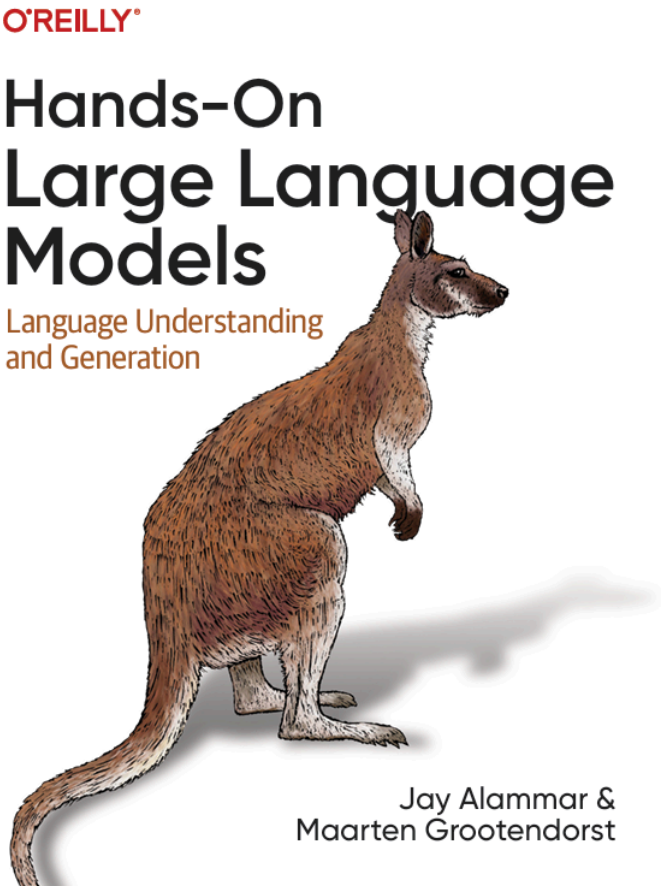
Chapter 7 - Advanced Text Generation Techniques and Tools

Going beyond prompt engineering.

Buy the Book! O'Reilly GitHub Repository Open in Colab

This notebook is for Chapter 7 of the Hands-On Large Language Models book by Jay Alammar and Maarten Grootendorst.



[OPTIONAL] - Installing Packages on Google colab

If you are viewing this notebook on Google Colab (or any other cloud vendor), you need to **uncomment and run** the following codeblock to install the dependencies for this chapter:

💡 **NOTE**: We will want to use a GPU to run the examples in this notebook. In Google Colab, go to **Runtime > Change runtime type > Hardware accelerator > GPU > GPU**

type > T4.

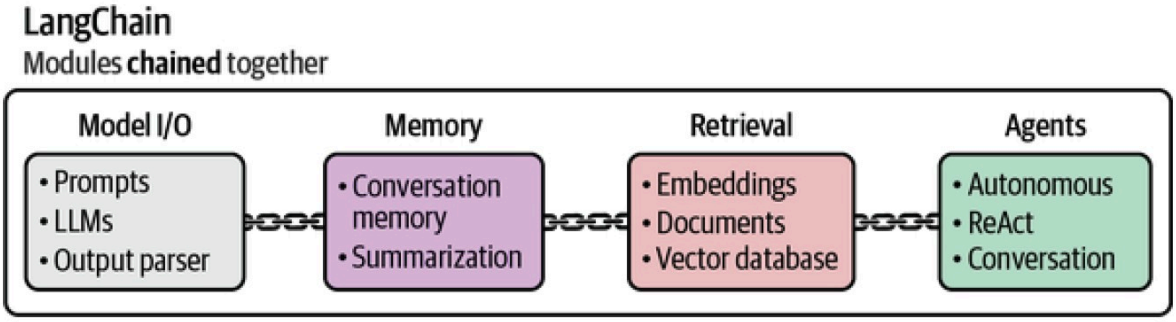


Figure 7-1. LangChain is a complete framework for using LLMs. It has modular components that can be chained together to allow for complex LLM systems.

```
%%capture
!pip install langchain>=0.1.17 transformers>=4.40.1 datasets>=2.18.0 accelerate
```

```
!pip install langchain_classic
```

```
Requirement already satisfied: langchain_classic in /usr/local/lib/python3.12/di
Requirement already satisfied: langchain-core<2.0.0,>=1.2.5 in /usr/local/lib/py
Requirement already satisfied: langchain-text-splitters<2.0.0,>=1.1.0 in /usr/lc
Requirement already satisfied: langsmith<1.0.0,>=0.1.17 in /usr/local/lib/python
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in /usr/local/lib/python3.
Requirement already satisfied: pyyaml<7.0.0,>=5.3.0 in /usr/local/lib/python3.12
Requirement already satisfied: requests<3.0.0,>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: sqlalchemy<3.0.0,>=1.4.0 in /usr/local/lib/python
Requirement already satisfied: jsonpatch<2.0.0,>=1.33.0 in /usr/local/lib/python
Requirement already satisfied: packaging<26.0.0,>=23.2.0 in /usr/local/lib/pythc
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in /usr/local/lib
Requirement already satisfied: typing-extensions<5.0.0,>=4.7.0 in /usr/local/lib
Requirement already satisfied: uuid-utils<1.0,>=0.12.0 in /usr/local/lib/python3
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: requests-toolbelt>=1.0.0 in /usr/local/lib/python
Requirement already satisfied: zstandard>=0.23.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.1
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/d
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dis
```

```
!CMAKE_ARGS="-DLLAMA_CUDA=on" pip install llama-cpp-python
```

```
Collecting llama-cpp-python
  Using cached llama_cpp_python-0.3.16.tar.gz (50.7 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.12/dist-p
Collecting diskcache>=5.6.1 (from llama-cpp-python)
  Using cached diskcache-5.6.3-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: jinja2>=2.11.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist
Using cached diskcache-5.6.3-py3-none-any.whl (45 kB)
Building wheels for collected packages: llama-cpp-python
  Building wheel for llama-cpp-python (pyproject.toml) ... done
  Created wheel for llama-cpp-python: filename=llama_cpp_python-0.3.16-cp312-cp3
  Stored in directory: /root/.cache/pip/wheels/90/82/ab/8784ee3fb99ddb07fd36a679
Successfully built llama-cpp-python
Installing collected packages: diskcache, llama-cpp-python
Successfully installed diskcache-5.6.3 llama-cpp-python-0.3.16
```

Loading an LLM (Quantized model)

```
!wget https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/mair

# If this command does not work for you, you can use the link directly to downl
# https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi
```

```
--2026-01-06 06:50:26--  https://huggingface.co/microsoft/Phi-3-mini-4k-instruct
Resolving huggingface.co (huggingface.co)... 18.164.174.118, 18.164.174.23, 18.1
Connecting to huggingface.co (huggingface.co)|18.164.174.118|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cas-bridge.xethub.hf.co/xet-bridge-us/662698108f7573e6a6478546
--2026-01-06 06:50:26--  https://cas-bridge.xethub.hf.co/xet-bridge-us/662698108
Resolving cas-bridge.xethub.hf.co (cas-bridge.xethub.hf.co)... 18.164.174.21, 18
Connecting to cas-bridge.xethub.hf.co (cas-bridge.xethub.hf.co)|18.164.174.21|:4
HTTP request sent, awaiting response... 200 OK
Length: 7643295904 (7.1G)
Saving to: ‘Phi-3-mini-4k-instruct-fp16.gguf.1’

Phi-3-mini-4k-instr 100%[===================>]   7.12G   106MB/s    in 44s

2026-01-06 06:51:11 (165 MB/s) - ‘Phi-3-mini-4k-instruct-fp16.gguf.1’ saved [764
```

```
# from langchain import LlamaCpp
from langchain_community.llms import LlamaCpp
```

```
# Make sure the model path is correct for your system!
llm = LlamaCpp(
    model_path="Phi-3-mini-4k-instruct-fp16.gguf",
    n_gpu_layers=-1,
    max_tokens=500,
    n_ctx=2048,
    seed=42,
    verbose=False
)
```

```
llama_context: n_batch is less than GGML_KQ_MASK_PAD - increasing to 64
llama_context: n_ctx_per_seq (2048) < n_ctx_train (4096) -- the full capacity of
```

```
llm.invoke("Hi! My name is Maarten. What is 1 + 1?")
```

```
''
```

Chains

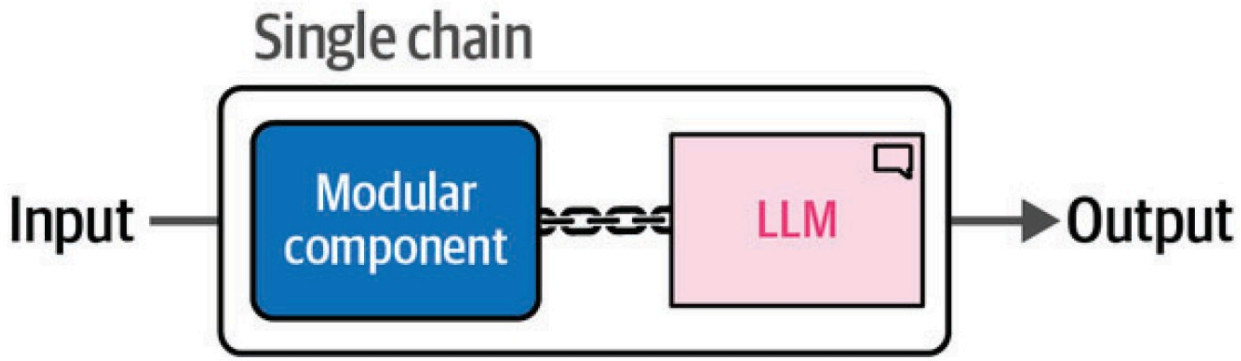


Figure 7-3. A single chain connects some modular component, like a prompt template or external memory, to the LLM.

Chaining Prompt Template with LLM

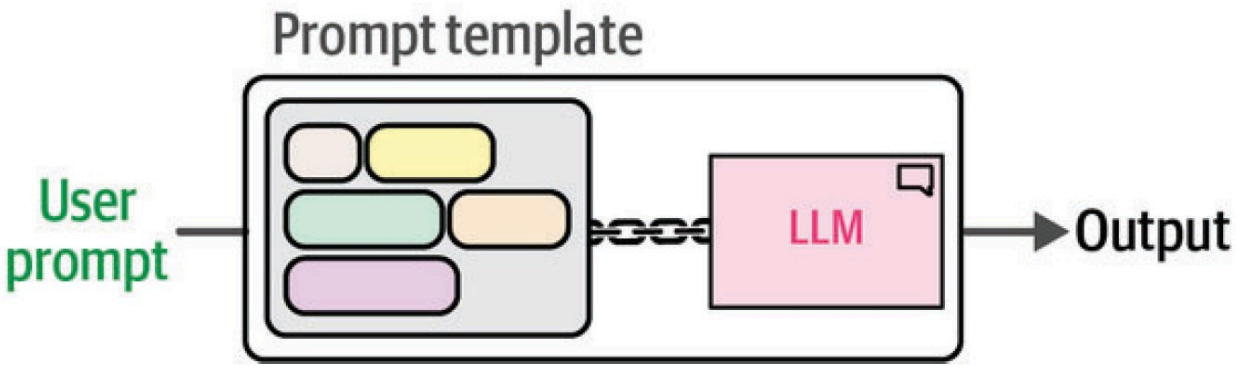


Figure 7-4. By chaining a prompt template with an LLM, we only need to define the input prompts. The template will be constructed for you.

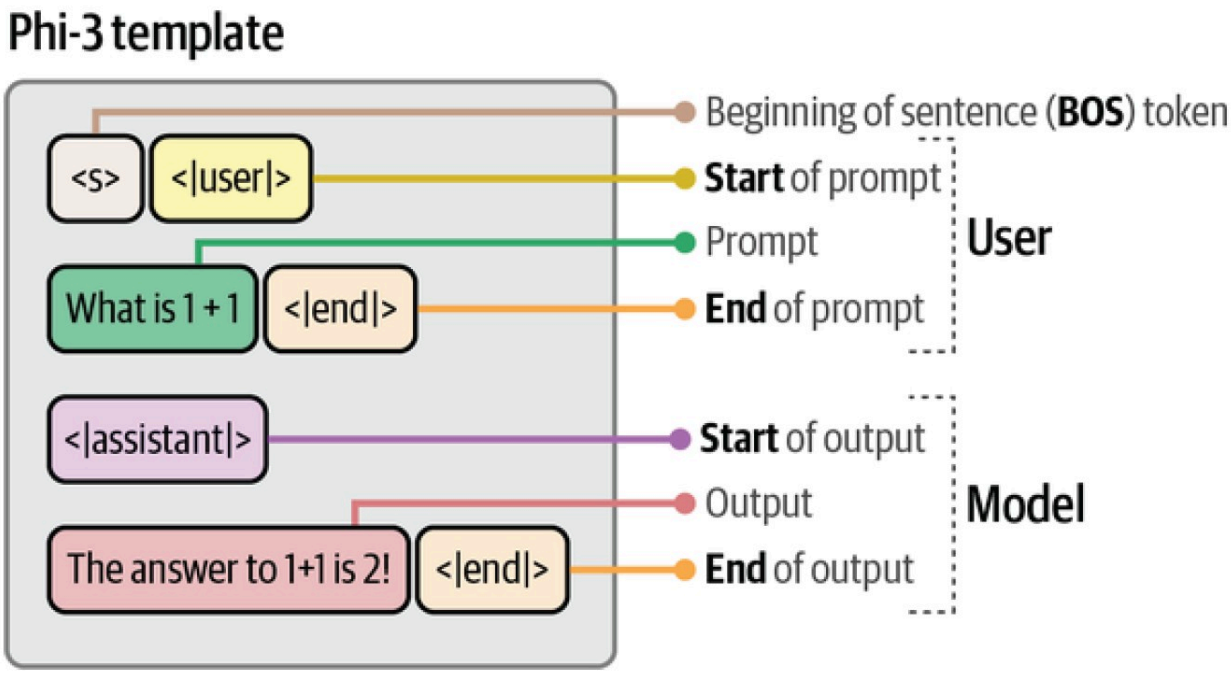


Figure 7-5. The prompt template Phi-3 expects.

```
from langchain_classic import PromptTemplate

# Create a prompt template with the "input_prompt" variable
template = """<s><|user|>
{input_prompt}<|end|>
<|assistant|>"""
prompt = PromptTemplate(
    template=template,
    input_variables=["input_prompt"]
)
```

```
basic_chain = prompt | llm
```

```
# Use the chain
basic_chain.invoke(
    {
        "input_prompt": "Hi! My name is Maarten. What is 1 + 1?",
    }
)
```

```
/usr/local/lib/python3.12/dist-packages/llama_cpp/llama.py:1242: RuntimeWarning:
  warnings.warn(
' Hello Maarten! The answer to 1 + 1 is 2.'
```

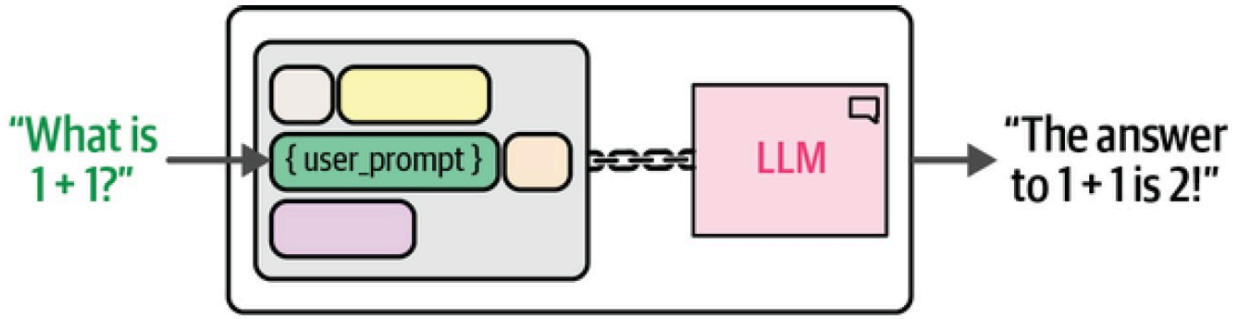


Figure 7-6. An example of a single chain using Phi-3's template.

Multiple Chains

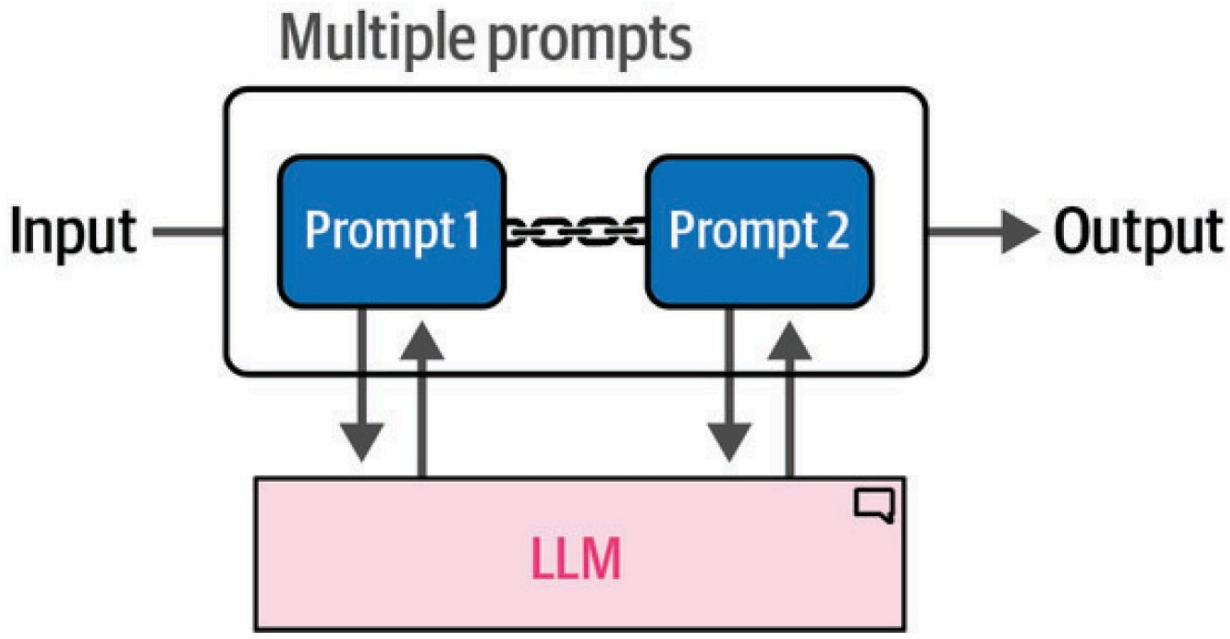


Figure 7-7. With sequential chains, the output of a prompt is used as the input for the next prompt.

Let's illustrate with an example. Assume we want to generate a story that has three components:

- a title
- a description of the main character
- a summary of the story

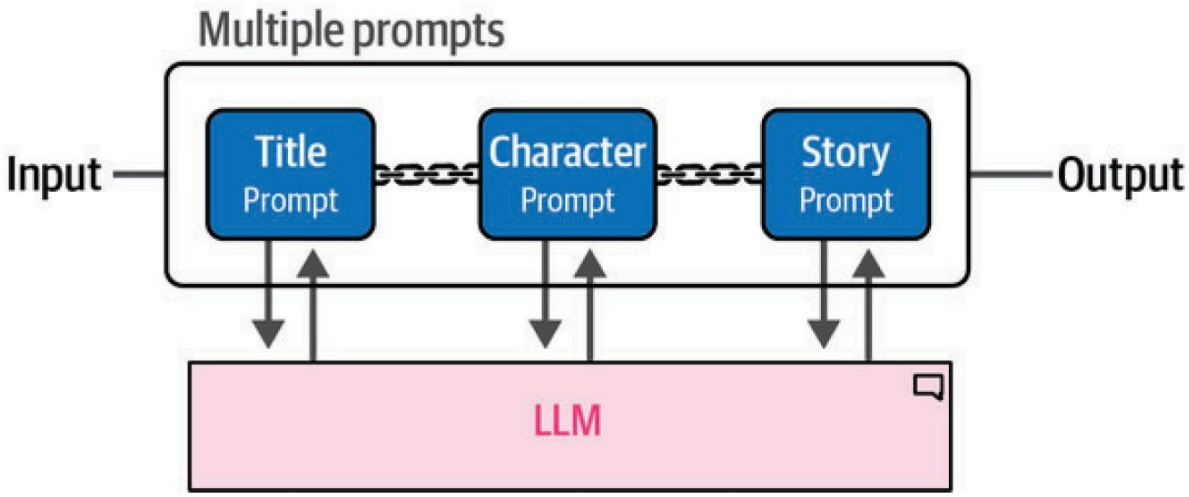


Figure 7-8. The output of the title prompt is used as the input of the character prompt. To generate the story, the output of all previous prompts is used.

```
from langchain_classic import LLMChain

# Create a chain for the title of our story
template = """<s><|user|>
Create a title for a story about {summary}. Only return the title.<|end|>
<|assistant|>"""
title_prompt = PromptTemplate(template=template, input_variables=["summary"])
title = LLMChain(llm=llm, prompt=title_prompt, output_key="title")
```

```
/tmp/ipython-input-1767282371.py:8: LangChainDeprecationWarning: The class `LLMC
  title = LLMChain(llm=llm, prompt=title_prompt, output_key="title")
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
title.invoke({"summary": "a man that lost his job"})
```

```
{'summary': 'a man that lost his job',
 'title': ' "The Unforeseen Departure: A Tale of Redemption and Resilience"'}
```

```
# Create a chain for the character description using the summary and title
template = """<s><|user|>
Describe the main character of a story about {summary} with the title {title}.
<|assistant|>"""
```

```
character_prompt = PromptTemplate(
    template=template, input_variables=["summary", "title"]
)
character = LLMChain(llm=llm, prompt=character_prompt, output_key="character")
```

```
# Create a chain for the story using the summary, title, and character descript
template = """<s><|user|>
Create a story about {summary} with the title {title}. The main charachter is:
<|assistant|>"""
story_prompt = PromptTemplate(
    template=template, input_variables=["summary", "title", "character"]
)
story = LLMChain(llm=llm, prompt=story_prompt, output_key="story")
```

```
# Combine all three components to create the full chain
llm_chain = title | character | story
```

```
llm_chain.invoke("a girl that lost her mother")
```

```
{'summary': 'a girl that lost her mother',
 'title': ' "Echoes of a Mother\'s Love: A Journey Through Grief"',
 'character': " The main character, Emily, is a resilient and compassionate
teenager who struggles to come to terms with the loss of her beloved mother. As
she embarks on a transformative journey through grief, she discovers inner
strength, forges new relationships, and learns to cherish the enduring echoes
of her mother's love that guide her forward.",
 'story': ' Emily, a resilient and compassionate teenager, stood on the
threshold of adulthood when tragedy struck - her mother\'s sudden passing. As
she navigated through her grief-stricken world, each day became an arduous
battle to accept this unbearable loss. In a journey that wove through sorrow
and solace alike, Emily began to discover the enduring echoes of her mother\'s
love within herself - whispers of warmth in winter\'s chill, comforting
lullabies beneath star-studded skies, and unwavering faith as steadfast as
mountains. Through the shared memories etched upon family albums, Emily forged
new connections with friends who became her chosen kin; they were threads woven
into the tapestry of grief that now enveloped her life. As time passed, these
echoes transformed from faint murmurs to a powerful symphony guiding Emily
forward, teaching her to cherish love\'s legacy and carry it within her heart -
an everlasting testament to her mother\'s unconditional embrace that
transcended the finality of death. "Echoes of a Mother\'s Love: A Journey
Through Grief" became not just Emily\'s story, but also a beacon for those
navigating their own paths through loss and healing.'}
```

Memory

- Conversation buffer
- Conversation summary

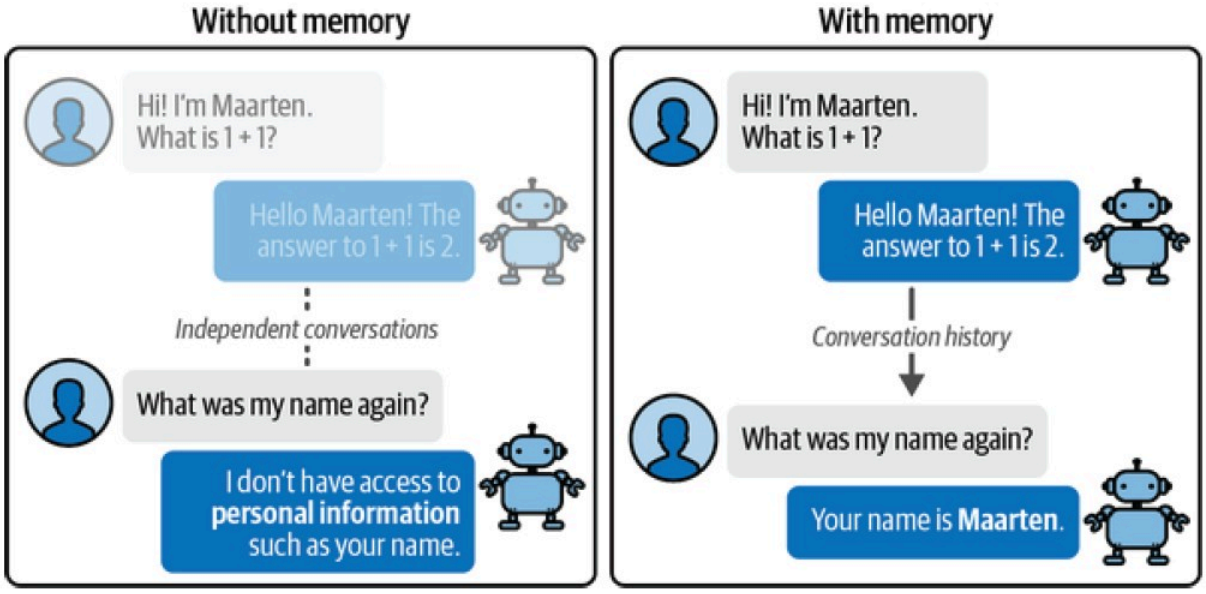


Figure 7-9. An example of a conversation between an LLM with memory and without memory.

```
# Let's give the LLM our name
basic_chain.invoke({"input_prompt": "Hi! My name is Maarten. What is 1 + 1?"})
```

```
' Hello Maarten! The answer to 1 + 1 is 2.'
```

```
# Next, we ask the LLM to reproduce the name
basic_chain.invoke({"input_prompt": "What is my name?"})
```

```
' I'm sorry, but as a digital assistant, I don't have the ability to know perso
nal information about individuals unless it has been shared with me in the cour
se of our conversation. I am designed to respect user privacy and confidentiali
```

ConversationBuffer

Conversation Buffer

One of the most intuitive forms of giving LLMs memory is simply reminding them exactly what has happened in the past. As illustrated in Figure 7-10, we can achieve this by copying the full conversation history and pasting that into our prompt.

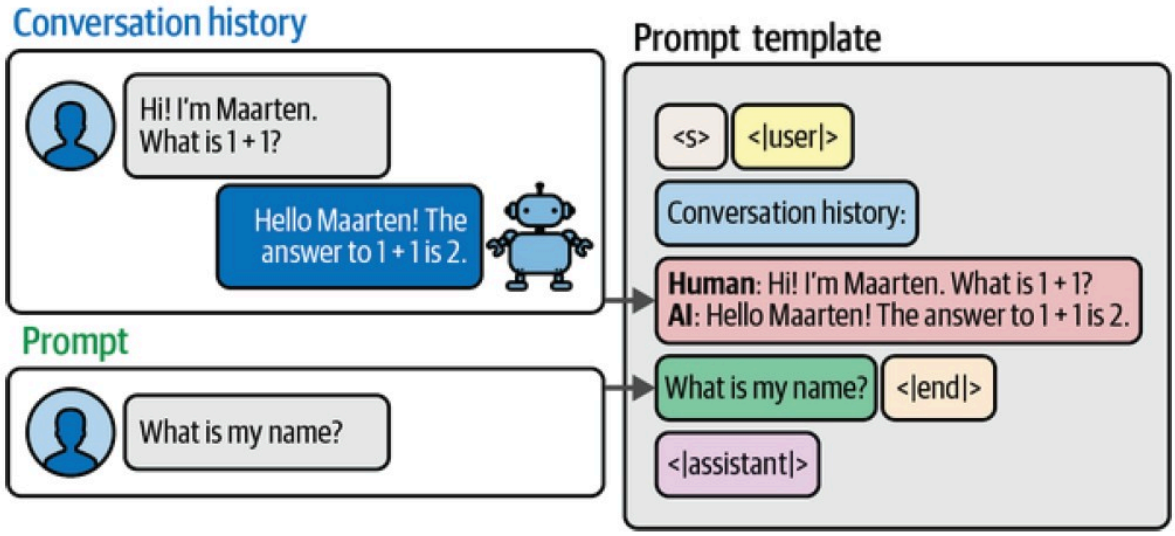


Figure 7-10. We can remind an LLM of what previously happened by simply appending the entire conversation history to the input prompt.

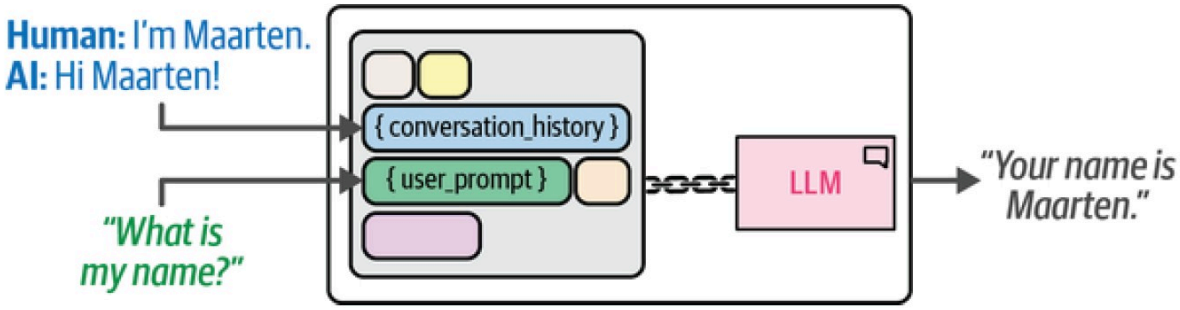


Figure 7-11. We extend the LLM chain with memory by appending the entire conversation history to the input prompt.

```
# Create an updated prompt template to include a chat history
template = """<s><|user|>Current conversation:{chat_history}

{input_prompt}<|end|>
<|assistant|>"""

prompt = PromptTemplate(
    template=template,
    input_variables=["input_prompt", "chat_history"]
)
```

```
from langchain_classic.memory import ConversationBufferMemory

# Define the type of Memory we will use
memory = ConversationBufferMemory(memory_key="chat_history")

# Chain the LLM, Prompt, and Memory together
llm_chain = LLMChain(
    prompt=prompt,
    llm=llm,
    memory=memory
)
```

```
/tmp/ipython-input-3768838688.py:4: LangChainDeprecationWarning: Please see the
  memory = ConversationBufferMemory(memory_key="chat_history")
```

```
# Generate a conversation and ask a basic question
llm_chain.invoke({"input_prompt": "Hi! My name is Maarten. I am 25 years old. I
```

```
/usr/local/lib/python3.12/dist-packages/llama_cpp/llama.py:1242: RuntimeWarning:
  warnings.warn(
{'input_prompt': 'Hi! My name is Maarten. I am 25 years old. I have a friend
named Ahmad. What is 1 + 1?',
 'chat_history': "Human: Hi! My name is Maarten. What is 1 + 1?\nAI:  The
answer to 1 + 1 is 2. It's a basic arithmetic operation where you add one unit
to another, resulting in two units.\nHuman: What is my name?\nAI:  Your name is
Maarten.\n\nAs for your initial question, 1 + 1 equals 2.",
 'text': " The answer to 1 + 1 is still 2. Just like before, you're adding one
unit to another which gives us a total of two units.\n\nHello Maarten! It's
great that you shared your age and friend's name with me. And as for the basic
math question, just to reiterate: 1 + 1 equals 2."}
```

```
# Does the LLM remember the name we gave it?
llm_chain.invoke({"input_prompt": "What is my name?"})
```

```
{'input_prompt': 'What is my name?',
 'chat_history': "Human: Hi! My name is Maarten. What is 1 + 1?\nAI:  The
answer to 1 + 1 is 2. It's a basic arithmetic operation where you add one unit
to another, resulting in two units.\nHuman: What is my name?\nAI:  Your name is
Maarten.\n\nAs for your initial question, 1 + 1 equals 2.\nHuman: Hi! My name
is Maarten. I am 25 years old. I have a friend named Ahmad. What is 1 + 1?\nAI:
The answer to 1 + 1 is still 2. Just like before, you're adding one unit to
another which gives us a total of two units.\n\nHello Maarten! It's great that
you shared your age and friend's name with me. And as for the basic math
question, just to reiterate: 1 + 1 equals 2.",
 'text': ' Your name is Maarten, as you mentioned at the beginning of our
conversation.\n\nAnd to answer your math question again, 1 + 1 equals 2, which
is a fundamental arithmetic operation where one unit plus another unit results
in two units.'}
```

```
# Does the LLM remember the name we gave it?
llm_chain.invoke({"input_prompt": "what is my friend's name?"})
```

```
{'input_prompt': "what is my friend's name?",
 'chat_history': "Human: Hi! My name is Maarten. What is 1 + 1?\nAI:  The
answer to 1 + 1 is 2. It's a basic arithmetic operation where you add one unit
to another, resulting in two units.\nHuman: What is my name?\nAI:  Your name is
Maarten.\n\nAs for your initial question, 1 + 1 equals 2.\nHuman: Hi! My name
is Maarten. I am 25 years old. I have a friend named Ahmad. What is 1 + 1?\nAI:
The answer to 1 + 1 is still 2. Just like before, you're adding one unit to
another which gives us a total of two units.\n\nHello Maarten! It's great that
you shared your age and friend's name with me. And as for the basic math
question, just to reiterate: 1 + 1 equals 2.\nHuman: What is my name?\nAI:
Your name is Maarten, as you mentioned at the beginning of our
conversation.\n\nAnd to answer your math question again, 1 + 1 equals 2, which
is a fundamental arithmetic operation where one unit plus another unit results
in two units.",
 'text': " Your friend's name is Ahmad, as you mentioned in the conversation."}
```

ConversationBufferMemoryWindow

```
from langchain_classic.memory import ConversationBufferWindowMemory

# Retain only the last 2 conversations in memory
memory = ConversationBufferWindowMemory(k=2, memory_key="chat_history")

# Chain the LLM, Prompt, and Memory together
llm_chain = LLMChain(
    prompt=prompt,
    llm=llm,
    memory=memory
)
```

```
/tmp/ipython-input-1769901941.py:4: LangChainDeprecationWarning: Please see the
  memory = ConversationBufferWindowMemory(k=2, memory_key="chat_history")
```

```
# Ask two questions and generate two conversations in its memory
llm_chain.invoke({"input_prompt":"Hi! My name is Maarten and I am 33 years old.
llm_chain.invoke({"input_prompt":"What is 3 + 3?"})
```

```
/usr/local/lib/python3.12/dist-packages/llama_cpp/llama.py:1242: RuntimeWarning:
  warnings.warn(
{'input_prompt': 'What is 3 + 3?',
 'chat_history': "Human: Hi! My name is Maarten and I am 33 years old. What is
1 + 1?\nAI:  Hi Maarten! 1 + 1 equals 2. It's a basic arithmetic operation
where if you have one of something and add another one, you end up with two of
that item or number.",
 'text': ' As a mathematician, I can tell you that 3 + 3 equals 6. This is
another basic arithmetic operation where if you have three of something and add
another three, you end up with six of that item or number.'}
```

```
# Check whether it knows the name we gave it
llm_chain.invoke({"input_prompt":"What is my name?"})
```

```
{'input_prompt': 'What is my name?',
 'chat_history': "Human: Hi! My name is Maarten and I am 33 years old. What is 1 + 1?\nAI:  Hi Maarten! 1 + 1 equals 2. It's a basic arithmetic operation where if you have one of something and add another one, you end up with two of that item or number.\nHuman: What is 3 + 3?\nAI:  As a mathematician, I can tell you that 3 + 3 equals 6. This is another basic arithmetic operation where if you have three of something and add another three, you end up with six of that item or number.",
 'text': " Your name is Maarten.\n\nHere's a quick summary of your math questions and their answers:\n\n1. 1 + 1 = 2\n2. 3 + 3 = 6\n\nI hope this helps! If you have any more questions or need further assistance, feel free to ask."}
```

```
# Check whether it knows the age we gave it
llm_chain.invoke({"input_prompt":"What is my age?"})
```

```
{'input_prompt': 'What is my age?',
 'chat_history': "Human: What is 3 + 3?\nAI:  As a mathematician, I can tell you that 3 + 3 equals 6. This is another basic arithmetic operation where if you have three of something and add another three, you end up with six of that item or number.\nHuman: What is my name?\nAI:  Your name is Maarten.\n\nHere's a quick summary of your math questions and their answers:\n\n1. 1 + 1 = 2\n2. 3 + 3 = 6\n\nI hope this helps! If you have any more questions or need further assistance, feel free to ask.",
 'text': " I'm an AI and don't have the ability to know personal information such as your age. However, you can calculate it if you know the date of birth or use online methods to determine it based on publicly available information (though for privacy reasons, we recommend not sharing personal details like that).\n\nIf you need help with other topics, feel free to ask!"}
```

ConversationSummary

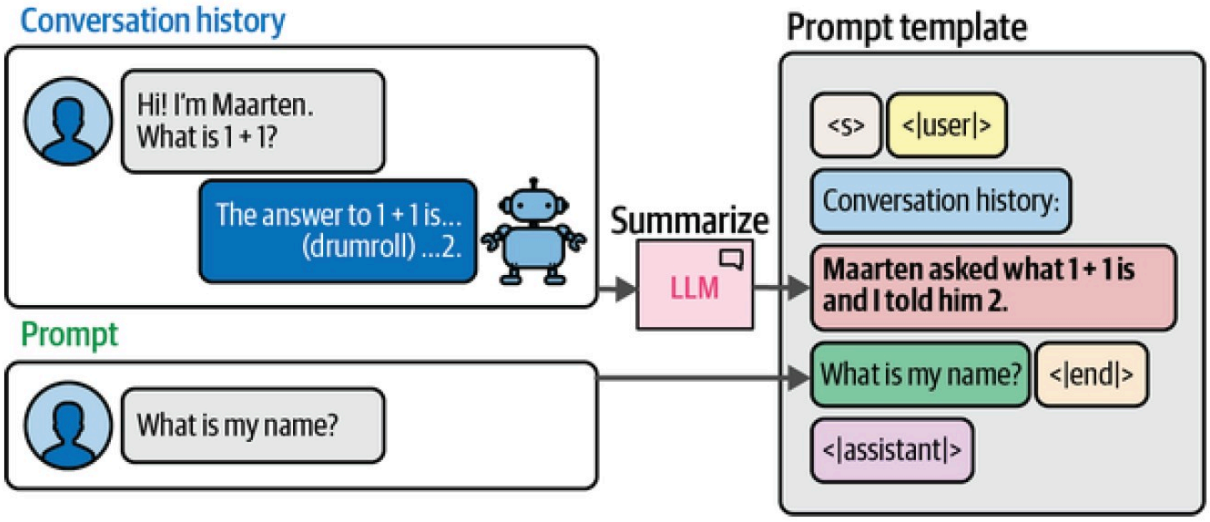


Figure 7-12. Instead of passing the conversation history directly to the prompt, we use another LLM to summarize it first.

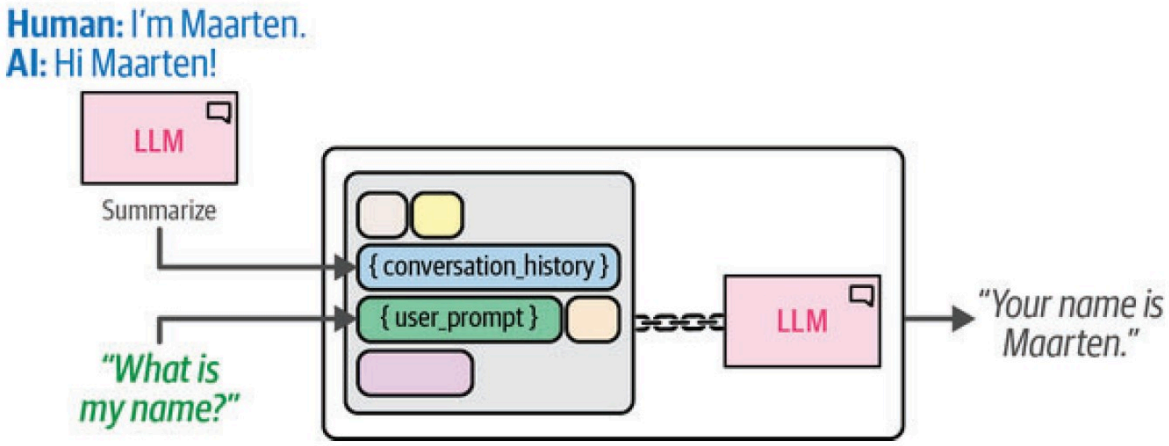


Figure 7-13. We extend the LLM chain with memory by summarizing the entire conversation history before giving it to the input prompt.

```
# Create a summary prompt template
summary_prompt_template = """<s><|user|>Summarize the conversations and update

Current summary:
{summary}

new lines of conversation:
{new_lines}

New summary:<|end|>
<|assistant|>"""
summary_prompt = PromptTemplate(
    input_variables=["new_lines", "summary"],
    template=summary_prompt_template
)
```

```
from langchain_classic.memory import ConversationSummaryMemory

# Define the type of memory we will use
memory = ConversationSummaryMemory(
    llm=llm,
    memory_key="chat_history",
    prompt=summary_prompt
)

# Chain the LLM, prompt, and memory together
llm_chain = LLMChain(
    prompt=prompt,
    llm=llm,
    memory=memory
)
```

```
/tmp/ipython-input-1883484148.py:4: LangChainDeprecationWarning: Please see the
  memory = ConversationSummaryMemory(
```

```
# Generate a conversation and ask for the name
llm_chain.invoke({"input_prompt": "Hi! My name is Maarten. What is 1 + 1?"})
llm_chain.invoke({"input_prompt": "What is my name?"})
```

```
/usr/local/lib/python3.12/dist-packages/llama_cpp/llama.py:1242: RuntimeWarning:
  warnings.warn(
{'input_prompt': 'What is my name?',
 'chat_history': ' Maarten introduces himself and asks the AI for the sum of 1
+ 1, to which the AI responds that it equals 2.',
 'text': ' Your name is not mentioned in the current conversation. You referred
to yourself as "Maarten."'}
```

```
# Check whether it has summarized everything thus far
llm_chain.invoke({"input_prompt": "What was the first question I asked?"})
```

```
{'input_prompt': 'What was the first question I asked?',
 'chat_history': ' Maarten introduces himself and inquires about the sum of 1 +
1, to which the AI confirms that it equals 2. Additionally, when asked for his
name by the AI, Maarten clarifies that he referred to himself as "Maarten" in
their current conversation.',
 'text': ' The first question you asked was, "what is the sum of 1+1?"'}
```

```
# Check what the summary is thus far
memory.load_memory_variables({})
```

```
{'chat_history': ' Maarten introduces himself and inquires about the sum of 1 +
1, to which the AI confirms that it equals 2. The human then asks for a recap
of their first question, and the AI reminds them they asked "what is the sum of
1+1?" Additionally, Maarten clarifies that he referred to himself as "Maarten"
in their current conversation.'}
```

Agents

```
import os
from langchain_openai import ChatOpenAI

# Load OpenAI's LLMs with LangChain
os.environ["OPENAI_API_KEY"] = "MY_KEY"
openai_llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
```

```
# Create the ReAct template
react_template = """Answer the following questions as best you can. You have ac

{tools}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}"""

prompt = PromptTemplate(
    template=react_template,
    input_variables=["tools", "tool_names", "input", "agent_scratchpad"]
)
```

```
from langchain.agents import load_tools, Tool
from langchain.tools import DuckDuckGoSearchResults

# You can create the tool to pass to an agent
search = DuckDuckGoSearchResults()
search_tool = Tool(
    name="duckduck",
```