a framework for developing applications powered by language models.

Are context-aware: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)

Reason: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

“链”（chain）是指将多个组件按照一定的顺序连接起来以完成特定任务的流程。这些组件可以是不同类型的工具或服务，例如数据处理、信息检索等。“代理”（agent）则是指能够执行一系列任务或操作的独立实体，它们通过使用链中定义的流程来处理输入信息，并产生输出。简而言之，链是任务执行的流程，而代理是执行这些流程的实体。

The LangChain libraries themselves are made up of several different packages.

* [**langchain-core**](https://github.com/langchain-ai/langchain/blob/master/libs/core): Base abstractions and LangChain Expression Language.
* [**langchain-community**](https://github.com/langchain-ai/langchain/blob/master/libs/community): Third party integrations.
* [**langchain**](https://github.com/langchain-ai/langchain/blob/master/libs/langchain): Chains, agents, and retrieval strategies that make up an application's cognitive architecture

Components fall into the following modules:

📃 Model I/O:

This includes prompt management, prompt optimization, a generic interface for all LLMs, and common utilities for working with LLMs.

📚 Retrieval:

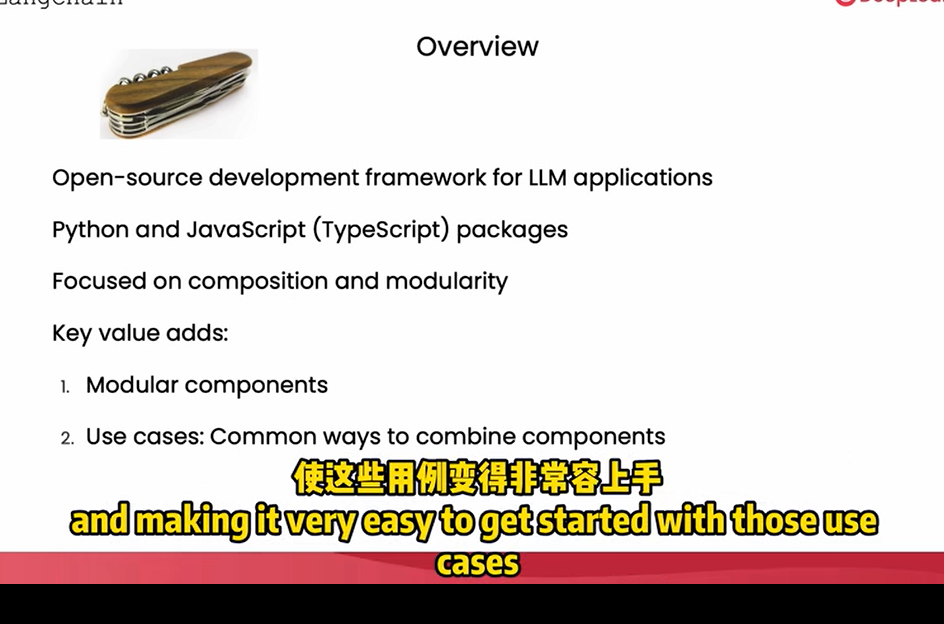
Data Augmented Generation involves specific types of chains that first interact with an external data source to fetch data for use in the generation step. Examples include summarization of long pieces of text and question/answering over specific data sources.

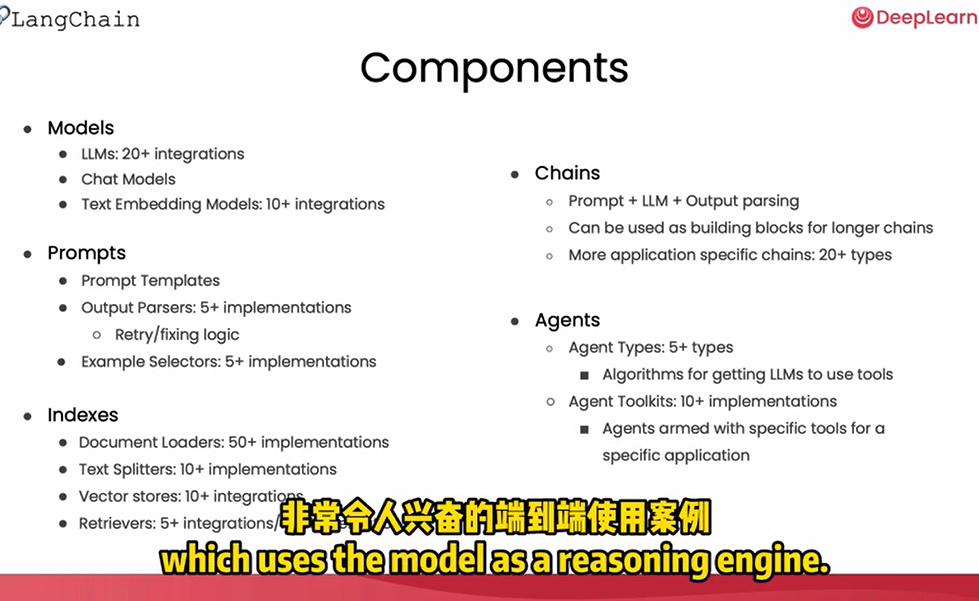
🤖 Agents:

Agents involve an LLM making decisions about which Actions to take, taking that Action, seeing an Observation, and repeating that until done. LangChain provides a standard interface for agents, a selection of agents to choose from, and examples of end-to-end agents.

吴恩达课程，一个简单的实践（langchain）

应用多次提示大模型，导致冗余





Models

Prompts

Parsers- output -structure

将粗鲁语言转换为礼貌

实现1：构建prompt- 产生响应

实现2：template 建立提示模板（更为方便地重复使用设计好的提示）（langchain有一些模板）

Thought action observation

输出解析器 langchain.output\_parsers，返回的就真的不是字符串了，之后使用会很方便

以下是一个实例

review\_template\_2 = """\

For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? \

Answer True if yes, False if not or unknown.

delivery\_days: How many days did it take for the product\

to arrive? If this information is not found, output -1.

price\_value: Extract any sentences about the value or price,\

and output them as a comma separated Python list.

text: {text}

{format\_instructions}

"""

prompt = ChatPromptTemplate.from\_template(template=review\_template\_2)

from langchain.output\_parsers import ResponseSchema

from langchain.output\_parsers import StructuredOutputParser

gift\_schema = ResponseSchema(name="gift",

description="Was the item purchased\

as a gift for someone else? \

Answer True if yes,\

False if not or unknown.")

delivery\_days\_schema = ResponseSchema(name="delivery\_days",

description="How many days\

did it take for the product\

to arrive? If this \

information is not found,\

output -1.")

price\_value\_schema = ResponseSchema(name="price\_value",

description="Extract any\

sentences about the value or \

price, and output them as a \

comma separated Python list.")

response\_schemas = [gift\_schema,

delivery\_days\_schema,

price\_value\_schema]

output\_parser = StructuredOutputParser.from\_response\_schemas(response\_schemas)

format\_instructions = output\_parser.get\_format\_instructions()

print(format\_instructions)

messages = prompt.format\_messages(text=customer\_review, format\_instructions=format\_instructions)

print(messages[0].content)

response = chat(messages)

print(response.content)

output\_dict = output\_parser.parse(response.content)

output\_dict