中医临床诊疗智能助手 - RAG 项目实战

这节课会带给你

1. 如何用你的垂域数据补充 LLM 的能力
2. 如何构建你的垂域（向量）知识库
3. 搭建一套完整 RAG 系统需要哪些模块
4. 搭建 RAG 系统时更多的有用技巧
5. 如何提升 RAG 检索的效果及优化实践

学习目标：

1. RAG 技术概述
2. RAG WorkFlow 及 RAG 工程化
3. 基于 LlamaIndex 快速构建 RAG 项目
4. 使用 LlamaIndex 存储和读取 Embedding 向量
5. 追踪哪些文档片段被用于检索增强生成
6. 深度剖析 RAG 检索底层实现细节
7. 自定义 RAG Prompt Template
8. RAG 项目企业级生产部署最佳实践

一、 RAG 技术概述

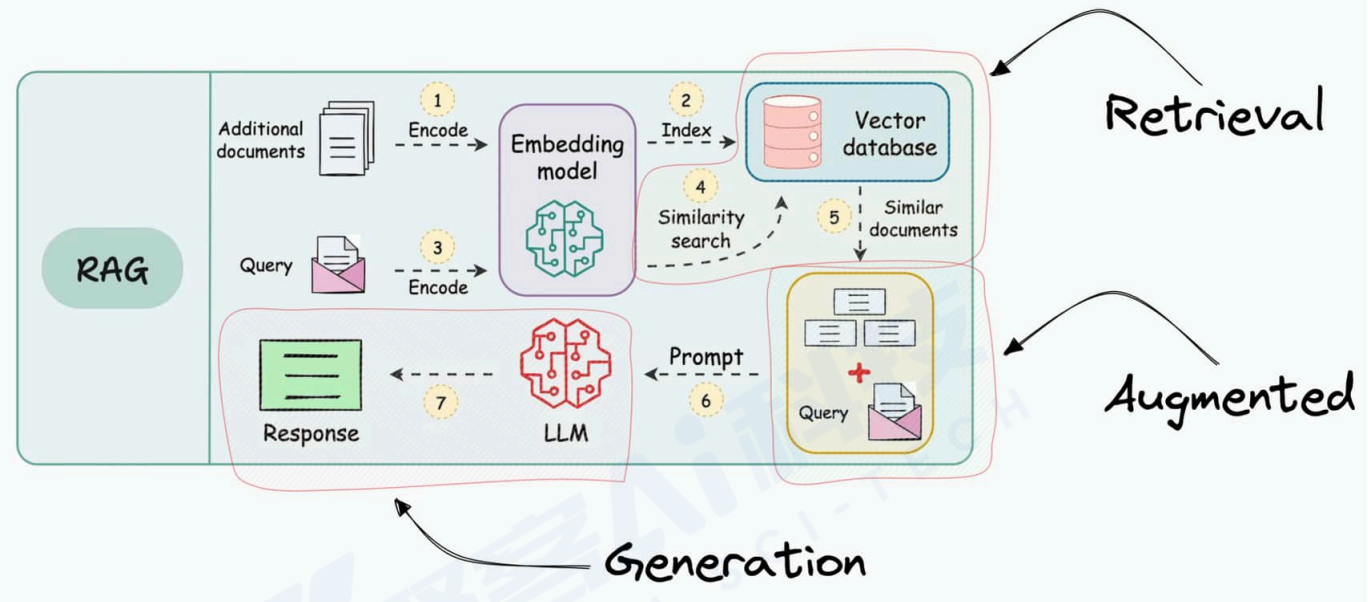
1.1 大模型目前固有的局限性

大语言模型（LLM）是概率生成系统

- **知识时效性**：模型知识截止于训练数据时间点（**联网搜索**）
- **推理局限性**：本质是概率预测而非逻辑运算，复杂数学推理易出错（**DeepSeek-R1的架构有所不同**）
- **专业领域盲区**：缺乏垂直领域知识
- **幻觉现象**：可能生成看似合理但实际错误的内容

1.2 什么是 RAG?

RAG（Retrieval Augmented Generation）顾名思义，通过**检索**的方法来增强**生成模型**的能力。



二、RAG 工程化

2.1 RAG系统的基本搭建流程

搭建过程：

1. 文档加载，并按一定条件**切割**成片段
2. 将切割的文本片段灌入**检索引擎**
3. 封装**检索接口**
4. 构建**调用流程**：Query -> 检索 -> Prompt -> LLM -> 回复

2.2 构建索引

2.3 检索和生成

三、项目环境配置

3.1 使用 conda 创建项目环境

```
# 创建环境
conda create -n tcm-ai-rag python=3.10

# 激活环境
conda activate tcm-ai-rag
```

3.2 安装项目所需依赖库

In [ ]:
```
# 安装 LlamaIndex 相关包
# !pip install llama-index
# !pip install llama-index-embeddings-huggingface
# !pip install llama-index-llms-huggingface
```

In [ ]:
```
# 安装 CUDA 版本 Pytorch
# !pip install torch==2.5.1 torchvision==0.20.1 torchaudio==2.5.1 --index-url https://download.pytorch.org/whl/cu118
```

四、模型下载

In [ ]:
```
# 安装 modelscope
# !pip install modelscope
```

4.1 下载 Embedding 模型权重

使用BAAI开源的中文bge模型作为embedding模型，使用modlescope提供的SDK将模型权重下载到本地服务器：

In [ ]:
```
# 使用 modelscope 提供的 sdk 进行模型下载
from modelscope import snapshot_download

# model_id 模型的id
# cache_dir 缓存到本地的路径
model_dir = snapshot_download(model_id="BAAI/bge-base-zh-v1.5", cache_dir="D:/AIProject/modelscope")
```

4.2 下载 LLM 大模型权重

使用阿里开源的通义千问大模型，使用modelscope提供的SDK将模型权重下载到服务器：

In [ ]:
```
from modelscope import snapshot_download

model_dir = snapshot_download(model_id="Qwen/Qwen2.5-7B-Instruct", cache_dir="D:/AIProject/modelscope")
```

五、构建中医临床诊疗术语证候问答

5.1 语料准备

本应用使用的文档是由国家卫生健康委员和国家中医药管理局发布的中医临床诊疗术语：

- 《中医临床诊疗术语第1部分：疾病》（修订版）.docx
- 《中医临床诊疗术语第2部分：证候》（修订版）.docx
- 《中医临床诊疗术语第3部分：治法》（修订版）.docx

部分内容展示：

4.1.1.2.1
气机阻滞证 syndrome/pattern of obstructed qi movement
泛指因各种原因导致气机不畅，或气郁而不散，阻滞脏腑、经络、官窍等所引起的一类证候。

4.1.1.2.1.1
气机郁滞证 syndrome/pattern of qi activity stagnation
因气机郁结，阻滞经络或脏腑官窍所致。临床以头颈肩背或胸胁脘腹等处闷胀，或攻窜作痛，常随紧张、抑郁等情绪缓解，或得太息、嗳气、肠鸣、矢气而减轻，脉弦，可伴见大便时秘或泻，小便不利，耳鸣、耳聋，嘶哑、呃逆等为特征的证候。

4.1.1.2.1.2
气滞耳窍证 syndrome/pattern of qi stagnation in the ears
因肝气郁结，气机不利，气滞耳窍所致。临床以突然耳窍失聪，或耳内堵塞，耳鸣，眩晕，脉弦，伴见胸胁胀闷，情绪抑郁等为特征的证候。

4.1.1.2.1.3
气滞声带证 syndrome/pattern of qi stagnation in the vocal fold
因气机阻滞，痹阻声带所致。临床以声音不扬、嘶哑，言语费劲或磕巴，脉弦，可伴见咽喉不适，胸闷、胁胀等为特征的证候。

5.2 基于 LlamaIndex 来快速构建知识库

5.2.1 导入所需的包

In [ ]:
```
import logging
import sys
import torch
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader, VectorStoreIndex, load_index_from_storage, StorageContext, QueryBundle
from llama_index.core.schema import MetadataMode
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.huggingface import HuggingFaceLLM
from llama_index.core.node_parser import SentenceSplitter
```

5.2.2 定义日志配置

In [2]:
```
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logging.getLogger().addHandler(logging.StreamHandler(stream=sys.stdout))
```

5.2.3 定义 System Prompt

In [3]:
```
SYSTEM_PROMPT = """You are a helpful AI assistant."""
query_wrapper_prompt = PromptTemplate(
    "[INST]<<SYS>>\n" + SYSTEM_PROMPT + "<</SYS>>\n\n{query_str}[/INST] "
)
```

5.2.4 使用 llama_index_llms_huggingface 调用本地大模型

In [ ]:
```
llm = HuggingFaceLLM(
    context_window=4096,
    max_new_tokens=2048,
    generate_kwargs={"temperature": 0.0, "do_sample": False},
    query_wrapper_prompt=query_wrapper_prompt,
    tokenizer_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    model_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    device_map="auto",
    model_kwargs={"torch_dtype": torch.float16},
)
Settings.llm = llm
```

注意：为了输出的可复现性

- 将大模型的temperature设置为0，do_sample设置为False，所以两次得到的输出基本相同；
- 如果将temperature设置为大于0的小数，do_sample设置为True，大模型每次的输出可能都是不一样的。
- 另外，如果你在实验时获得的输出与文中的输出不一致，这也是正常的，这与多个因素有关。

5.2.5 使用 llama_index_embeddings_huggingface 调用本地 embedding 模型

In [ ]:
```
Settings.embed_model = HuggingFaceEmbedding(
    model_name="D:/AIProject/modelscope/BAAI/bge-base-zh-v1___5"
)
```

5.2.6 读取文档

In [6]:
```
documents = SimpleDirectoryReader("./documents", required_exts=[".txt"]).load_data()
```

5.2.7 对文档进行切分，将切分后的片段转化为embedding向量，构建向量索引

In [ ]:
```
index = VectorStoreIndex.from_documents(documents, transformations=[SentenceSplitter(chunk_size=256)])
```

SentenceSplitter 参数详细设置：

预设会以 1024 个 token 为界切割片段，每个片段的开头重叠上一个片段的 200 个 token 的内容。

```
    chunk_size=1024,    # 切片 token 数限制
    chunk_overlap=200,  # 切片开头与前一片段尾端的重复 token 数
    paragraph_separator='\n\n\n', # 段落的分界
    secondary_chunking_regex='[^,.;。？！]+[,.;。？！]?' # 单一句子的样式
    separator=' ', # 最小切割的分界字元
```

5.2.8 构建查询引擎

In [8]:
```
query_engine = index.as_query_engine(similarity_top_k=5)
```

5.2.9 生成答案

In [ ]:
```
response = query_engine.query("不耐疲劳，口燥、咽干可能是哪些证候？")

print(response)
```

六、使用LlamaIndex存储和读取embedding向量

6.1 上面面临的问题

- 使用llama-index-llms-huggingface构建本地大模型时，会花费相当一部分时间
- 在对文档进行切分，将切分后的片段转化为embedding向量，构建向量索引时，会花费大量的时间

6.2 向量存储

In [ ]:
```
# 将embedding向量和向量索引存储到文件中
# ./doc_emb 是存储路径
index.storage_context.persist(persist_dir='./doc_emb')

# 很方便的集成目前主流的向量数据集 chroma
```

找到刚才定义的persist_dir所在的路径，可以发现该路径下有以下几个文件：

- **default_vector_store.json**：用于存储embedding向量
- **docstore.json**：用于存储文档切分出来的片段
- graph_store.json：用于存储知识图数据
- image__vector_store.json：用于存储图像数据
- **index_store.json**：用于存储向量索引

在上述代码中，我们只用到了纯文本文档，所以生成出来的graph_store.json和image__vector_store.json中没有数据。

6.3 从向量数据库检索

将embedding向量和向量索引存储到文件中后，我们就不需要重复地执行对文档进行切分，将切分后的片段转化为embedding向量，构建向量索引的操作了。

以下代码演示了如何使用LlamaIndex读取结构化文件中的embedding向量和向量索引数据：

In [ ]:
```
# 从存储文件中读取embedding向量和向量索引
storage_context = StorageContext.from_defaults(persist_dir="doc_emb")

# 根据存储的embedding向量和向量索引重新构建检索索引
index = load_index_from_storage(storage_context)

# 构建查询引擎
query_engine = index.as_query_engine(similarity_top_k=5)

# 查询获得答案
response = query_engine.query("不耐疲劳，口燥、咽干可能是哪些证候？")
print(response)
```

七、追踪哪些文档片段被检索

In [ ]:
```
# 从存储文件中读取embedding向量和向量索引
storage_context = StorageContext.from_defaults(persist_dir="doc_emb")

# 根据存储的embedding向量和向量索引重新构建检索索引
index = load_index_from_storage(storage_context)

# 构建查询引擎
query_engine = index.as_query_engine(similarity_top_k=5)

# 获取我们抽取出的相似度 top 5 的片段
contexts = query_engine.retrieve(QueryBundle("不耐疲劳，口燥、咽干可能是哪些证候？"))
print('-'*10 + 'ref' + '-'*10)
for i, context in enumerate(contexts):
    print('*'*10 + f'chunk {i} start' + '*'*10)
    content = context.node.get_content(metadata_mode=MetadataMode.LLM)
    print(content)
    print('*' * 10 + f'chunk {i} end' + '*' * 10)
print('-'*10 + 'ref' + '-'*10)

# 查询获得答案
response = query_engine.query("不耐疲劳，口燥、咽干可能是哪些证候？")
print(response)
```

八、RAG 检索底层实现细节

知道了如何追踪哪些文档片段被用于检索增强生成，但我们仍不知道RAG过程中到底发生了什么，为什么大模型能够根据检索出的文档片段进行回复？

In [ ]:
```
import logging
import sys
import torch
from llama_index.core import PromptTemplate, Settings, StorageContext, load_index_from_storage
from llama_index.core.callbacks import LlamaDebugHandler, CallbackManager
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.huggingface import HuggingFaceLLM

# 定义日志
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logging.getLogger().addHandler(logging.StreamHandler(stream=sys.stdout))


# 定义system prompt
SYSTEM_PROMPT = """You are a helpful AI assistant."""
query_wrapper_prompt = PromptTemplate(
    "[INST]<<SYS>>\n" + SYSTEM_PROMPT + "<</SYS>>\n\n{query_str}[/INST] "
)

# 使用llama-index创建本地大模型
llm = HuggingFaceLLM(
    context_window=4096,
    max_new_tokens=2048,
    generate_kwargs={"temperature": 0.0, "do_sample": False},
    query_wrapper_prompt=query_wrapper_prompt,
    tokenizer_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    model_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    device_map="auto",
    model_kwargs={"torch_dtype": torch.float16},
)
Settings.llm = llm

# 使用LlamaDebugHandler构建事件回溯器，以追踪LlamaIndex执行过程中发生的事件
llama_debug = LlamaDebugHandler(print_trace_on_end=True)
callback_manager = CallbackManager([llama_debug])
Settings.callback_manager = callback_manager

# 使用llama-index-embeddings-huggingface构建本地embedding模型
Settings.embed_model = HuggingFaceEmbedding(
    model_name="D:/AIProject/modelscope/BAAI/bge-base-zh-v1___5"
)

# 从存储文件中读取embedding向量和向量索引
storage_context = StorageContext.from_defaults(persist_dir="doc_emb")
index = load_index_from_storage(storage_context)
# 构建查询引擎
query_engine = index.as_query_engine(similarity_top_k=5)

# 查询获得答案
response = query_engine.query("不耐疲劳，口燥、咽干可能是哪些证候？")
print(response)

# get_llm_inputs_outputs 返回每个LLM调用的开始/结束事件
event_pairs = llama_debug.get_llm_inputs_outputs()
# print(event_pairs[0][1].payload.keys())
print(event_pairs[0][1].payload["formatted_prompt"])
```

九、自定义 Prompt

LlamaIndex中提供的prompt template都是英文的，该如何使用中文的prompt template呢?

In [ ]:
```
import logging
import sys
```

```python
import torch
from llama_index.core import PromptTemplate, Settings, StorageContext, load_index_from_storage
from llama_index.core.callbacks import LlamaDebugHandler, CallbackManager
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.huggingface import HuggingFaceLLM

# 定义日志
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logging.getLogger().addHandler(logging.StreamHandler(stream=sys.stdout))


# 定义system prompt
SYSTEM_PROMPT = """你是一个医疗人工智能助手。"""
query_wrapper_prompt = PromptTemplate(
    "[INST]<<SYS>>\n" + SYSTEM_PROMPT + "<</SYS>>\n\n{query_str}[/INST] "
)

# 定义qa prompt
qa_prompt_tmpl_str = (
    "上下文信息如下。\n"
    "---------------------\n"
    "{context_str}\n"
    "---------------------\n"
    "请根据上下文信息而不是先验知识来回答以下的查询。"
    "作为一个医疗人工智能助手，你的回答要尽可能严谨。\n"
    "Query: {query_str}\n"
    "Answer: "
)
qa_prompt_tmpl = PromptTemplate(qa_prompt_tmpl_str)

# 定义refine prompt
refine_prompt_tmpl_str = (
    "原始查询如下：{query_str}"
    "我们提供了现有答案：{existing_answer}"
    "我们有机会通过下面的更多上下文来完善现有答案（仅在需要时）。"
    "------------"
    "{context_msg}"
    "------------"
    "考虑到新的上下文，优化原始答案以更好地回答查询。 如果上下文没有用，请返回原始答案。"
    "Refined Answer:"
)
refine_prompt_tmpl = PromptTemplate(refine_prompt_tmpl_str)

# 使用llama-index-llm-huggingface调用本地大模型
llm = HuggingFaceLLM(
    context_window=4096,
    max_new_tokens=2048,
    generate_kwargs={"temperature": 0.0, "do_sample": False},
    query_wrapper_prompt=query_wrapper_prompt,
    tokenizer_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    model_name='D:/AIProject/modelscope/Qwen/Qwen2___5-7B-Instruct',
    device_map="auto",
    model_kwargs={"torch_dtype": torch.float16},
)
Settings.llm = llm

# 使用LlamaDebugHandler构建事件回溯器，以追踪LlamaIndex执行过程中发生的事件
llama_debug = LlamaDebugHandler(print_trace_on_end=True)
callback_manager = CallbackManager([llama_debug])
Settings.callback_manager = callback_manager

# 使用llama-index-embeddings-huggingface调用本地embedding模型
Settings.embed_model = HuggingFaceEmbedding(
    model_name="D:/AIProject/modelscope/BAAI/bge-base-zh-v1___5"
)

# 从存储文件中读取embedding向量和向量索引
storage_context = StorageContext.from_defaults(persist_dir="doc_emb")
index = load_index_from_storage(storage_context)

# 构建查询引擎
query_engine = index.as_query_engine(similarity_top_k=5)

# 输出查询引擎中所有的prompt类型
prompts_dict = query_engine.get_prompts()
print(list(prompts_dict.keys()))

# 更新查询引擎中的prompt template
query_engine.update_prompts(
    {"response_synthesizer:text_qa_template": qa_prompt_tmpl,
     "response_synthesizer:refine_template": refine_prompt_tmpl}
)

# 查询获得答案
response = query_engine.query("不耐疲劳，口燥、咽干可能是哪些证候？")
print(response)

# 输出formatted_prompt
event_pairs = llama_debug.get_llm_inputs_outputs()
print(event_pairs[0][1].payload["formatted_prompt"])
```