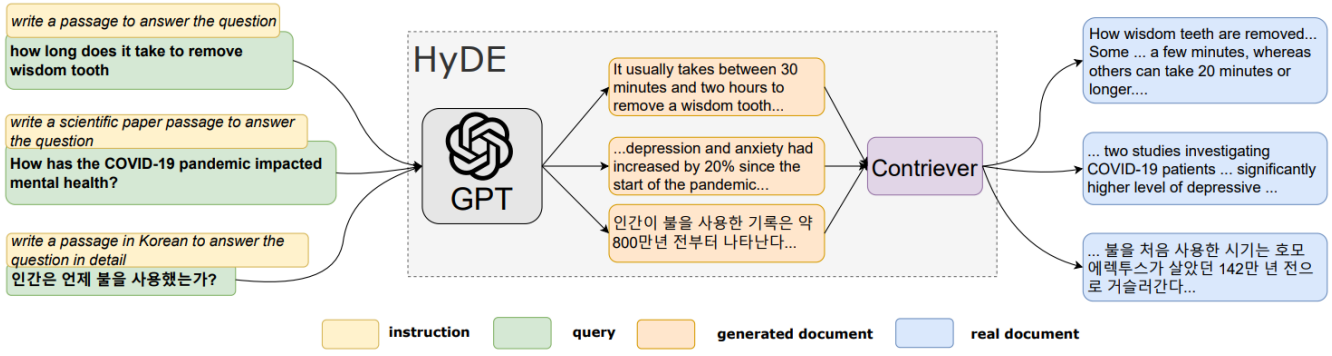
🦜Langchain 0.2 RAG 快速入门 #6 HyDE

Hypothetical Document Embeddings

HyDE不是基于原始问题生成查询，而是专注于为给定的查询生成假设性文档。生成这类假设性文档背后的直觉是，它们的嵌入向量可以用来在语料库嵌入空间中识别一个邻域，基于向量相似性检索出相似的真正文文档。在这种情况下，RAG将能够根据假设性文档检索到更多相关的文档，从而准确回答用户查询。



生成假设文档

```python
from langchain.prompts import ChatPromptTemplate

hyde_prompt = ChatPromptTemplate.from_template(
    """
    请撰写一段科学论文的内容来回答以下问题：
    问题：{question}\n
    段落：
    """
)

generate_doc_chain = (
        {'question': RunnablePassthrough()}
        | hyde_prompt
        | glm4_air_model
        | StrOutputParser()
)
question = "工作满意度指数有哪些？"
res = generate_doc_chain.invoke(question)
```

```
19
20   retrieval_chain = generate_doc_chain | retriever
21   retireved_docs = retrieval_chain.invoke({"question": question})
22   retireved_docs
```

RAG chain

```
1 template = """Answer the following question based on the provided context:
2
3     {context}
4
5     Question: {question}
6     """
7
8     prompt = ChatPromptTemplate.from_template(template)
9
10    final_rag_chain = (
11            prompt
12            | glm4_air_model
13            | StrOutputParser()
14    )
15
16    res = final_rag_chain.invoke({"context": retireved_docs, "question":
  question})
17    print(res)
```

完整代码

```
1 import os
2 from uuid import uuid4
3
4
5 # 从我本地的一个文件，导入我设置的ZHIPU_AK
6 from ai_docs_cfg import ZHIPU_AK,MY_LANGCHAIN_API_KEY
7
8 from langchain_community.document_loaders import PyPDFLoader, DirectoryLoader
9 from langchain.text_splitter import RecursiveCharacterTextSplitter
10
11 from langchain_community.vectorstores import Chroma
12 from langchain_core.output_parsers import StrOutputParser
13 from langchain_core.runnables import RunnablePassthrough
14 from langchain_openai import ChatOpenAI
15 from langchain import hub
16 from langchain.load import loads, dumps
```

```python
from typing import List

# Langsmith 配置，不用可注掉
unique_id = uuid4().hex[0:8]
os.environ["LANGCHAIN_PROJECT"] = f" RAG - {unique_id}"
# os.environ["LANGCHAIN_TRACING_V2"] = 'true'
os.environ["LANGCHAIN_API_KEY"] = MY_LANGCHAIN_API_KEY

if __name__ == '__main__':
# 初始化 模型
glm4_air_model = ChatOpenAI(
    model_name="glm-4-0520",
    openai_api_base="https://open.bigmodel.cn/api/paas/v4",
    openai_api_key=ZHIPU_AK,
)

from langchain_community.embeddings.huggingface import HuggingFaceEmbeddings

# 模型所在路径
embedding_model_name = 'D:\LLM\\bce_modesl\\bce-embedding-base_v1'

# 使用 GPU
embedding_model_kwargs = {'device': 'cuda:0'}

# 使用 CPU
# embedding_model_kwargs = {'device': 'cpu'}

# 配置
embedding_encode_kwargs = {'batch_size': 32, 'normalize_embeddings': True, }

# 初始化
embed_model = HuggingFaceEmbeddings(
    model_name=embedding_model_name,
    model_kwargs=embedding_model_kwargs,
    encode_kwargs=embedding_encode_kwargs
)

data_path = 'D:\\LLM\\my_projects\\new_langchain_test\\AI_docs\\data'
vdb_path =
'D:\\LLM\\my_projects\\new_langchain_test\\AI_docs\\data\\vectorstore'

loader = DirectoryLoader(data_path, glob="*.pdf", loader_cls=PyPDFLoader)
documents = loader.load()

# Split text into chunks
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=20)
text_chunks = text_splitter.split_documents(documents)
```

```
63
64 # 创建向量数据库
65 vectorstore = Chroma(persist_directory=vdb_path,
   embedding_function=embed_model)
66
67 # 索引数据
68 #res = vectorstore.add_documents(text_chunks)
69
70 # 检索器
71 retriever = vectorstore.as_retriever(search_kwargs={'k': 5})
72
73 from langchain.prompts import ChatPromptTemplate
74
75 hyde_prompt = ChatPromptTemplate.from_template(
76     """
77     请撰写一段科学论文的内容来回答以下问题:
78     问题: {question}\n
79     段落:
80     """
81 )
82
83 generate_doc_chain = (
84         {'question': RunnablePassthrough()}
85         | hyde_prompt
86         | glm4_air_model
87         | StrOutputParser()
88 )
89 question = "工作满意度指数有哪些? "
90 res = generate_doc_chain.invoke(question)
91
92 retrieval_chain = generate_doc_chain | retriever
93 retireved_docs = retrieval_chain.invoke({"question": question})
94 retireved_docs
95
96 template = """Answer the following question based on the provided context:
97
98 {context}
99
100 Question: {question}
101 """
102
103 prompt = ChatPromptTemplate.from_template(template)
104
105 final_rag_chain = (
106         prompt
107         | glm4_air_model
108         | StrOutputParser()
109 )
```

```
110
111 res = final_rag_chain.invoke({"context": retireved_docs, "question":
    question})
112 print(res)
113
```