Manipulating inputs & output

RunnableParallel can be useful for manipulating the output of one Runnable to match the input format of the next Runnable in a sequence.

Here the input to prompt is expected to be a map with keys “context” and “question”. The user input is just the question. So we need to get the context using our retriever and passthrough the user input under the “question” key.

```
%pip install --upgrade --quiet  langchain
langchain-openai
```

```
from langchain_community.vectorstores import
FAISS
from langchain_core.output_parsers import
StrOutputParser
from langchain_core.prompts import
ChatPromptTemplate
from langchain_core.runnables import
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()

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

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

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

TIP

Note that when composing a RunnableParallel with another Runnable we don’t even need to wrap our dictionary in the RunnableParallel class — the type conversion is handled for us. In the context of a chain, these are equivalent:

```
{"context": retriever, "question":
RunnablePassthrough()}
```

```
RunnableParallel({"context": retriever,
"question": RunnablePassthrough()})
```

```
RunnableParallel(context=retriever,
question=RunnablePassthrough())
```

Using itemgetter as shorthand

Note that you can use Python’s `itemgetter` as shorthand to extract data from the map when combining with `RunnableParallel`. You can find more information about itemgetter in the Python Documentation.

In the example below, we use itemgetter to extract specific keys from the map:

```
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 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}

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.'
```

Parallelize steps

RunnableParallel (aka. RunnableMap) makes it easy to execute multiple Runnables in parallel, and to return the output of these Runnables as a map.

```
from langchain_core.prompts import
ChatPromptTemplate
from langchain_core.runnables import
RunnableParallel
from langchain_openai import ChatOpenAI

model = ChatOpenAI()
joke_chain =
ChatPromptTemplate.from_template("tell me a
joke about {topic}") | model
poem_chain = (
    ChatPromptTemplate.from_template("write a
2-line poem about {topic}") | model
)

map_chain = RunnableParallel(joke=joke_chain,
poem=poem_chain)

map_chain.invoke({"topic": "bear"})
```

```
{'joke': AIMessage(content="Why don't bears
wear shoes?\n\nBecause they have bear
feet!"),
 'poem': AIMessage(content="In the wild's
embrace, bear roams free,\nStrength and
grace, a majestic decree.")}
```

Parallelism

RunnableParallel are also useful for running independent processes in parallel, since each Runnable in the map is executed in parallel. For example, we can see our earlier `joke_chain`, `poem_chain` and `map_chain` all have about the same runtime, even though `map_chain` executes both of the other two.

```
%%timeit

joke_chain.invoke({"topic": "bear"})
```

```
958 ms ± 402 ms per loop (mean ± std. dev. of
7 runs, 1 loop each)
```

```
%%timeit

poem_chain.invoke({"topic": "bear"})
```

```
1.22 s ± 508 ms per loop (mean ± std. dev. of
7 runs, 1 loop each)
```

```
%%timeit

map_chain.invoke({"topic": "bear"})
```

```
1.15 s ± 119 ms per loop (mean ± std. dev. of
7 runs, 1 loop each)
```