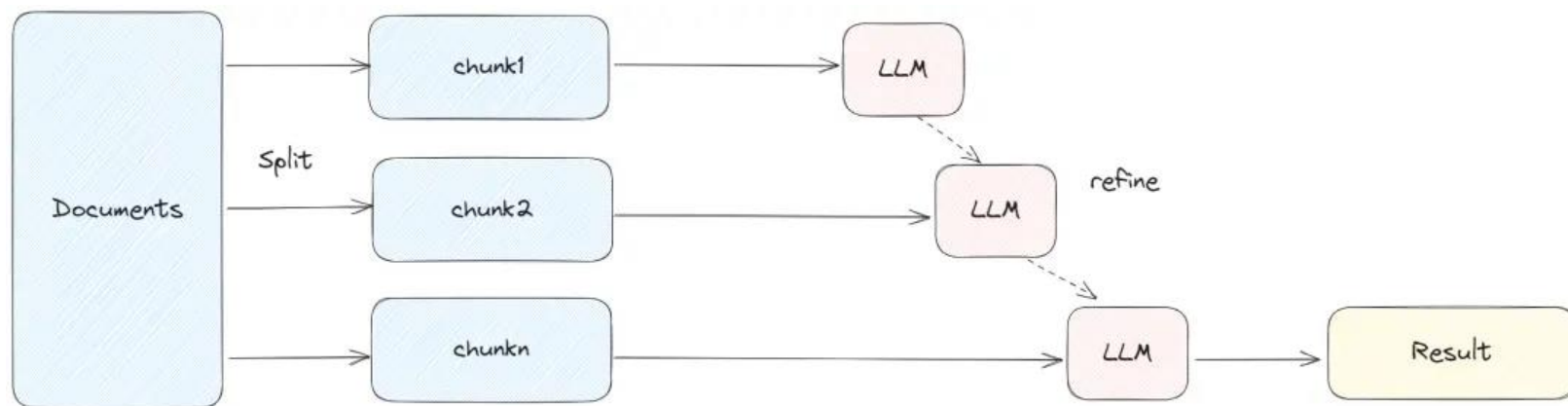
调用 LLM 的次数也比较少，能使用 stuff 的就使用这种方式。



2) map_reduce

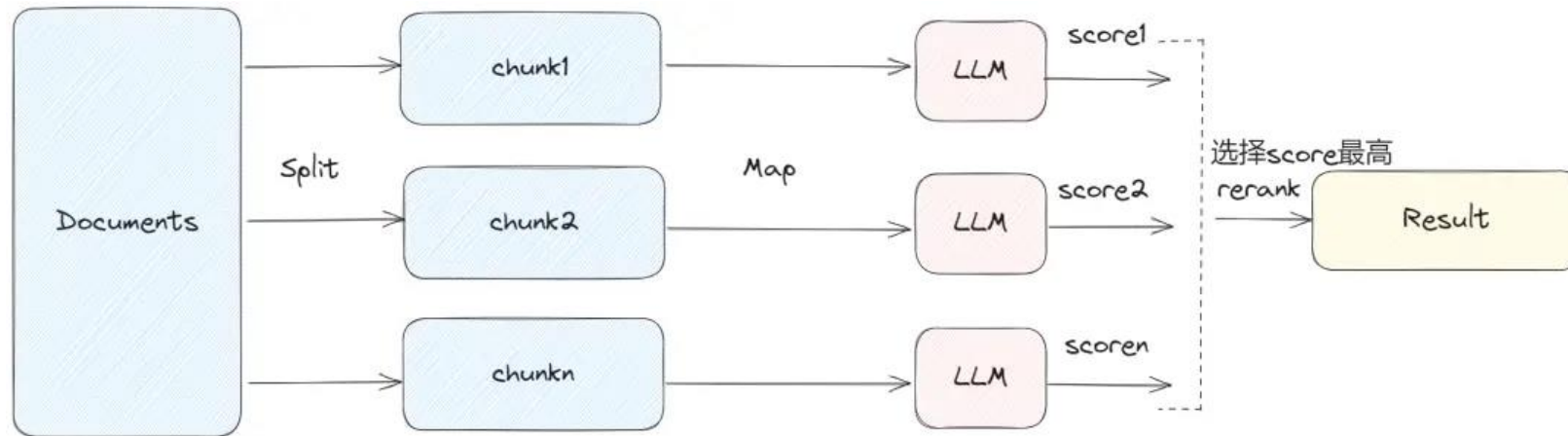
将每个 document 单独处理，可以并发进行调用。但是每个文档之间缺少上下文。

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。

CASE: 迪士尼RAG助手

CASE：迪士尼RAG助手

TO DO：搭建迪士尼RAG助手

为迪士尼构建一个7x24小时在线的AI客服助手：

- 自动化解答高频问题：如票务、入园须知、会员权益等，降低人工客服压力。
- 提供准确的回答：确保所有回答均来自官方知识库，避免信息错误或过时。
- 处理多模态查询：不仅能回答文本问题，还能理解并回应关于图片（如活动海报）的查询。



CASE：迪士尼RAG助手

Thinking：挑战都有哪些？

知识来源多样化：知识库包含多种格式的文档，如 PDF 格式的官方规定、Word 格式的内部FAQ、网页公告、以及包含大量图片和表格的活动介绍文件。

- **非结构化数据处理**：如何有效提取并理解 PDF 和 Word 文档中的表格与图片信息，这是RAG成功的关键。
- **知识的有效组织**：如何将海量、零散的知识点切片（Chunking）并建立索引，确保检索的准确性。
- **保证答案的有效性**：如何让最终生成的答案严格基于检索到的内容，避免LLM出现幻觉。

CASE：迪士尼RAG助手

Thinking：技术选型是怎样的？

- 文档处理库: PyMuPDF (处理PDF), python-docx (处理Word), pytesseract (OCR识别图片中的文字)。
- 文本Embedding模型: text-embedding-v4 (性能优秀，支持可变维度)。
- 图像Embedding模型: CLIP (由OpenAI开发，能同时理解图片和文本，是多模态RAG的核心)。
- 向量数据库/库: FAISS，作为向量检索引擎的核心，性能极高

注意：在生产环境中，可以使用Milvus, ChromaDB 或 Elasticsearch，他们提供了完整数据管理服务

- LLM: Qwen-turbo 用于最终答案的生成。
- 流程编排框架: 不依赖于LangChain，直接使用底层API

格式处理相关 (.docx)

```
def parse_docx(file_path):
    doc = DocxDocument(file_path)
    content_chunks = []

    for element in doc.element.body:
        if element.tag.endswith('p'): # 段落处理
            paragraph_text = ""
            for run in element.findall('.//w:t', {'w':
'http://schemas.openxmlformats.org/wordprocessingml/2006/main'}):
                paragraph_text += run.text if run.text else ""
            if paragraph_text.strip():
                content_chunks.append({"type": "text", "content":
paragraph_text.strip()})
        elif element.tag.endswith('tbl'): # 表格处理
            # 转换为Markdown格式
            md_table = []
            table = [t for t in doc.tables if t._element is element][0]
```

```
            if table.rows:
                header = [cell.text.strip() for cell in table.rows[0].cells]
                md_table.append("| " + " | ".join(header) + " |")
                md_table.append("| " + "---| "*len(header))

                for row in table.rows[1:]:
                    row_data = [cell.text.strip() for cell in row.cells]
                    md_table.append("| " + " | ".join(row_data) + " |")

            table_content = "\n".join(md_table)
            if table_content.strip():
                content_chunks.append({"type": "table", "content":
table_content})

    return content_chunks
```

格式处理相关 (.docx)

- 整体功能:

函数 `parse_docx` 读取一个 `.docx` 文件，遍历文件中的所有元素 (段落和表格)，并将它们提取成一个个独立的内容区块 (chunks)。

- 段落处理 (`if element.tag.endswith('p')`):

找出文档中的所有段落。

它会逐一提取每个段落内的纯文本内容，去除多余的空白，然后将其标记为 `"type": "text"` 并存储起来。

- 表格处理 (`elif element.tag.endswith('tbl')`):

专门处理文档中的“表格”。

它会将 Word 原生的表格格式转换为通用的 Markdown 格式。程序会先读取表头，然后逐行读取单元格数据，最后组合成 Markdown 表格字符串，并标记为 `"type": "table"`。

格式处理相关 (.pdf)

```
def parse_pdf(file_path, image_dir):
    doc = fitz.open(file_path)
    content_chunks = []

    for page_num, page in enumerate(doc):
        # 提取文本
        text = page.get_text("text")
        content_chunks.append({"type": "text", "content": text, "page":
page_num + 1})

        # 提取图片
        for img_index, img in enumerate(page.get_images(full=True)):
            xref = img[0]
            base_image = doc.extract_image(xref)
            image_bytes = base_image["image"]
            image_ext = base_image["ext"]
```

```
            image_path = os.path.join(image_dir,
f"{os.path.basename(file_path)}_p{page_num+1}_{img_index}.{image_ext}")

            with open(image_path, "wb") as f:
                f.write(image_bytes)
                content_chunks.append({"type": "image", "path": image_path,
"page": page_num + 1})

    return content_chunks
```

格式处理相关 (.pdf)

- 整体功能:

`parse_pdf` 使用 `fitz` (PyMuPDF) 库打开并逐页读取 PDF 文档。它的核心目标是将 PDF 这种复合式文档，拆解成纯文本和独立的图片文件，以便 RAG 模型后续能分别处理和理解。

- 文本提取 (`page.get_text("text")`):

代码遍历每一页，使用 `get_text()` 方法抓取该页所有的纯文本内容。

它将每页的文本保存为一个独立的区块 (`chunk`)，并附上页码。

- 图片提取 (`page.get_images()`):

侦测并提取页面中的所有嵌入图片。

它将每张图片的二进制数据读取出来，以唯一的文件名（包含原始文件名、页码和图片索引）保存到指定的 `image_dir` 目录下。同时，它会记录图片的存储路径。

格式处理相关 (.jpg, .png, .jpeg)

```
def image_to_text(image_path):
    try:
        image = Image.open(image_path)
        # OCR文字识别
        ocr_text = pytesseract.image_to_string(image,
lang='chi_sim+eng').strip()
        return {"ocr": ocr_text}
    except Exception as e:
        print(f"处理图片失败 {image_path}: {e}")
        return {"ocr": ""}
```

- `image_to_text` 的作用是“看图识字”。

它接收一个图片文件的路径，通过OCR，提取图片中包含的文字信息，适合扫描版 PDF 或截图。

- OCR 技术 (`pytesseract.image_to_string`):

代码调用了 `pytesseract` 库，它是 Tesseract OCR 引擎的 Python 封装。

`image_to_string` 方法是执行识别的核心。

`lang='chi_sim+eng'` 参数告诉 OCR 引擎要同时识别简体中文和英文两种语言

模态处理相关 (文本)

- 文本Embedding

```
def get_text_embedding(text):  
    response = client.embeddings.create(  
        model="text-embedding-v4",  
        input=text,  
        dimensions=1024  
    )  
    return response.data[0].embedding
```

维度：1024维

模型：text-embedding-v4

适用：语义相似度计算

模态处理相关（图像）

- CLIP图像Embedding

```
def get_image_embedding(image_path):  
    image = Image.open(image_path)  
    inputs = clip_processor(images=image, return_tensors="pt")  
    with torch.no_grad():  
        image_features = clip_model.get_image_features(**inputs)  
    return image_features[0].numpy()
```

维度：512维

模型：CLIP ViT-Base-Patch32

适用：跨模态检索

- CLIP文本Embedding（用于图像检索）

```
def get_clip_text_embedding(text):  
    inputs = clip_processor(text=text, return_tensors="pt")  
    with torch.no_grad():  
        text_features = clip_model.get_text_features(**inputs)  
    return text_features[0].numpy()
```

与图像向量在同一空间

支持跨模态相似度计算

用于文本到图像的检索

混合检索策略

```
# 文本检索
```

```
query_text_vec =
```

```
np.array([get_text_embedding(query)]).astype('float32')
```

```
distances, text_ids = text_index.search(query_text_vec, k)
```

```
# 图像检索
```

```
if any(keyword in query.lower() for keyword in image_keywords):
```

```
    query_clip_vec =
```

```
    np.array([get_clip_text_embedding(query)]).astype('float32')
```

```
    distances, image_ids = image_index.search(query_clip_vec, 2)
```

```
# 混合排序
```

```
text_results = [(match, distance, "text") for match, distance in
```

```
zip(matches, distances)]
```

```
image_results = [(match, distance, "image") for match, distance in
```

```
zip(matches, distances)]
```

```
# 优先文本，强制包含图像
```

```
for result, distance, result_type in sorted(text_results,
```

```
key=lambda x: x[1]):
```

```
    retrieved_context.append(result)
```

```
if image_results:
```

```
    best_image = min(image_results, key=lambda x: x[1])
```

```
    retrieved_context.append(best_image[0])
```


混合检索策略

结合文本和图像的搜索结果，为LLM提供更丰富的上下文：

- 并行检索：文本与图像分路进行

代码首先会无条件地执行文本检索。它将用户查询（`query`）向量化，然后在文本索引库 (`text_index`) 中搜索最相似的 `k` 个文本片段。

与此同时，它会进行一次判断：仅当用户查询包含“海报”、“图片”等预设的图像关键词时，才会触发图像检索。图像检索使用专门的 CLIP 模型生成向量，并在图像索引库 (`image_index`) 中搜索最相关的图片。

- 排序策略：“文本优先，强制包含最佳图片”

采用了一种有偏向的策略。

首先，将所有检索到的文本结果按照相关性（距离 `distance`）从高到低排序。

然后，如果图像检索被触发，它会从图片结果中选出唯一一张最相关的图片（`best_image`）

CASE：迪士尼RAG助手（用户问题1）

CLIP 模型加载成功。

--- 步骤 1 & 2: 正在解析、Embedding并索引知识库 ---

- 正在处理: 1-上海迪士尼门票规则.docx
- 正在处理: 2-迪士尼老人票价规定.docx
- 正在处理: 3-迪士尼乐园游玩攻略清单.docx
- 正在处理: 4-上海迪士尼乐园酒店会员制度.docx
- 正在处理独立图片文件...
 - 处理图片: 1-聚在一起说奇妙.jpg
 - 处理图片: 2-万圣节.jpeg

索引构建完成。共索引 30 个文本片段和 2 张图片。

=====

迪士尼客服RAG助手已准备就绪，开始模拟提问。

=====

--- 收到用户提问: '我想了解一下迪士尼门票的退款流程' ---

- 步骤 1: 向量化查询并进行检索...
 - 文本检索命中 (ID: 0, 距离: 0.5303)
 - 文本检索命中 (ID: 1, 距离: 0.6309)
 - 文本检索命中 (ID: 2, 距离: 0.6513)
- 步骤 2: 构建 Prompt...

--- Prompt Start ---

你是一个迪士尼客服助手。请根据以下背景知识，用友好和专业的语气回答用户的问题。请只使用背景知识中的信息，不要自行发挥。

[背景知识]

背景知识 1 (来源: 1-上海迪士尼门票规则.docx):

上海迪士尼门票规则

CASE：迪士尼RAG助手（用户问题1）

背景知识 2 (来源: 1-上海迪士尼门票规则.docx):

上海迪士尼乐园门票分为一日票、两日票和特定日票三种类型。一日票可在购买时选定日期使用，价格根据季节浮动，平日成人票475元起，高峰日659元起，特别高峰日799元起。两日票需连续两天使用，总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段，需注意门票标注的有效期限。

背景知识 3 (来源: 1-上海迪士尼门票规则.docx):

购票渠道以官方渠道为主，包括上海迪士尼官网、官方App、微信公众号及小程序。第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识。所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

[用户问题]

我想了解一下迪士尼门票的退款流程

--- Prompt End ---

- 步骤 3: 调用 LLM 生成最终答案...

--- 最终答案 ---

根据目前的门票规则，上海迪士尼乐园的门票一旦购买，通常是不支持退款的。因此，在购票前请您务必确认好出行日期及行程安排。

如您遇到特殊情况需要处理门票，建议您尽快通过购票渠道联系官方客服，以获取最合适的解决方案。购票时请注意查看所购门票的具体条款和条件，以便了解相关的使用规则。

希望以上信息对您有帮助，如有其他问题，欢迎随时咨询！

CASE：迪士尼RAG助手（用户问题2）

--- 收到用户提问: '最近万圣节的活动海报是什么' ---

- 步骤 1: 向量化查询并进行检索...
- 文本检索命中 (ID: 12, 距离: 1.1134)
- 文本检索命中 (ID: 8, 距离: 1.1237)
- 文本检索命中 (ID: 13, 距离: 1.2508)
- 检测到图像查询关键词, 执行图像检索...
- 图像检索命中 (ID: 30, 距离: 165.7778)
- 步骤 2: 构建 Prompt...

--- Prompt Start ---

你是一个迪士尼客服助手。请根据以下背景知识，用友好和专业的语气回答用户的问题。请只使用背景知识中的信息，不要自行发挥。

[背景知识]

背景知识 1 (来源: 1-上海迪士尼门票规则.docx):

疫情防控常态化期间仍需备好随申码，体温检测异常者需在临时观察区等待复查。雨天政策照常运营，雷暴天气部分户外项目暂停，可凭票根在官网申请雨天补偿券。季节性活动期间票价可能上浮，如万圣节夜场票需单独购买。

背景知识 2 (来源: 1-上海迪士尼门票规则.docx):

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批。

背景知识 3 (来源: 1-上海迪士尼门票规则.docx):

票务诈骗常见于二手平台，注意识别票面二维码真伪。转赠门票需在官方渠道操作，私下转让可能导致门票失效。员工折扣票仅限在职员工直系亲属使用，需提前报备个人信息。

CASE：迪士尼RAG助手（用户问题2）

背景知识 4 (来源: 独立图片: 1-聚在一起说奇妙.jpg):

找到一张相关图片，图片路径: `disney_knowledge_base\images\1-聚在一起说奇妙.jpg`。图片上的文字是:"

[用户问题]

最近万圣节的活动海报是什么

--- Prompt End ---

- 步骤 3: 调用 LLM 生成最终答案...

--- 最终答案 ---

关于上海迪士尼最近万圣节活动的海报，我目前无法直接展示图片或文件。不过，根据背景信息，万圣节期间可能会有特别的季节性活动，包括夜场票需单独购买。

您可以通过以下方式查看最新活动海报：

1. 访问上海迪士尼的官方网站或官方社交媒体账号。
2. 前往迪士尼乐园现场，海报通常会在园区内显著位置展示。

如果需要进一步帮助，请告诉我！

(同时，我为您找到了相关图片，路径为:

`disney_knowledge_base\images\1-聚在一起说奇妙.jpg`)

CASE：迪士尼RAG助手（用户问题3）

--- 收到用户提问: '迪士尼年卡有什么优惠' ---

- 步骤 1: 向量化查询并进行检索...

- 文本检索命中 (ID: 14, 距离: 0.6444)

- 文本检索命中 (ID: 27, 距离: 0.6526)

- 文本检索命中 (ID: 5, 距离: 0.7085)

- 步骤 2: 构建 Prompt...

--- Prompt Start ---

你是一个迪士尼客服助手。请根据以下背景知识，用友好和专业的语气回答用户的问题。请只使用背景知识中的信息，不要自行发挥。

[背景知识]

背景知识 1 (来源: 2-迪士尼老人票价规定.docx):

迪士尼老人票价规定

背景知识 2 (来源: 4-上海迪士尼乐园酒店会员制度.docx):

首先迪士尼vip服务大致分为三种，一种是尊享卡（园区内十二个热门项目可单独购买）入园以后在上海迪士尼度假区app绑定门票以后可购买，这相当于单次的快速通道，价格的话每天都会有波动，客流量小的日子单次80一位，客流量大的日子会有单次100，120，150，180等价格不等，具体以app公布为准。尊享卡由于需要入园购买，所以入园比较晚的朋友可能会买不到，因为如果尊享卡出售的多了大家都购买排队也就长了，所以官方为了控制尊享卡通道排队时长都是限量出售的，售罄就没了。比较容易售罄的是热门项目，飞跃地平线小矮人矿山车等。但是特别喜欢某个小项目的，就可以多次购买同一个项目的尊享卡，多次进入，曾经淡季时候我儿子把飞跃地平线连刷了五次，当天尊享卡120一次，小孩子还必须有一个大人陪同，尊享卡是有单独的通道的，一般在正常的免费排队通道旁边，也就是大家通常说的vip通道。

CASE：迪士尼RAG助手（用户问题3）

背景知识 3 (来源: 1-上海迪士尼门票规则.docx):

年卡分为宝石卡、珍珠卡、翡翠卡三种，价格从2399元至4399元不等。年卡用户享提前入园、商品餐饮折扣、专属活动等权益。年卡激活需本人持证件办理，补卡费200元，转让需支付卡面价30%手续费。

[用户问题]

迪士尼年卡有什么优惠

--- Prompt End ---

- 步骤 3: 调用 LLM 生成最终答案...

--- 最终答案 ---

您好！上海迪士尼的年卡分为宝石卡、珍珠卡和翡翠卡三种类型，价格从2399元至4399元不等。年卡用户可以享受以下优惠：

1. ****提前入园****：年卡用户可以享受提前入园的权益。
2. ****商品和餐饮折扣****：年卡用户在购买商品和餐饮时可以享受专属折扣。
3. ****专属活动****：年卡用户有机会参与一些专属活动。

激活年卡时，需要本人持证件办理。如果年卡遗失需要补卡，补卡费用为200元。年卡转让需支付卡面价格的30%作为手续费。

希望这些信息对您有帮助！如有其他问题，欢迎随时咨询。

切片策略

Thinking: 知识切片是RAG系统的核心环节，直接影响检索质量和回答准确性。都有哪些常用的切片策略？

方法1：改进的固定长度切片

方法2：语义切片

方法3：LLM语义切片

方法4：层次切片

方法5：滑动窗口切片

切片策略

方法1：改进的固定长度切片

- 在固定长度基础上，**优先在句子边界进行切分，通过重叠机制确保上下文连续性**，同时保持长度可控。
- 实现简单，处理速度快，长度统一，适合技术文档和规范文件。
- 适用场景：需要统一处理长度的场景，批量处理大量文档。

方法2：语义切片

- 基于自然语言处理，按句子、段落等语义单位进行切分。保持语义完整性，避免在句子中间断开，确保每个切片都是完整的语义单元。**无重叠，按语义边界切分。**
- 语义保持好，检索准确性高，但长度可能不均匀。
- 适用场景：适用于自然语言文本，需要保持语义完整性的场景。

切片策略

方法3: LLM语义切片

- 利用LLM的语义理解能力，在保持语义完整性的同时实现精确的长度控制。
- 语义理解能力强，分割点选择智能，但依赖GPU，成本较高。
- 适用场景：高质量要求的场景，复杂语义结构，有预算支持的项目。

方法4: 层次切片

- 基于文档的层次结构（标题、章节、段落）进行切分。便于理解文档的逻辑关系。
- 保持文档结构，支持层次化查询，但依赖文档格式。
- 适用场景：结构化文档（手册、规范），多级标题的文档。

切片策略

方法5：滑动窗口切片

- 使用固定大小的窗口在文本上滑动，产生重叠的切片。通过重叠机制确保上下文连续性，减少信息丢失，提高检索召回率。
- 保持上下文连续性，减少信息丢失，但产生大量重叠内容。
- 适用场景：需要重叠信息的场景，长文档处理，需要保持上下文的场景。

切片策略对比

Thinking: 针对某个知识库，不同的切片策略结果如何，优缺点是怎样的？

迪士尼乐园提供多种门票类型以满足不同游客需求。一日票是最基础的门票类型，可在购买时选定日期使用，价格根据季节浮动。两日票需要连续两天使用，总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段，需注意门票标注的有效期限。

购票渠道以官方渠道为主，包括上海迪士尼官网、官方App、微信公众号及小程序。第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识。所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批。

切片策略对比（改进的固定长度切片）

方法1：改进的固定长度切片

块 1 (292 字符):

迪士尼乐园提供多种门票类型以满足不同游客需求。一日票是最基础的门票类型，可在购买时选定日期使用，价格根据季节浮动。两日票需要连续两天使用，总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段，需注意门票标注的有效期限。

购票渠道以官方渠道为主，包括上海迪士尼官网、官方App、微信公众号及小程序。第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识。所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。

块 2 (80 字符):

需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批。

1-固定长度切片.py

切片策略对比（语义切片）

方法2：语义切片

块 1 (287 字符):

迪士尼乐园提供多种门票类型以满足不同游客需求 一日票是最基础的门票类型，可在购买时选定日期使用，价格根据季节浮动 两日票 需要连续两天使用，总价比购买两天单日票优惠约9折 特定日票包含部分节庆活动时段，需注意门票标注的有效期限 购票渠道以官方渠道 为主，包括上海迪士尼官网、官方App、微信公众号及小程序 第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识 所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件 生日福利需在官方渠道登记，可获赠生日徽章和甜品券 半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐

块 2 (29 字符):

军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批

切片策略对比 (LLM切片)

方法3: LLM切片

```
prompt = f"""
```

请将以下文本按照语义完整性进行切片，每个切片不超过{max_chunk_size}字符。

要求：

1. 保持语义完整性
2. 在自然的分割点切分
3. 返回JSON格式的切片列表，格式如下：

```
{{  
  "chunks": [  
    "第一个切片内容",  
    "第二个切片内容",  
    ...  
  ]  
}}
```

```
}}
```

文本内容：

```
{text}
```

请返回JSON格式的切片列表：

```
"""
```

切片策略对比 (LLM切片)

LLM切片结果:

块 1 (57 字符): 迪士尼乐园提供多种门票类型以满足不同游客需求。一日票是最基础的门票类型,可在购买时选定日期使用,价格根据季节浮动。

块 2 (56 字符): 两日票需要连续两天使用,总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段,需注意门票标注的有效期限。

块 3 (71 字符): 购票渠道以官方渠道为主,包括上海迪士尼官网、官方App、微信公众号及小程序。第三方平台如飞猪、携程等合作代理商也可购票,但需认准官方授权标识。

块 4 (50 字符): 所有电子票需绑定身份证件,港澳台居民可用通行证,外籍游客用护照,儿童票需提供出生证明或户口本复印件。

块 5 (54 字符): 生日福利需在官方渠道登记,可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票,含皇家宴会厅双人餐。

块 6 (30 字符): 军人优惠现役及退役军人凭证件享8折,需至少提前3天登记审批。

切片策略对比（层次切片）

方法4：层次切片

块 1 (11 字符):

迪士尼乐园门票指南

块 2 (219 字符):

一、门票类型介绍

1. 基础门票类型

迪士尼乐园提供多种门票类型以满足不同游客需求。一日票是最基础的门票类型，可在购买时选定日期使用，价格根据季节浮动。两日票需要连续两天使用，总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段，需注意门票标注的有效期限。

2. 特殊门票类型

年票适合经常游玩的游客，提供更多优惠和特权。VIP门票包含快速通道服务，可减少排队时间。团体票适用于10人以上团队，享受团体折扣。

切片策略对比（层次切片）

块 3 (214 字符):

二、购票渠道与流程

1. 官方购票渠道

购票渠道以官方渠道为主，包括上海迪士尼官网、官方App、微信公众号及小程序。这些渠道提供最可靠的服务和最新的票务信息。

2. 第三方平台

第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识。建议优先选择官方渠道以确保购票安全。

3. 证件要求

所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

切片策略对比（层次切片）

块 4 (264 字符):

三、入园须知

1. 入园时间

乐园通常在上午8:00开园，晚上8:00闭园，具体时间可能因季节和特殊活动调整。建议提前30分钟到达园区。

2. 安全检查

入园前需要进行安全检查，禁止携带危险物品、玻璃制品等。建议轻装简行，提高入园效率。

3. 园区服务

园区内提供寄存服务、轮椅租赁、婴儿车租赁等服务，可在游客服务中心咨询详情。

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。

军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批。

切片策略对比（滑动窗口切片）

方法5：滑动窗口切片

块 1 (299 字符): 迪士尼乐园提供多种门票类型以满足不同游客需求。一日票是最基础的门票类型，可在购买时选定日期使用，价格根据季节浮动。两日 票需要连续两天使用，总价比购买两天单日票优惠约9折。特定日票包含部分节庆活动时段，需注意门票标注的有效期限。

购票渠道以官方渠道为主，包括上海迪士尼官网、官方App、微信公众号及小程序。第三方平台如飞猪、携程等合作代理商也可购票，但需 认准官方授权标识。所有电子票需绑定身份证件，港澳台居民可用通行证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。军人优惠现役及

块 2 (173 字符): 小程序。第三方平台如飞猪、携程等合作代理商也可购票，但需认准官方授权标识。所有电子票需绑定身份证件，港澳台居民可用通行 证，外籍游客用护照，儿童票需提供出生证明或户口本复印件。

生日福利需在官方渠道登记，可获赠生日徽章和甜品券。半年内有效结婚证持有者可购买特别套票，含皇家宴会厅双人餐。军人优惠现役及退役军人凭证件享8折，需至少提前3天登记审批。

块 3 (23 字符): 退役军人凭证件享8折，需至少提前3天登记审批。

切片策略对比

策略	语义保持	长度控制	实现复杂度	处理速度	适用场景
改进固定长度	⚠️ 中等	✅ 优秀	✅ 简单	✅ 快速	技术文档、规范
语义切片	✅ 优秀	⚠️ 中等	⚠️ 中等	⚠️ 中等	自然语言文本
LLM语义切片	✅ 优秀	✅ 优秀	❌ 复杂	❌ 慢	高质量要求
层次切片	✅ 优秀	❌ 差	⚠️ 中等	⚠️ 中等	结构化文档
滑动窗口	⚠️ 中等	✅ 优秀	✅ 简单	✅ 快速	长文档处理

打卡：迪士尼RAG助手



搭建完整的Disney RAG助手（原生RAG应用）

Step1, 数据层

- 文档处理：解析Word文档(.docx)，提取文本段落和表格（转为Markdown格式）
- 图像处理：支持图片OCR（Tesseract）和CLIP视觉特征提取

Step2. 向量化层

- 文本Embedding：使用阿里云百炼的 text-embedding-v4 模型（1024维）
- 图像Embedding：使用CLIP模型提取图像特征（512维）
- 双索引系统：FAISS分别构建文本和图像的向量索引

Step3. 检索层

- 混合检索：文本查询使用语义相似度检索，图像查询使用CLIP文本编码器
- 关键词触发：检测特定关键词（如"海报"、"图片"）触发图像检索

Step4. 生成层：将检索到的上下文组织成结构化提示

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。

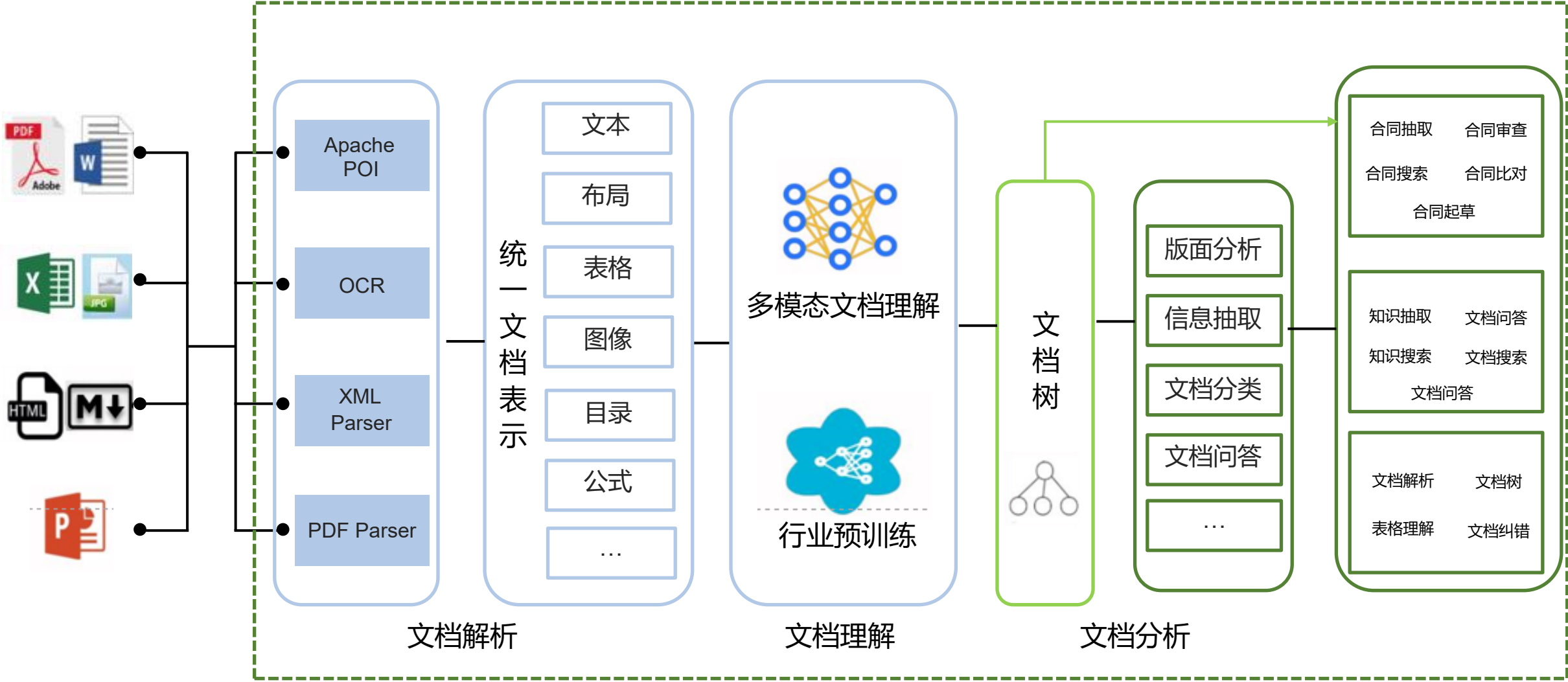
5. 数据治理框架

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，从而导致错误的答案。
- **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