14.11.23

OPENCAMPUS.sh

Practical Engineering with LLMs

Introduction to LangChain

TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

QUIZ ON INTRODUCTION TO LANGCHAIN

https://forms.office.com/r/9WLqzWrxfL

TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

HOMEWORK PRESENTATION

Tasks:

1. Create a simple sketch showing the flow of the application, the inputs and outputs of the components of the application, and briefly explain in 1-2 sentences what the application does.

2. Analyze the code and answer the following questions with a brief explanation: …

3. Make the following changes to the code: …

4. Get creative and create your own SequentialChain with at least 3 chains.

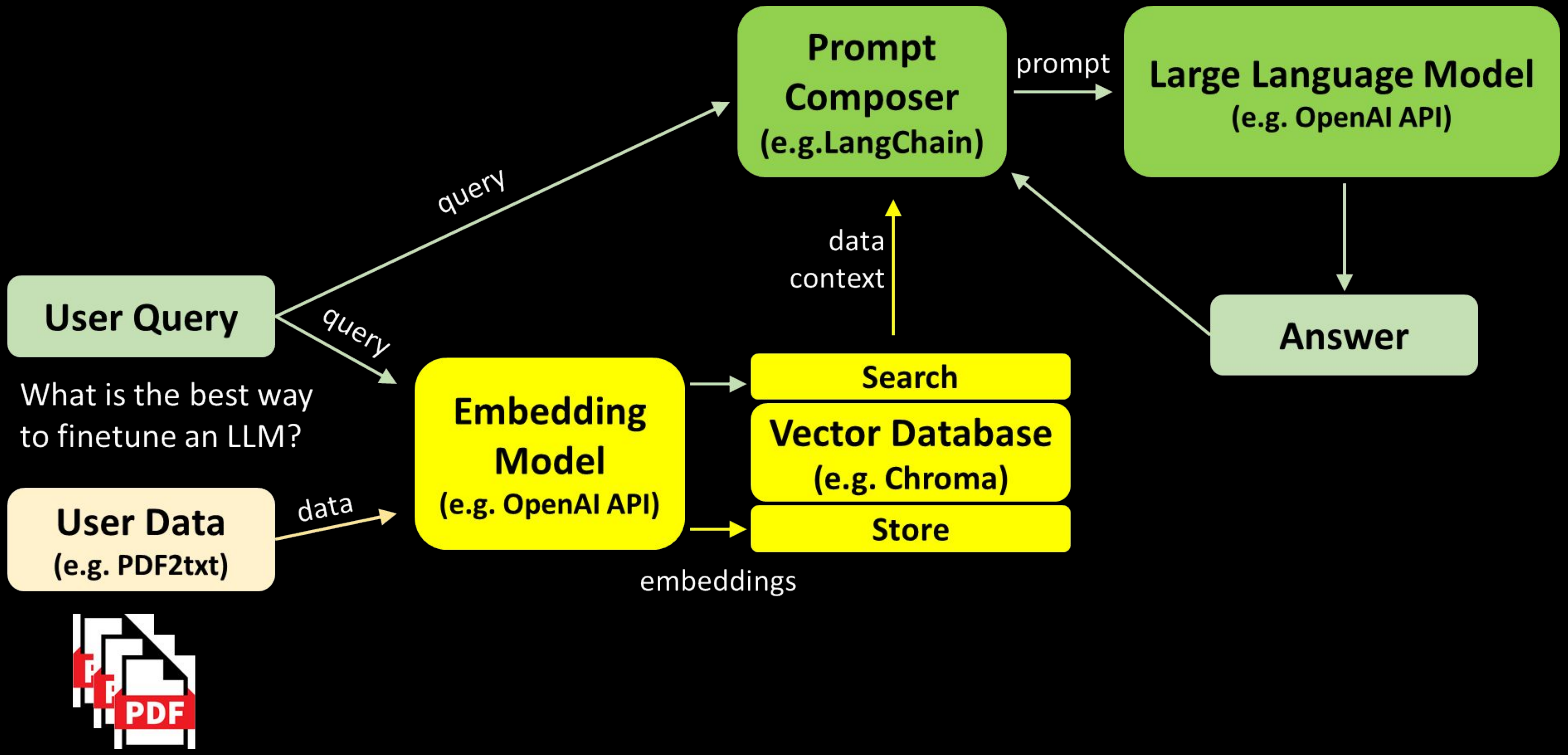
TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

QUICK RECAP- LangChain

What can LangChain do for you?

What LangChain can do for you ...


Prompt Composer (e.g.LangChain)
prompt
Large Language Model (e.g. OpenAI API)
query
data context
User Query
query
Answer
What is the best way to finetune an LLM?
Embedding Model (e.g. OpenAI API)
Search
Vector Database (e.g. Chroma)
Store
User Data (e.g. PDF2txt)
data
embeddings
PDF

https://www.youtube.com/watch?v=dXxQ0LR-3Hg

What LangChain can do for you …

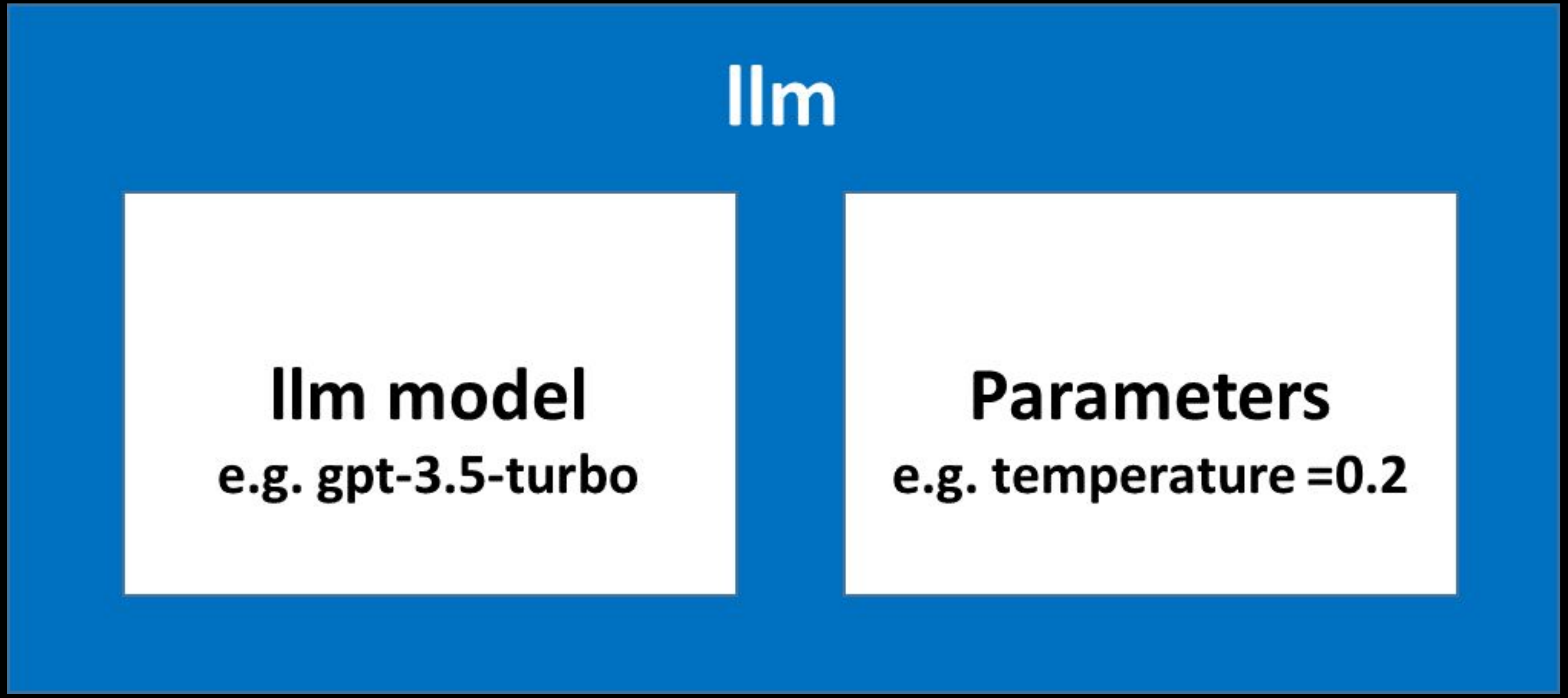
- Model, Prompts, and Parsers
- Manage memory
- Chains
- (Q&A over Documents)
- (Evaluation)
- Agents

QUICK RECAP- LangChain

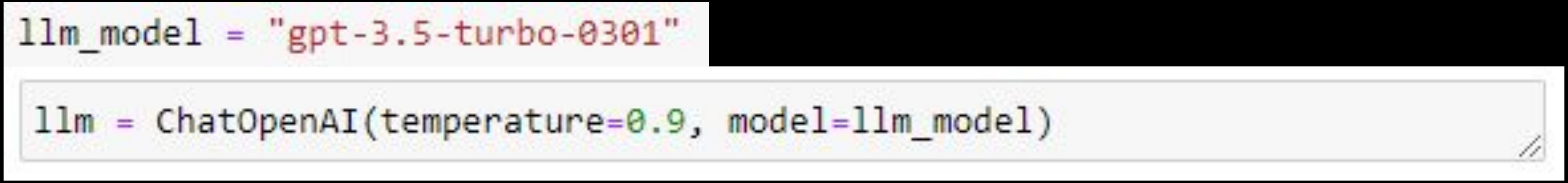
Model, Prompts & Parsers

QUICK RECAP- LangChain

Model, Prompts & Parsers



+ API-key as environment

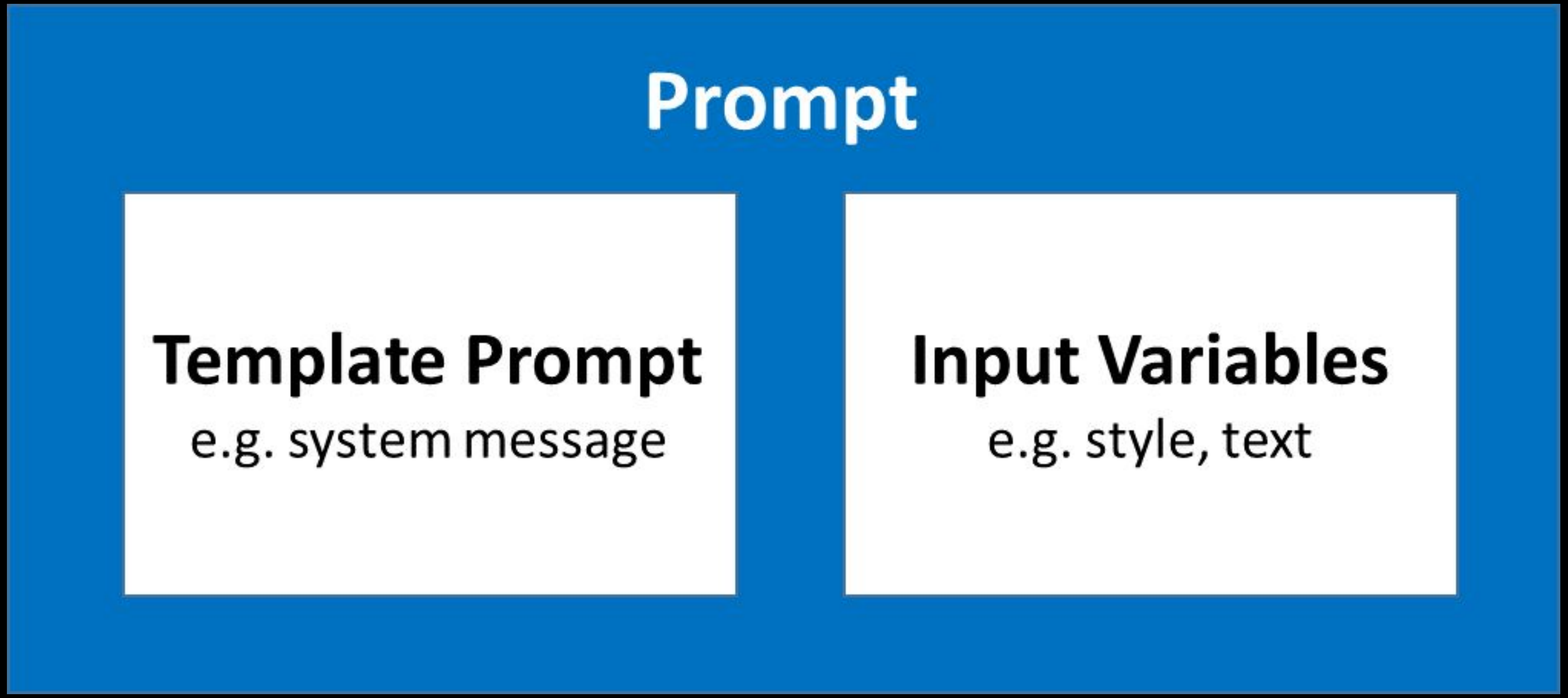


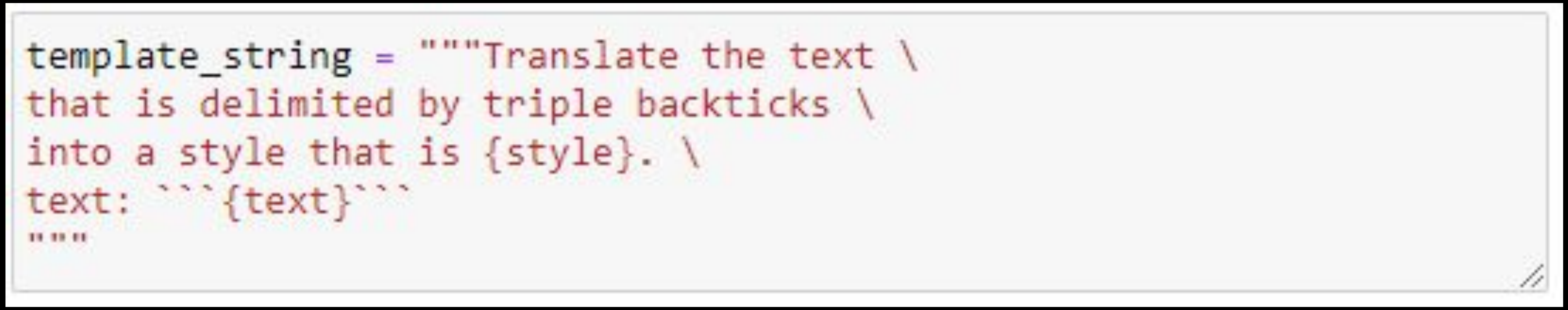
```
llm_model = "gpt-3.5-turbo-0301"
```

```
llm = ChatOpenAI(temperature=0.9, model=llm_model)
```

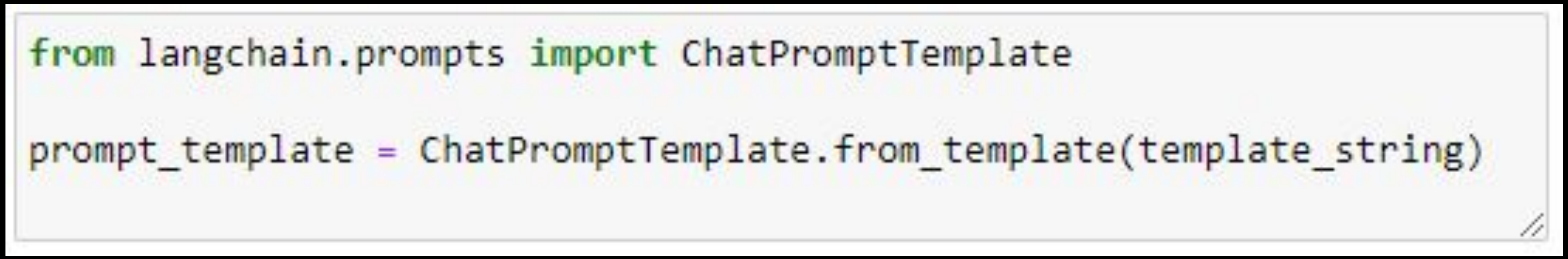
QUICK RECAP- LangChain

Model, Prompts & Parsers





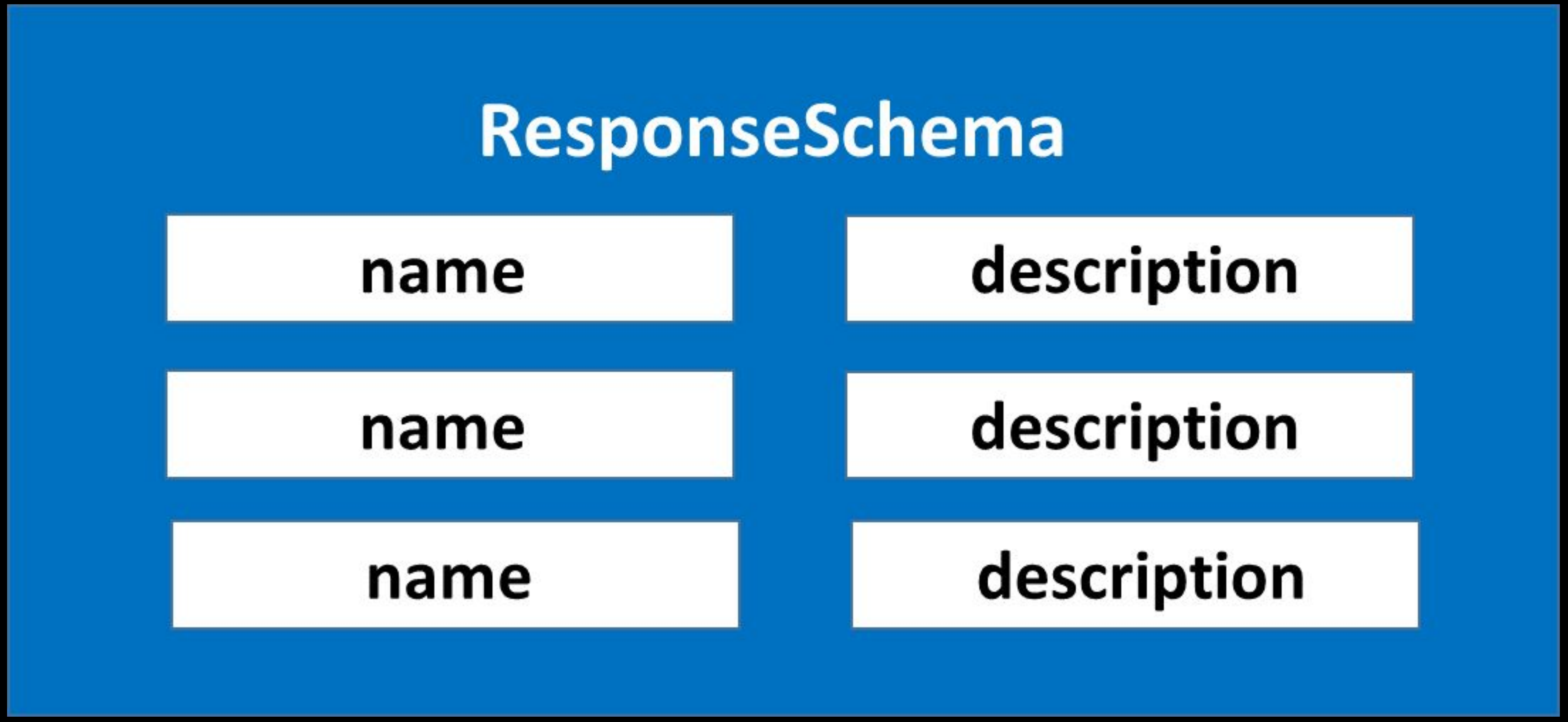
```
template_string = """Translate the text \
that is delimited by triple backticks \
into a style that is {style}. \
text: ```{text}```
"""
```

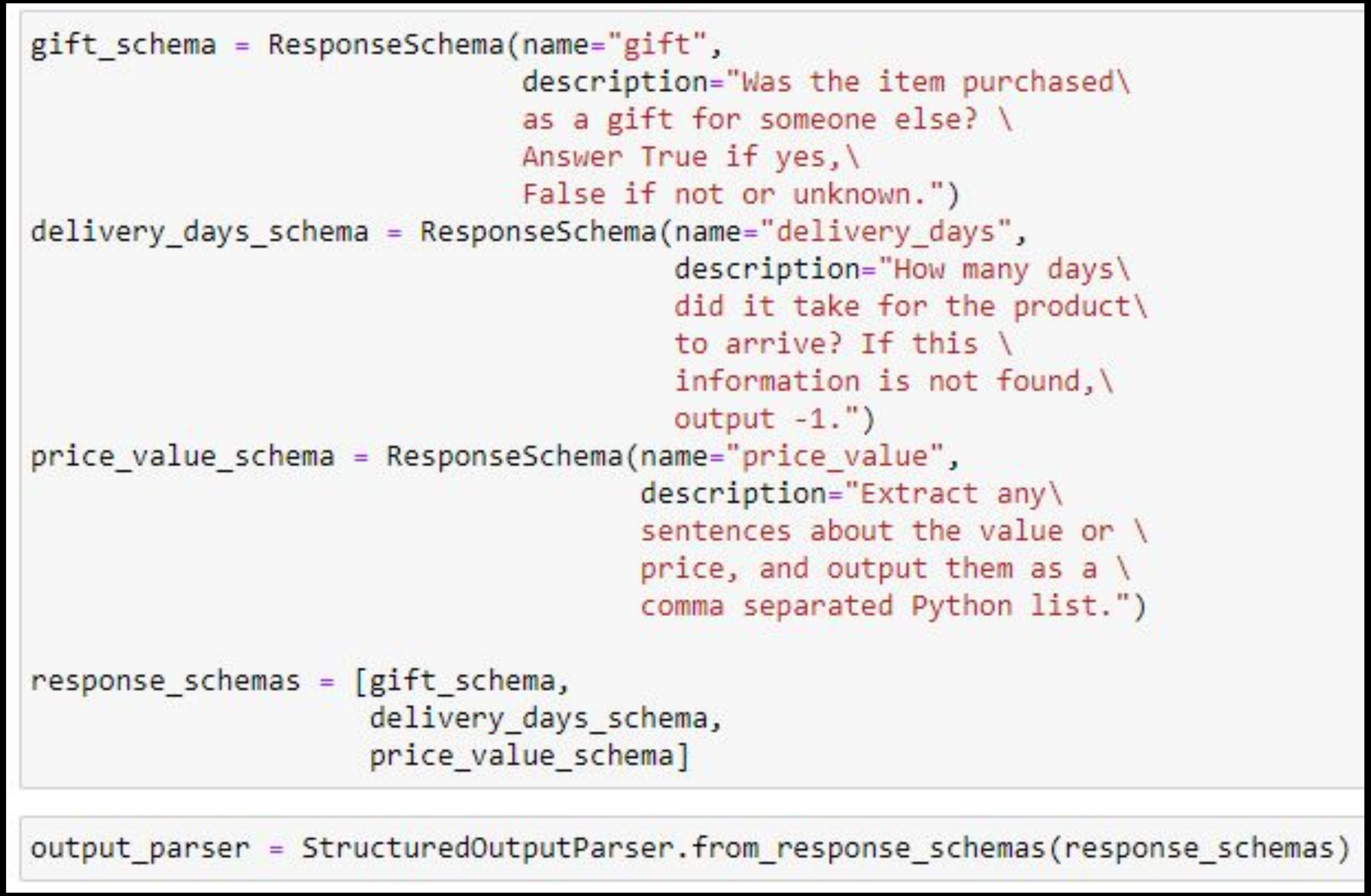


```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(template_string)
```

QUICK RECAP- LangChain

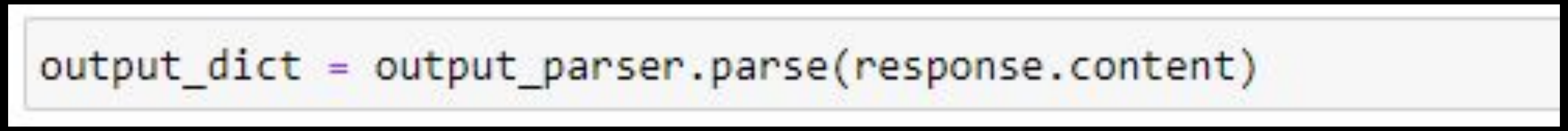
Model, Prompts & Parsers





```
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)
```



```
output_dict = output_parser.parse(response.content)
```

QUICK RECAP- LangChain

Model, Prompts & Parsers

```python
# This import connects your application with OpenAI's language models, enabling a variety of language processing tasks.
from langchain.llms import OpenAI

# This object allows integration of OpenAI's chat models in applications for creating interactive chatbots or conversational agents.
from langchain.chat_models import ChatOpenAI

# ChatPromptTemplate aids in creating, managing, and customizing structured templates for efficient interactions with language models, particularly in chat-based contexts.
from langchain.prompts import ChatPromptTemplate
```

```python
# ResponseSchema defines the expected structure of language model responses, helping in organizing and effectively interpreting output data.
from langchain.output_parsers import ResponseSchema

# StructuredOutputParser facilitates parsing language model outputs into structured formats like dictionaries or JSON for ease of use in applications.
from langchain.output_parsers import StructuredOutputParser
```

QUICK RECAP- LangChain

Model, Prompts & Parsers

from langchain.llms import **OpenAI**

This import connects your application with OpenAI's language models, enabling a variety of language processing tasks.

from langchain.chat_models import **ChatOpenAI**

This object allows integration of OpenAI's chat models in applications for creating interactive chatbots or conversational agents.

from langchain.prompts import **ChatPromptTemplate**

ChatPromptTemplate aids in creating, managing, and customizing structured templates for efficient interactions with language models, particularly in chat-based contexts.

QUICK RECAP- LangChain

Model, Prompts & Parsers

from langchain.output_parsers import **ResponseSchema**

ResponseSchema defines the expected structure of language model responses, helping in organizing and effectively interpreting output data.

from langchain.output_parsers import **StructuredOutputParser**

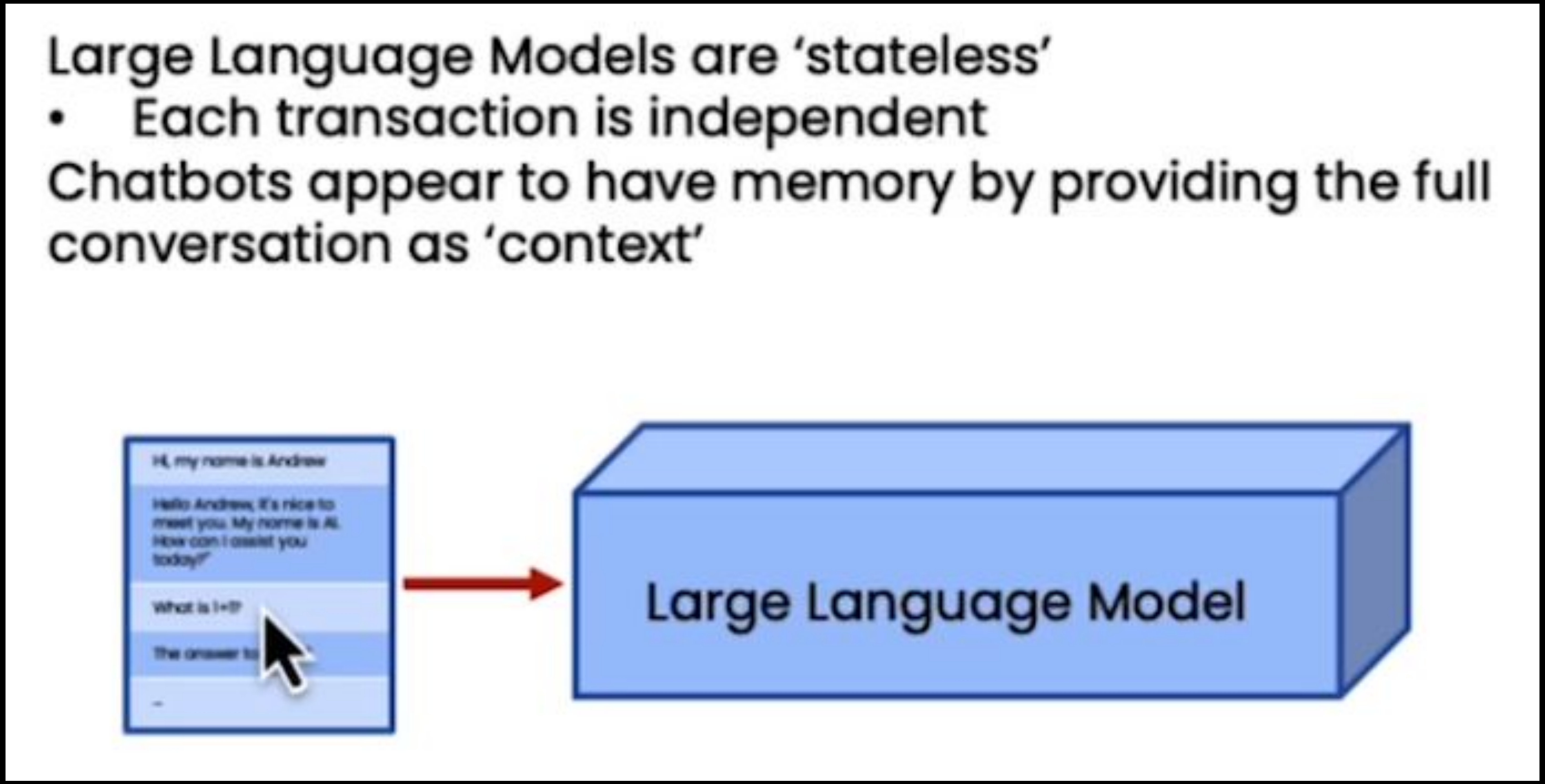
StructuredOutputParser facilitates parsing language model outputs into structured formats like dictionaries or JSON for ease of use in applications.

QUICK RECAP- LangChain

Memory

QUICK RECAP- LangChain

Memory



Large Language Models are 'stateless'

- Each transaction is independent

Chatbots appear to have memory by providing the full conversation as 'context'

Hi, my name is Andrew

Hello Andrew, it's nice to meet you. My name is AI. How can I assist you today?"

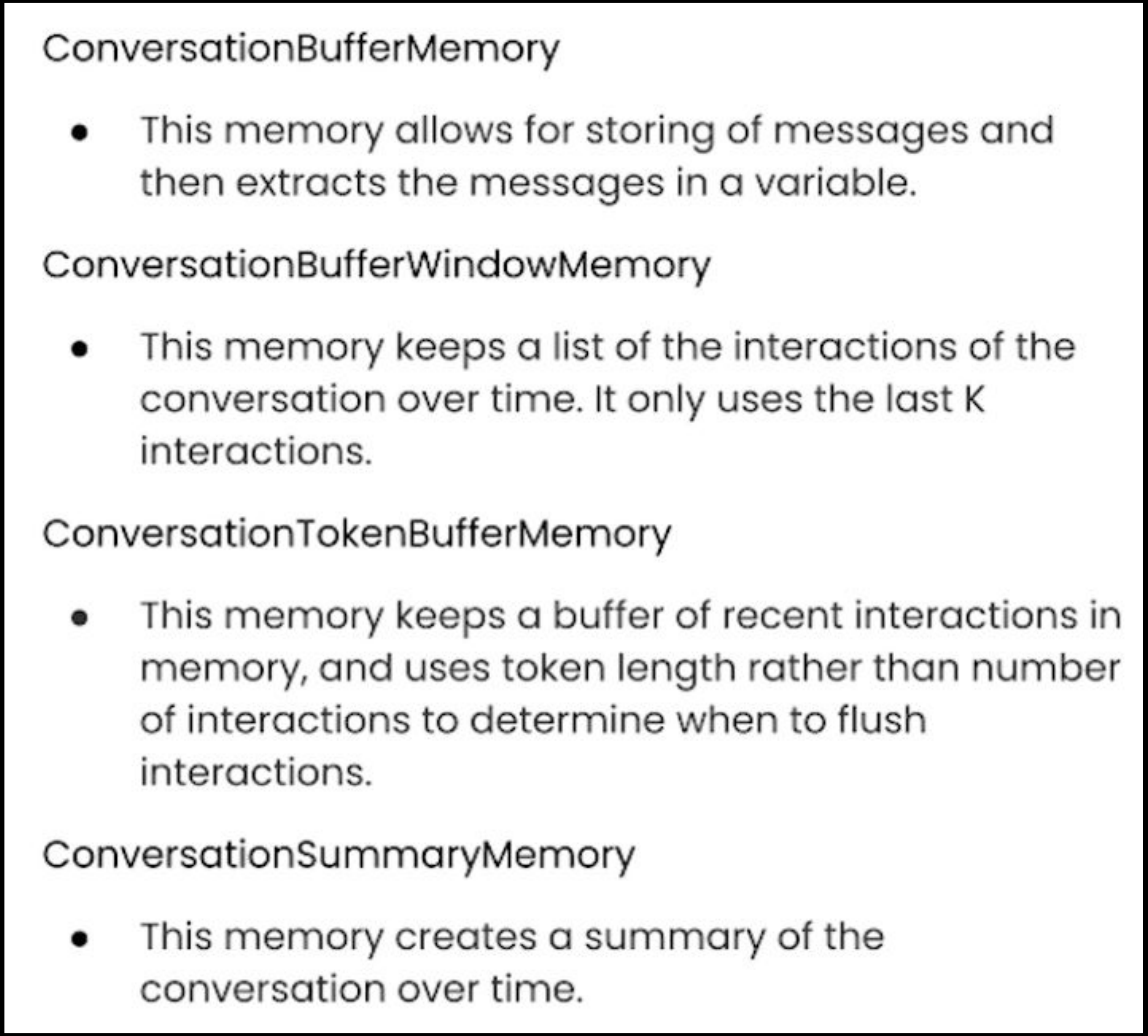
What is 1+1?

The answer to

...

QUICK RECAP- LangChain

Memory



ConversationBufferMemory

- This memory allows for storing of messages and then extracts the messages in a variable.

ConversationBufferWindowMemory

- This memory keeps a list of the interactions of the conversation over time. It only uses the last K interactions.

ConversationTokenBufferMemory

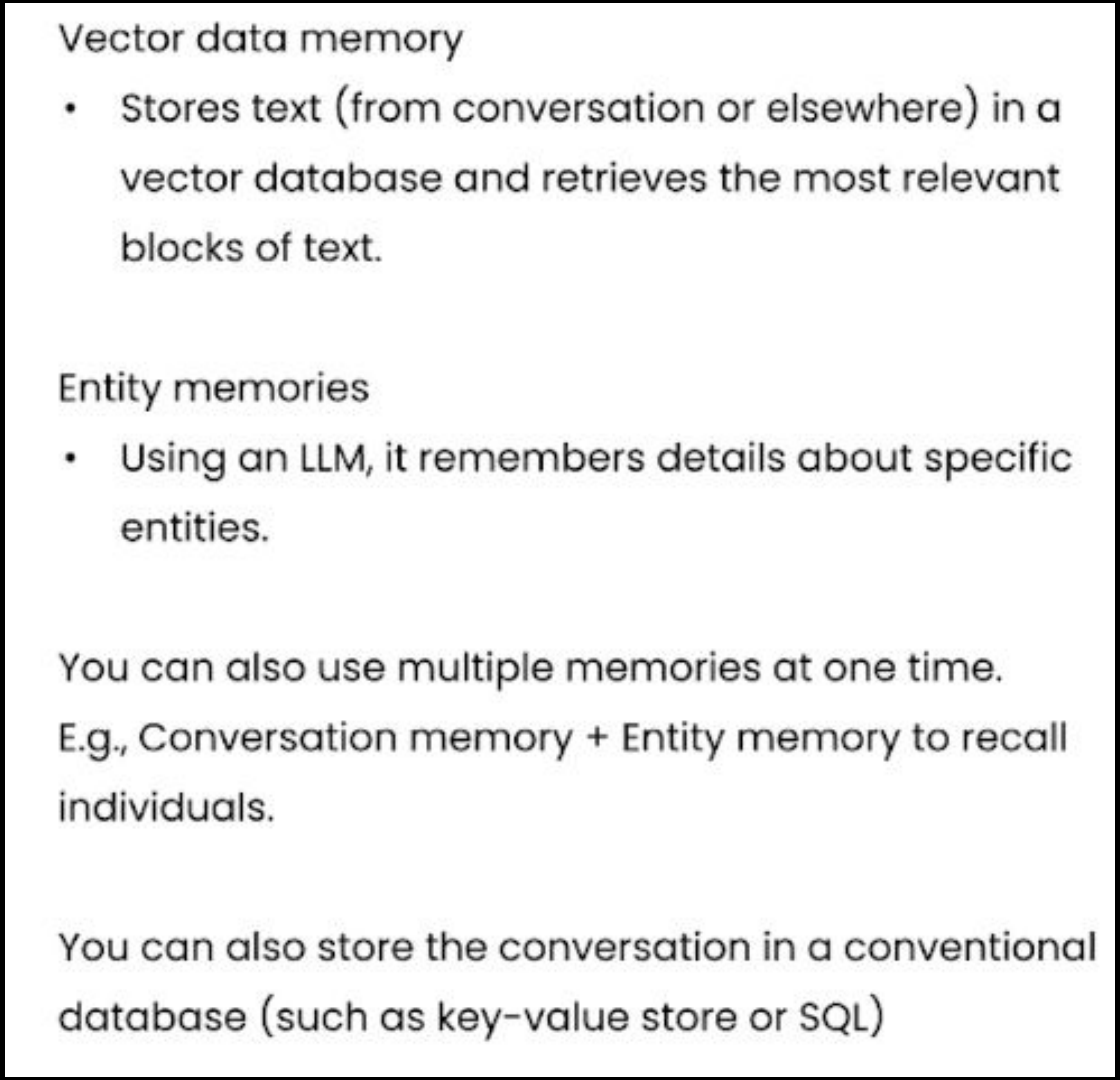
- This memory keeps a buffer of recent interactions in memory, and uses token length rather than number of interactions to determine when to flush interactions.

ConversationSummaryMemory

- This memory creates a summary of the conversation over time.

QUICK RECAP- LangChain

Memory



Vector data memory

- Stores text (from conversation or elsewhere) in a vector database and retrieves the most relevant blocks of text.

Entity memories

- Using an LLM, it remembers details about specific entities.

You can also use multiple memories at one time.
E.g., Conversation memory + Entity memory to recall individuals.

You can also store the conversation in a conventional database (such as key-value store or SQL)

QUICK RECAP- LangChain

Memory

```python
# ConversationChain manages and orchestrates conversational flow, ensuring coherent and contextually relevant interactions.
from langchain.chains import ConversationChain
```

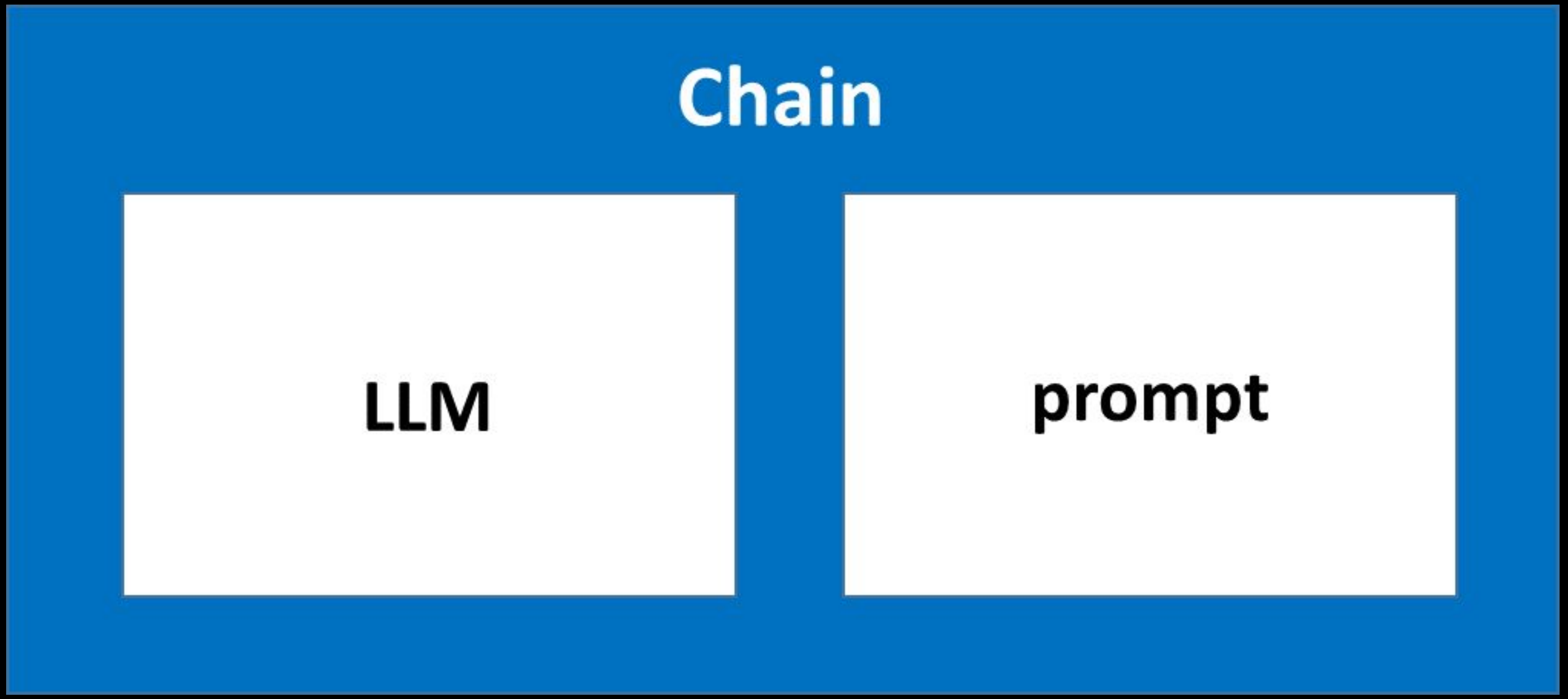
```python
# Stores the entire history of a conversation, maintaining context and continuity throughout interactions.
from langchain.memory import ConversationBufferMemory

# Tracks specific tokens or parts of a conversation for more detailed control and recall of conversation elements.
from langchain.memory import ConversationTokenBufferMemory

# Stores and recalls specific segments of a conversation, focusing on the most recent or relevant parts.
from langchain.memory import ConversationBufferWindowMemory

# Stores summarized versions of conversations for efficient recall of key points or themes without processing the entire history.
from langchain.memory import ConversationSummaryBufferMemory
```

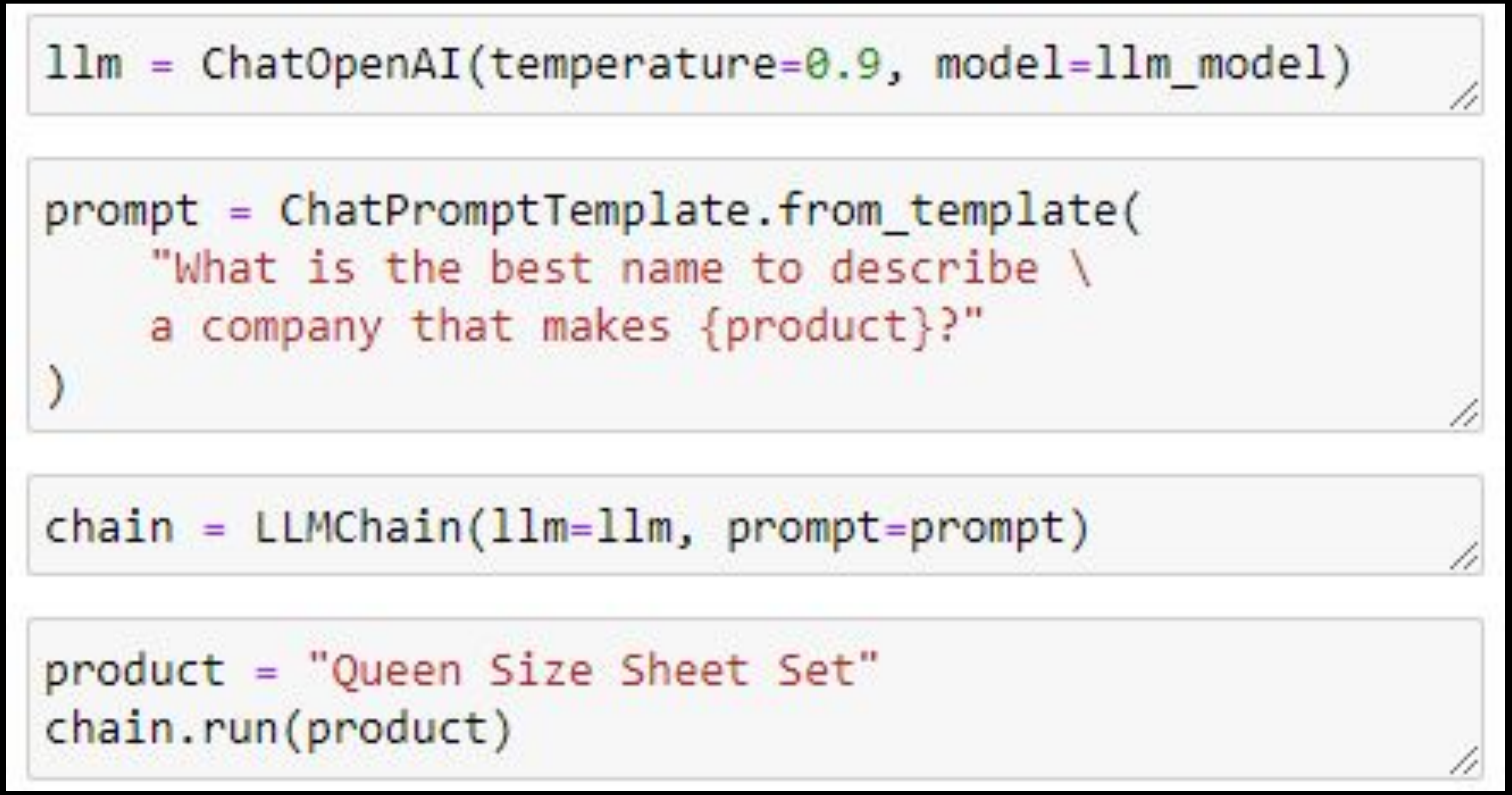
QUICK RECAP- LangChain

Chains

QUICK RECAP- LangChain

Chains





```
llm = ChatOpenAI(temperature=0.9, model=llm_model)
```

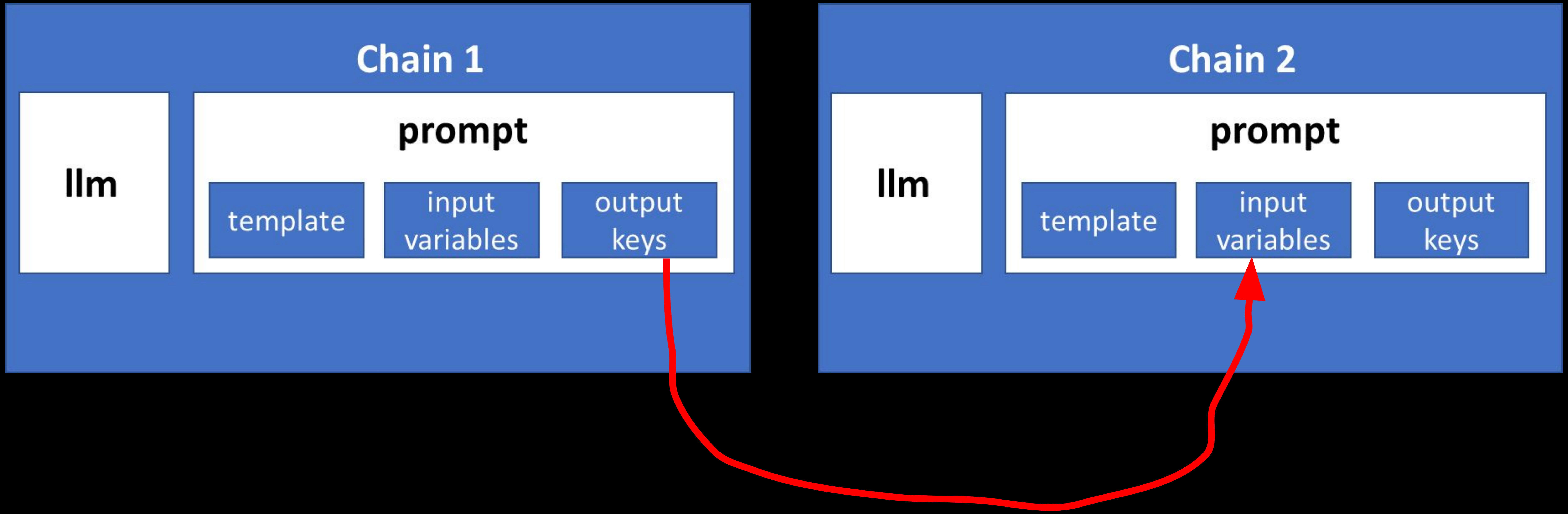
```
prompt = ChatPromptTemplate.from_template(
    "What is the best name to describe \
    a company that makes {product}?"
)
```

```
chain = LLMChain(llm=llm, prompt=prompt)
```

```
product = "Queen Size Sheet Set"
chain.run(product)
```

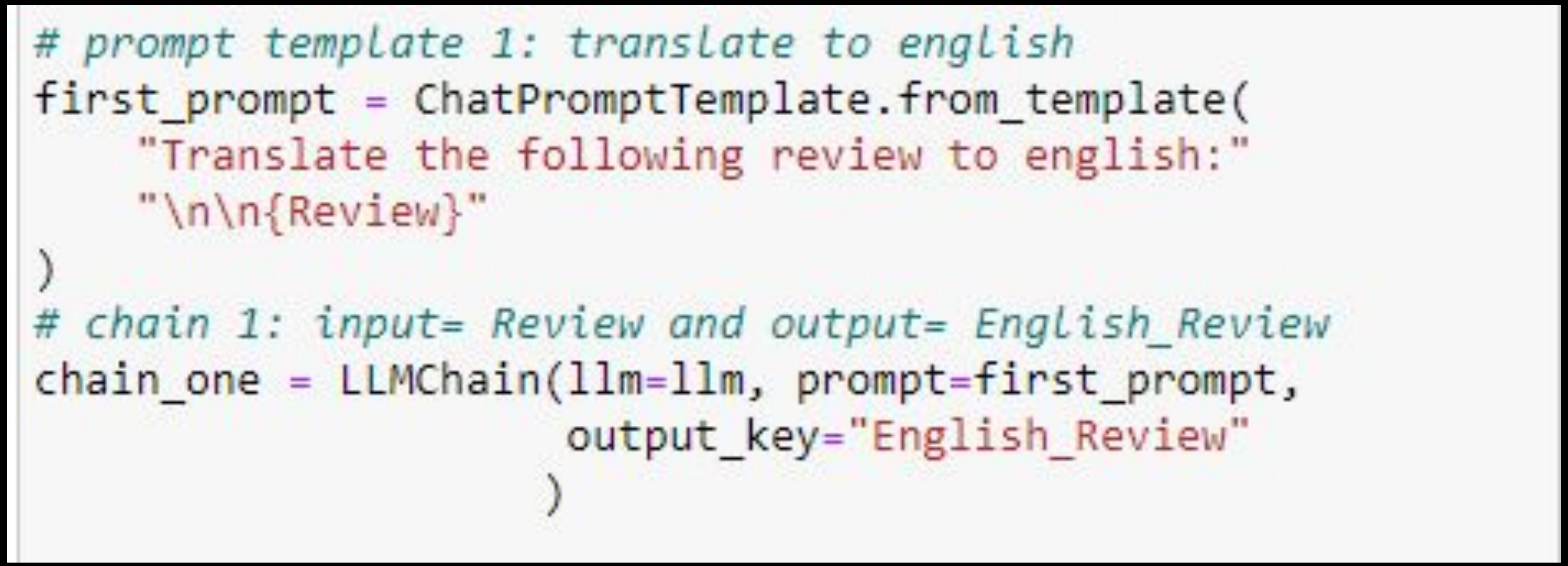
QUICK RECAP- LangChain

Chains

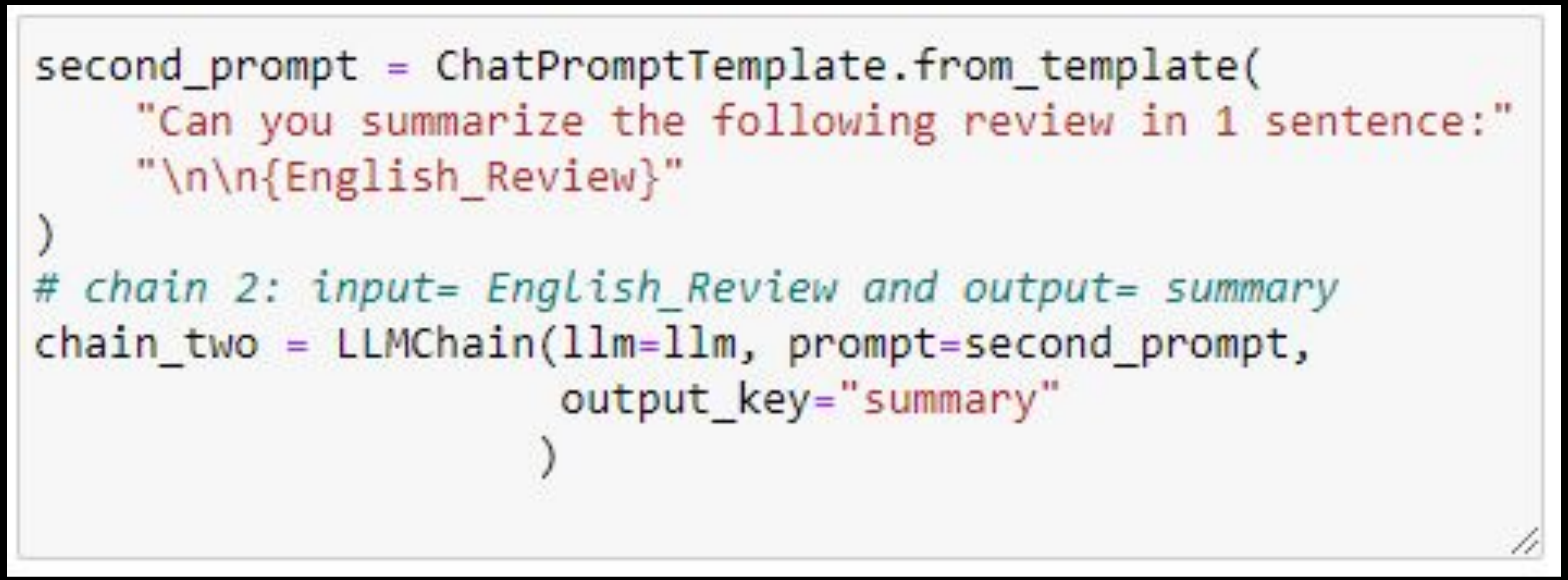


QUICK RECAP- LangChain

Chains



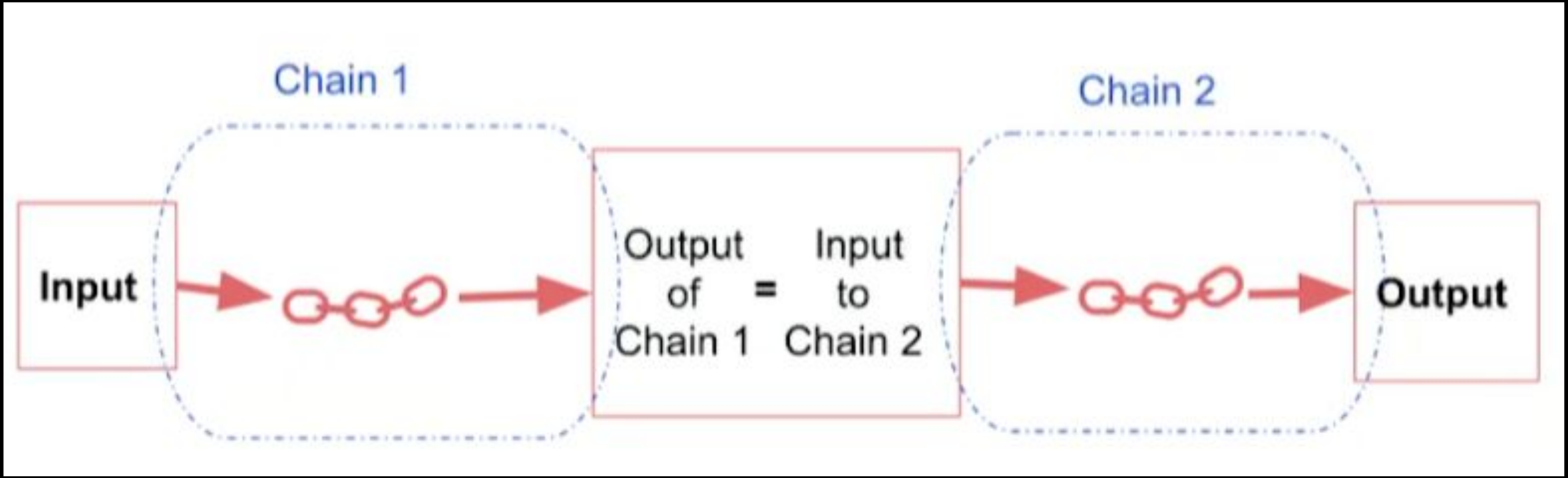
```python
# prompt template 1: translate to english
first_prompt = ChatPromptTemplate.from_template(
    "Translate the following review to english:"
    "\n\n{Review}"
)
# chain 1: input= Review and output= English_Review
chain_one = LLMChain(llm=llm, prompt=first_prompt,
                     output_key="English_Review"
                    )
```



```python
second_prompt = ChatPromptTemplate.from_template(
    "Can you summarize the following review in 1 sentence:"
    "\n\n{English_Review}"
)
# chain 2: input= English_Review and output= summary
chain_two = LLMChain(llm=llm, prompt=second_prompt,
                     output_key="summary"
                    )
```

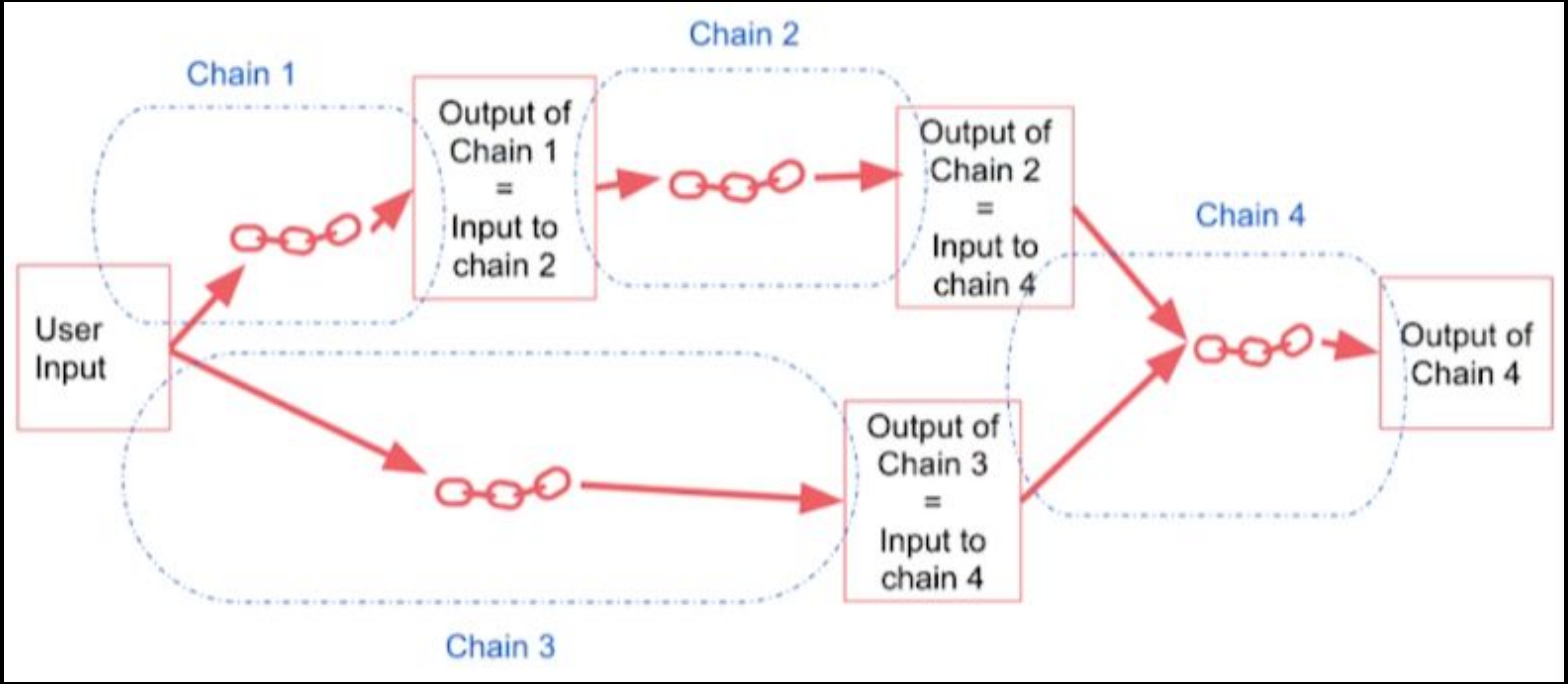
QUICK RECAP- LangChain

Chains



QUICK RECAP- LangChain

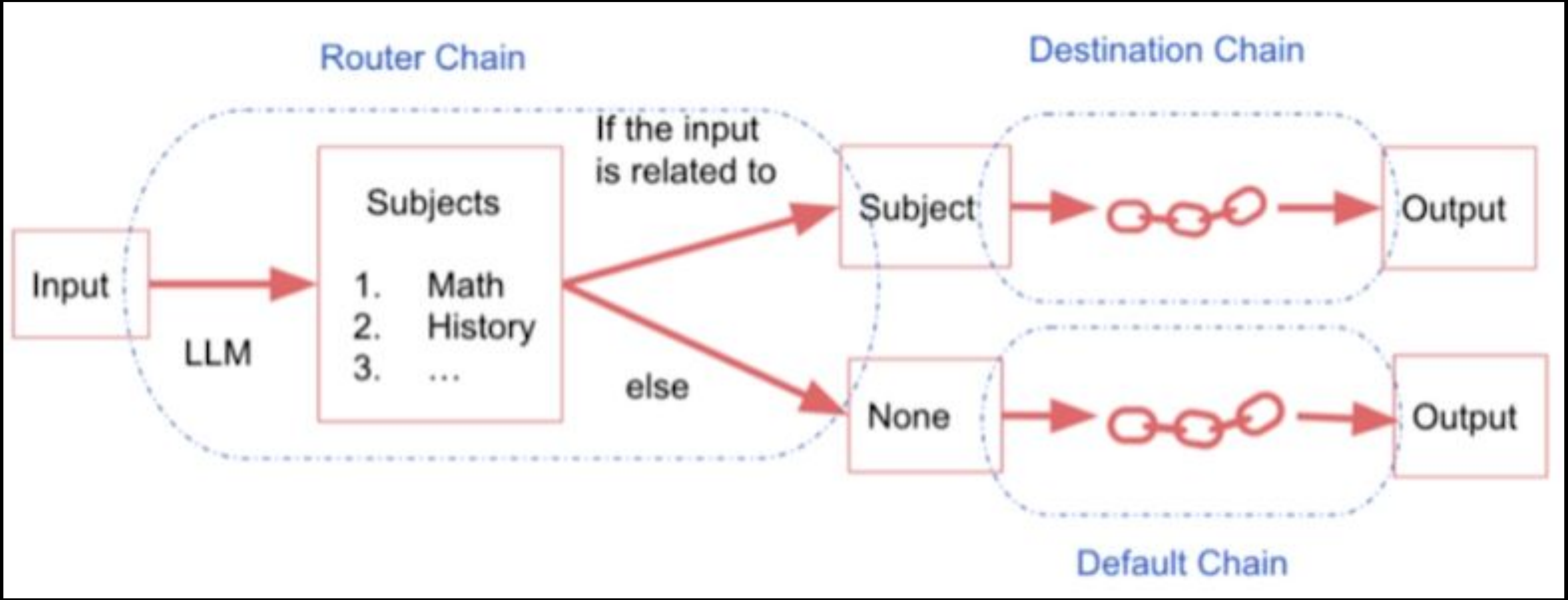
Chains



Chain 1
Chain 2
Chain 3
Chain 4
User Input
Output of Chain 1 = Input to chain 2
Output of Chain 2 = Input to chain 4
Output of Chain 3 = Input to chain 4
Output of Chain 4

QUICK RECAP- LangChain

Chains



QUICK RECAP- LangChain

Chains

```
# PromptTemplate is used for creating and managing customizable prompt templates for various interactions with language models.
from langchain.prompts import PromptTemplate

# Allows chaining multiple operations involving language models, enabling streamlined execution of complex tasks.
from langchain.chains import LLMChain
```

```
# Provides a simple method to execute a series of tasks or interactions in sequence, suitable for straightforward workflows.
from langchain.chains import SimpleSequentialChain

# Offers a more complex structure for executing sequences of tasks, allowing conditional and intricate operations.
from langchain.chains import SequentialChain
```

QUICK RECAP- LangChain

Chains

```python
# Enables using multiple prompts in a chain for varied and dynamic language model interactions.
from langchain.chains.router import MultiPromptChain

# LLMRouterChain manages task routing to different language model chains
from langchain.chains.router.llm_router import LLMRouterChain,

# RouterOutputParser structures outputs from these chains
from langchain.chains.router.llm_router import RouterOutputParser
```

QUICK RECAP- LangChain

Chains

```python
# Enables using multiple prompts in a chain for varied and dynamic language model interactions.
from langchain.chains.router import MultiPromptChain

# LLMRouterChain manages task routing to different language model chains
from langchain.chains.router.llm_router import LLMRouterChain,

# RouterOutputParser structures outputs from these chains
from langchain.chains.router.llm_router import RouterOutputParser
```

QUICK RECAP- LangChain

Chains

from langchain.prompts import PromptTemplate

PromptTemplate is used for creating and managing customizable prompt templates for various interactions with language models.

from langchain.chains import LLMChain

Allows chaining multiple operations involving language models, enabling streamlined execution of complex tasks.

QUICK RECAP- LangChain

Chains

from langchain.chains import **SimpleSequentialChain**

Provides a simple method to execute a series of tasks or interactions in sequence, suitable for straightforward workflows.

from langchain.chains import **SequentialChain**

Offers a more complex structure for executing sequences of tasks, allowing conditional and intricate operations.

QUICK RECAP- LangChain

Chains

from langchain.chains.router import **MultiPromptChain**

Enables using multiple prompts in a chain for varied and dynamic language model interactions.

from langchain.chains.router.llm_router import **LLMRouterChain**,

LLMRouterChain manages task routing to different language model chains

from langchain.chains.router.llm_router import **RouterOutputParser**

RouterOutputParser structures outputs from these chains

QUICK RECAP- LangChain

Q&A Chains

QUICK RECAP- LangChain

Q&A Chains

from langchain.chains import **RetrievalQA**

RetrievalQA facilitates question-answering tasks by retrieving relevant information from documents or data sources.

from langchain.document_loaders import **CSVLoader**

CSVLoader handles loading and processing data from CSV files.

QUICK RECAP- LangChain

Q&A Chains

from langchain.vectorstores import **DocArrayInMemorySearch**

Enables in-memory storage and searching of document vectors for efficient information retrieval.

from langchain.indexes import **VectorstoreIndexCreator**

Aids in creating indexes for vector stores to optimize document vector search and retrieval.

from langchain.embeddings import **OpenAIEmbeddings**

Utilizes OpenAI's models to convert text into vector representations for tasks like semantic search or document clustering.

QUICK RECAP- LangChain

Evaluation

QUICK RECAP- LangChain

Evaluation

from langchain.evaluation.qa import **QAGenerateChain**

Automates generation of QA pairs for training or testing language models in QA tasks.

import langchain; **langchain.debug = True**

Activates debug mode for detailed insights into LangChain's internal processes, aiding in troubleshooting.

from langchain.evaluation.qa import **QAEvalChain**

Evaluates language model performance in QA tasks, assessing comprehension and response quality to questions.

QUICK RECAP- LangChain

Agents

QUICK RECAP- LangChain

Example of how a agent could work

1. The agent gets your query.
2. The agent generates a thought. The thought is the first part of the prompt. Its goal is to break the query down into smaller steps.
3. Based on the thought, the agent chooses the tool to use and generates a text input for that tool.
4. Based on the output the agent gets from the tool, it either stops and returns the answer or repeats the process from step 2.
5. The agent iterates over the steps as many times as it needs to generate an accurate answer.

QUICK RECAP- LangChain

Example of an agent prompt (Zero-shot ReACT) part 1:

You are a helpful and knowledgeable agent. To achieve your goal of answering complex questions correctly, you have access to the following tools:

{tool_names_with_descriptions}

Use the following format:

Question: the question to be answered
Thought: Reason if you have the final answer. If yes, answer the question. If not, find out the missing information needed to answer it.
[…]

QUICK RECAP- LangChain

Example of an agent prompt (Zero-shot ReACT) part 2:

```
[...]
Tool: pick one of {tool_names}
Tool Input: the input for the tool
Observation: the tool will respond with the result
...

Final Answer: the final answer to the question, make it short (1-5 words)
Thought, Tool, Tool Input, and Observation steps can be repeated multiple times, but sometimes we
can find an answer in the first pass
---

Question: {query}
Thought: Let's think step-by-step, I first need to
```

https://prompthub.deepset.ai/?prompt=deepset%2Fzero-shot-react

QUICK RECAP- LangChain

Agents

```
# Creates Python-based agents capable of executing Python code and interacting within applications.
from langchain.agents.agent_toolkits import create_python_agent

# These functions prepare agents by loading tools and initializing them for application use.
from langchain.agents import load_tools, initialize_agent

# Defines various agent types based on their capabilities and roles in applications.
from langchain.agents import AgentType
```

QUICK RECAP- LangChain

Agents

```python
# Simulates a Python REPL for dynamic code execution by agents.
from langchain.tools.python.tool import PythonREPLTool

# Provides an interface for running Python scripts or commands within applications.
from langchain.python import PythonREPL

# Offers functionalities for creating and managing tools used by agents in various tasks.
from langchain.agents import tool
```

QUICK RECAP- LangChain

Agents - (Custom) Tools

```python
from langchain.tools import tool

@tool
def search_api(query: str) -> str:
    """Searches the API for the query."""
    return f"Results for query {query}"
```

QUICK RECAP- LangChain

Agents

from langchain.agents.agent_toolkits import **create_python_agent**

Creates Python-based agents capable of executing Python code and interacting within applications.

from langchain.agents import **load_tools, initialize_agent**

These functions prepare agents by loading tools and initializing them for application use.

from langchain.agents import **AgentType**

Defines various agent types based on their capabilities and roles in applications.

QUICK RECAP- LangChain

Agents

from langchain.tools.python.tool import **PythonREPLTool**

Simulates a Python REPL for dynamic code execution by agents.

from langchain.python import **PythonREPL**

Provides an interface for running Python scripts or commands within applications.

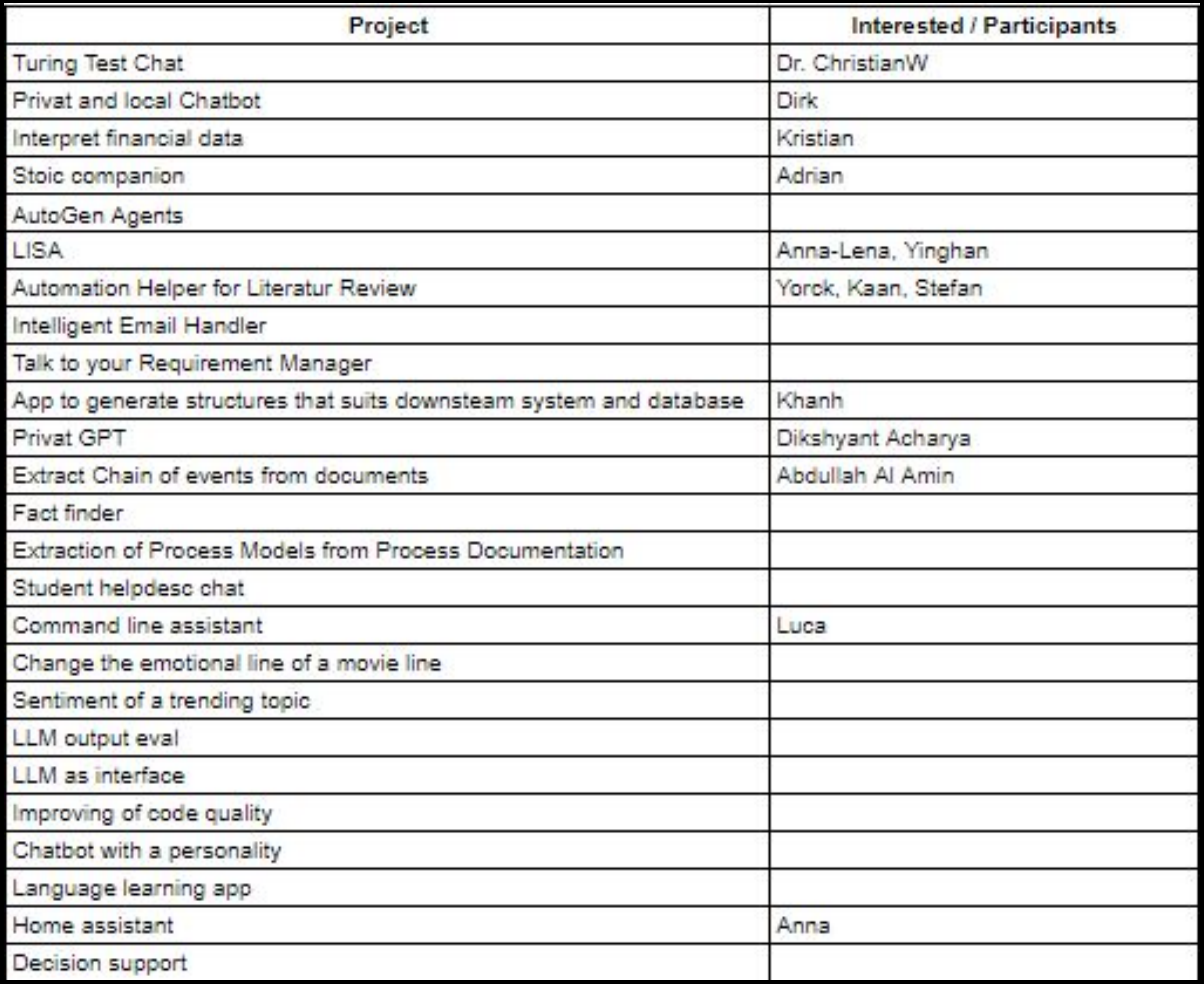
from langchain.agents **import tool**

Offers functionalities for creating and managing tools used by agents in various tasks.

TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

Overview of Project Ideas & Pitches



Project	Interested / Participants
Turing Test Chat	Dr. ChristianW
Privat and local Chatbot	Dirk
Interpret financial data	Kristian
Stoic companion	Adrian
AutoGen Agents	
LISA	Anna-Lena, Yinghan
Automation Helper for Literatur Review	Yorck, Kaan, Stefan
Intelligent Email Handler	
Talk to your Requirement Manager	
App to generate structures that suits downsteam system and database	Khanh
Privat GPT	Dikshyant Acharya
Extract Chain of events from documents	Abdullah Al Amin
Fact finder	
Extraction of Process Models from Process Documentation	
Student helpdesc chat	
Command line assistant	Luca
Change the emotional line of a movie line	
Sentiment of a trending topic	
LLM output eval	
LLM as interface	
Improving of code quality	
Chatbot with a personality	
Language learning app	
Home assistant	Anna
Decision support	

Overview of Project Ideas & Pitches

https://padlet.com/christianwiesner1/project-cards-8dphuws9a20ma2nf

TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

Breakout Session: Building Project Groups

TODAY'S SCHEDULE

- Quiz
- Homework presentation
- Short Recap on LangChain
- Overview of Project Ideas & Pitches
- Breakout Session: Building Project Groups
- Homework for Next Week

Homework Until Next Week

- Watch and work through the **course “LangChain: Chat with your Data”** on Deeplearning.AI
- Work through the **tutorial** [LangChain tutorial](url) and understand the code
- Modify the code according to the **Jupyter notebook**

- Get together in your **project group**, get to know each other and define a Minimum Viable Product of the project you want to build.