retrieve_note_with_paragraph

July 19, 2024

[1]:
```python
import os
from dotenv import load_dotenv
from langchain_community.chat_models import ChatOllama
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import FAISS
from pathlib import Path
from langchain.docstore.document import Document

from langchain_core.output_parsers import StrOutputParser
from langchain_teddynote import logging

from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_core.prompts import PromptTemplate
```

[2]:
```python
load_dotenv()
os.environ["LANGCHAIN_TRACING_V2"] = os.getenv("LANGCHAIN_TRACING_V2")
os.environ["LANGCHAIN_API_KEY"] = os.getenv("LANGCHAIN_API_KEY")
logging.langsmith("jeniffer_RAG")
```

```
LangSmith      .
[   ]
jeniffer_RAG
```

[4]:
```python
notebook_dir = Path(os.getcwd())
project_dir = notebook_dir.parent
data_dir = project_dir / "data"
file_path = data_dir / "paragrah.txt"

print(file_path)
```

```
C:\Users\cywell\Documents\dev\ai\jeniffer\data\paragrah.txt
```

[7]:
```python
docs = []

with open(file_path, "r", encoding="utf-8") as file:
    for line in file:
        if line.strip():  #
            docs.append(Document(page_content=line.strip()))
```

```
print("         ")
```

[8]:
```
embeddings = OllamaEmbeddings(model="EEVE:latest")
```

[10]:
```
vectorstore = FAISS.from_documents(documents=docs, embedding=embeddings)
print("        ")
```

[11]:
```
vectorstore.save_local("fasis_paragraph")
print("        ")
```

[12]:
```
retriever = vectorstore.as_retriever()
```

[13]:
```
prompt = PromptTemplate.from_template(
    """You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Answer in Korean.

#Question:
{question}
#Context:
{context}

#Answer:"""
)
```

[14]:
```
llm = ChatOllama(model="EEVE:latest")
```

[15]:
```
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

[17]:
```
print("  ")
question = "             "
response = chain.invoke(question)
print(response)
```

```
---------------------------------------------------------------------------
```

```
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[17], line 3
      1 print("  ")
      2 question = "              "
----> 3 response = chain.invoke(question)
      4 print(response)

File ~\Documents\dev\ai\jeniffer\.
 ↪venv\Lib\site-packages\langchain_core\runnables\base.py:2822, in␣
 ↪RunnableSequence.invoke(self, input, config, **kwargs)
   2818 config = patch_config(
   2819     config, callbacks=run_manager.get_child(f"seq:step:{i+1}")
   2820 )
   2821 if i == 0:
-> 2822     input = step.invoke(input, config, **kwargs)
   2823 else:
   2824     input = step.invoke(input, config)

File ~\Documents\dev\ai\jeniffer\.
 ↪venv\Lib\site-packages\langchain_core\runnables\base.py:3511, in␣
 ↪RunnableParallel.invoke(self, input, config)
   3498     with get_executor_for_config(config) as executor:
   3499         futures = [
   3500             executor.submit(
   3501                 step.invoke,
   (…)
   3509             for key, step in steps.items()
   3510         ]
-> 3511         output =␣
 ↪{key: future.result() for key, future in zip(steps, futures)}
   3512 # finish the root run
   3513 except BaseException as e:

File ~\Documents\dev\ai\jeniffer\.
 ↪venv\Lib\site-packages\langchain_core\runnables\base.py:3511, in <dictcomp>(.0)
   3498     with get_executor_for_config(config) as executor:
   3499         futures = [
   3500             executor.submit(
   3501                 step.invoke,
   (…)
   3509             for key, step in steps.items()
   3510         ]
-> 3511         output = {key: future.result() for key, future in zip(steps,␣
 ↪futures)}
   3512 # finish the root run
   3513 except BaseException as e:
```

```
File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.254
 ↪0_x64__qbz5n2kfra8p0\Lib\concurrent\futures\_base.py:451, in Future.
 ↪result(self, timeout)
    448 elif self._state == FINISHED:
    449     return self.__get_result()
--> 451 self._condition.wait(timeout)
    453 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:
    454     raise CancelledError()

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.254
 ↪0_x64__qbz5n2kfra8p0\Lib\threading.py:327, in Condition.wait(self, timeout)
    325 try:    # restore state no matter what (e.g., KeyboardInterrupt)
    326     if timeout is None:
--> 327         waiter.acquire()
    328         gotit = True
    329     else:

KeyboardInterrupt:
```

```
[ ]:
```