Prompt + LLM

The most common and valuable composition is taking:

`PromptTemplate` / `ChatPromptTemplate` -> `LLM` / `ChatModel` -> `OutputParser`

Almost any other chains you build will use this building block.

PromptTemplate + LLM

The simplest composition is just combining a prompt and model to create a chain that takes user input, adds it to a prompt, passes it to a model, and returns the raw model output.

Note, you can mix and match PromptTemplate/ChatPromptTemplates and LLMs/ChatModels as you like here.

%pip install –upgrade –quiet langchain langchain-openai

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

prompt =
ChatPromptTemplate.from_template("tell me a
joke about {foo}")
model = ChatOpenAI()
chain = prompt | model
```

```
chain.invoke({"foo": "bears"})
```

```
AIMessage(content="Why don't bears wear
shoes?\n\nBecause they have bear feet!",
additional_kwargs={}, example=False)
```

Often times we want to attach kwargs that'll be passed to each model call. Here are a few examples of that:

Attaching Stop Sequences

```
chain = prompt | model.bind(stop=["\n"])
```

```
chain.invoke({"foo": "bears"})
```

```
AIMessage(content='Why did the bear never
wear shoes?', additional_kwargs={},
example=False)
```

Attaching Function Call information

```
functions = [
    {
        "name": "joke",
        "description": "A joke",
        "parameters": {
            "type": "object",
            "properties": {
                "setup": {"type": "string",
"description": "The setup for the joke"},
                "punchline": {
                    "type": "string",
                    "description": "The
punchline for the joke",
                },
            },
            "required": ["setup",
"punchline"],
        },
    }
]
chain = prompt | model.bind(function_call=
{"name": "joke"}, functions=functions)
```

```
chain.invoke({"foo": "bears"}, config={})
```

```
AIMessage(content='', additional_kwargs=
{'function_call': {'name': 'joke',
'arguments': '{\n  "setup": "Why don\'t bears
wear shoes?",\n  "punchline": "Because they
have bear feet!"\n}'}}, example=False)
```

PromptTemplate + LLM + OutputParser

We can also add in an output parser to easily transform the raw LLM/ChatModel output into a more workable format

```
from langchain_core.output_parsers import
StrOutputParser

chain = prompt | model | StrOutputParser()
```

Notice that this now returns a string - a much more workable format for downstream tasks

```
chain.invoke({"foo": "bears"})
```

```
"Why don't bears wear shoes?\n\nBecause they
have bear feet!"
```

Functions Output Parser

When you specify the function to return, you may just want to parse that directly

```
from
langchain.output_parsers.openai_functions
import JsonOutputFunctionsParser

chain = (
    prompt
    | model.bind(function_call={"name":
"joke"}, functions=functions)
    | JsonOutputFunctionsParser()
)
```

```
chain.invoke({"foo": "bears"})
```

```
{'setup': "Why don't bears like fast food?",
 'punchline': "Because they can't catch it!"}
```

```
from langchain.output_parsers.openai_functions
import JsonKeyOutputFunctionsParser

chain = (
    prompt
    | model.bind(function_call={"name":
"joke"}, functions=functions)
    |
JsonKeyOutputFunctionsParser(key_name="setup")
)
```

```
chain.invoke({"foo": "bears"})
```

```
"Why don't bears wear shoes?"
```

Simplifying input

To make invocation even simpler, we can add a `RunnableParallel` to take care of creating the prompt input dict for us:

```
from langchain_core.runnables import
RunnableParallel, RunnablePassthrough
```

```
map_ =
RunnableParallel(foo=RunnablePassthrough())
chain = (
    map_
    | prompt
    | model.bind(function_call={"name":
"joke"}, functions=functions)
    |
JsonKeyOutputFunctionsParser(key_name="setup")
)
```

```
chain.invoke("bears")
```

```
"Why don't bears wear shoes?"
```

Since we're composing our map with another Runnable, we can even use some syntactic sugar and just use a dict:

```
chain = (
    {"foo": RunnablePassthrough()}
    | prompt
    | model.bind(function_call={"name":
"joke"}, functions=functions)
    |
JsonKeyOutputFunctionsParser(key_name="setup")
)
```

```
chain.invoke("bears")
```

```
"Why don't bears like fast food?"
```