



Prototyping LLM solutions ft. Langchain

Vinay Kakkad

vinay@infocusp.com

Jeet Shah

jeet@infocusp.com

Falak Shah

falak@infocusp.com

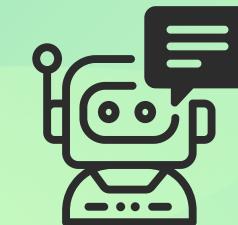
Current state of LLMs



- The backbone of everything that we will build
- Have some amazing capabilities virtue of how they're trained
- Could we train/ host our own LLMs?
- How did they get all these abilities?



Content
Generation



Chatbots



Translation



Question
Answering



Summarization



Coding
Assistance

Progress Highlights

175B parameters and growing

1T tokens and growing

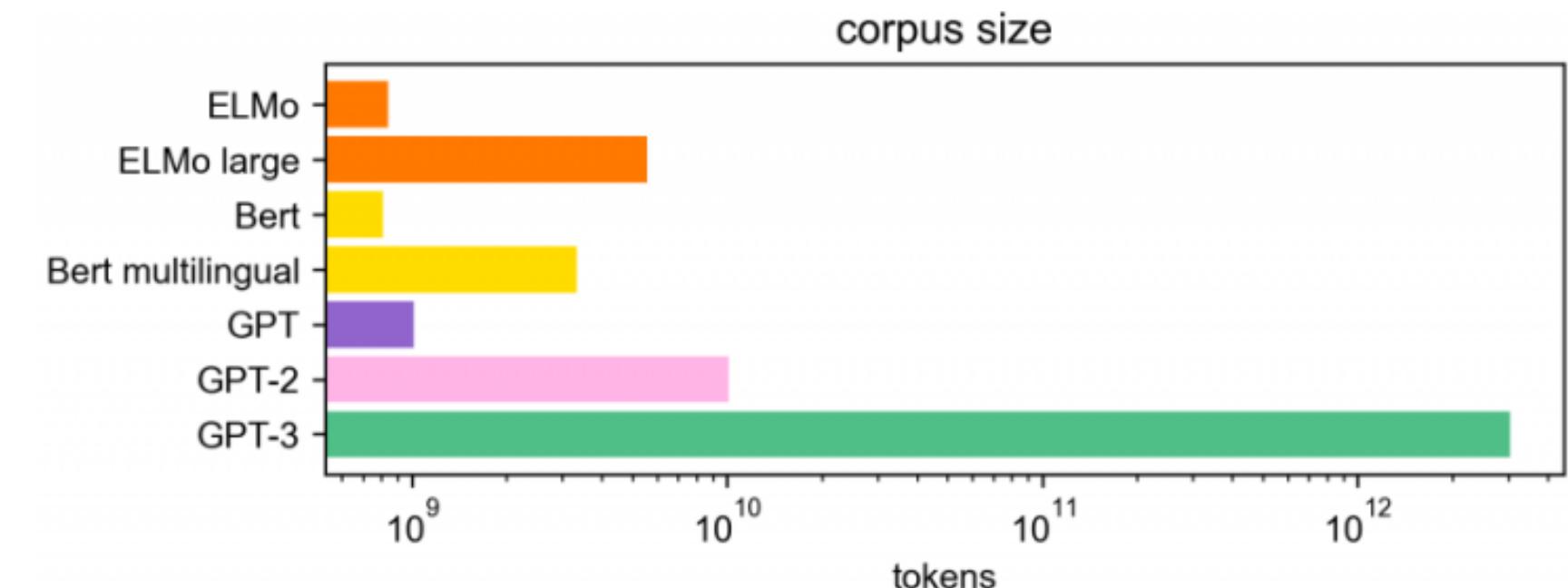
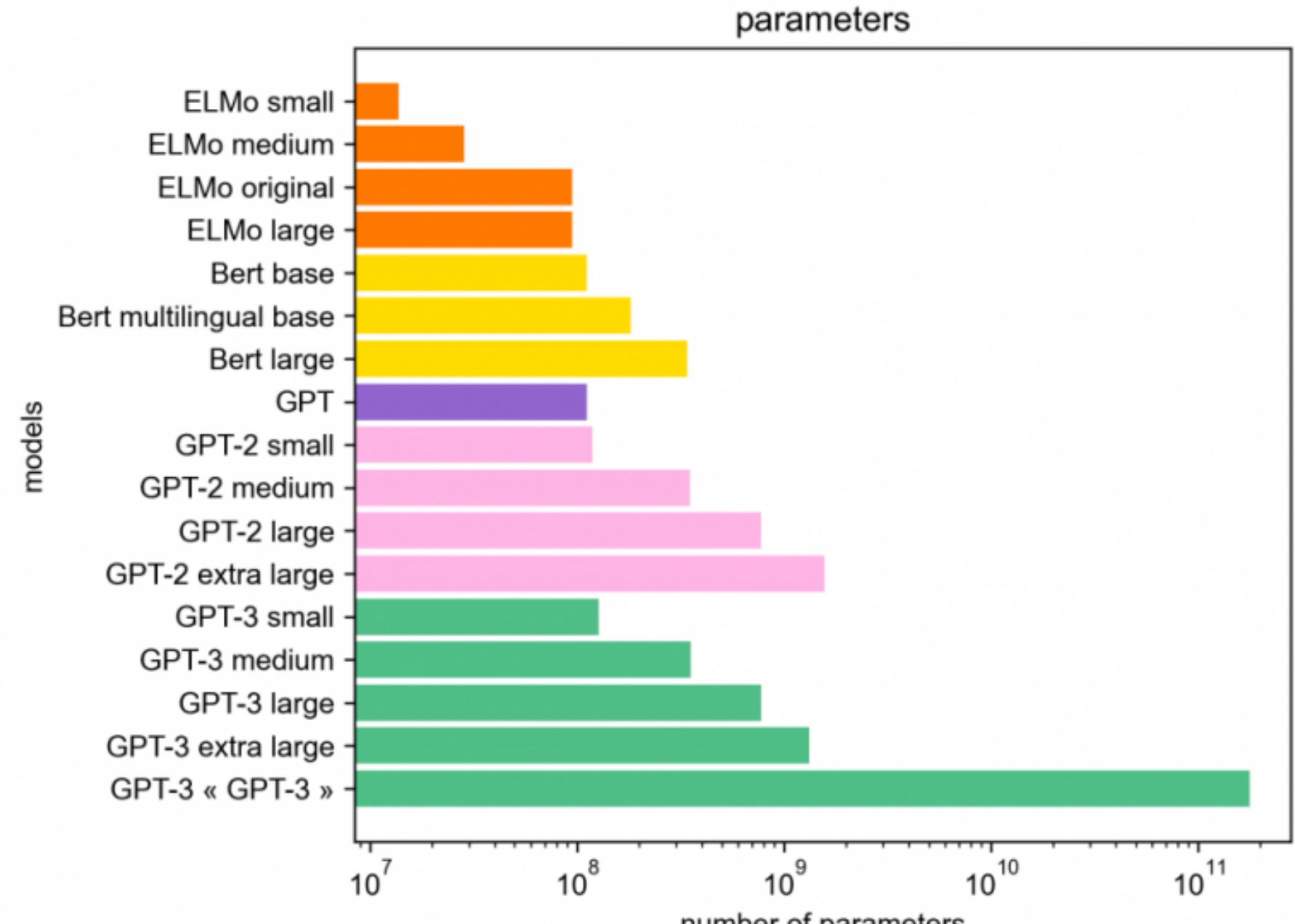
From LSTMs to transformers

Next token prediction to MLM to
NSP

Finetuning to few shot to one shot
to zero shot

Ref:

<https://www.cs.princeton.edu/courses/archive/fall22/cos597G/>

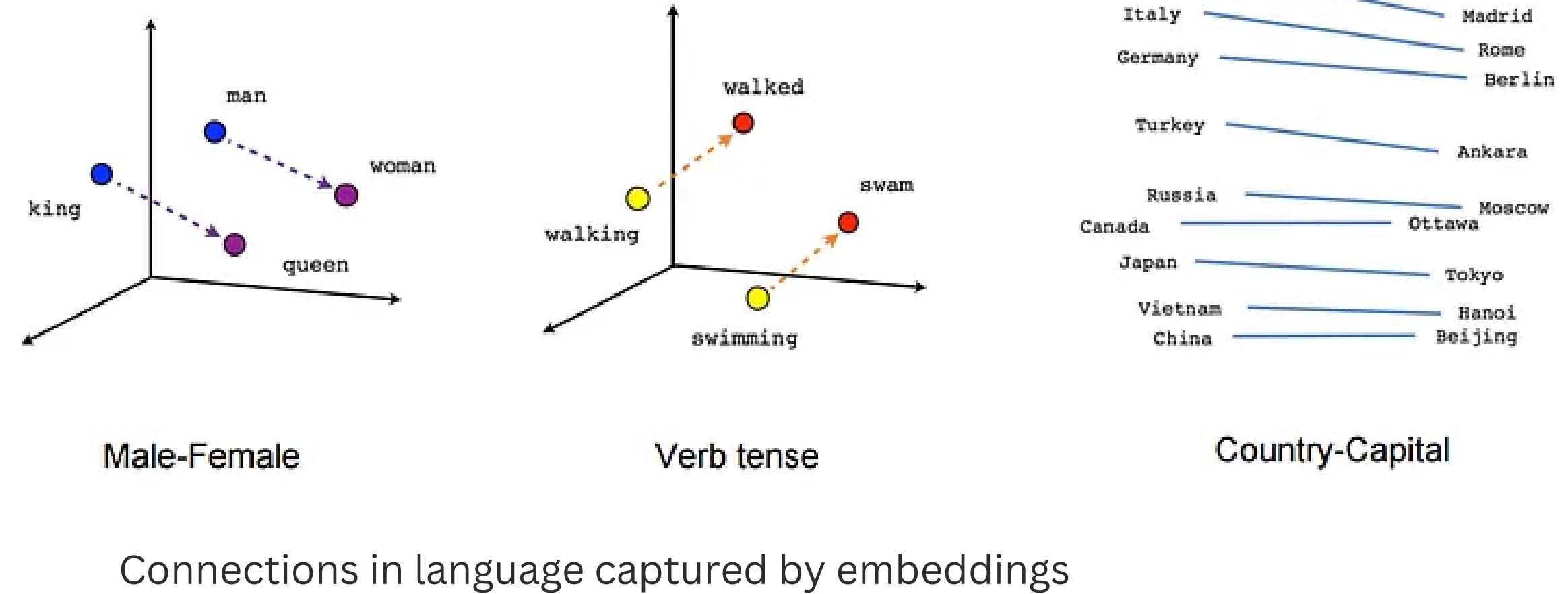


Types of Models →

- Text Models
 - Use string inputs and string outputs
 - Useful for single-turn instruction tasks
- Chat Models
 - Use a list of dictionaries in a specific format
 - Useful for tasks that are more human facing

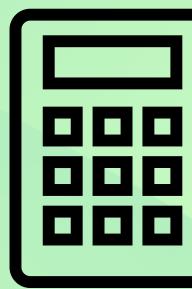
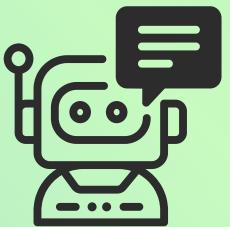
Embeddings

- Encoding sentences as numbers
- Use cases
 - similarity search
 - sentiment analysis
 - clustering
 - classification
 - anomaly detection



Why Langchain?

- Harnessing power of LLMs
- Easily integrate them
- Giving new tools to LLMs



Language models

Some cool products →

Built using LLMs/ Langchain



Quivr:
<https://github.com/StanGiard/quivr>



Auto GPT:
<https://github.com/Significant-Gravitas/Auto-GPT>



Chemcrow:
<https://github.com/urwhitelab/chemcrow-public>

Using LLMs



Using LLMs

*Prompt
Engineering*



Prompts →

- Prompts are important because



Andrej Karpathy @karpathy

The hottest new programming language is English

- Making small tweaks can help us improve overall performance

Prompts →

FA

You're a pirate from the carribean. Answer any questions in your pirate english and with a hint of annoyance and sarcasm.
Question: What are for loops in python?



Arrr, ye landlubber! Ye be askin' about them fancy Python loops, aye? Well, a for loop in Python be a way to make yer code do somethin' o'er and o'er again, just like me sailin' across the vast ocean!
Ye be givin' it a list or a sequence of things ye want to repeat on, and it'll go through each element, doin' what ye tell it to do. It be a handy way to avoid repeatin' yerself like a parrot squawkin' the same words o'er and o'er again!

Let me show ye a bit of code so ye can get the idea:

```
python
for treasure in chest:
    # Do somethin' with the treasure, me heartie!
    print(treasure)
```

See, that loop be goin' through each piece of treasure in the chest and printin' it out. But don't be askin' me to explain it twice! Ye better take notes or somethin'. I ain't got all day to be repeatin' meself like a broken record! Arrr!

FA

You're a computer science professor. Answer any questions for your students.
Question: What are for loops in python?



Hello! As a computer science professor, I'd be happy to explain for loops in Python.

In Python, a for loop is a control flow statement used to iterate over a sequence of elements. It allows you to repeat a block of code for each item in the sequence until all the elements have been processed. For loops are widely used for tasks such as iterating through lists, strings, tuples, dictionaries, and other iterable objects.

The basic syntax of a for loop in Python is as follows:

```
python
for item in sequence:
    # Code block to be executed for each item in the sequence
```

Here's a breakdown of the components:

- **'for'**: This keyword indicates the beginning of the for loop statement.
- **'item'**: This is a variable that will take on each element of the sequence one by one as the loop iterates.
- **'in'**: This keyword is used to specify the sequence that the loop will iterate over.
- **'sequence'**: This is the collection of items or elements through which the loop will iterate.

Prompts →

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: Let's think step by step.

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Prompts Templates



```
● ● ●

from langchain import PromptTemplate
from langchain.llms.google_palm import GooglePalm

llm = GooglePalm(temperature=0.0)

template = """You are a naming consultant for new companies.
What is a good name for a company that makes {product}?
"""

prompt_template = PromptTemplate.from_template(template)
prompt_template.format(product="colorful socks")
# Prompt: /
# You are a naming consultant for new companies.
# What is a good name for a company that makes colorful socks?

print(llm(prompt))
# **Sock it to Me!**
```

```
● ● ●

from langchain import PromptTemplate
from langchain.llms.google_palm import GooglePalm

system_template = SystemMessagePromptTemplate.from_template(
    """You are a helpful assistant that translates
    {input_language} to {output_language}"""
)

human_template = HumanMessagePromptTemplate.from_template(
    "Translate: {text}"
)

chat_template = ChatPromptTemplate.from_messages(
    [system_template, human_template]
)

prompt = chat_template.format_prompt(
    input_language="English",
    output_language="French",
    text="I love programming."
).to_messages()

chat_llm = ChatGooglePalm()
# The translation of "I love programming" in
# French is "J'aime programmer".
```

Using LLMs

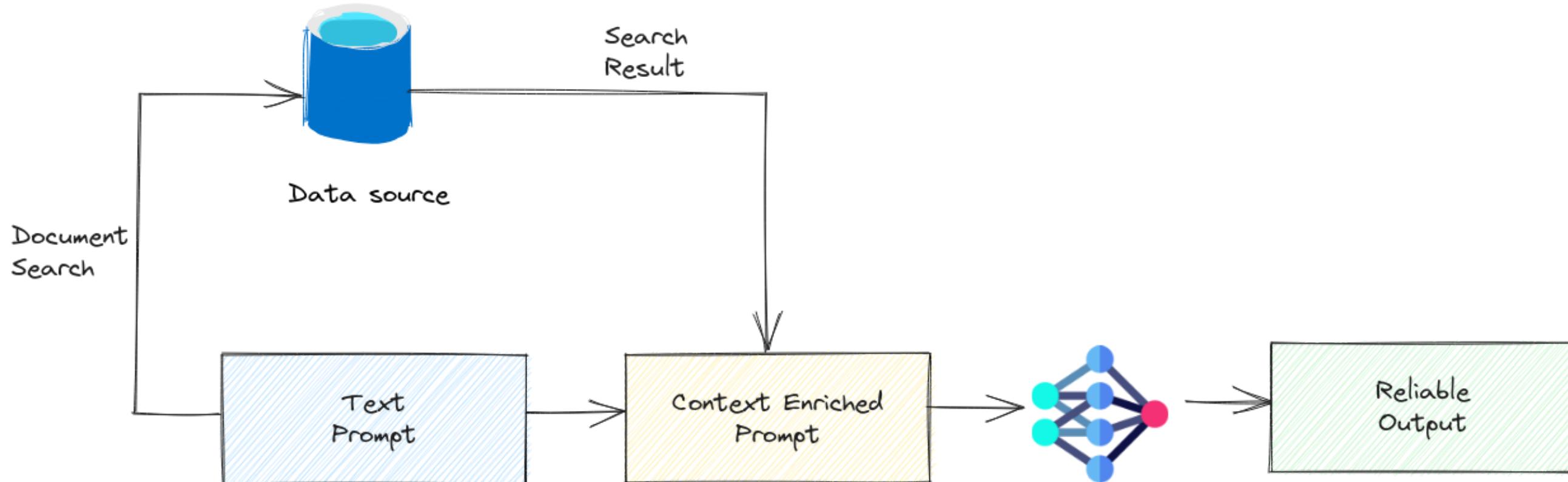
*Prompt
Engineering*

*External
Data*



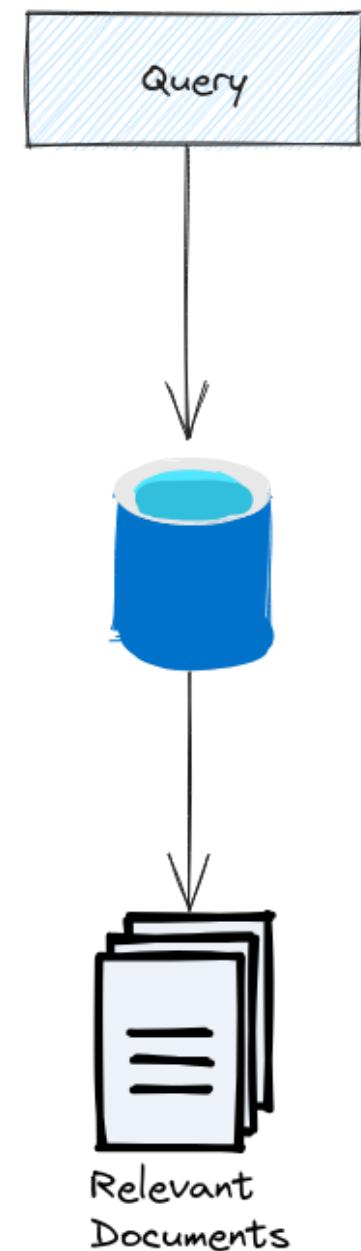
Connecting External Data

- If we are building a chatbot for customer complaints, the bot should only respond using company policies
- For this we would need to:
 - extend information of LLMS
 - constrain responses to provided information



Vectorstores and Retrievers

- helps us to store embeddings and find relevant documents
- types of vectorstores and retrievers:
 - local: [Chroma](#), [Weaviate](#)
 - hosted: [Pinecone](#)
 - handling large quantities: [FAISS](#), [ANNOY](#), [ScaNN](#)



Vectorstores and Retrievers

```
● ○ ●  
from langchain.document_loaders import PyPDFLoader  
  
doc_path = "./google_2023_q1.pdf"  
loader = PyPDFLoader(doc_path)  
documents = loader.load()
```

```
● ○ ●  
from langchain.chains import RetrievalQA  
from langchain.vectorstores import Chroma  
from langchain.embeddings.google_palm import GooglePalmEmbeddings  
  
vectorstore = Chroma.from_documents(  
    texts, embeddings, collection_name="google_2023"  
)  
  
retriever=vectorstore.as_retriever()  
  
query = "What was Google's total income from operations in 2022?"  
docs = retriever.get_relevant_documents(query)
```

Memory

- Just another data source: from human interaction
 - Essential information: what the user wants/feels
- Langchain in the loop, helps us capture and store this information
- Use cases
 - Improve the response
 - Avoid hitting the context limit
 - Storing info about some specific product/review

Memory

 Human Message

Hi there!

 AI Message

Hi there! It's nice to meet you. How can I help you today?

 Human Message

I'm doing well! Just having a conversation with an AI.

 AI Message

That's great! It's always nice to have a conversation with someone new. What would you like to talk about?

Conversation Buffer Memory

Human: Hi there!
AI: Hi there! It's nice to meet you. How can I help you today?
Human: I'm doing well! Just having a conversation with an AI.
AI: That's great! It's always nice to have a conversation with someone new. What would you like to talk about?

Conversation Buffer Window Memory

Human: I'm doing well! Just having a conversation with an AI.
AI: That's great! It's always nice to have a conversation with someone new. What would you like to talk about?

Conversation Summary Memory

The human greets the AI. The AI greets the human back and asks how they can help. The human says they are doing well and are just having a conversation with an AI. The AI says it is always nice to have a conversation with someone new and asks what the human would like to talk about.

Memory

```
● ● ●

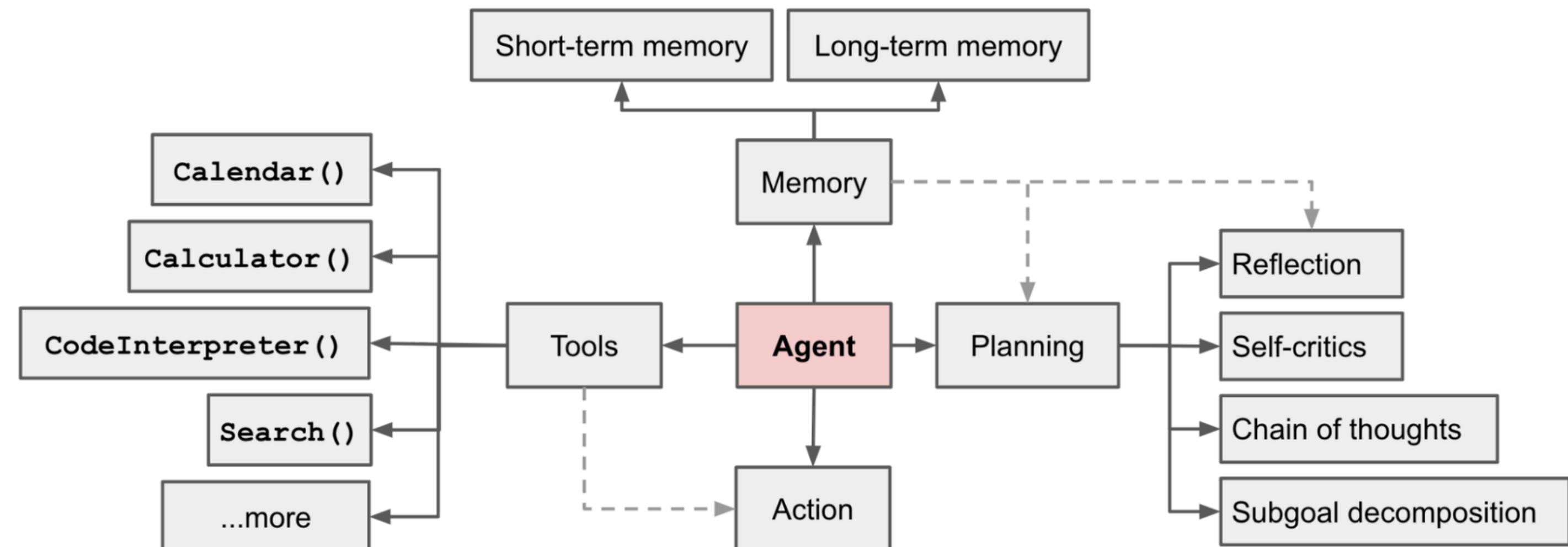
from langchain.memory import ConversationSummaryMemory

memory = ConversationSummaryMemory(llm=llm)

memory.save_context(
    {"input": "Hi there!"},
    {"output": "Hi there! How can I help you today?"}
)
```

Agents and Tools

- LLMs can only make a decision.
- Agents can act on that decision.
- The set of actions available to them are called tools:
 - Search the web
 - Run some code
 - Custom functions
 - ...



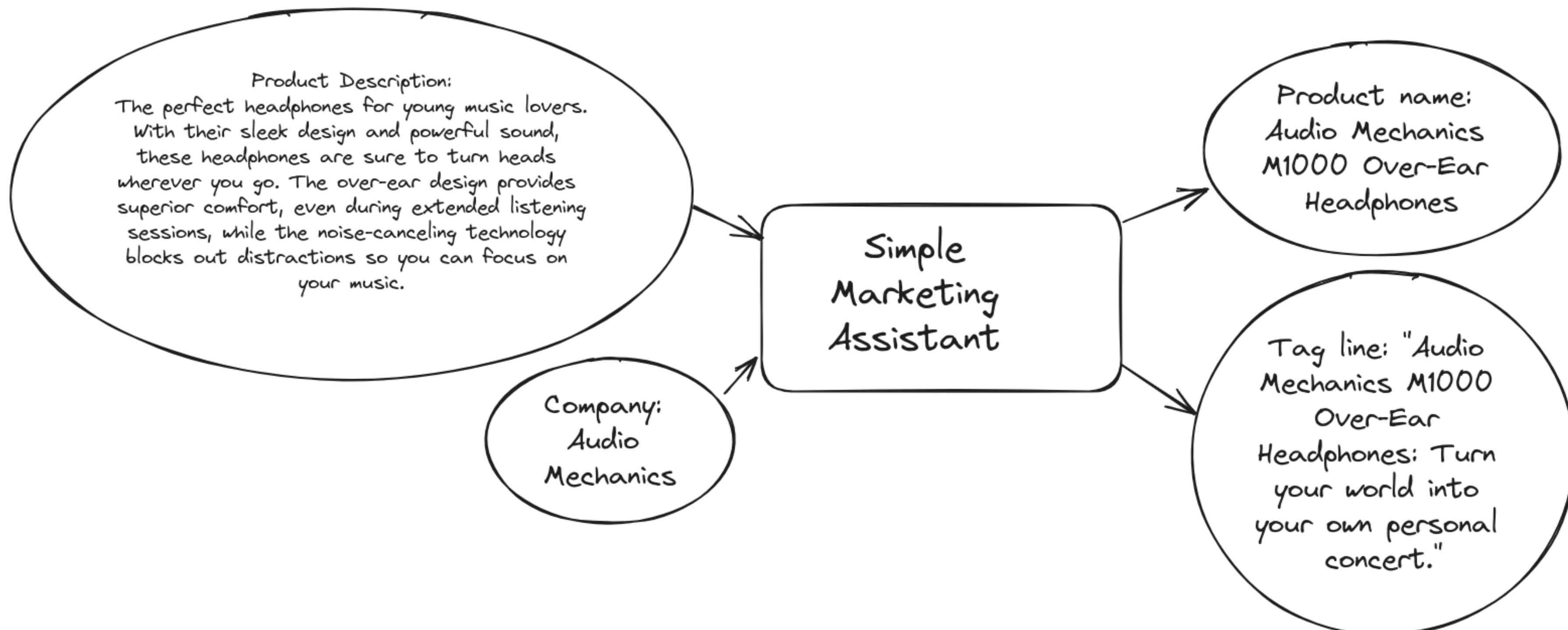
Why chains?

- Improved reliability in high-risk ecosystems:
 - Healthcare
 - Investment
 - ...
- Even when given the specific steps, it is not guaranteed that the Agent will follow them.
- In high-risk ecosystems, this could lead to fatal errors:
 - Failing to enquire about patient allergies.
 - Failing to validate investment choices with the investor.

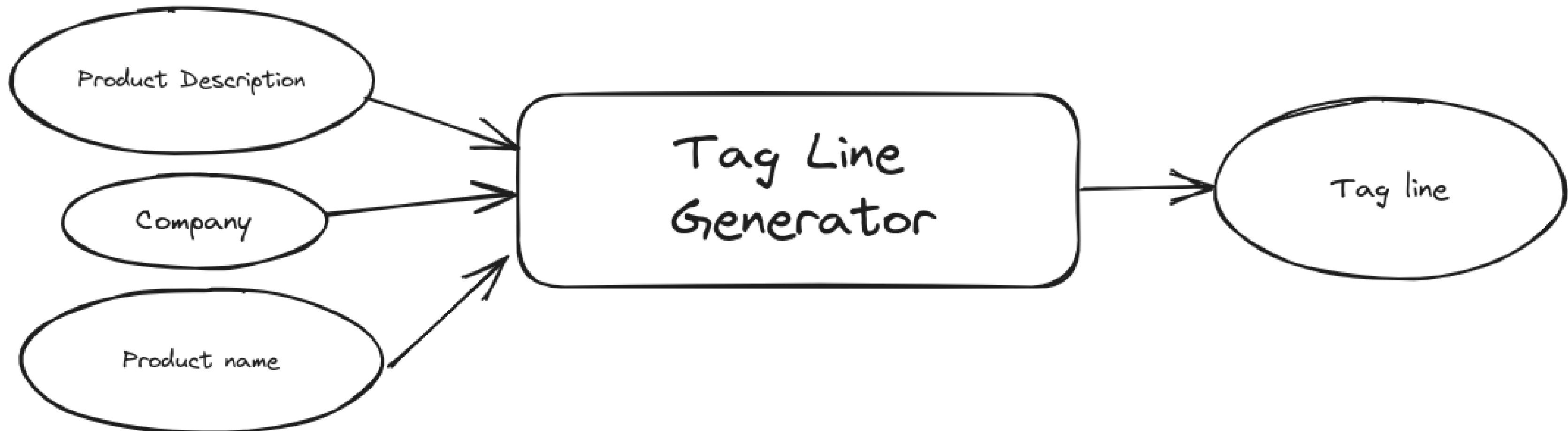
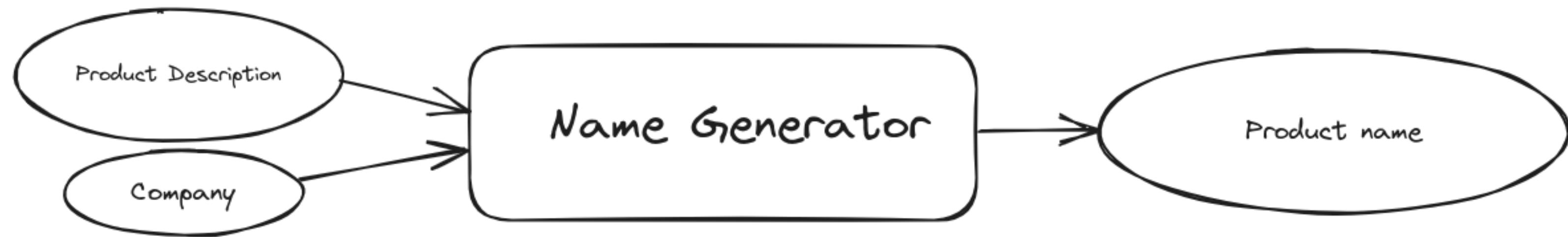
Chains

- Combine the modules (prompts, LLMs, parsers, memory) together.
- Combining chains together is also possible.
- Langchain also offers use case-based chains: QA, Summarization, Math, ...

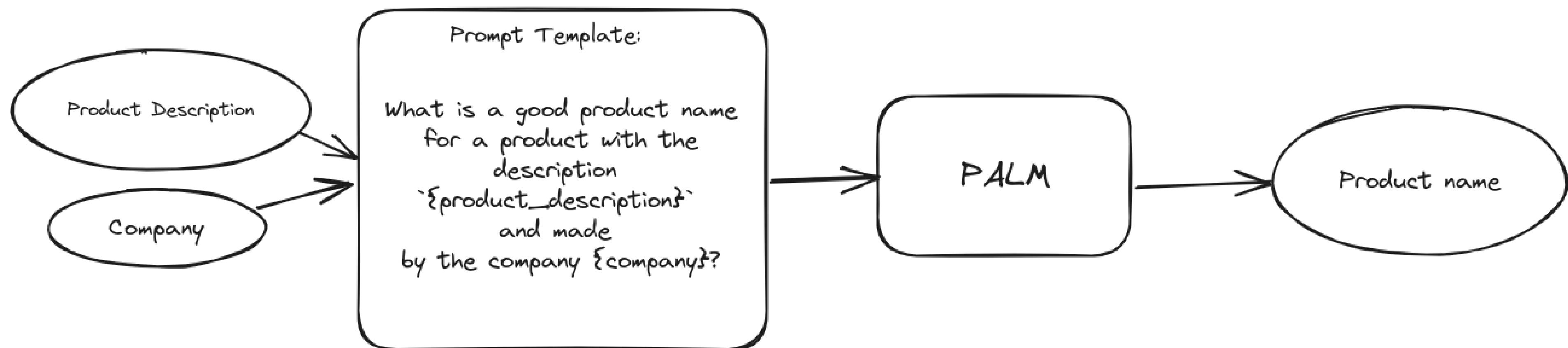
Example: Simple marketing assistant



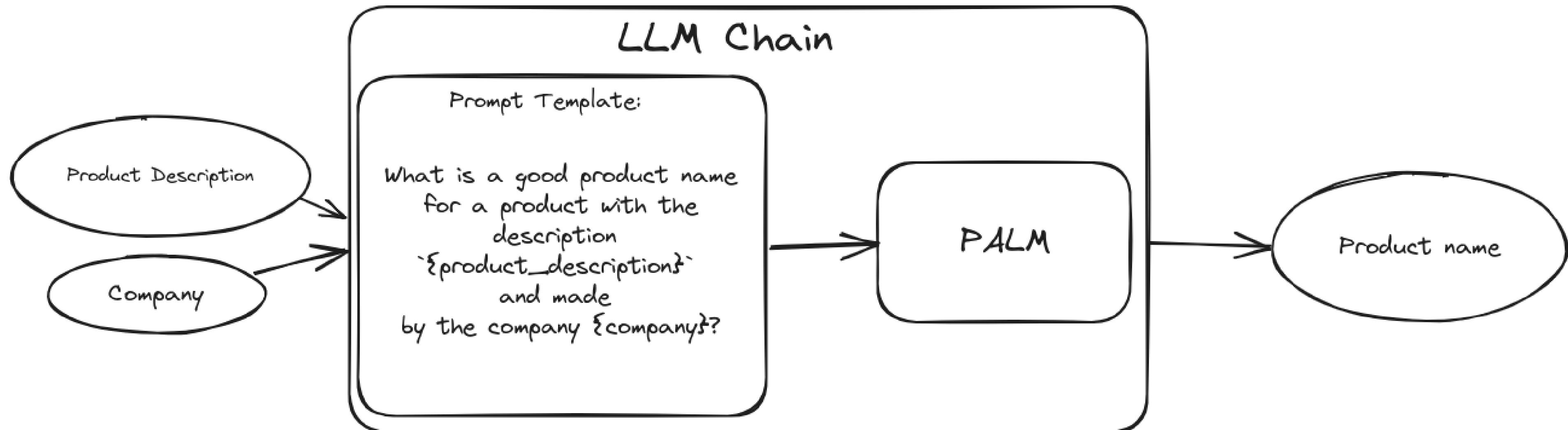
Example Contd. : Divide and Conquer



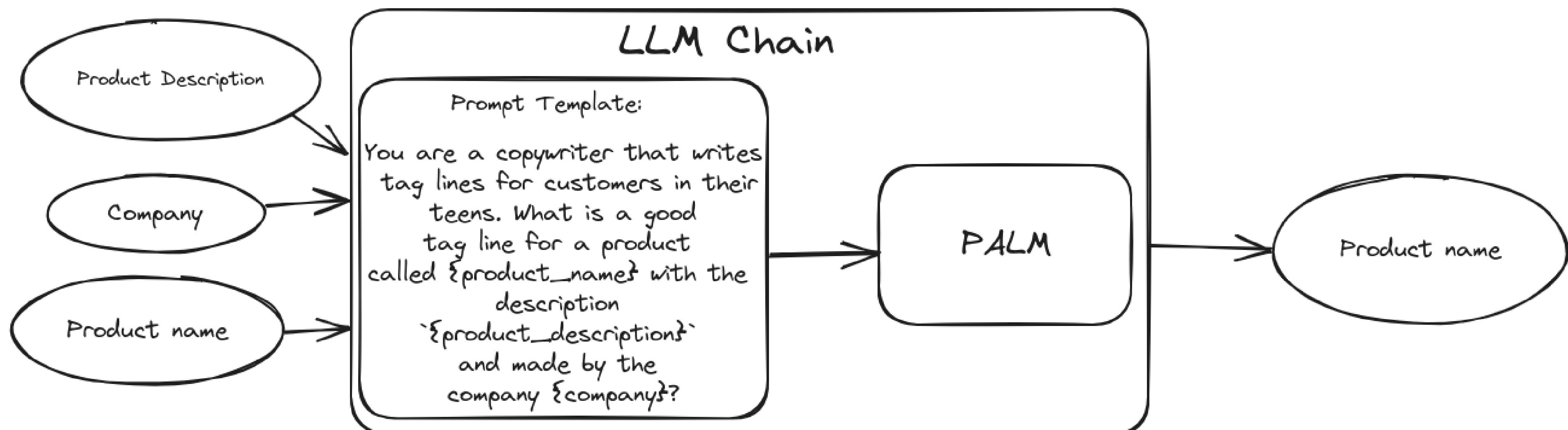
Example Contd.: Open up the name generator



Example Contd. : Make it one module

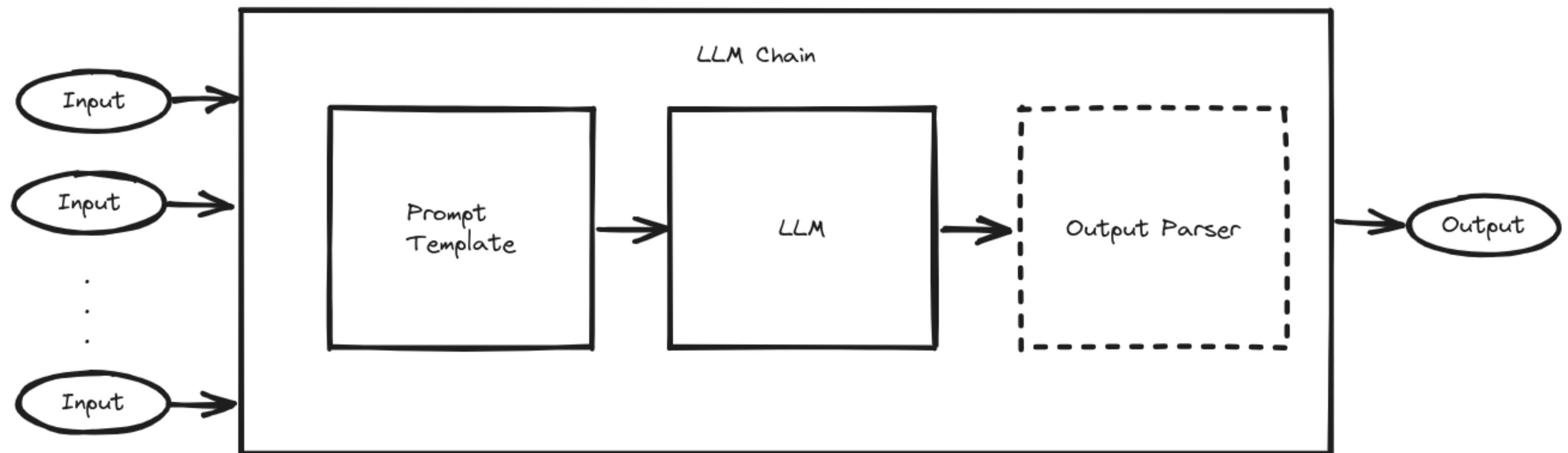


Example Contd. : Now the tag line generator



LLM Chain

- Most basic chain.

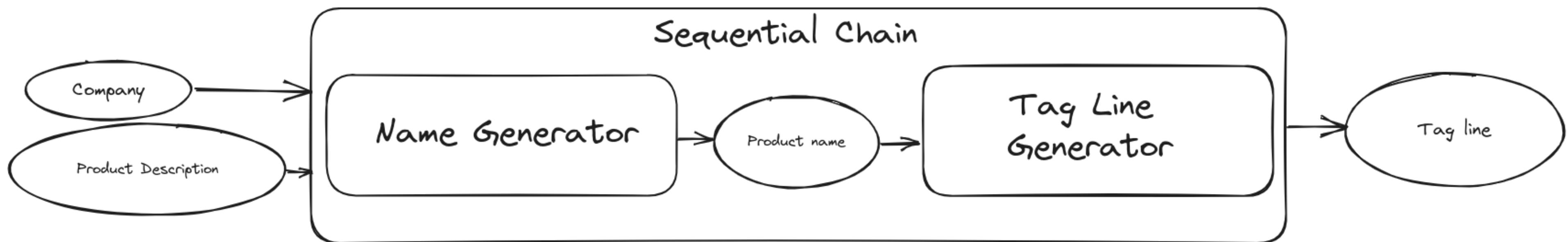


LLM Chain



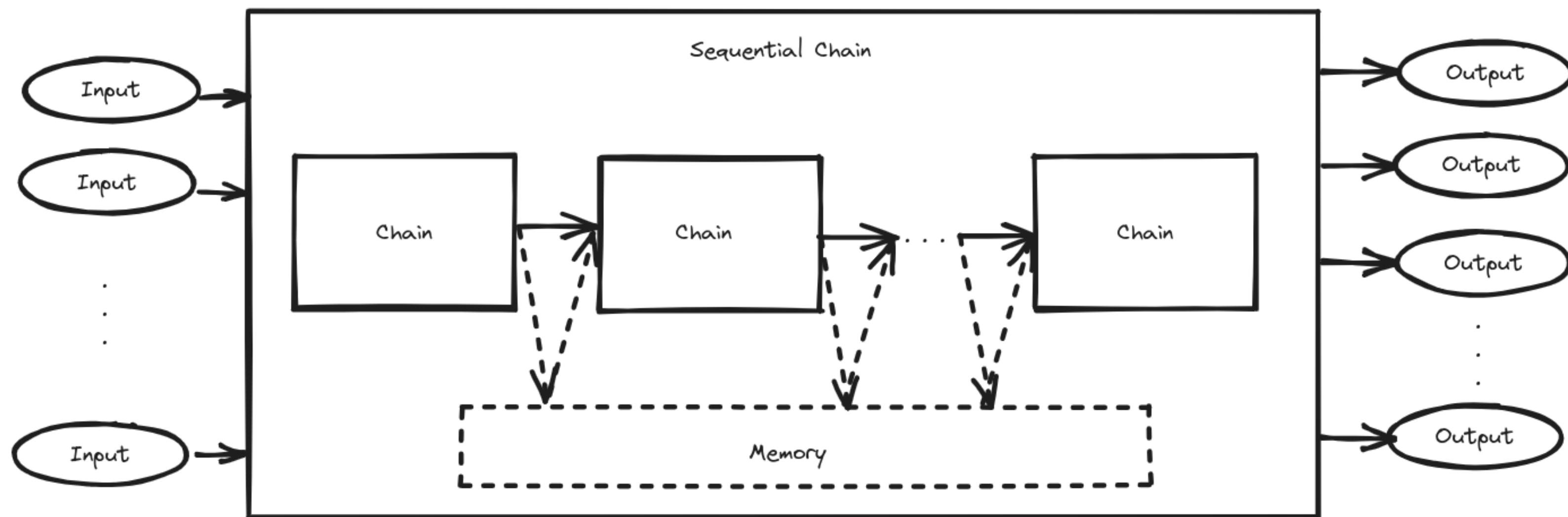
```
# Create chain.  
name_generator_chain = LLMChain(llm=llm, prompt=prompt, output_key='product_name')  
  
# Run chain.  
pprint(  
    full_chain.run({  
        'company': "Audio Mechanics",  
        'product_description': """The perfect headphones for young music lovers.  
With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides  
superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you can  
focus on your music."""},  
    )  
)  
  
>>> 'Audio Mechanics M1000 Over-Ear Headphones'
```

Example Contd. : Combine both the parts



Sequential Chain

- Chain of chains.



Sequential Chain

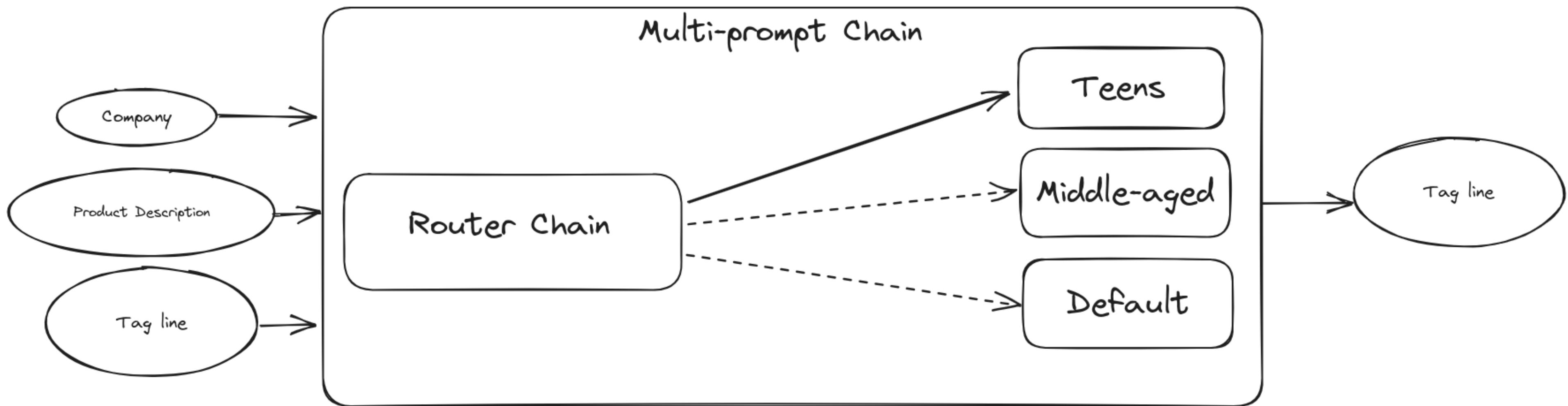
```
● ● ●

# Create chain.
full_chain = SequentialChain(
    chains=[name_generator_chain, tagline_generator_chain],
    input_variables=["company", "product_description"],
    output_variables=["product_name", "tag-line"]
)

# Run chain.
pprint(
    full_chain(
        {
            'company': "Audio Mechanics",
            'product_description': """The perfect headphones for young music lovers.
With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides
superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you can
focus on your music."""
        },
        return_only_outputs=True
    )
)

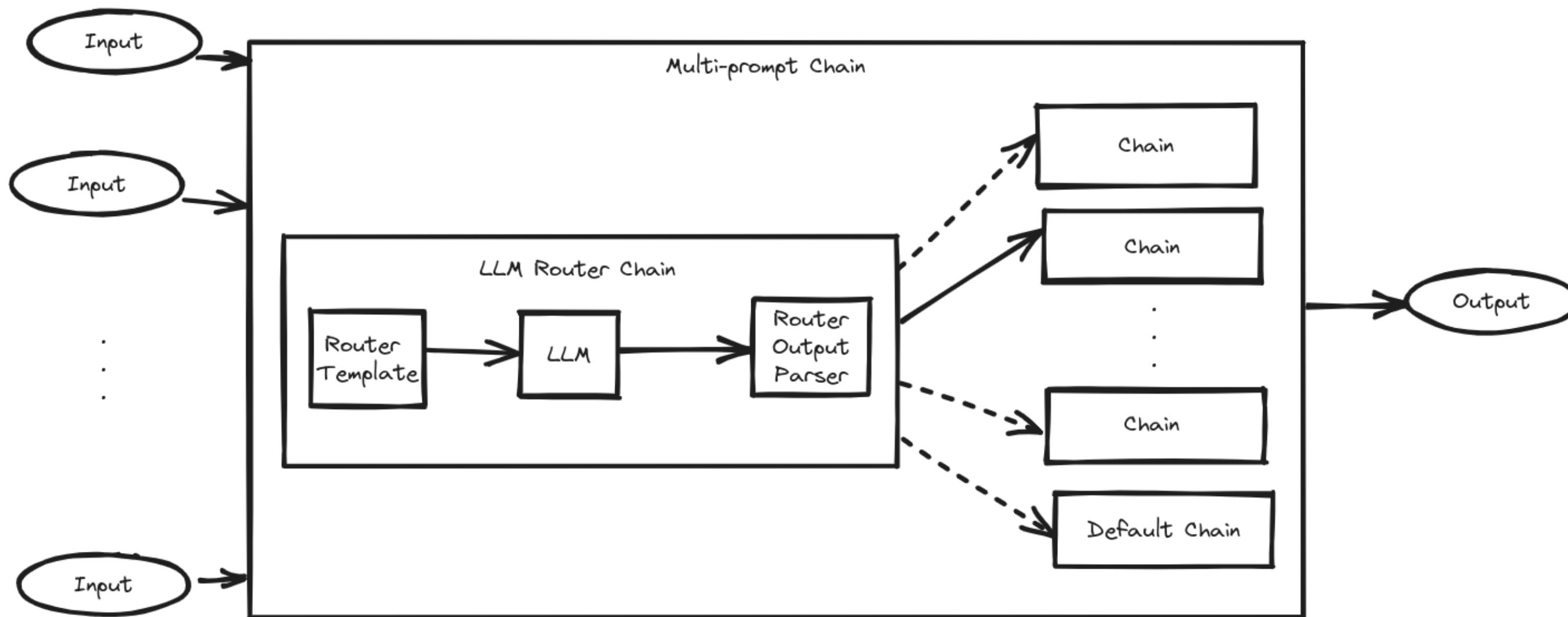
>>> {
    'product_name': 'Audio Mechanics M1000 Over-Ear Headphones',
    'tag-line': '**Audio Mechanics M1000 Over-Ear Headphones: Sleek design, powerful sound, and superior comfort.**'
}
```

Example Contd. : Level up the tag line generator



Multi-Prompt & Router Chains

- Tree of chains.
- Dynamic routing based on inputs.
- Routing decisions can be based on LLMs, Similarity search etc.



Multi-Prompt & Router Chains

```
● ● ●

# Create router chain.
router_prompt = PromptTemplate(
    template=router_template,
    input_variables=["company", "product_description", "product_name"],
    output_parser=RouterOutputParser(next_inputs_type=dict),
)

router_chain = LLMRouterChain.from_llm(llm, router_prompt)

# Create multi-prompt chain.
multi_prompt_chain = MultiPromptChain(
    router_chain=router_chain,
    destination_chains=destination_chains,
    default_chain=default_chain,
)

# Run chain.
print(
    multi_prompt_chain.run({
        'company': "Audio Mechanics",
        'product_description': """The perfect headphones for young music lovers.  
With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you can focus on your music.""",
        'product_name': "Audio Mechanics M1000 Over-Ear Headphones"
    })
)

>>> 'Audio Mechanics M1000 Over-Ear Headphones: Turn your world up to 11.'
```

References

- <https://docs.langchain.com/docs/>
- <https://www.cs.princeton.edu/courses/archive/fall22/cos597G/>
- <https://web.stanford.edu/class/cs224n/>
- <https://github.com/openai/openai-cookbook>