deepseek实验三：基于deepseek-chat和Rag-fusion构建问答系统

准备环境

下载向量数据库chromadb

```
pip install chromadb
```

下载langchain相关库

```
pip install langchain
```

```
pip install langchain_openai
```

```
pip install langchain_community
```

```
pip install langchain_core
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain.prompts import ChatPromptTemplate
from operator import itemgetter
from langchain.embeddings import HuggingFaceBgeEmbeddings
from langchain_openai import ChatOpenAI
from langchain_community.document_loaders import WebBaseLoader, DirectoryLoader
from langchain.load import dumps, loads
from langchain_core.documents import Document
import os
import json
```

初始化BGE模型加载路径

```
bge_model_path = "./bge-small-zh-v1.5/"
```

定义RRF算法函数

```
#定义RRF算法函数
def reciprocal_rank_fusion(results: list[list], k=60):
    """ Reciprocal_rank_fusion that takes multiple lists of ranked documents
        and an optional parameter k used in the RRF formula """

    # Initialize a dictionary to hold fused scores for each unique document
    fused_scores = {}

    # Iterate through each list of ranked documents
    for docs in results:
        # Iterate through each document in the list, with its rank (position in
        for rank, doc in enumerate(docs):
            # Convert the document to a string format to use as a key (assumes d
            doc_str = dumps(doc)
            # If the document is not yet in the fused_scores dictionary, add it
            if doc_str not in fused_scores:
                fused_scores[doc_str] = 0
            # Retrieve the current score of the document, if any
            previous_score = fused_scores[doc_str]
            # Update the score of the document using the RRF formula: 1 / (rank
            fused_scores[doc_str] += 1 / (rank + k)

    # Sort the documents based on their fused scores in descending order to get
    reranked_results = [
        (loads(doc), score)
        for doc, score in sorted(fused_scores.items(), key=lambda x: x[1], rever
    ]

    # Return the reranked results as a list of tuples, each containing the docum
    return reranked_results
```

加载bge embedding模型

```
# 0.加载bge embedding模型
```

```
bge_embeddings = HuggingFaceBgeEmbeddings(model_name=bge_model_path)
```

处理文档

加载文档

```
#  1.加载文档
loader = DirectoryLoader('./data', glob="**/*.txt")
docs = loader.load()
```

创建文档分割器，并分割文档

```
# 2.创建文档分割器，并分割文档
text_splitter = RecursiveCharacterTextSplitter(chunk_size=512,chunk_overlap=0)
splits = text_splitter.split_documents(docs)
```

创建向量数据库

```
# 3.创建向量数据库
vectorstore = Chroma.from_documents(documents=splits,embedding=bge_embeddings)
```

创建检索器

```
# 4.创建检索器
retriever = vectorstore.as_retriever()
```

RAG-fusion处理过程

第一步，创建一个生成多重查询的chain, 该chain会根据用户的query生成4个多角度的query, 这些多角度的query是对用户原始query的补充。
请注意，该chain不执行最后的生成步骤(不会将top k的检索结果喂给LLM)

```
template1 = """You are a helpful assistant that generates multiple search querie
Generate multiple search queries related to: {question} \n
Output (4 queries):"""
prompt_rag_fusion = ChatPromptTemplate.from_template(template1)
```

```
generate_queries = (
    prompt_rag_fusion
    | ChatOpenAI(model="deepseek-chat", api_key="sk-f70da689860944fca980b2ee34f3
    | StrOutputParser()
    | (lambda x: x.split("\n"))
)
```

第二步，我们现在可以将它们放在一起并定义完整的用于检索的chain。该chain由generate_queries， retriever.map()， reciprocal_rank_fusion三部分组成，其中generate_queries会生成4个多角度的query, retriever.map()的作用是根据generate_queries的结果映射出4个retriever(可以理解为同时复制出4个retriever)与中generate_queries会生成4个query对应，并为每个query检索出来的一组相关文档集(默认为4个相关文档)，那么4个query总共可以生成16个相关文档。这16个相关文档集最后会经过RRF算法从新排序后输出最终的4个相关度最高的文档。

```
retrieval_chain_rag_fusion = generate_queries | retriever.map() | reciprocal_rar
template2 = """Answer the following question based on this context:
{context}
Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template2)
```

回答生成

```
final_rag_chain = (
   {"context": retrieval_chain_rag_fusion,
    "question": itemgetter("question")}
   | prompt
   | ChatOpenAI(model="deepseek-chat", api_key="sk-f70da689860944fca980b2ee34f3b
   | StrOutputParser()
)

question='恐龙是怎么灭绝的'
final_rag_chain.invoke({"question":question})
```

```
'根据提供的文档内容，恐龙的灭绝主要是由于约0.65亿年前一颗直径约10公里的小行星撞击地球所引发
```