实操练习指南

基础任务（15分钟）

Task 1: 创建并配置环境（5分钟）

参考：

```
1. 创建项目目录结构（已有）
mkdir -p happyCake/{app,knowledge_base/{products,orders,faq}}
2. 创建并激活conda环境
conda create -n openai-env python=3.10
conda activate openai-env
3. 安装依赖包
pip install langchain>=0.2.0 langchain-openai langchain-community langchain-core
langchain-text-splitters python-dotenv faiss-cpu dashscope gradio==3.50.2 -i
https://pypi.tuna.tsinghua.edu.cn/simple
4. 配置环境变量
创建 .env 文件并添加以下内容：
OPENAI_API_KEY=your_api_key
OPENAI_API_BASE="https://api.deepseek.com"
DASHSCOPE_API_KEY="sk-62244f697212473f8624c61ab08b362f" #培训期间共用此key，你也可在
https://dashscope.aliyun.com/ 注册自己的key
```

Task 2: 知识库导入（5分钟）

参考代码，使用DirectoryLoader加载知识库，并使用RecursiveCharacterTextSplitter分割文档，使用DashScopeEmbeddings进行向量嵌入，最后保存到FAISS中。
因环境差异，可能出错，借助AI IDE工具，修改代码，解决问题。

```
from langchain_community.document_loaders import DirectoryLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS
def build_knowledge_base():
# 1. 加载文档
loader = DirectoryLoader(
'knowledge_base',
glob="/.md",
show_progress=True
)
documents = loader.load()
# 2. 分割文档
splitter = RecursiveCharacterTextSplitter(
chunk_size=500,
chunk_overlap=50
)
texts = splitter.split_documents(documents)
# 3. 创建向量存储
```

```
embeddings = DashScopeEmbeddings(
    model="text-embedding-v1"
)
vector_store = FAISS.from_documents(texts, embeddings)
# 4. 保存向量存储
vector_store.save_local("knowledge_base/vector_store")
#运行此函数并验证向量存储是否成功创建
```

Task 3: 基本查询（5分钟）

参考：

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
def test_basic_query():
# 1. 加载向量存储
embeddings = OpenAIEmbeddings()
vector_store = FAISS.load_local(
"knowledge_base/vector_store",
embeddings
)
# 2. 创建检索器
retriever = vector_store.as_retriever(search_kwargs={"k": 3})
# 3. 测试查询
test_questions = [
"黑森林蛋糕的价格是多少？",
"你们的蛋糕保质期是多久？",
"可以提前多久预订蛋糕？"
]
# 4. 执行查询并打印结果
for question in test_questions:
docs = retriever.get_relevant_documents(question)
print(f"\n问题: {question}")
print("相关文档片段:")
for i, doc in enumerate(docs, 1):
print(f"\n{i}. {doc.page_content[:200]}...")
#运行测试并检查结果的相关性
```

进阶任务（15分钟）

Task 4: 添加新的知识文档（5分钟）

完善新增加的文档内容，并测试新添加的内容是否可被检索。

```
1. 创建新的产品文档
knowledge_base/products/seasonal_cakes.md
def add_new_document():
"""
创建一个新的季节限定蛋糕文档，包含以下内容：
1. 至少3个季节限定产品
```

```
每个产品的详细描述
价格和供应时间
预订说明
"""
# 编写文档内容,这一步可使用AI辅助完成
content = """# 季节限定蛋糕系列
## 1. 春日樱花蛋糕
产品描述：使用日本进口樱花粉末...
供应时间：3月-4月
价格：...
## 2. 夏日西瓜慕斯
...
"""
# 将内容写入文件
with open("knowledge_base/products/seasonal_cakes.md", "w", encoding="utf-8") as f:
f.write(content)
# 重新构建知识库
build_knowledge_base()
#实现函数并测试新添加的内容是否可被检索
```

Task 5: 提示词优化（5分钟）

借助AI，根据提示词书写原则，优化提示词，使回答更准确，或者更加拟人化。

```
def create_optimized_prompt():
# 1. 创建更细致的提示模板
template = """你是甜品日记蛋糕店的专业客服。
请根据以下上下文信息，用专业且友善的语气回答客户的问题。
回答要求：
如果是询问产品，请包含价格和预订建议
如果是询问保质期，请强调储存条件
如果信息不完整，可以主动推荐相关产品
如果完全没有相关信息，请礼貌说明并建议咨询客服
上下文信息：
{context}
客户问题：{question}
回答："""
# 2. 创建新的提示模板
prompt = ChatPromptTemplate.from_template(template)
# 3. 测试优化后的提示词
test_cases = [
"有什么季节限定蛋糕？",
"生日蛋糕需要提前多久预订？",
"可以定制图案吗？"
]
return prompt, test_cases
#实现并测试优化后的提示词效果
```

Task 6: 自定义响应格式（5分钟）

调整提示词，定义输出格式，使回答更加美观。

```
class CustomResponseFormatter:
@staticmethod
def format_product_response(response: str) -> str:
"""
将产品相关的回答格式化为更易读的形式
任务：
1. 识别回答中的产品名称、价格和描述
使用emoji和分隔符美化输出
添加预订提示信息
"""
    # 实现格式化逻辑
    formatted = "🎂 产品信息\n"
    formatted += "================\n"
    # ... 添加格式化逻辑 ...
    return formatted

#实现格式化器并测试效果
```

练习评估

基础任务

- [ ] 成功创建并配置环境
- [ ] 知识库正确导入并可检索
- [ ] 基本查询能返回相关结果

进阶任务

- [ ] 新知识文档被正确添加和索引
- [ ] 优化后的提示词能产生更好的回答
- [ ] 响应格式美观且信息完整

挑战任务（可选）

1. 实现多轮对话记忆功能
2. 添加简单的意图识别
3. 实现查询结果重排序