Chains

Using an LLM in isolation is fine for simple applications, but more complex applications require chaining LLMs - either with each other or with other components.

LangChain provides the **Chain** interface for such "chained" applications. We define a Chain very generically as a sequence of calls to components, which can include other chains. The base interface is simple:

```
class Chain(BaseModel, ABC):
    """Base interface that all chains should implement."""

    memory: BaseMemory
    callbacks: Callbacks

    def __call__(
        self,
        inputs: Any,
        return_only_outputs: bool = False,
        callbacks: Callbacks = None,
    ) -> Dict[str, Any]:
        ...
```

This idea of composing components together in a chain is simple but powerful. It drastically simplifies and makes more modular the implementation of complex applications, which in turn makes it much easier to debug, maintain, and improve your applications.

For more specifics check out:

- How-to for walkthroughs of different chain features
- Foundational to get acquainted with core building block chains
- Document to learn how to incorporate documents into chains
- Popular chains for the most common use cases
- Additional to see some of the more advanced chains and integrations that you can use out of the box

Why do we need chains?

Chains allow us to combine multiple components together to create a single, coherent application. For example, we can create a chain that takes user input, formats it with a PromptTemplate, and then passes the formatted response to an LLM. We can build more complex chains by combining multiple chains together, or by combining chains with other components.

Get started

Using `LLMChain`

The `LLMChain` is most basic building block chain. It takes in a prompt template, formats it with the user input and returns the response from an LLM.

To use the `LLMChain`, first create a prompt template.

```
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

llm = OpenAI(temperature=0.9)
prompt = PromptTemplate(
```

```
    input_variables=["product"],
    template="What is a good name for a company that makes {product}?",
)
```

We can now create a very simple chain that will take user input, format the prompt with it, and then send it to the LLM.

```
from langchain.chains import LLMChain
chain = LLMChain(llm=llm, prompt=prompt)

# Run the chain only specifying the input variable.
print(chain.run("colorful socks"))
```

```
    Colorful Toes Co.
```

If there are multiple variables, you can input them all at once using a dictionary.

```
prompt = PromptTemplate(
    input_variables=["company", "product"],
    template="What is a good name for {company} that makes {product}?",
)
chain = LLMChain(llm=llm, prompt=prompt)
print(chain.run({
    'company': "ABC Startup",
    'product': "colorful socks"
    }))
```

```
    Socktopia Colourful Creations.
```

You can use a chat model in an `LLMChain` as well:

```
from langchain.chat_models import ChatOpenAI
from langchain.prompts.chat import (
    ChatPromptTemplate,
    HumanMessagePromptTemplate,
)
human_message_prompt = HumanMessagePromptTemplate(
        prompt=PromptTemplate(
            template="What is a good name for a company that makes {product}?",
            input_variables=["product"],
        )
    )
chat_prompt_template = ChatPromptTemplate.from_messages([human_message_prompt])
chat = ChatOpenAI(temperature=0.9)
chain = LLMChain(llm=chat, prompt=chat_prompt_template)
print(chain.run("colorful socks"))
```

```
    Rainbow Socks Co.
```