



# HOW TO BUILD A CHATBOT

Session 2 -  
Introduction to  
LangChain

# SESSION 2

## AGENDA



**1**

Langchain Ecosystem

**2**

Introduction of LangChain

**3**

Key Components of LangChain

# LANGCHAIN ECOSYSTEM

## LangChain:

- Core framework for building LLM-powered apps with modular components.

## LangGraph:

- Visualizes and organizes connections between chains and agents.

## Integrations:

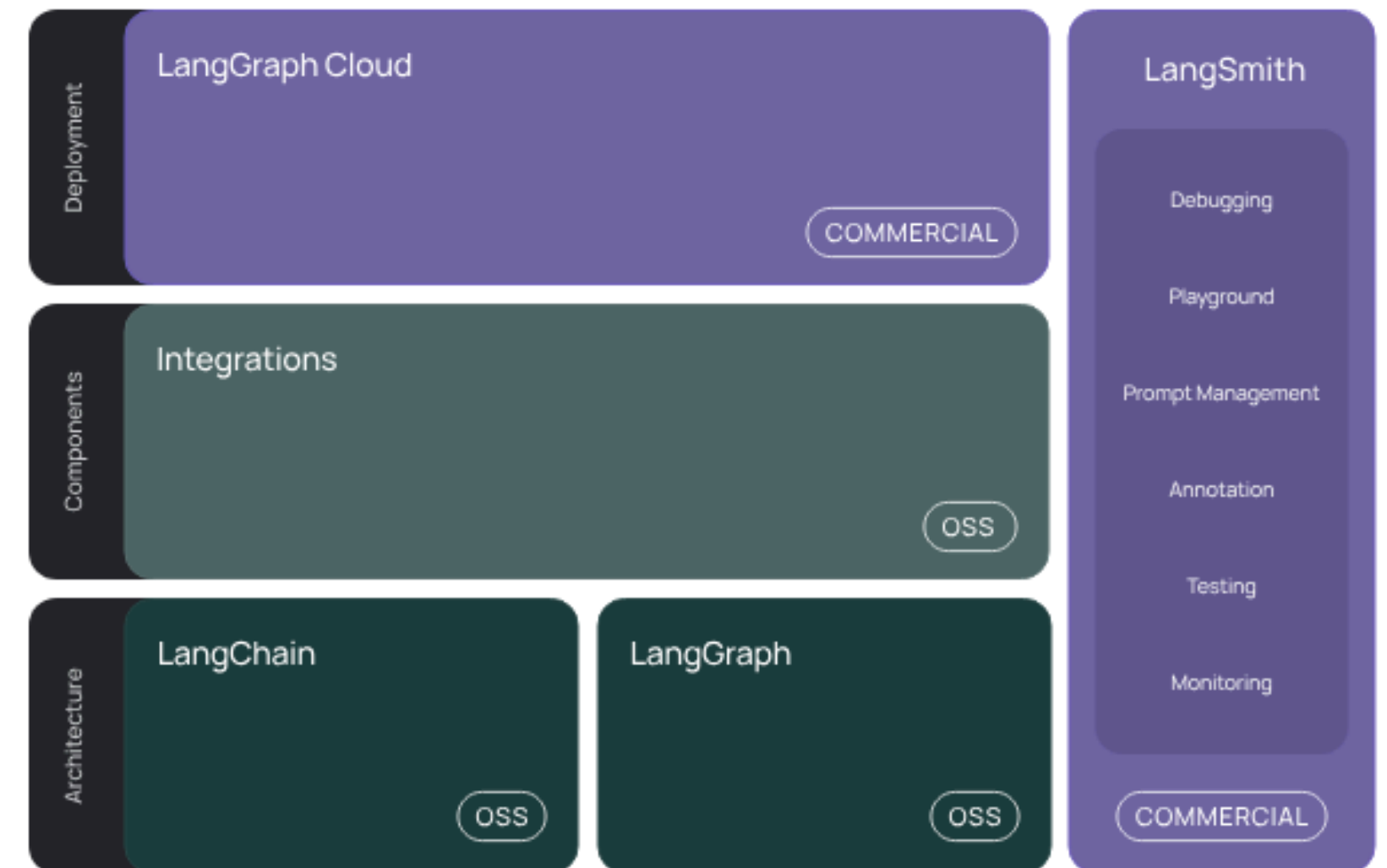
- Connects LLMs with external services and data sources.

## LangSmith:

- Tool for testing, debugging, and optimizing LLM applications.

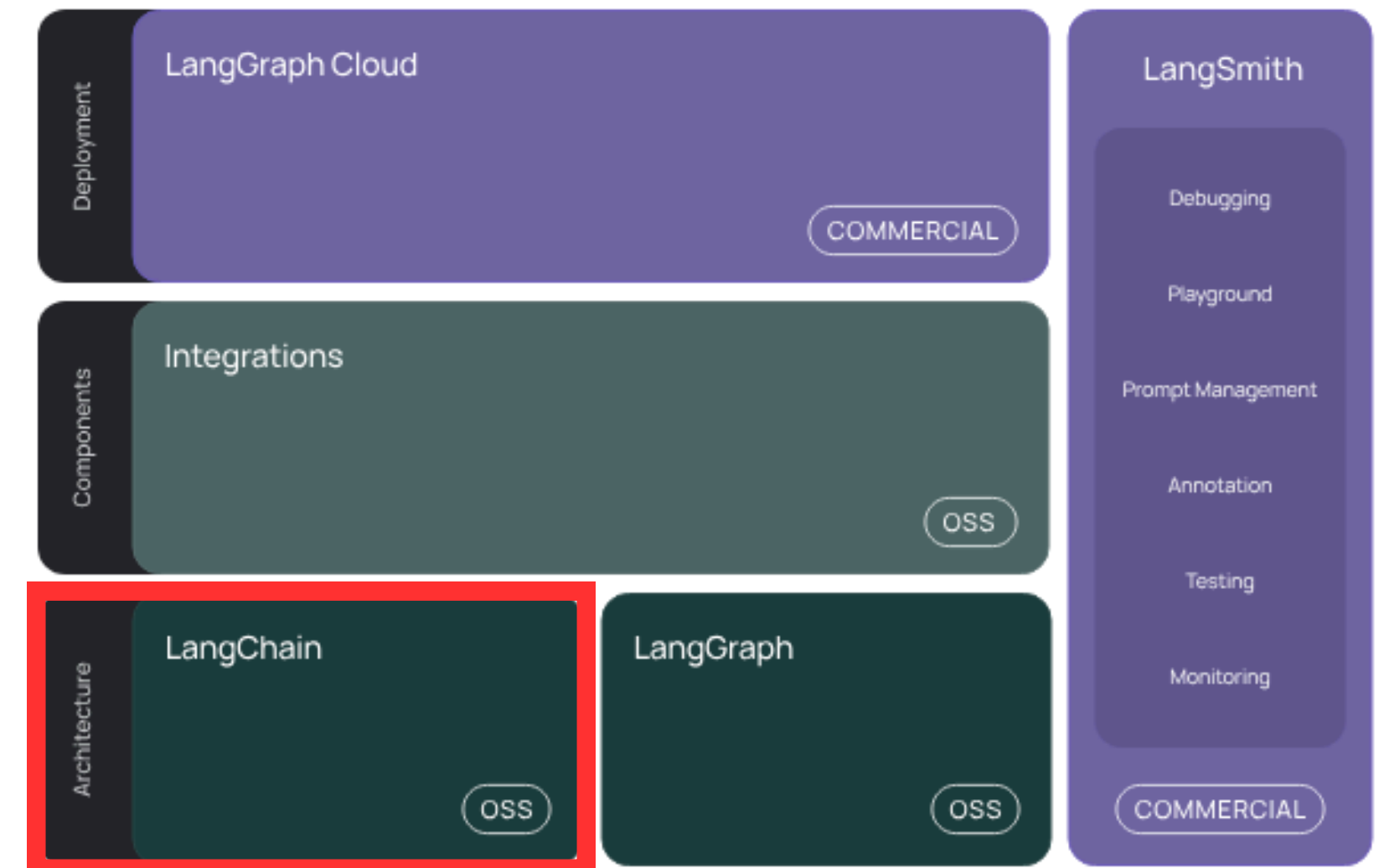
## LangGraph Cloud:

- Cloud platform for managing and deploying LLM workflows.



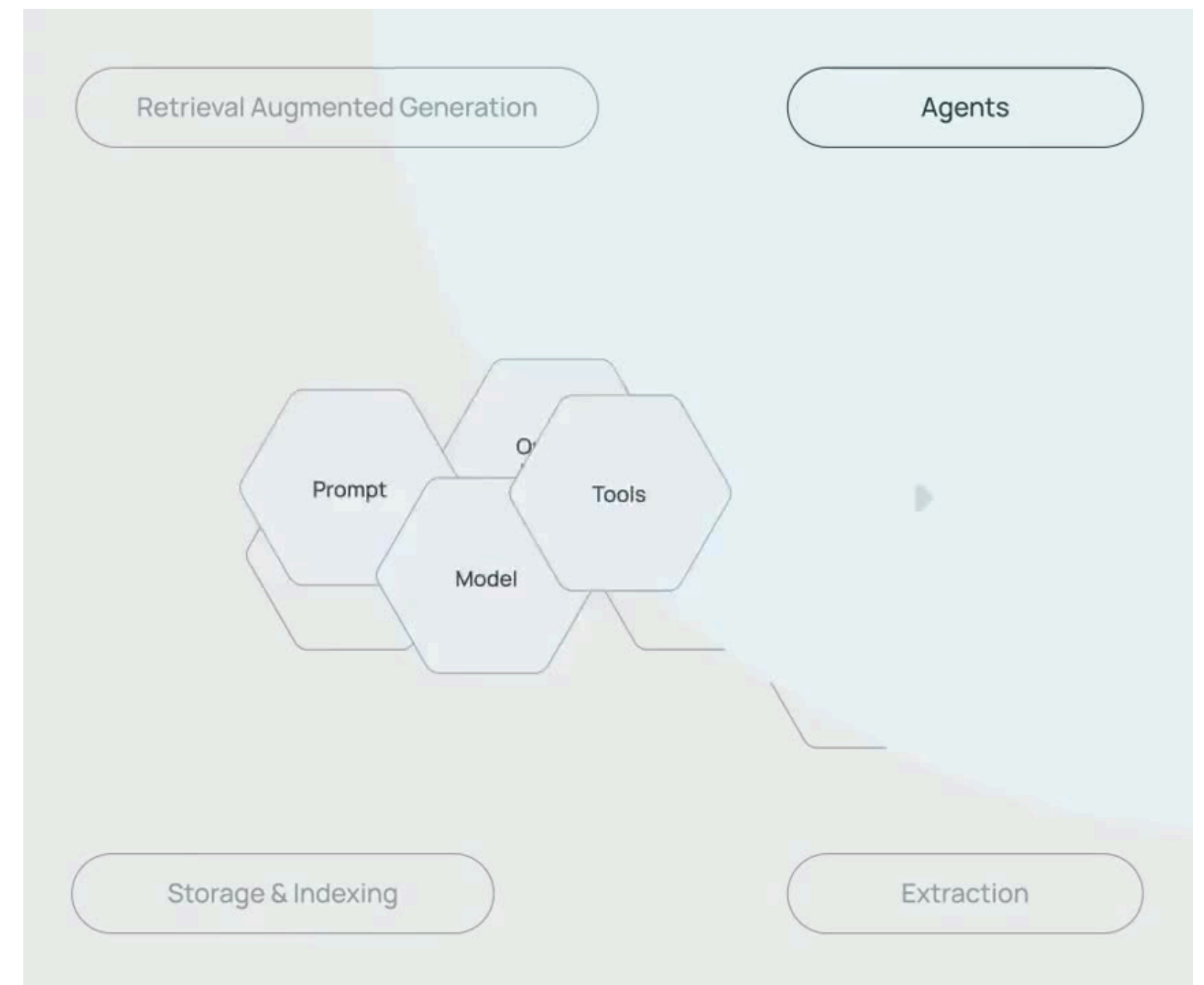
# INTRODUCTION LANGCHAIN

- Open-source Python framework, 2022
- Unified interface to interact with LLMs from different providers
- Easy integration of external data sources and services.
- Building LLM based workflows and applications
- Typical Use Cases:
  - Chatbots / Virtual Assistants (e.g. support)
  - Intelligent search engines (e.g. natural language search)
  - ...



# KEY COMPONENTS OF LANGCHAIN

- LLMs / Chat Models
- Prompt Templates
- Chains
- Agents
- Memory
- Tools / Toolkits



# KEY COMPONENTS OF LANGCHAIN

## Integration of different LLM Provider

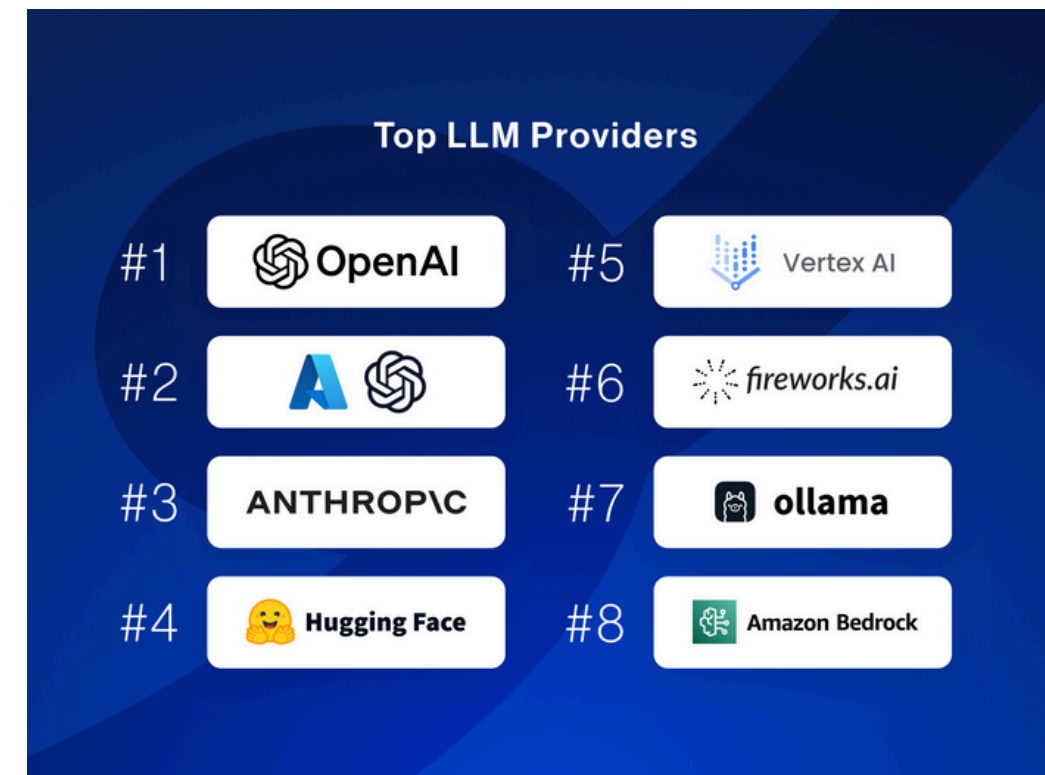
### LLMs

Language model for various text-based tasks (e.g., translation, summarization)



### Chat Models

Language model optimized for dialogue and maintaining context in conversations.



[2]

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o",
    temperature=0,
    max_tokens=None,
    timeout=None,
    max_retries=2,
    # api_key="...", # if you prefer to pass api key in directly instead of using env vars
    # base_url="...",
    # organization="...",
    # other params...
)
```



# KEY COMPONENTS OF LANGCHAIN

## Prompt Templates - Structure LLM Input

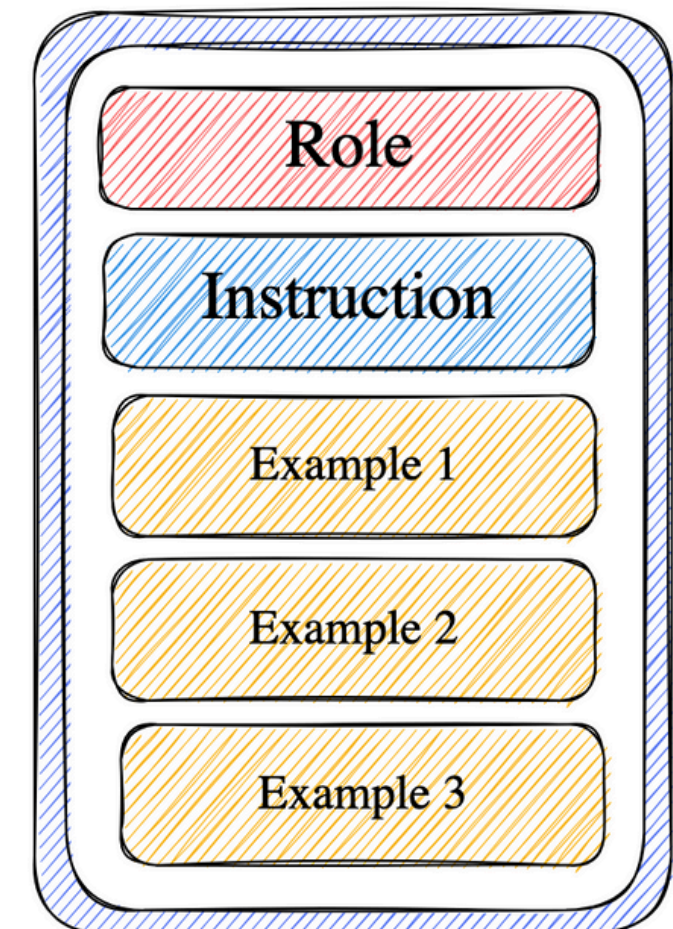
```
# In addition to Human/AI/Tool/Function messages,
# you can initialize the template with a MessagesPlaceholder
# either using the class directly or with the shorthand tuple syntax:

template = ChatPromptTemplate([
    ("system", "You are a helpful AI bot."),
    # Means the template will receive an optional list of messages under
    # the "conversation" key
    ("placeholder", "{conversation}")
    # Equivalently:
    # MessagesPlaceholder(variable_name="conversation", optional=True)
])

prompt_value = template.invoke(
    {
        "conversation": [
            ("human", "Hi!"),
            ("ai", "How can I assist you today?"),
            ("human", "Can you make me an ice cream sundae?"),
            ("ai", "No.")
        ]
    }
)

# Output:
# ChatPromptValue(
#   messages=[
#     SystemMessage(content='You are a helpful AI bot.'),
#     HumanMessage(content='Hi!'),
#     AIMessage(content='How can I assist you today?'),
#     HumanMessage(content='Can you make me an ice cream sundae?'),
#     AIMessage(content='No.'),
#   ]
#)
```

A Combined Techniques Prompt

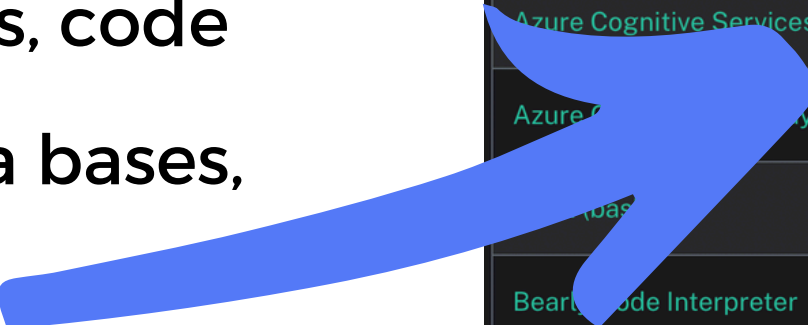


[3]

# KEY COMPONENTS OF LANGCHAIN

## Tools - Enhance LLM Functionality

- Tools provide modular interfaces to integrate external services like databases and APIs.
- Tools can be combined with models to extend their capability.
- LangChain offers tools like search engines, code interpreter, office365, web browsing, data bases, wheater APIs, stock APIs, ...



Name	Description
<a href="#">AINetwork Toolkit</a>	AI Network is a layer 1 blockchain designed to accommodate large-scal...
<a href="#">Alpha Vantage</a>	Alpha Vantage Alpha Vantage provides realtime and historical financia...
<a href="#">Amadeus Toolkit</a>	This notebook walks you through connecting LangChain to the Amadeus t...
<a href="#">ArXiv</a>	This notebook goes over how to use the arxiv tool with an agent.
<a href="#">AskNews</a>	AskNews infuses any LLM with the latest global news (or historical ne...
<a href="#">AWS Lambda</a>	Amazon AWS Lambda is a serverless computing service provided by Amazo...
<a href="#">Azure AI Services Toolkit</a>	This toolkit is used to interact with the Azure AI Services API to ac...
<a href="#">Azure Cognitive Services Toolkit</a>	This toolkit is used to interact with the Azure Cognitive Services AP...
<a href="#">Azure Container Apps dynamic sessions</a>	Azure Container Apps dynamic sessions provides a secure and scalable ...
<a href="#">Bearly Code Interpreter</a>	Giving agents access to the shell is powerful (though risky outside a...
<a href="#">Bing Search</a>	Bearly Code Interpreter allows for remote execution of code. This mak...
<a href="#">Brave Search</a>	Bing Search is an Azure service and enables safe, ad-free, location-a...
<a href="#">Brave Search</a>	This notebook goes over how to use the Brave Search tool.



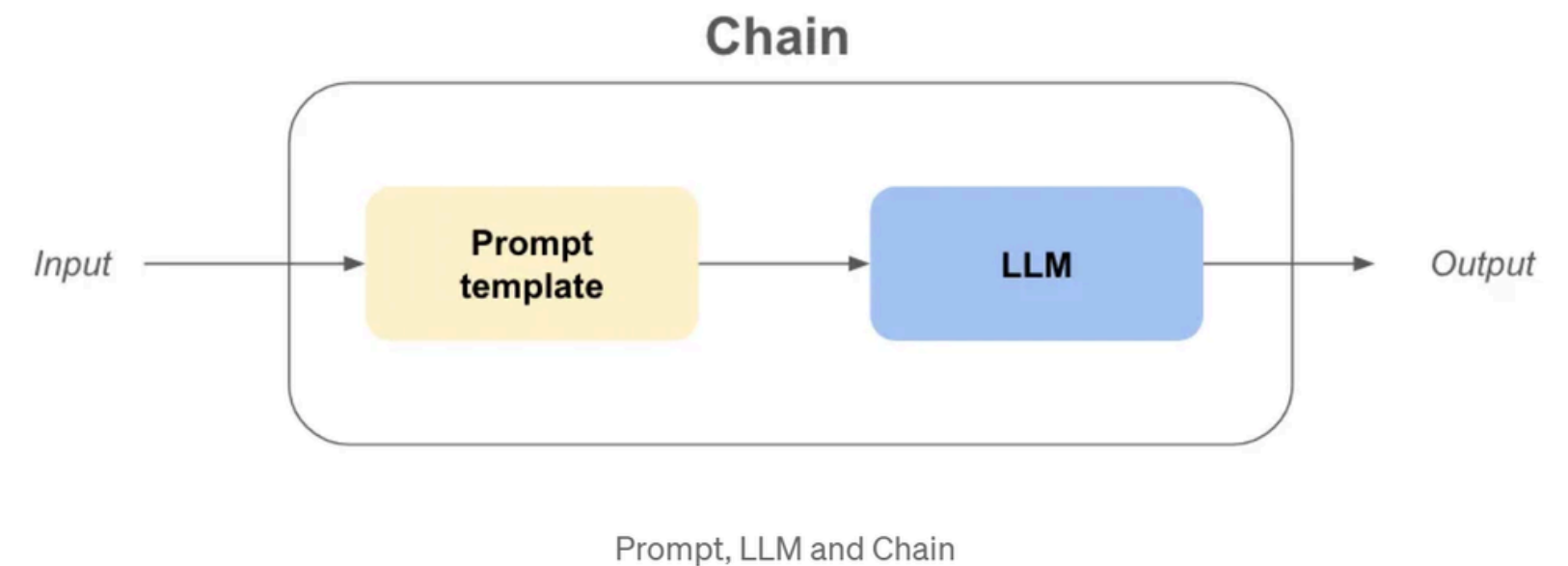
# KEY COMPONENTS OF LANGCHAIN

## Chains - Streamline LLM Pipelines

- Compose modular logic components into reusable pipelines.
- Chains are sequences of components with calls to components
- Steps can be added, removed, and swapped.
- Use pre-built chains or implement custom chains

### Example:

A simple chain might involve passing a formatted prompt to an LLM.



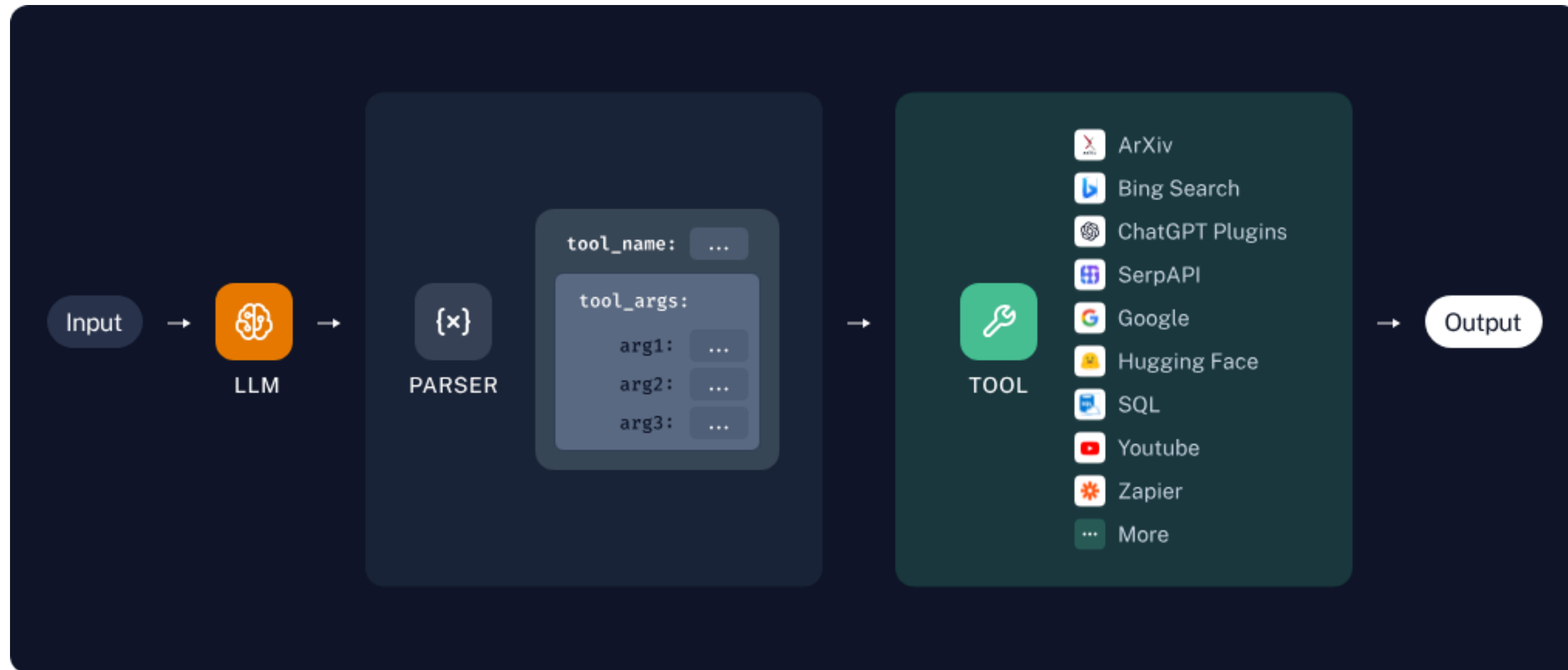
```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_template("tell me a joke about {topic}")

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

# KEY COMPONENTS OF LANGCHAIN

## Chains - Tool Integration



[4]

-> Chain: fixed sequence of actions, LLM parses tool parameters to call a fixed tool.

# KEY COMPONENTS OF LANGCHAIN

## Agents - Tool Integration



-> Agent: LLM itself decide which tools to use and in what order (loop). [4]

# KEY COMPONENTS OF LANGCHAIN

## 1. Implement Tools

```
from langchain_core.tools import tool

@tool
def multiply(first_int: int, second_int: int) -> int:
    """Multiply two integers together."""
    return first_int * second_int

@tool
def add(first_int: int, second_int: int) -> int:
    "Add two integers."
    return first_int + second_int

@tool
def exponentiate(base: int, exponent: int) -> int:
    "Exponentiate the base to the exponent power."
    return base**exponent

tools = [multiply, add, exponentiate]
```

## 2. Create and invoke Agent

```
# Construct the tool calling agent
agent = create_tool_calling_agent(llm, tools, prompt)
```

```
# Create an agent executor by passing in the agent and tools
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

```
agent_executor.invoke(
    {
        "input": "Take 3 to the fifth power and multiply that by the sum of twelve and three, then"
    }
)
```

## Agents - Tool Integration

## 3. Agent Output

```
[1m> Entering new AgentExecutor chain... [0m
[32;1m [1;3m
Invoking: `exponentiate` with `{'base': 3, 'exponent': 5}`
responded: [{'text': "Okay, let's break this down step-by-step:", 'type': 'text'}, {'id': 'toolu_01Cj

[0m [38;5;200m [1;3m243 [0m [32;1m [1;3m
Invoking: `add` with `{'first_int': 12, 'second_int': 3}`
responded: [{'text': '3 to the 5th power is 243.', 'type': 'text'}, {'id': 'toolu_01EKqn4E5w3Zj7bQ8s8

[0m [33;1m [1;3m15 [0m [32;1m [1;3m
Invoking: `multiply` with `{'first_int': 243, 'second_int': 15}`
responded: [{'text': '12 + 3 = 15', 'type': 'text'}, {'id': 'toolu_017VZJgZBYbwMo2KGD6o6hsQ', 'input'

[0m [36;1m [1;3m3645 [0m [32;1m [1;3m
Invoking: `multiply` with `{'first_int': 3645, 'second_int': 3645}`
responded: [{'text': '243 * 15 = 3645', 'type': 'text'}, {'id': 'toolu_01RtFCcQgbVGya3NVDgTYKTa', 'in

[0m [36;1m [1;3m13286025 [0m [32;1m [1;3mSo 3645 squared is 13,286,025.

Therefore, the final result of taking 3 to the 5th power (243), multiplying by 12 + 3 (15), and then

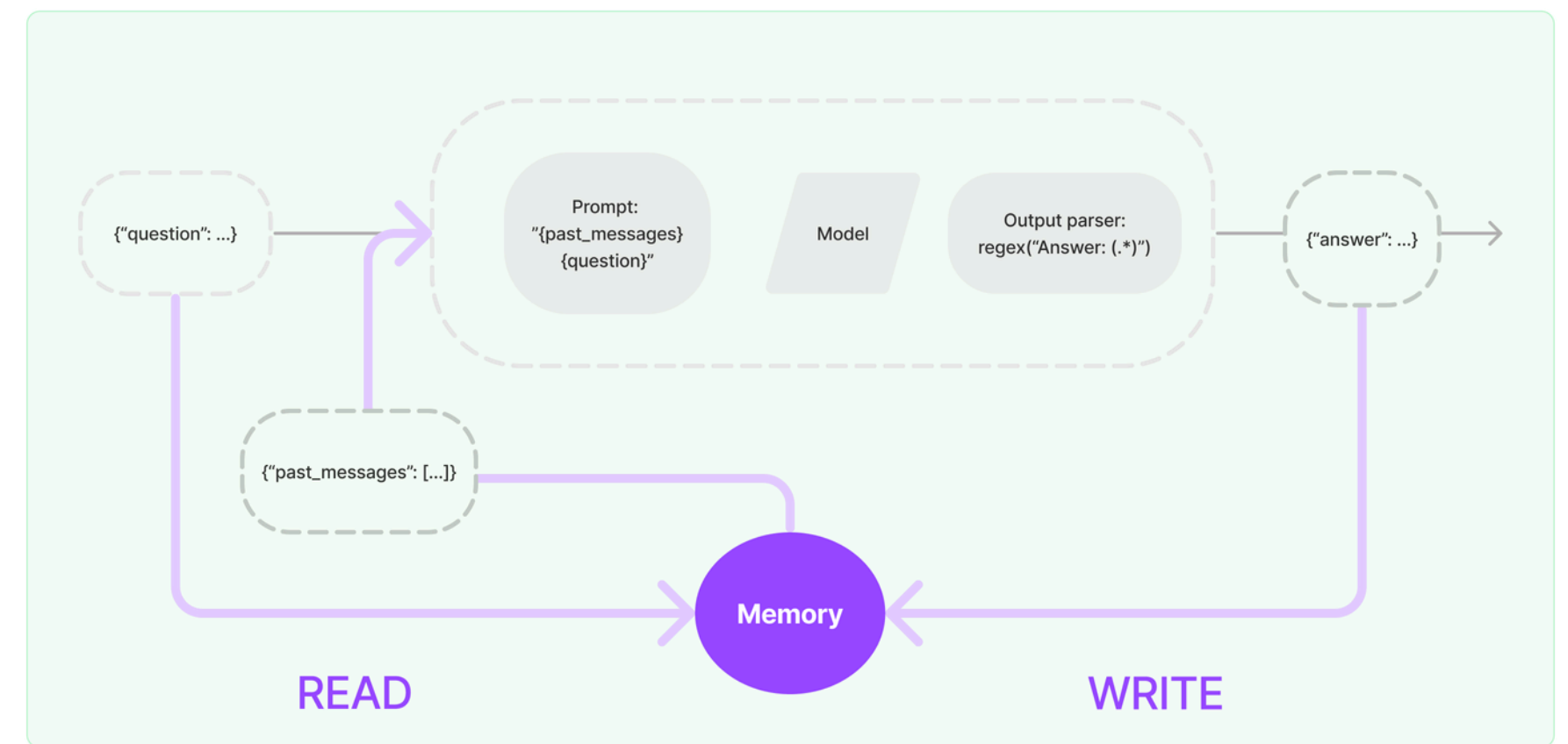
[1m> Finished chain. [0m
```



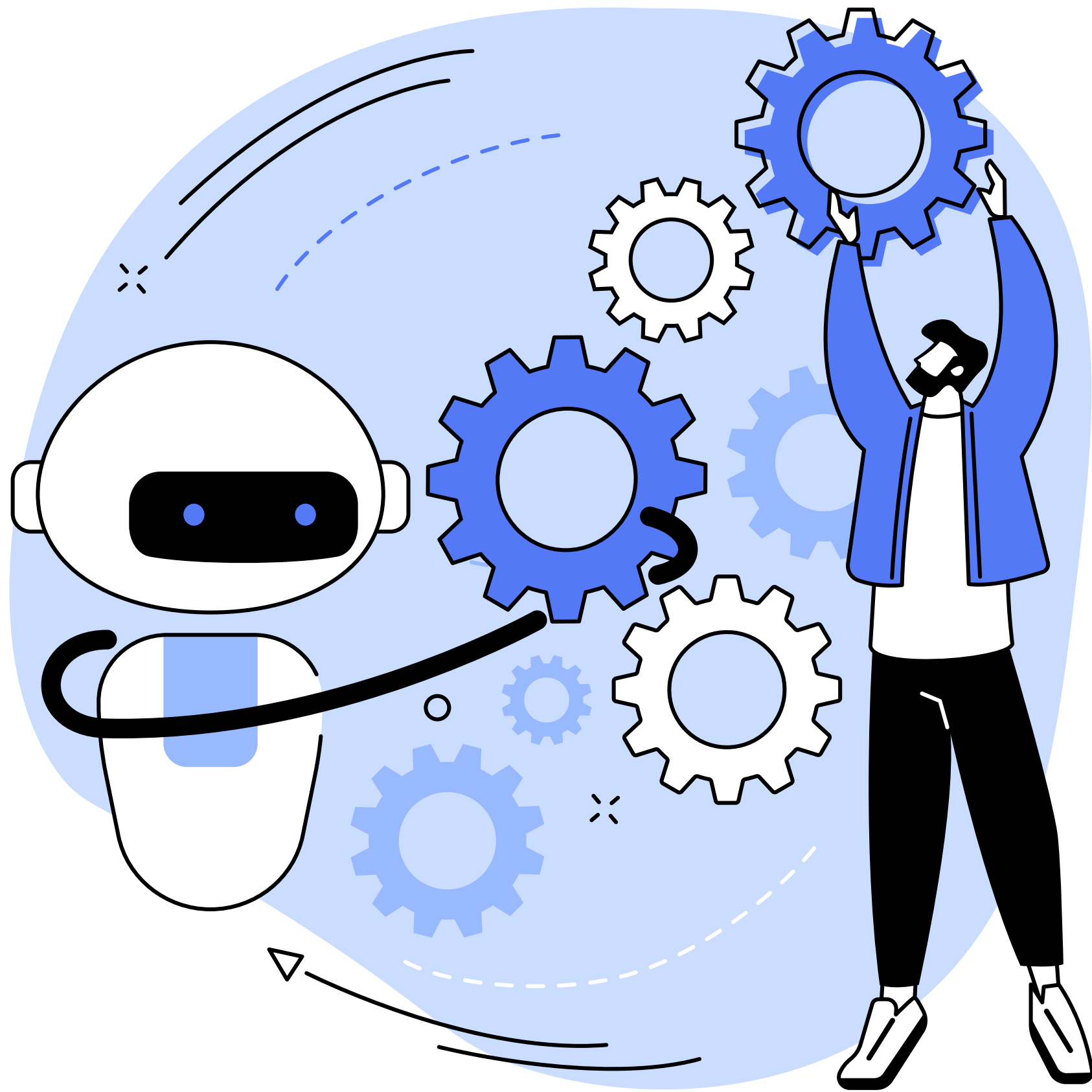
# KEY COMPONENTS OF LANGCHAIN

## Memory - Enhancing Conversational Applications

- Ability to persist information between interactions.
- Memory improves LLM responses by providing relevant historical information.
- Memory reduces redundant LLM calls, saving costs and improving performance.
- Storage in-memory or in databases







**IT'S YOUR TURN**

## Sources:

[1]: LangChain: [https://cdn.prod.website-files.com/65b8cd72835ceeacd4449a53/6695b116b0b60c78fd4ef462\\_15.07.24%20-Updated%20stack%20diagram%20-%20lightfor%20website-3.webp](https://cdn.prod.website-files.com/65b8cd72835ceeacd4449a53/6695b116b0b60c78fd4ef462_15.07.24%20-Updated%20stack%20diagram%20-%20lightfor%20website-3.webp)

[2]: [https://www.google.com/url?sa=i&url=https%3A%2F%2Ftwitter.com%2Fkalyan\\_kpl%2Fstatus%2F1757039040996802775&psig=AOvVaw1J64GLqe82hfc9y0Gccy5s&ust=1726389391094000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCMj6hOWDwogDFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Ftwitter.com%2Fkalyan_kpl%2Fstatus%2F1757039040996802775&psig=AOvVaw1J64GLqe82hfc9y0Gccy5s&ust=1726389391094000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCMj6hOWDwogDFQAAAAAdAAAAABAE)

[3]: <https://learnprompting.org/de/docs/basics/prompting>

[4]: <https://python.langchain.com/>