



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

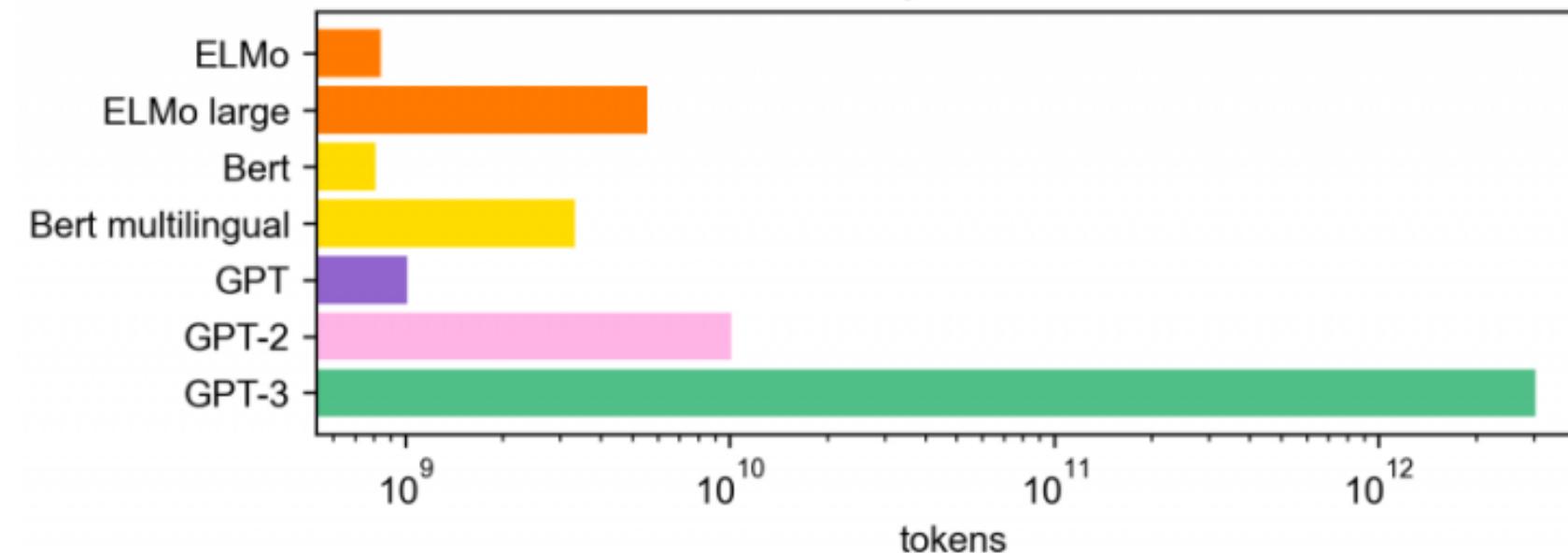
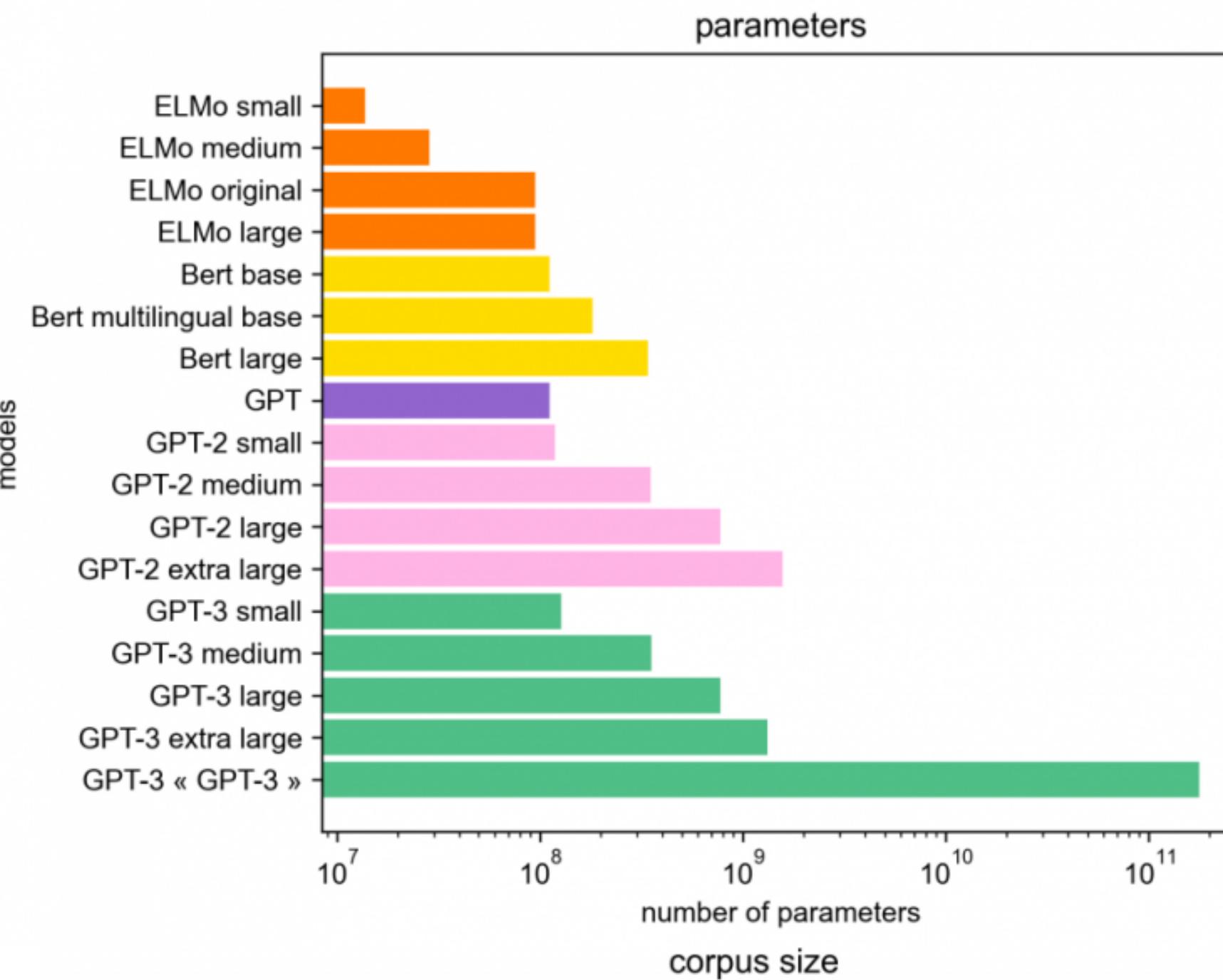
From LSTMs to transformers

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/>

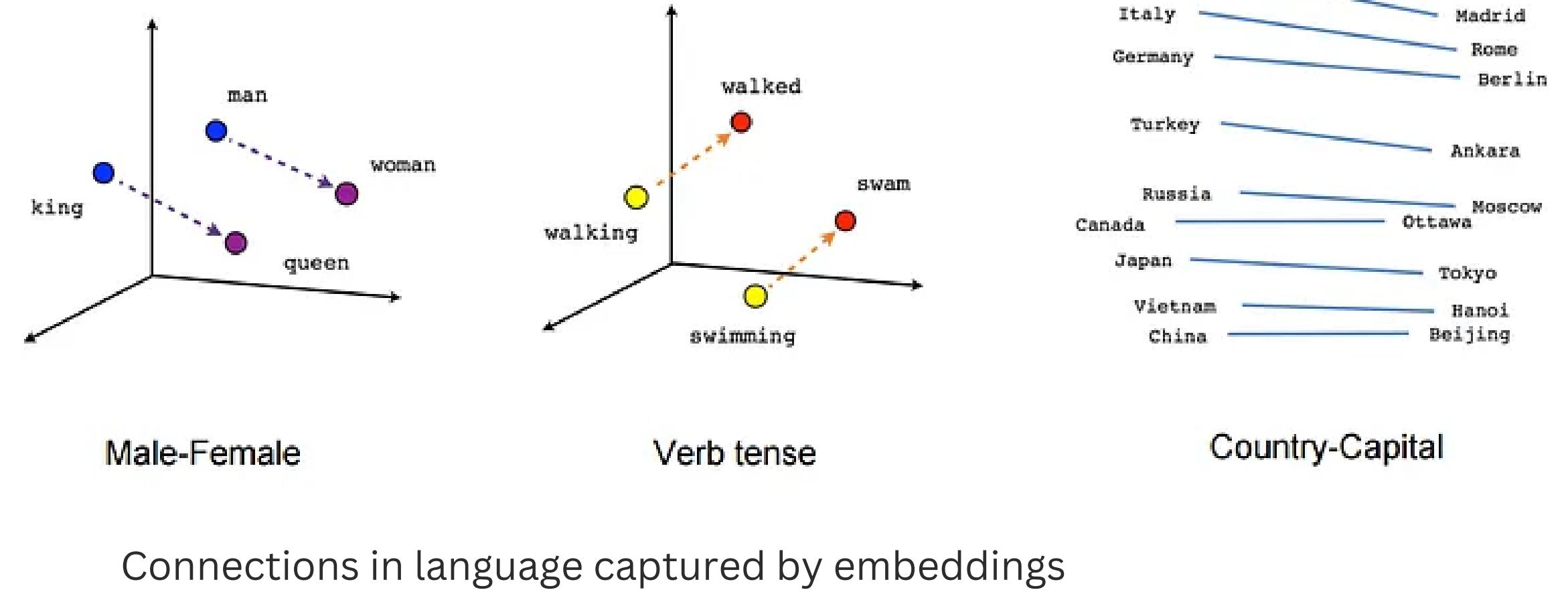


Types of Models

- 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

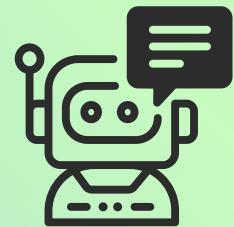
Embeddings

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

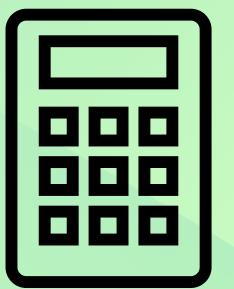


Why Langchain?

- 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/StanGillard/quivr>



Auto GPT:

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



Chemcrow:

<https://github.com/ur-whitelab/chemcrow-public>

Prompts

- Prompts are important because



Andrej Karpathy @karpathy

The hottest new programming language is English

Source: <https://twitter.com/karpathy/status/1617979122625712128>

- Making small tweaks can help us improve overall performance

Adding Style

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.

Let's think step by step

Prompt

I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?

Response

11 apples

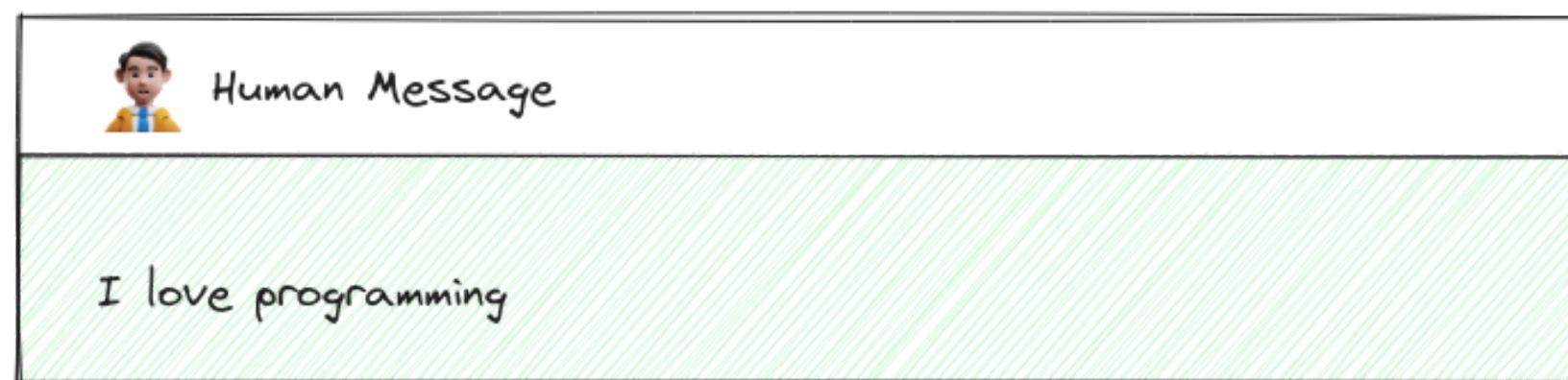
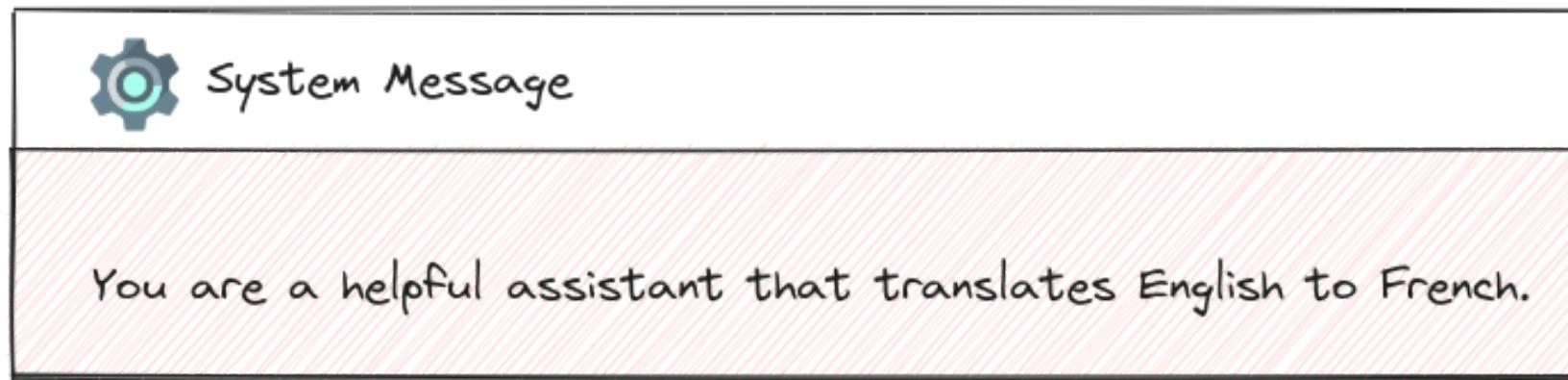
Prompt

I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?
Let's think step by step.

Response

First, you started with 10 apples.
You gave away 2 apples to the neighbor and 2 to the repairman, so you had 6 apples left.
Then you bought 5 more apples, so now you had 11 apples.
Finally, you ate 1 apple, so you would remain with 10 apples.

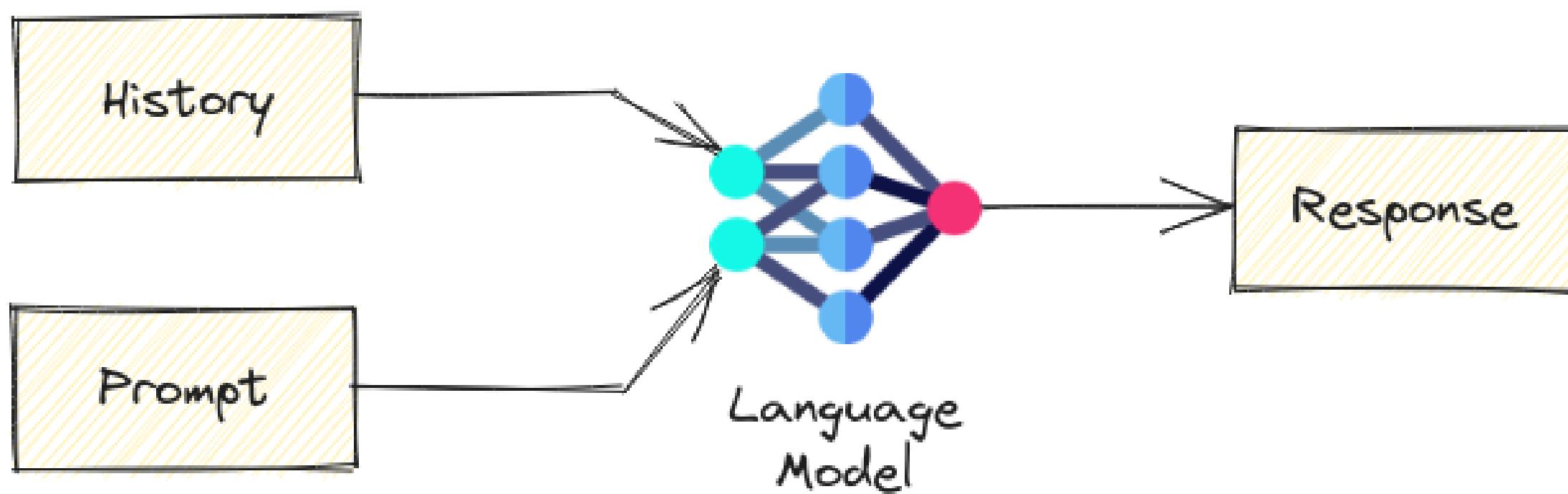
LangChain Implementation



```
1 system_template = SystemMessagePromptTemplate.from_template(  
2     """You are a helpful assistant that translates  
3     {input_language} to {output_language}."""  
4 )  
5  
6 human_template = HumanMessagePromptTemplate.from_template(  
7     "Translate: {text}"  
8 )  
9  
10 chat_prompt = ChatPromptTemplate.from_messages(  
11     [system_template, human_template]  
12 )  
13  
14 prompt = chat_prompt.format_prompt(  
15     input_language="English",  
16     output_language="French",  
17     text="I love programming."  
18 ).to_messages()  
19  
20 chat_llm = ChatGooglePalm()  
21  
22 chat_llm(prompt)  
23 # The translation of "I love programming" in  
24 # French is "J'aime programmer".
```

Memory

- History of the conversation between the user and the LLM
- Helps us retain the context of the conversation
- For chat applications, we pass in the history with the query



Types of 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.

LangChain Implementation



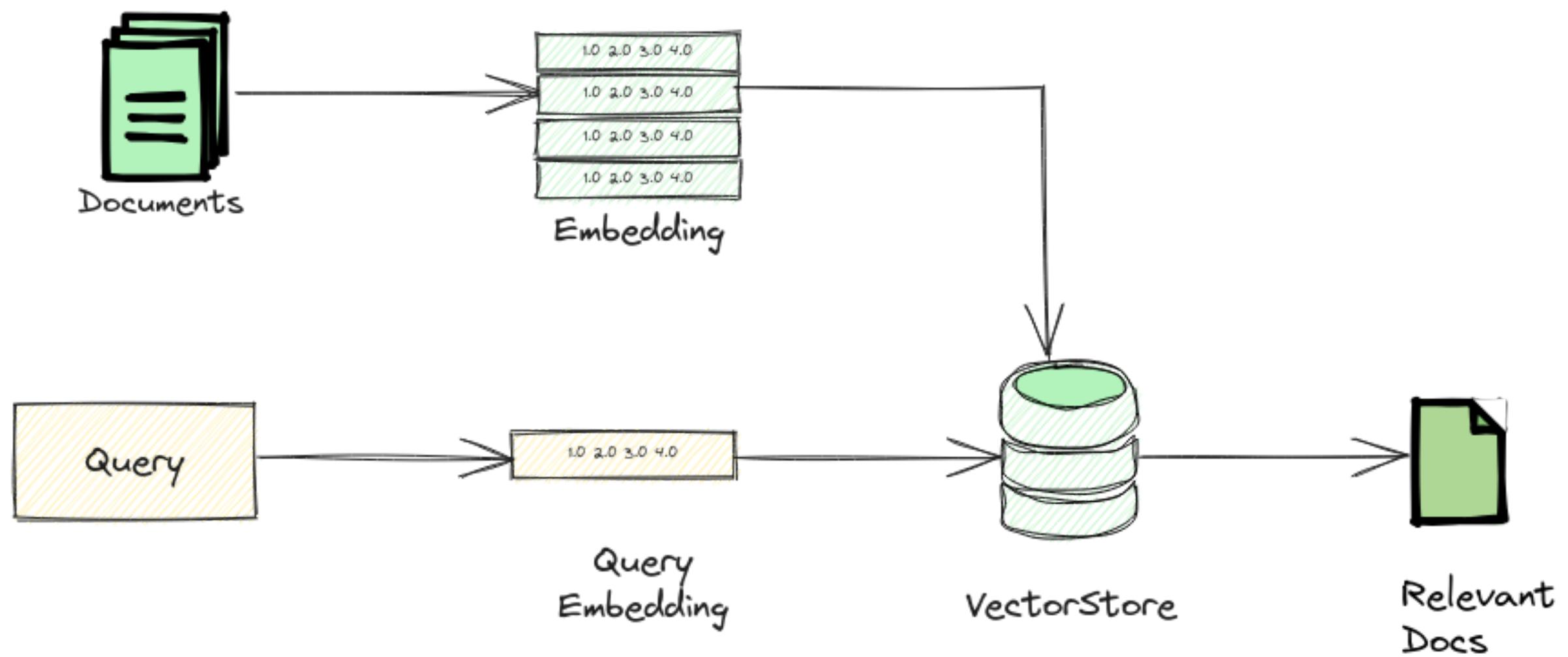
```
1 from langchain.memory import ConversationBufferMemory
2
3 memory = ConversationBufferMemory()
4
5 memory.save_context(
6     {"input": "hi"},
7     {"output": "whats up"}
8 )
```



```
1 from langchain.memory import ConversationSummaryMemory
2
3 memory = ConversationSummaryMemory(llm=llm)
4
5 memory.save_context(
6     {"input": "Hi there!"},
7     {"output": "Hi there! How can I help you today?"}
8 )
```

Connecting External Data

- Using information beyond the training data of LLMs
- Constrain responses to the provided information



LangChain Implementation



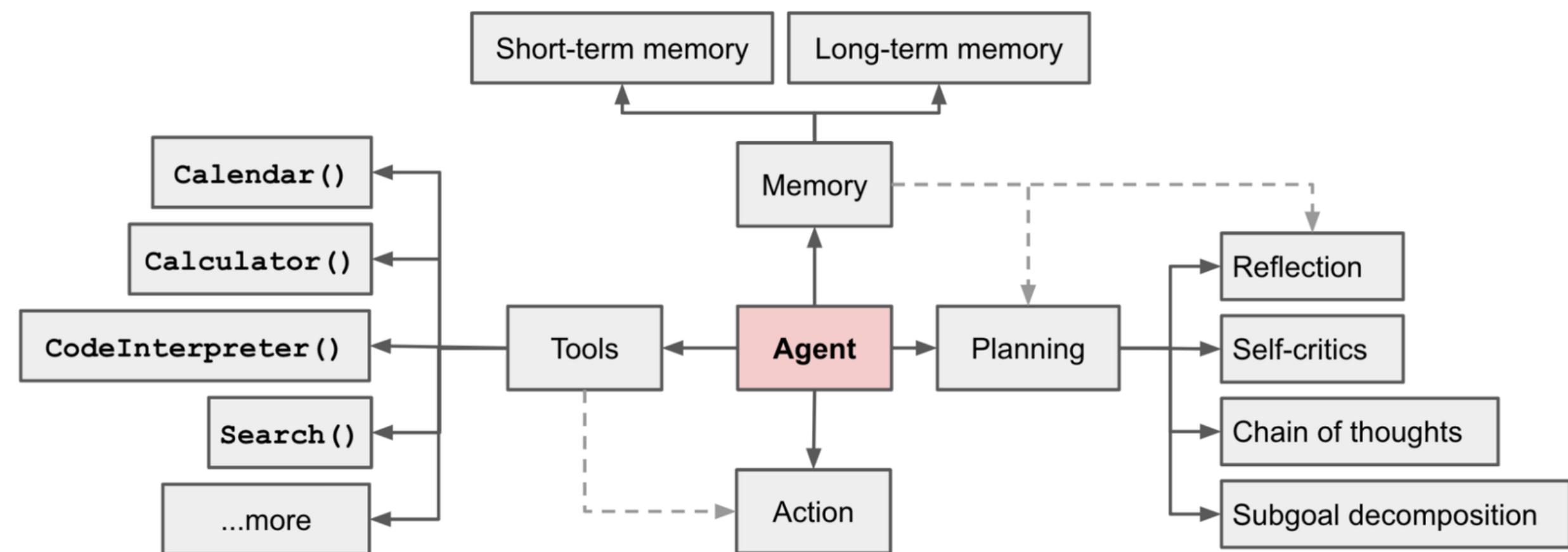
```
1 embeddings = GooglePalmEmbeddings()  
2  
3 vectorstore = Chroma.from_documents(  
4     texts, embeddings, collection_name="google_2023"  
5 )  
6  
7 retriever=vectorstore.as_retriever()  
8  
9 query = "What was Google's total income from operations in 2022?"  
10 docs = retriever.get_relevant_documents(query)
```

Types of vectorstores and retrievers:

- local: [Chroma](#), [Weaviate](#)
- hosted: [Pinecone](#)
- large quantities: [FAISS](#), [ANNOY](#), [ScaNN](#)

Agents and Tools

- Using LLMs as planners
- Enabling them to act using functions -> *tools*
- Using LLMs to reflect



Source: <https://lilianweng.github.io/posts/2023-06-23-agent/>

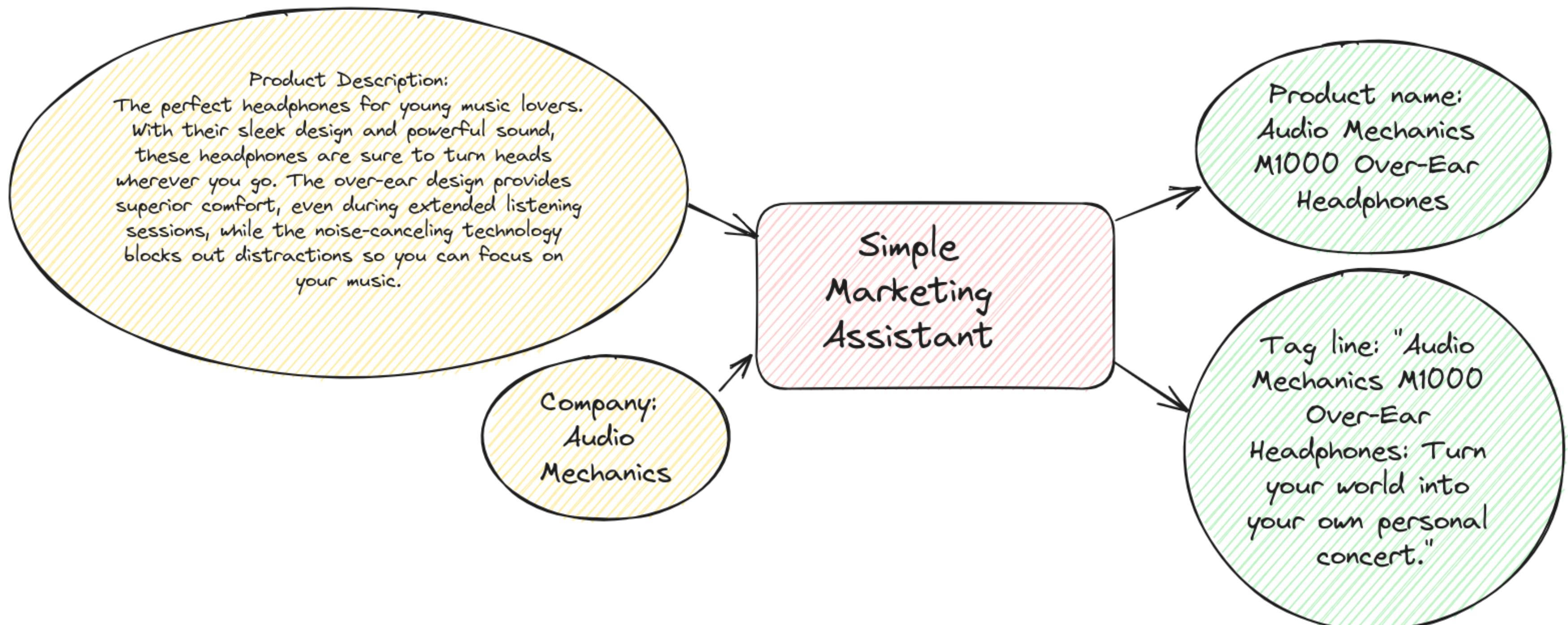
Why chains?

- 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.

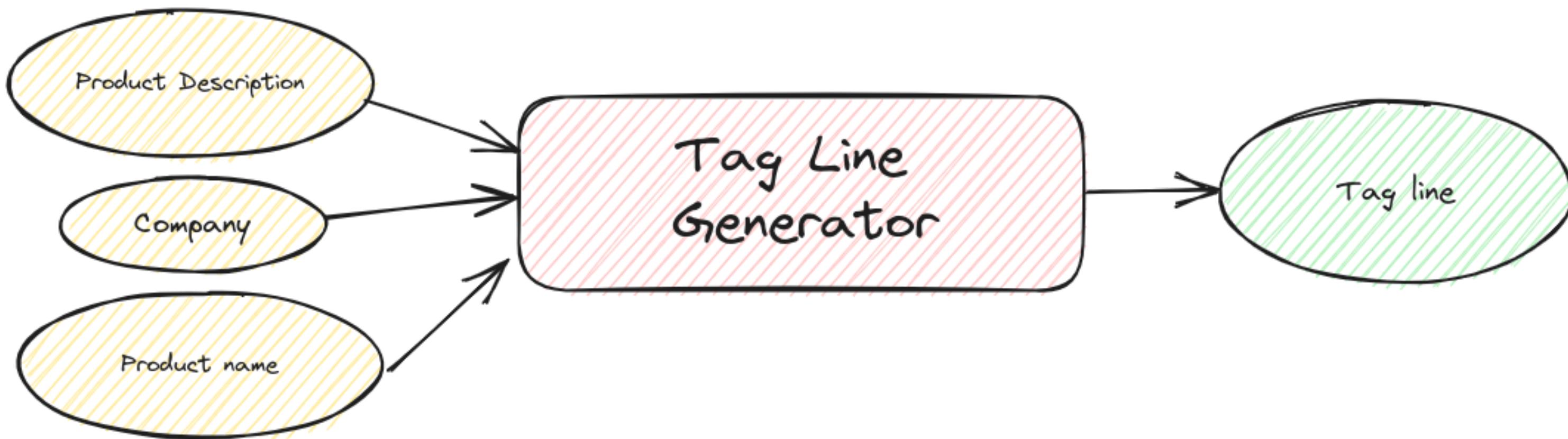
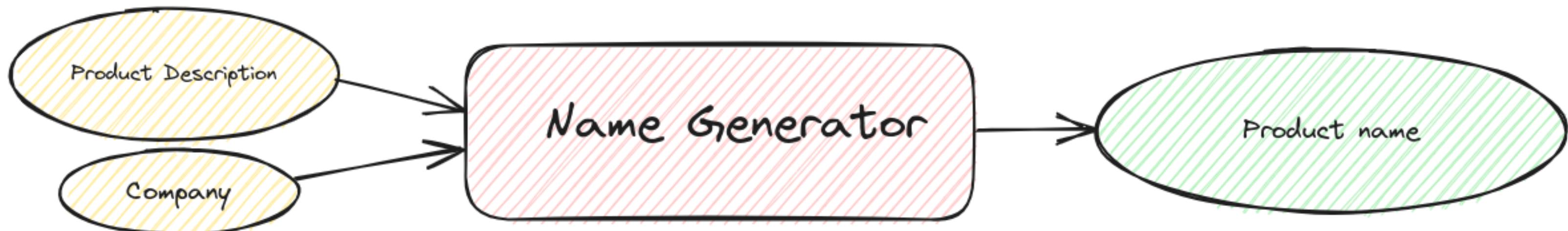
Chains

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

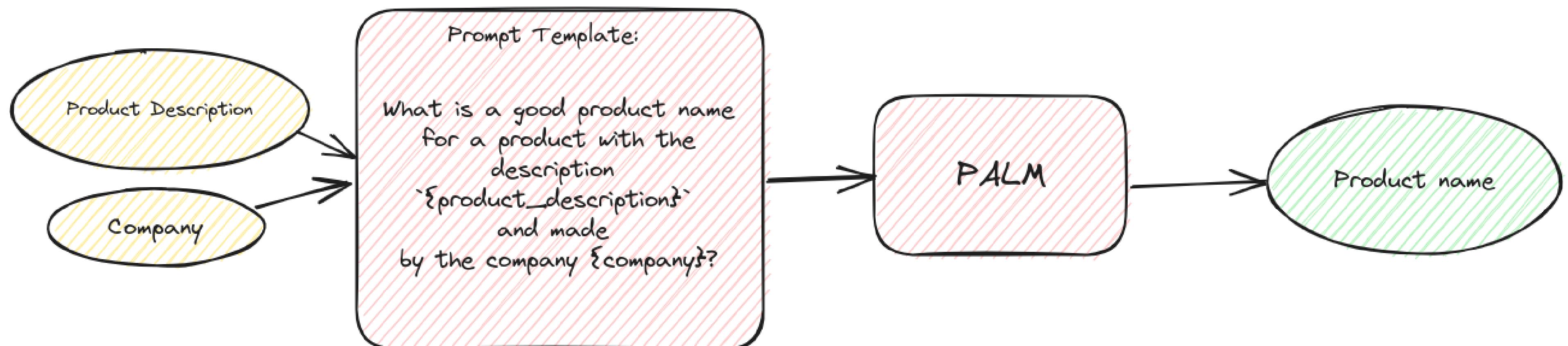
Example: Simple marketing assistant



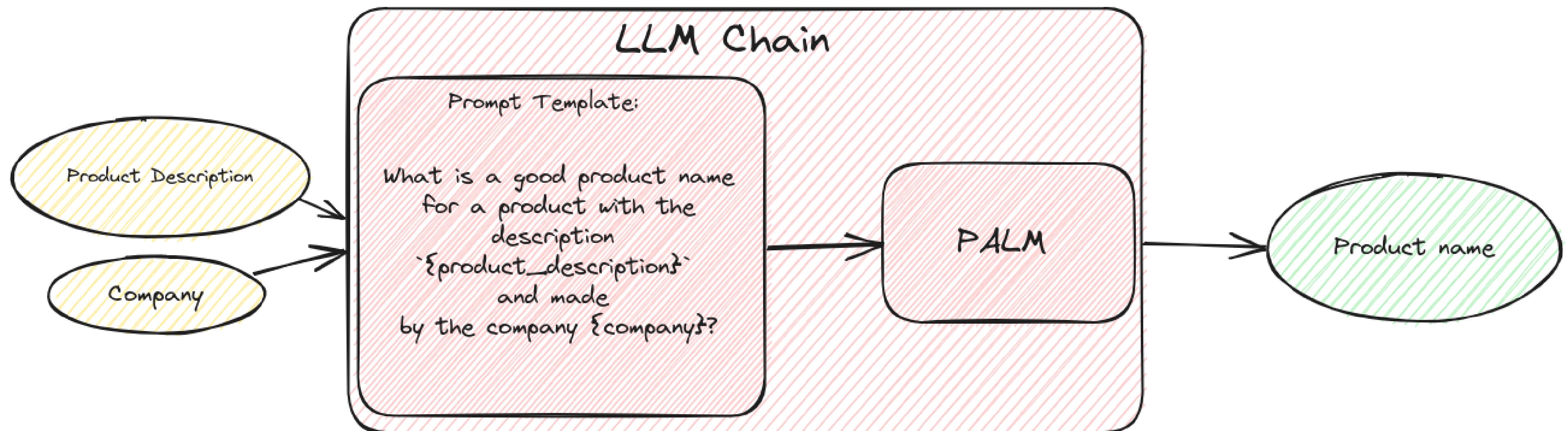
Example Contd. : Divide and Conquer



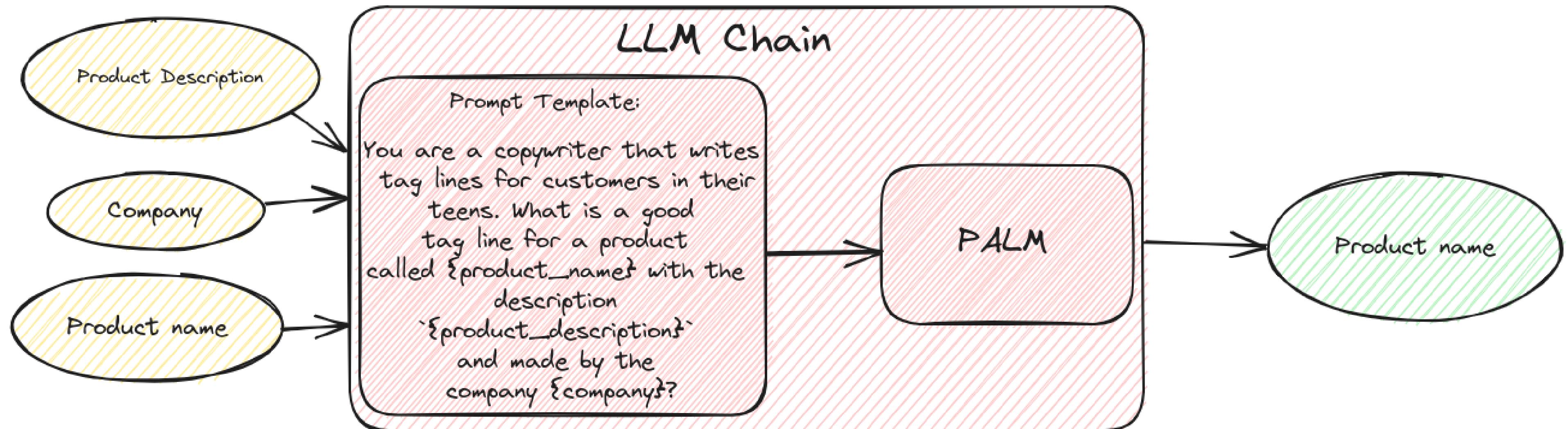
Example Contd.: Open up the name generator



Example Contd. : Make it one module

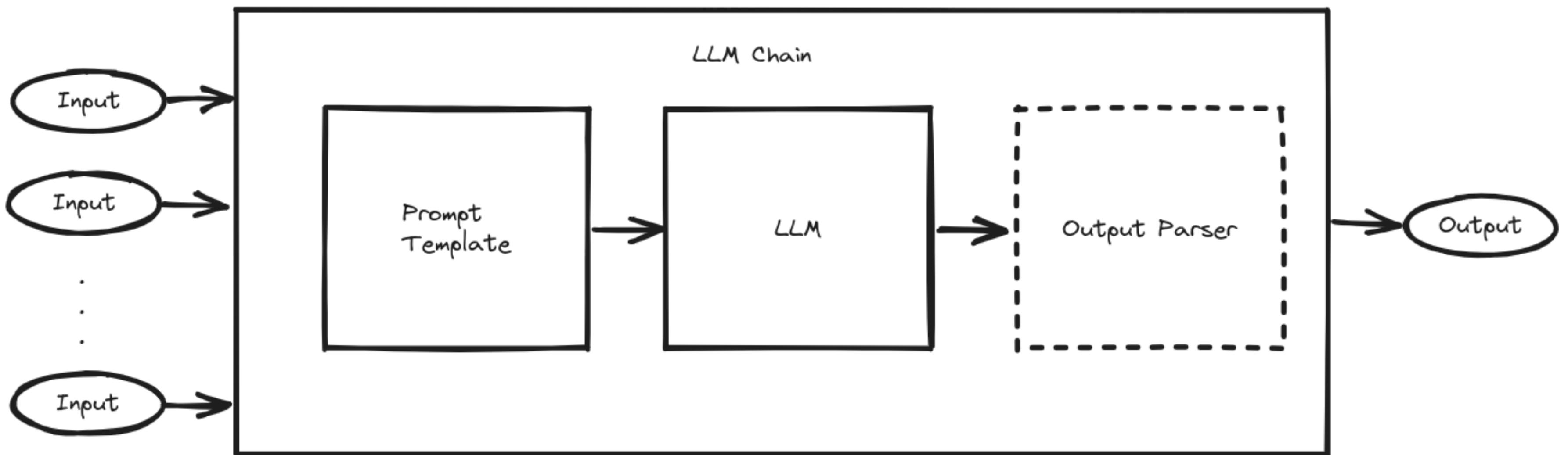


Example Contd. : Now the tag line generator



LLM Chain

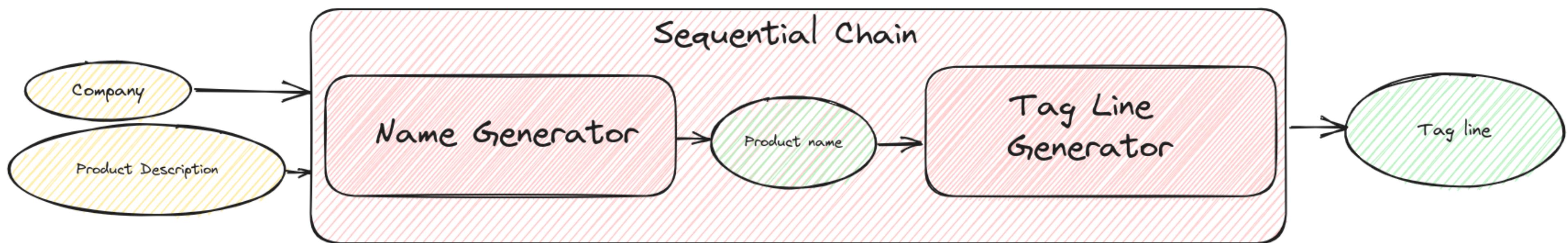
- Most basic chain.



LLM Chain

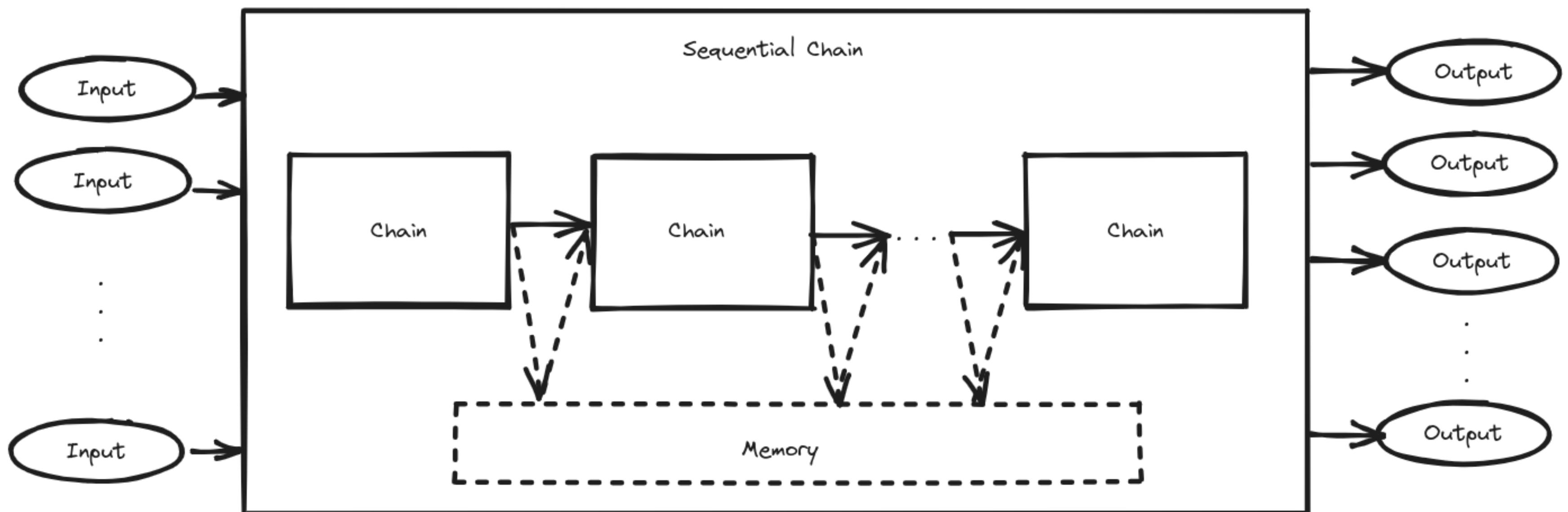
```
1 # Create chain.
2 name_generator_chain = LLMChain(llm=llm, prompt=prompt, output_key='product_name')
3
4 # Run chain.
5 print(
6     name_generator_chain.run({
7         'company': 'Audio Mechanics',
8         'product_description': '''The perfect headphones for young music lovers.
9 With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides
10 superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you
11 can focus on your music.'''
12     })
13 )
14
15 >>> 'Audio Mechanics M1000 Over-Ear Headphones'
```

Example Contd. : Combine both the parts



Sequential Chain

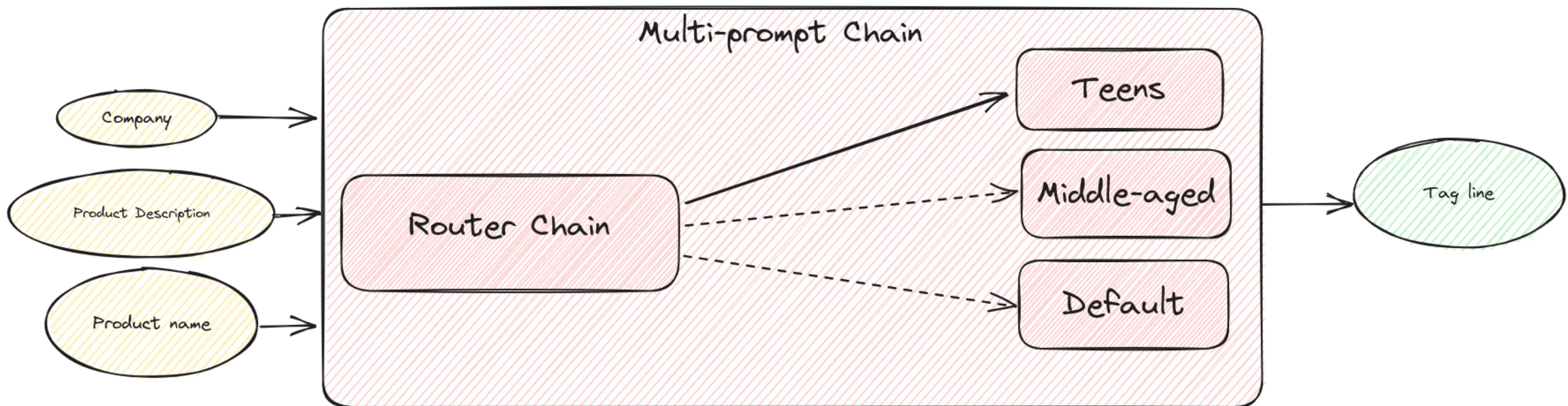
- Chain of chains.



Sequential Chain

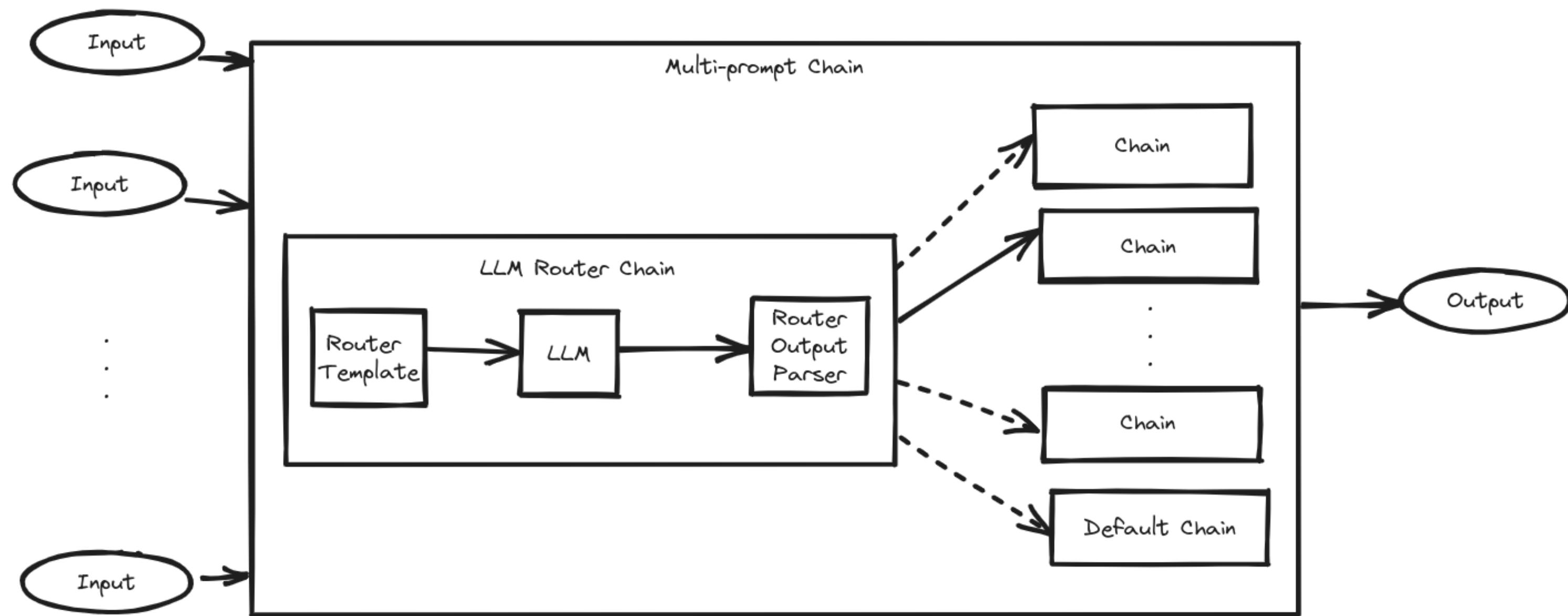
```
 1 # Create chain.
 2 full_chain = SequentialChain(
 3     chains=[name_generator_chain, tagline_generator_chain],
 4     input_variables=["company", "product_description"],
 5     output_variables=["product_name", "tag-line"],
 6 )
 7
 8 # Run chain.
 9 pprint(
10     full_chain(
11         {
12             'company': 'Audio Mechanics',
13             'product_description': '''The perfect headphones for young music lovers.
14 With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides
15 superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you
16 can focus on your music.'''
17         },
18         return_only_outputs=True
19     )
20 )
21
22 >>> {
23     'product_name': 'Audio Mechanics M1000 Over-Ear Headphones',
24     'tag-line': 'Audio Mechanics M1000 Over-Ear Headphones: Sleek design, powerful sound, and superior comfort.'
25 }
```

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

```
 1 # Create router chain.
 2 router_prompt = PromptTemplate(
 3     template=router_template,
 4     input_variables=["company", "product_description", "product_name"],
 5     output_parser=RouterOutputParser(next_inputs_type=dict),
 6 )
 7
 8 router_chain = LLMRouterChain.from_llm(llm, router_prompt)
 9
10 # Create multi-prompt chain.
11 multi_prompt_chain = MultiPromptChain(
12     router_chain=router_chain,
13     destination_chains=destination_chains,
14     default_chain=default_chain,
15 )
16
17 # Run chain.
18 print(
19     multi_prompt_chain.run({
20         'company': 'Audio Mechanics',
21         'product_description': '''The perfect headphones for young music lovers.
22 With their sleek design and powerful sound, these headphones are sure to turn heads wherever you go. The over-ear design provides
23 superior comfort, even during extended listening sessions, while the noise-canceling technology blocks out distractions so you
24 can focus on your music.''' ,
25         'product_name': 'Audio Mechanics M1000 Over-Ear Headphones'
26     })
27 )
28
29 >>> 'Audio Mechanics M1000 Over-Ear Headphones: Turn your world into your own personal concert.'
```

Material Links

Github repo: [bit.ly/lctutorial](https://github.com/infocusp/lctutorial)



References & Resources

- <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>
- <https://www.promptingguide.ai/techniques/cot>
- <https://lilianweng.github.io/posts/2023-06-23-agent/>