RAG

Let’s look at adding in a retrieval step to a prompt and LLM, which adds up to a “retrieval-augmented generation” chain

```
%pip install --upgrade --quiet  langchain
langchain-openai faiss-cpu tiktoken
```

```
from operator import itemgetter

from langchain_community.vectorstores import
FAISS
from langchain_core.output_parsers import
StrOutputParser
from langchain_core.prompts import
ChatPromptTemplate
from langchain_core.runnables import
RunnableLambda, RunnablePassthrough
from langchain_openai import ChatOpenAI,
OpenAIEmbeddings
```

```
vectorstore = FAISS.from_texts(
    ["harrison worked at kensho"],
embedding=OpenAIEmbeddings()
)
retriever = vectorstore.as_retriever()

template = """Answer the question based only
on the following context:
{context}

Question: {question}
"""
prompt =
ChatPromptTemplate.from_template(template)

model = ChatOpenAI()
```

```
chain = (
    {"context": retriever, "question":
RunnablePassthrough()}
    | prompt
    | model
    | StrOutputParser()
)
```

```
chain.invoke("where did harrison work?")
```

```
'Harrison worked at Kensho.'
```

```
template = """Answer the question based only
on the following context:
{context}

Question: {question}

Answer in the following language: {language}
"""
prompt =
ChatPromptTemplate.from_template(template)

chain = (
    {
        "context": itemgetter("question") |
retriever,
        "question": itemgetter("question"),
        "language": itemgetter("language"),
    }
    | prompt
    | model
    | StrOutputParser()
)
```

```
chain.invoke({"question": "where did harrison
work", "language": "italian"})
```

```
'Harrison ha lavorato a Kensho.'
```

Conversational Retrieval Chain

We can easily add in conversation history. This primarily means adding in chat_message_history

```
from langchain.schema import format_document
from langchain_core.messages import
AIMessage, HumanMessage, get_buffer_string
from langchain_core.runnables import
RunnableParallel
```

```
from langchain.prompts.prompt import
PromptTemplate

_template = """Given the following
conversation and a follow up question,
rephrase the follow up question to be a
standalone question, in its original
language.

Chat History:
{chat_history}
Follow Up Input: {question}
Standalone question:"""
CONDENSE_QUESTION_PROMPT =
PromptTemplate.from_template(_template)
```

```
template = """Answer the question based only
on the following context:
{context}

Question: {question}
"""
ANSWER_PROMPT =
ChatPromptTemplate.from_template(template)
```

```
DEFAULT_DOCUMENT_PROMPT =
PromptTemplate.from_template(template="
{page_content}")


def _combine_documents(
    docs,
document_prompt=DEFAULT_DOCUMENT_PROMPT,
document_separator="\n\n"
):
    doc_strings = [format_document(doc,
document_prompt) for doc in docs]
    return
document_separator.join(doc_strings)
```

```
_inputs = RunnableParallel(

standalone_question=RunnablePassthrough.assign(
        chat_history=lambda x:
```

```
get_buffer_string(x["chat_history"])
    )
    | CONDENSE_QUESTION_PROMPT
    | ChatOpenAI(temperature=0)
    | StrOutputParser(),
)
_context = {
    "context":
itemgetter("standalone_question") | retriever |
_combine_documents,
    "question": lambda x:
x["standalone_question"],
}
conversational_qa_chain = _inputs | _context |
ANSWER_PROMPT | ChatOpenAI()
```

```
conversational_qa_chain.invoke(
    {
        "question": "where did harrison
work?",
        "chat_history": [],
    }
)
```

```
AIMessage(content='Harrison was employed at
Kensho.')
```

```
conversational_qa_chain.invoke(
    {
        "question": "where did he work?",
        "chat_history": [
            HumanMessage(content="Who wrote
this notebook?"),
            AIMessage(content="Harrison"),
        ],
    }
)
```

```
AIMessage(content='Harrison worked at
Kensho.')
```

With Memory and returning source documents

This shows how to use memory with the above. For memory, we need to manage that outside at the memory. For returning the retrieved documents, we just need to pass them through all the way.

```
from operator import itemgetter

from langchain.memory import
ConversationBufferMemory
```

```
memory = ConversationBufferMemory(
    return_messages=True,
output_key="answer", input_key="question"
)
```

```
# First we add a step to load memory
# This adds a "memory" key to the input object
loaded_memory = RunnablePassthrough.assign(

chat_history=RunnableLambda(memory.load_memory_
| itemgetter("history"),
)
# Now we calculate the standalone question
standalone_question = {
    "standalone_question": {
        "question": lambda x: x["question"],
        "chat_history": lambda x:
get_buffer_string(x["chat_history"]),
    }
    | CONDENSE_QUESTION_PROMPT
    | ChatOpenAI(temperature=0)
    | StrOutputParser(),
}
# Now we retrieve the documents
retrieved_documents = {
    "docs": itemgetter("standalone_question") |
retriever,
    "question": lambda x: x["standalone_questio
}
```

```
# Now we construct the inputs for the final pro
final_inputs = {
    "context": lambda x: _combine_documents(x["
    "question": itemgetter("question"),
}
# And finally, we do the part that returns the
answer = {
    "answer": final_inputs | ANSWER_PROMPT |
ChatOpenAI(),
    "docs": itemgetter("docs"),
}
# And now we put it all together!
final_chain = loaded_memory | standalone_questi
retrieved_documents | answer
```

```
inputs = {"question": "where did harrison
work?"}
result = final_chain.invoke(inputs)
result
```

```
{'answer': AIMessage(content='Harrison was
employed at Kensho.'),
 'docs': [Document(page_content='harrison
worked at kensho')]}
```

```
# Note that the memory does not save
automatically
```

```
# This will be improved in the future
# For now you need to save it yourself
memory.save_context(inputs, {"answer":
result["answer"].content})
```

```
memory.load_memory_variables({})
```

```
{'history': [HumanMessage(content='where did
harrison work?'),
  AIMessage(content='Harrison was employed at
Kensho.')]}
```

```
inputs = {"question": "but where did he
really work?"}
result = final_chain.invoke(inputs)
result
```

```
{'answer': AIMessage(content='Harrison
actually worked at Kensho.'),
 'docs': [Document(page_content='harrison
worked at kensho')]}
```