



# 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

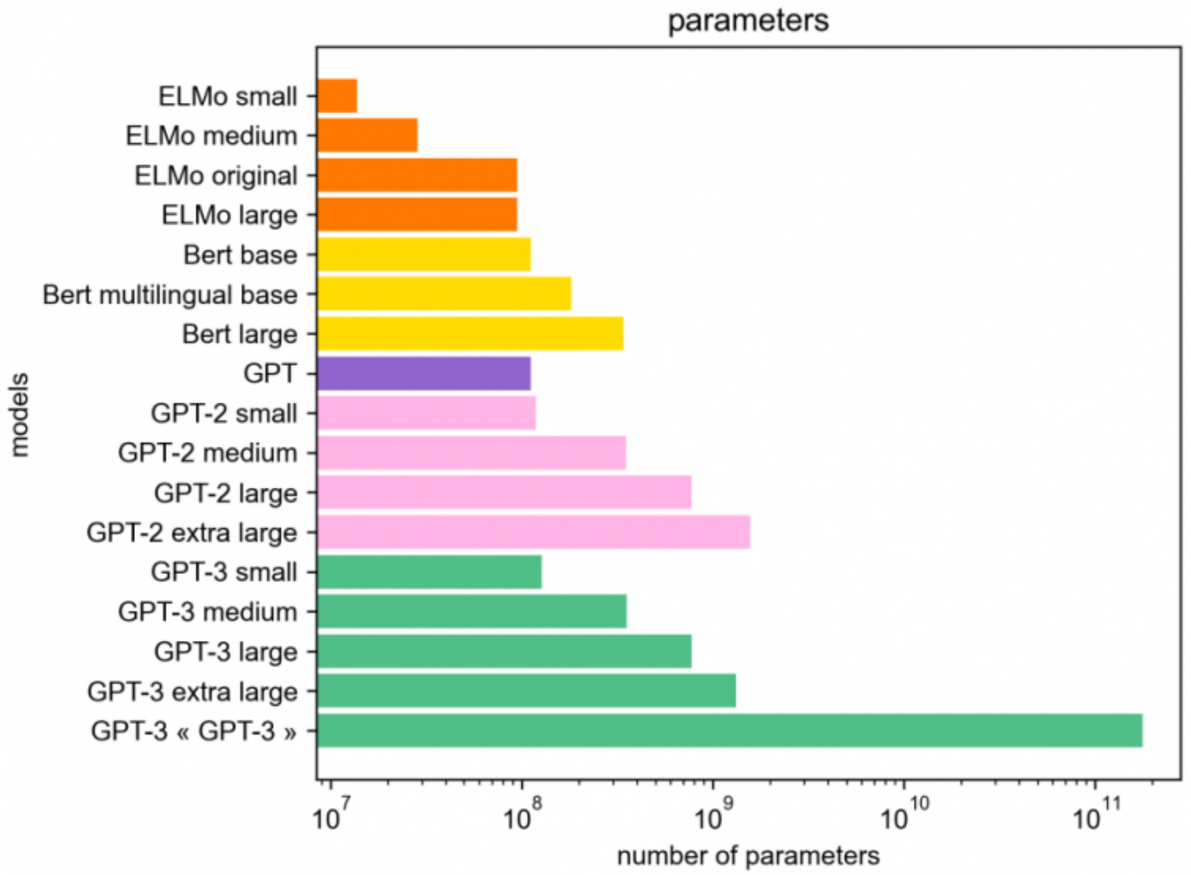
1T tokens and growing

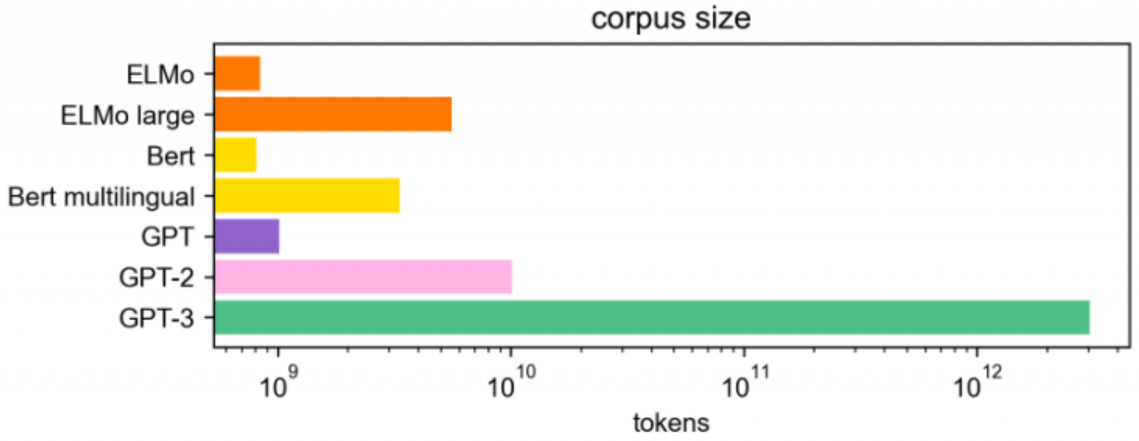
From LSTMs to transformers

Next token prediction to MLM to NSP

Finetuning to few shot to one shot to zero shot

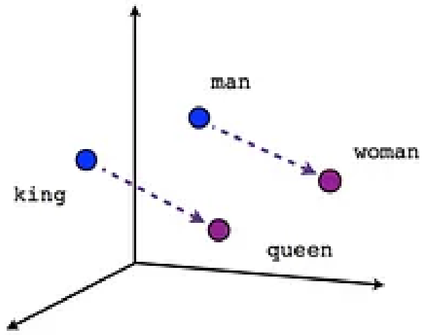
*Ref: https://www.cs.princeton.edu/courses/archive/fall22/cos597G/*



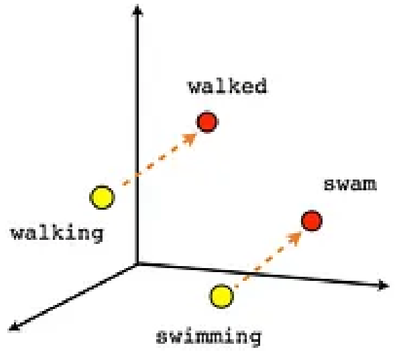



# Embeddings

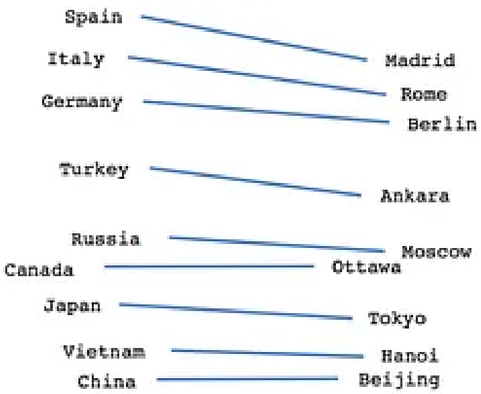
- Encoding sentences as numbers
- Use cases
  - similarity search
  - sentiment analysis
  - clustering
  - classification
  - anomaly detection




Male-Female




Verb tense




Country-Capital

*Connections in language captured by embeddings*

*Image from : Link*

# Why Langchain?

- Harnessing power of LLMs
- Easily integrate them
- Giving new tools to LLMs
- Ready made setup for simpler flows

# Prompts

- Prompts are important because
- Making small tweaks can help us improve overall performance


*Source: https://twitter.com/karpathy/status/1617979122625712128*

# Adding Style

infocusp Innovations

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 Copy code

```
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.

# 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 Copy code

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

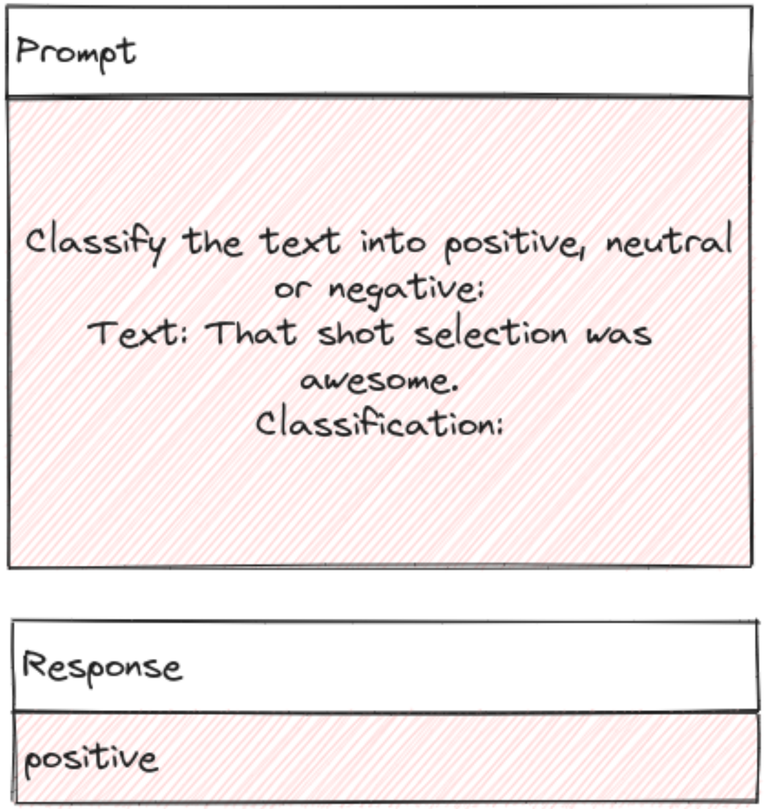
See, that loop be goin' through each piece of treasure in the chest and printin' it out. But don't be askin' me to explain it twice! Ye better take notes or somethin'. I ain't got all day to be repeatin' meself like a broken record! Arrr!

# How to prompt better?

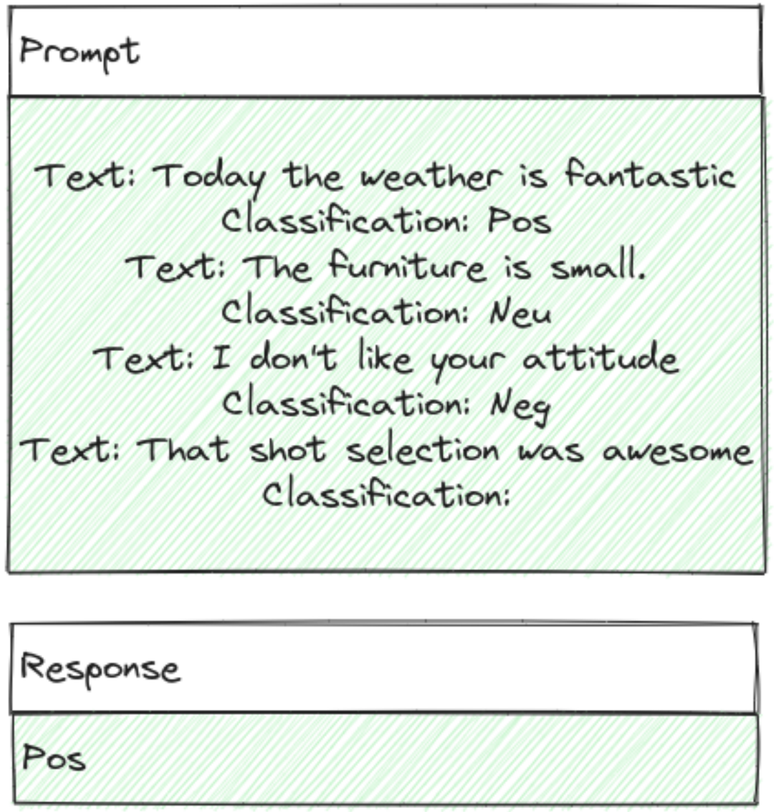
- Give clearer instructions:
  - This requires clarity on the objective.
  - This will generally be an iterative process.
- Few-Shot Prompting: Provide examples to improve reliability and include information that is difficult to convey otherwise.
- Chain-of-Thought Prompting: Give the LLM some time and space to reason the answer. This is helpful for complex tasks.
- Use CAPITAL LETTERS for stronger enforcement.
- Some LLMs are sensitive to minor changes in the prompt. Either avoid such LLMs or be very careful with the prompts.

# Few-Shot Prompting

## Zero-Shot




## Few-Shot



# Chain-of-Thought (CoT) Prompting

infocusp Innovations

**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**

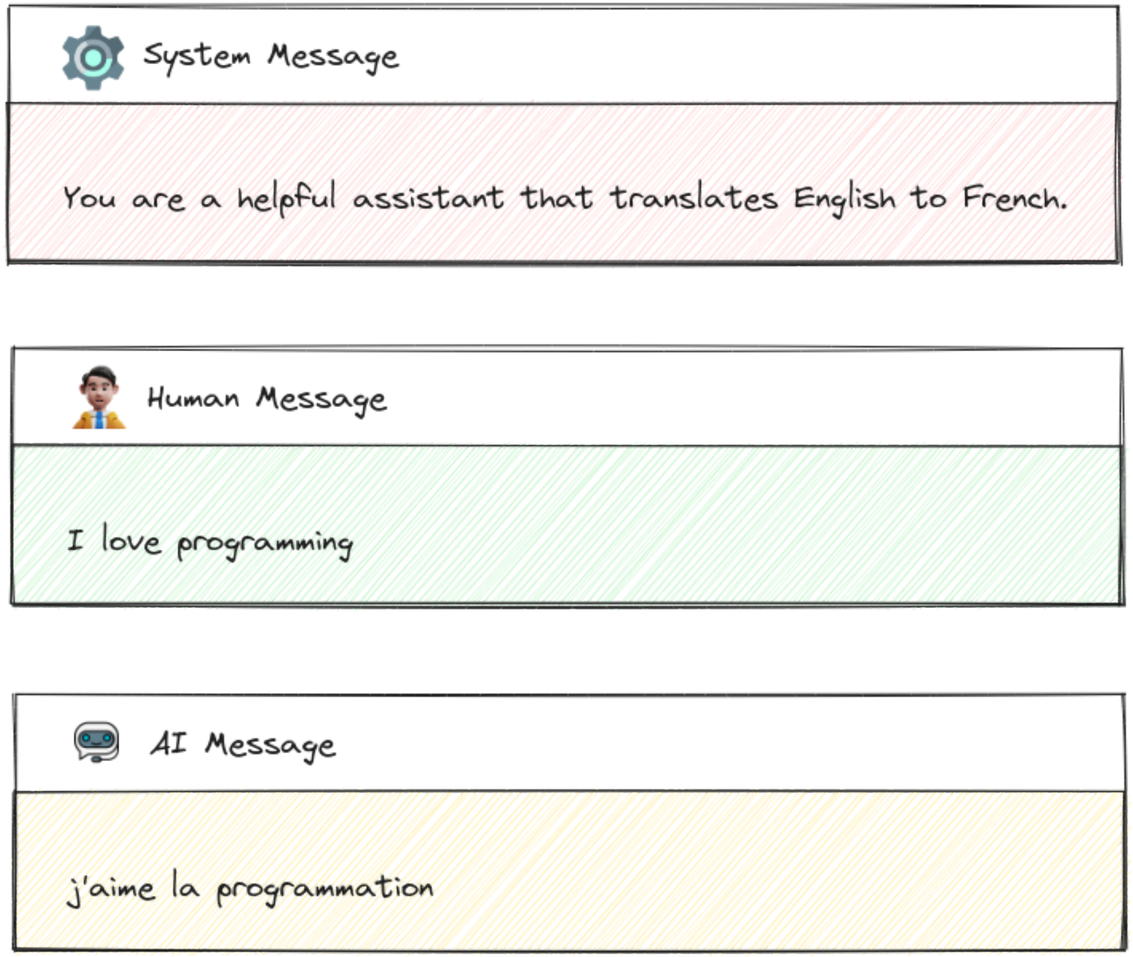
I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?
Let's think step by step.

**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.

- We can go a step further and list out the steps.
- This can also be combined with few-shot prompting.

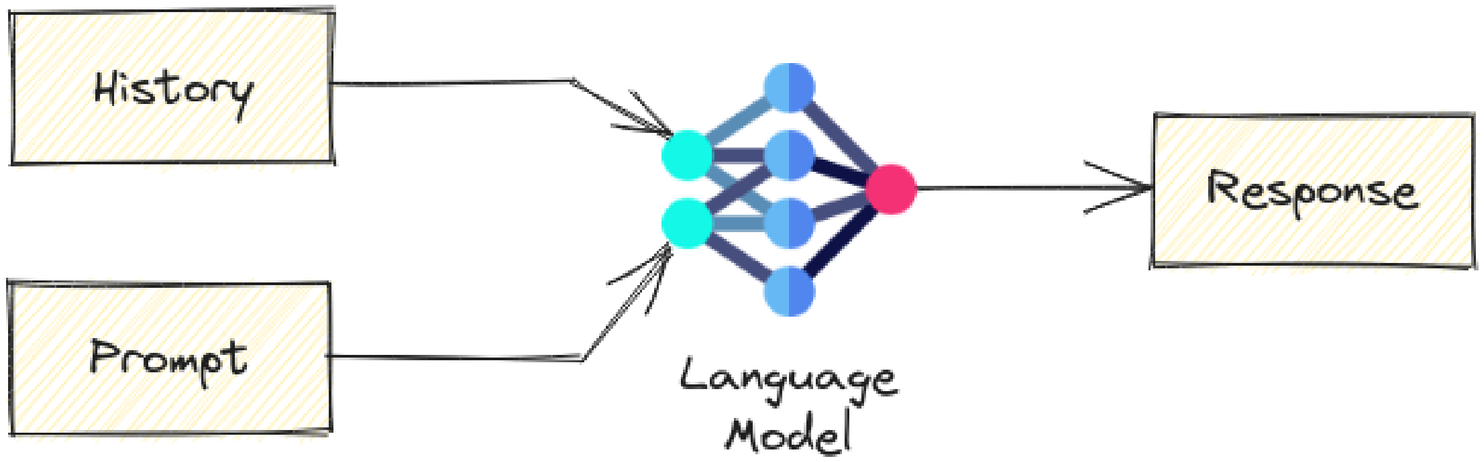
# Chat Workflow




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

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

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

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

chat_llm = ChatGooglePalm()

chat_llm(prompt)
# The translation of "I love programming" in
# 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

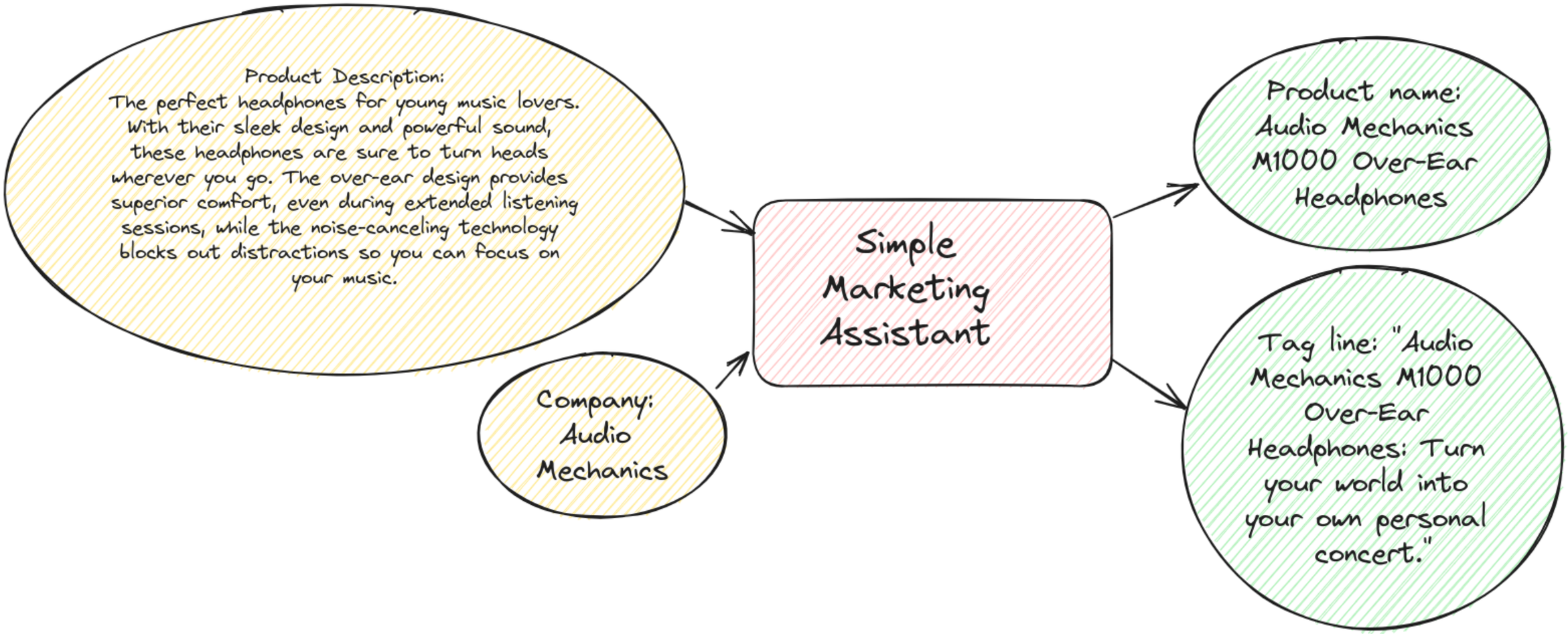
```python
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory()

memory.save_context(
  {"input": "hi"},
  {"output": "whats up"}
)
```

```python
from langchain.memory import ConversationSummaryMemory

memory = ConversationSummaryMemory(llm=llm)

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

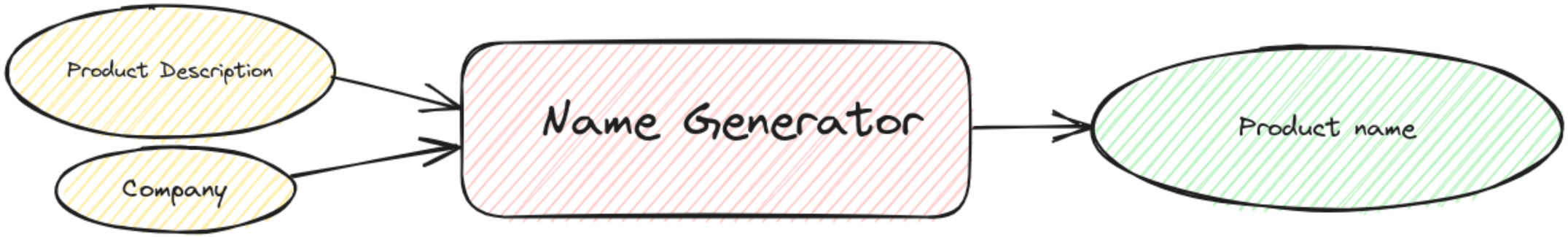
# Chains

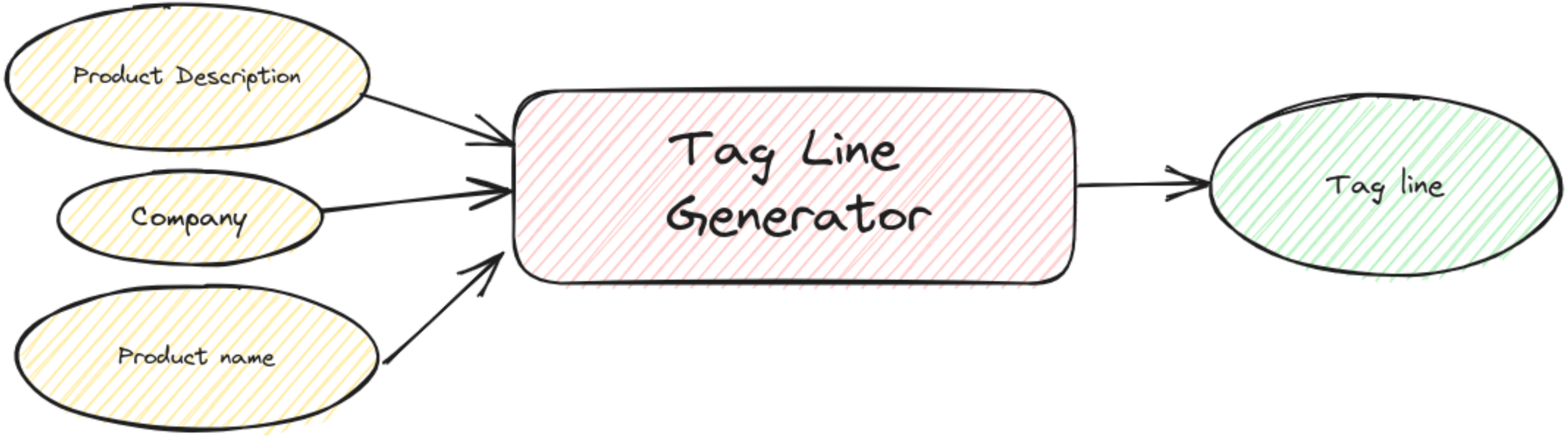
- Combine the modules (prompts, LLMs, parsers, memory) together.
- Combining chains together is also possible.
- Langchain also offers use case-based chains: QA, Summarization, Math, ...

# Example: Simple marketing assistant

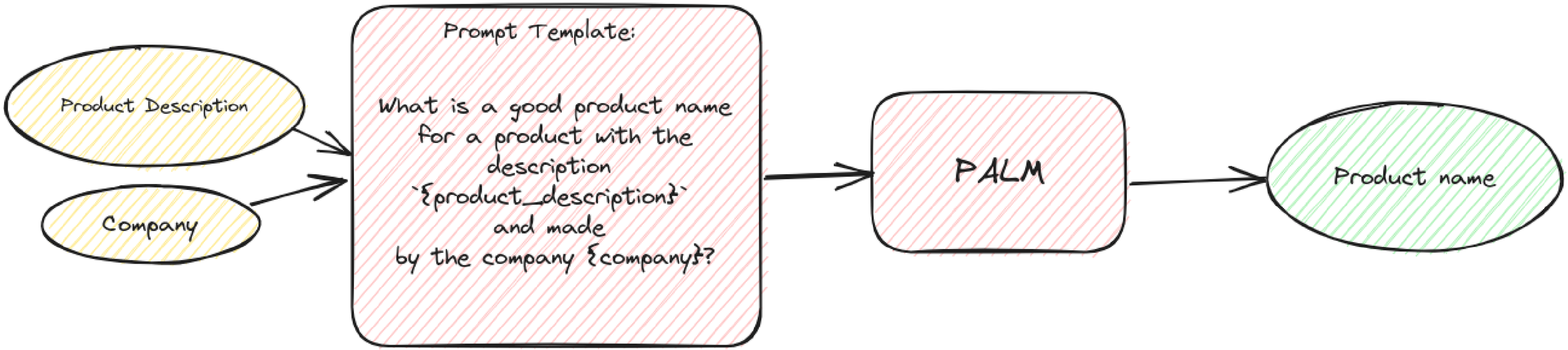


# Example Contd. : Divide and Conquer

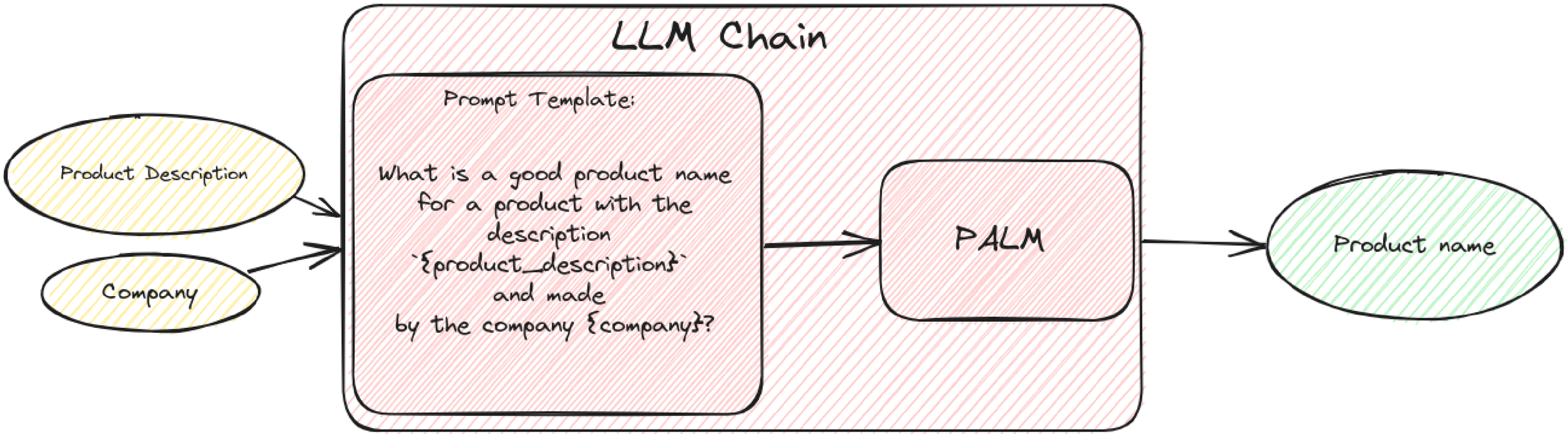


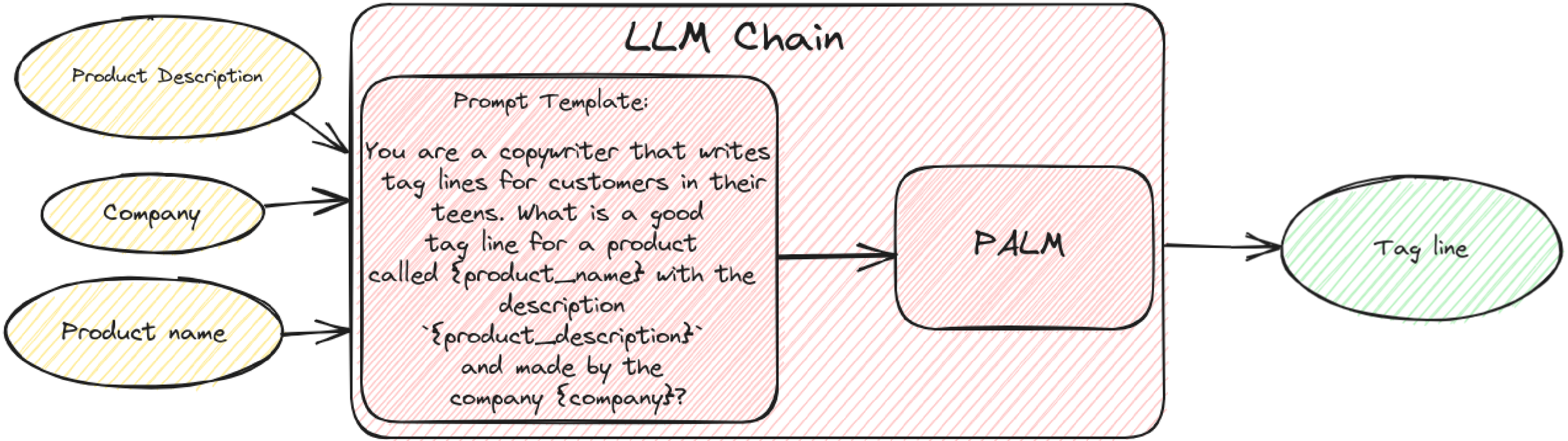
# Example Contd.: Open up the name generator



# Example Contd. : Make it one module

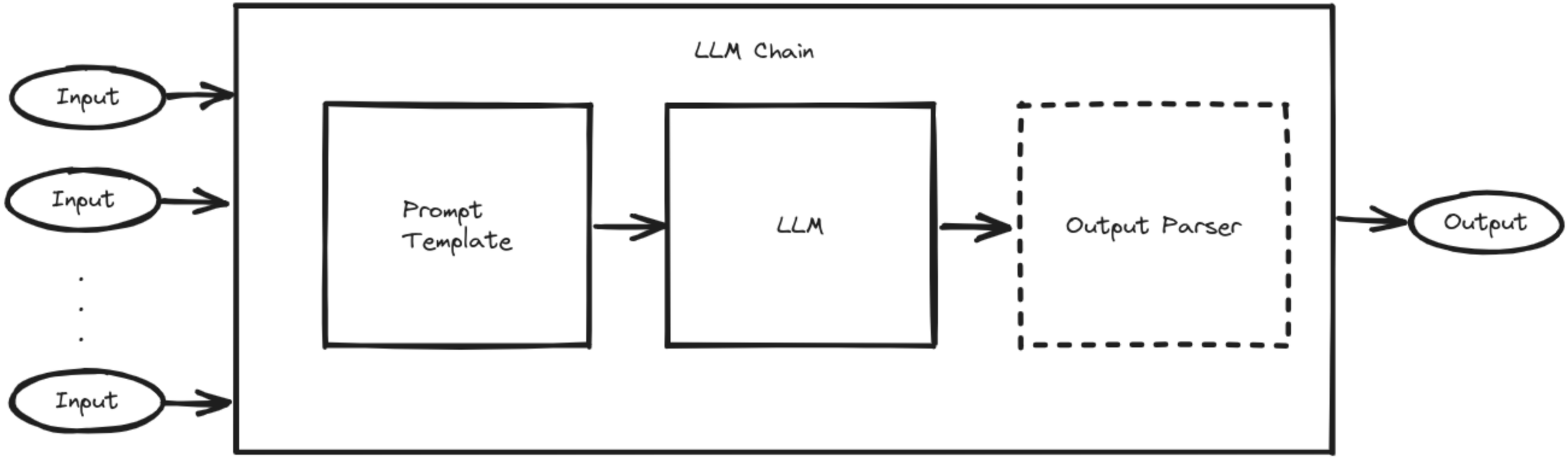


# Example Contd. : Now the tag line generator



# LLM Chain

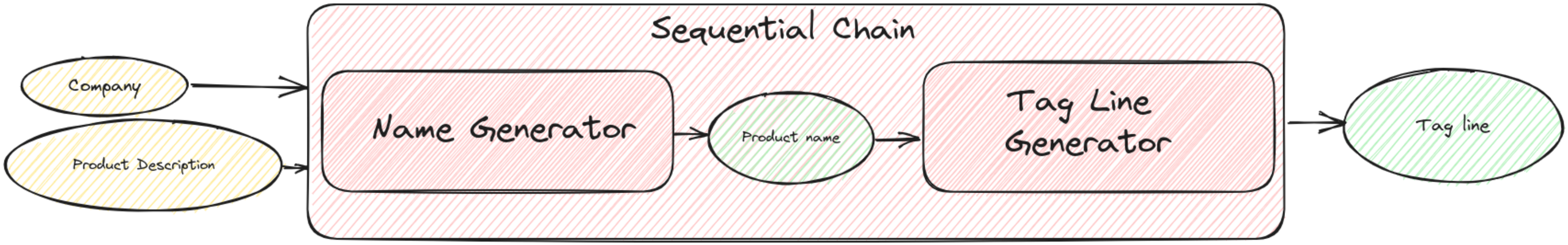
- Most basic chain.



# LLM Chain

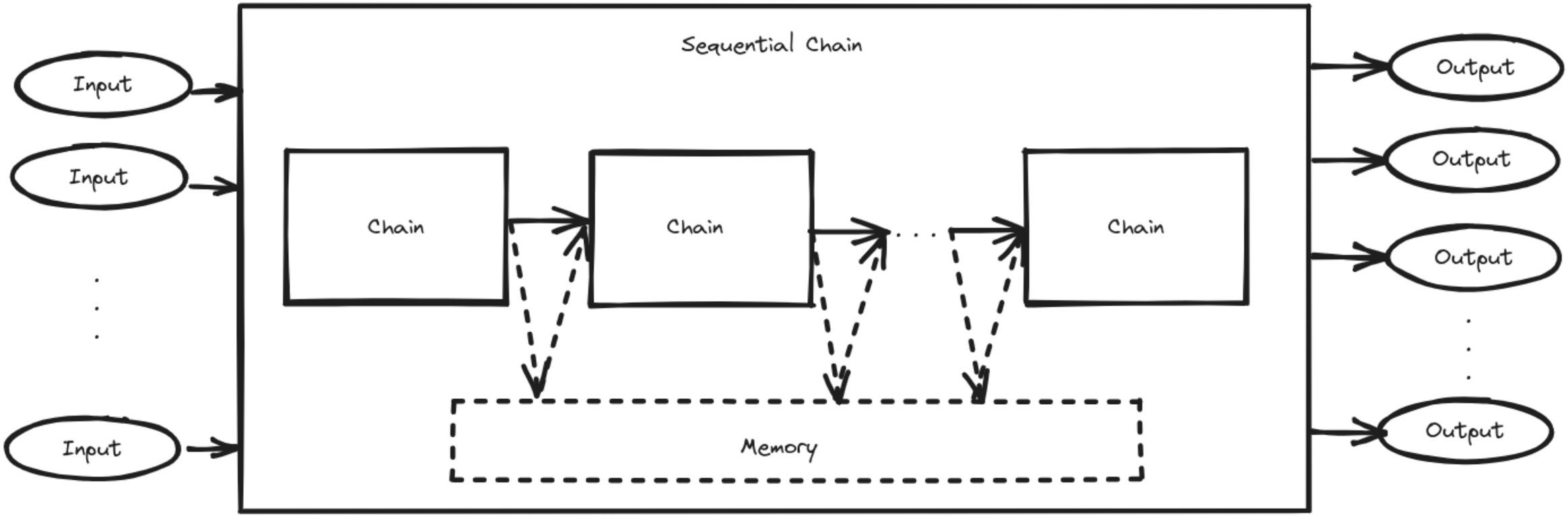
```
# Create chain.
name_generator_chain = LLMChain(llm=llm, prompt=prompt, output_key='product_name')

# Run chain.
print(
    name_generator_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

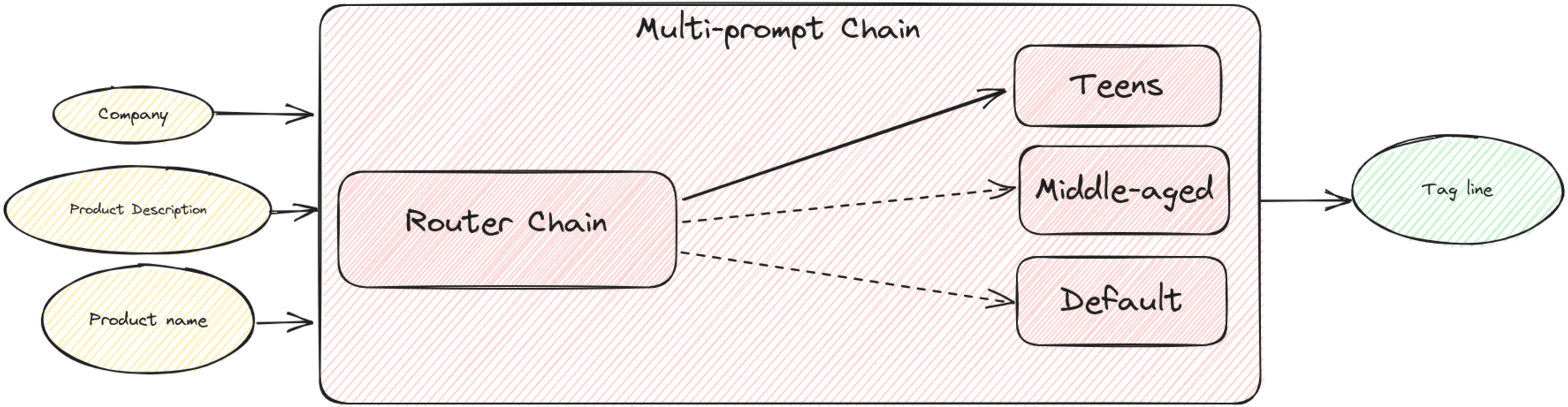
- Chain of chains.



# Sequential Chain

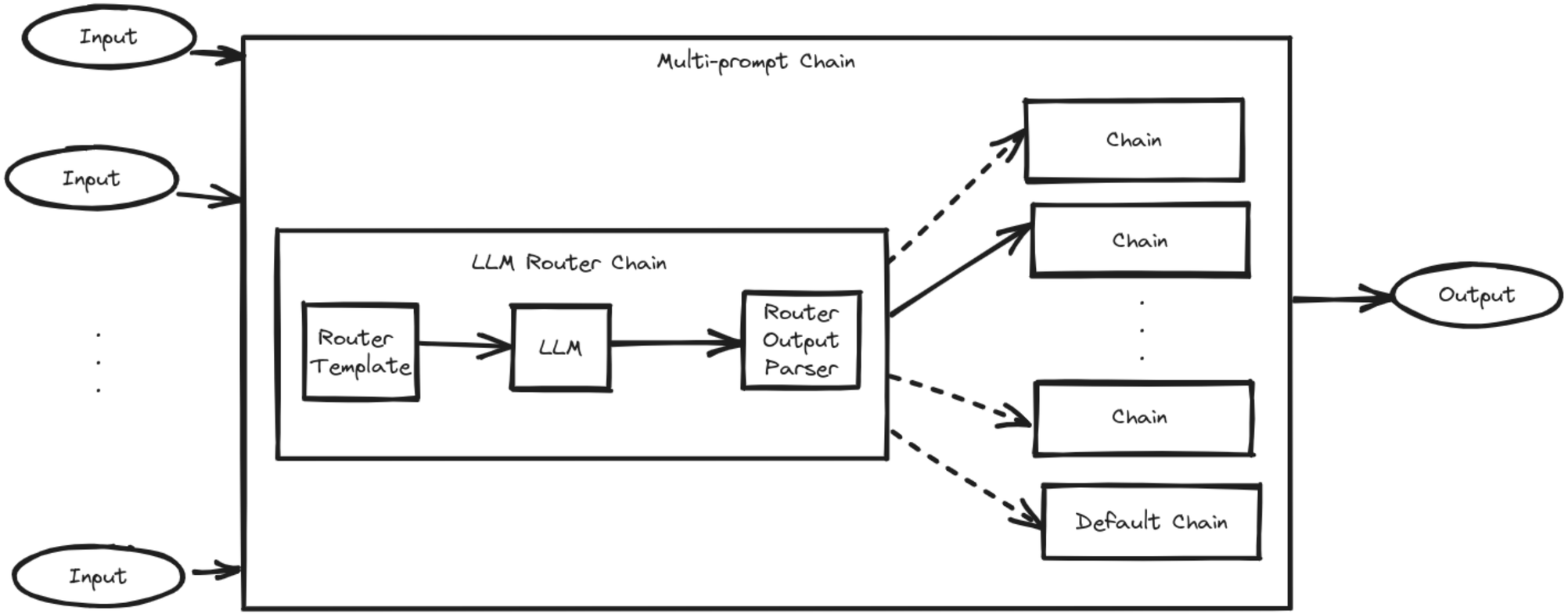
```
# Create chain.
full_chain = SequentialChain(
    chains=[name_generator_chain, tagline_generator_chain],
    input_variables=["company", "product_description"],
    output_variables=["product_name", "tag-line"],
)

# 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.

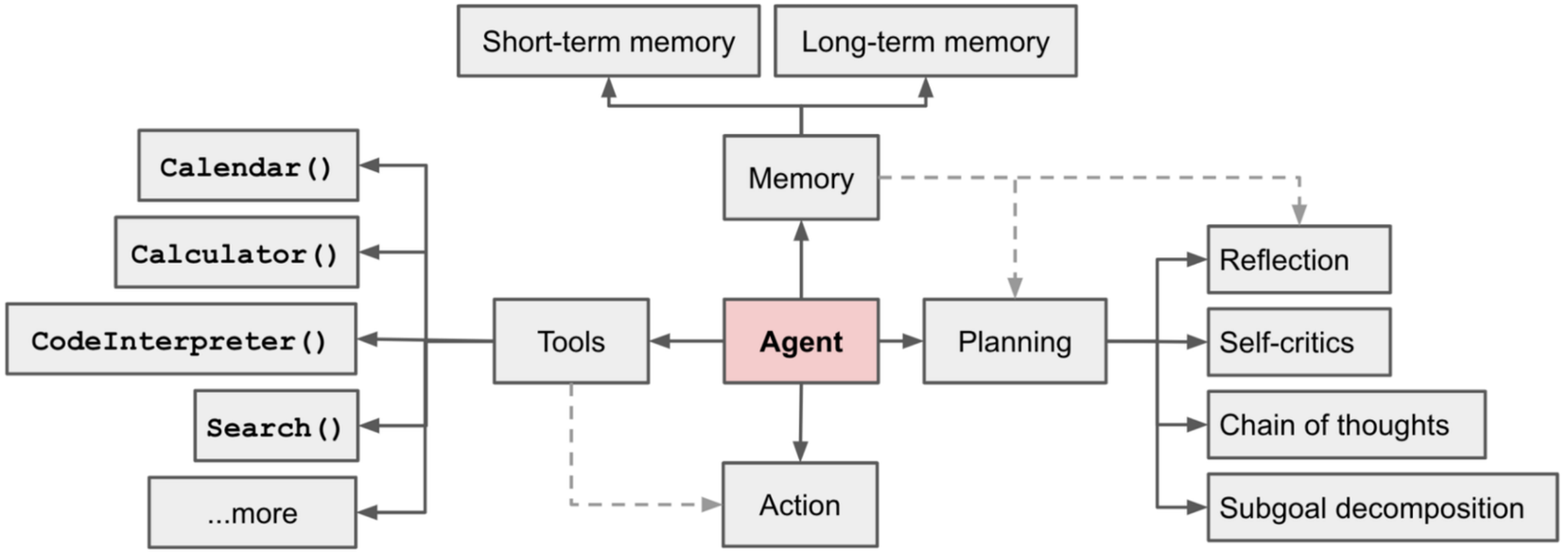


# Multi-Prompt & Router Chains

```
# 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 into your own personal concert.'
```

# 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.

# Retrieval Augmented Generation

Generative capabilities of LLMs are limited by their training data, which leads to the following issues:

- Lack of knowledge
- Hallucinations
- Irrelevant answers

*Chat 1*

*Chat 2*

```
What is RAG?
```

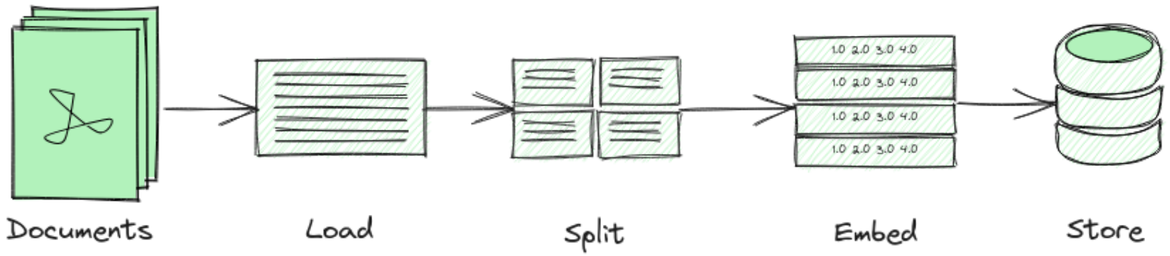
```
RAG can have several meanings. Here are a few possible
interpretations:

1. Red, Amber, Green (RAG): In project management ...
2. RAG Rating: This is a rating system used ...
3. Recombinase-activating gene (RAG): In the context of
genetics and immunology ...
4. RAG Magazine: There have been various magazines ...
```

```
Context: ...
What is RAG?
```

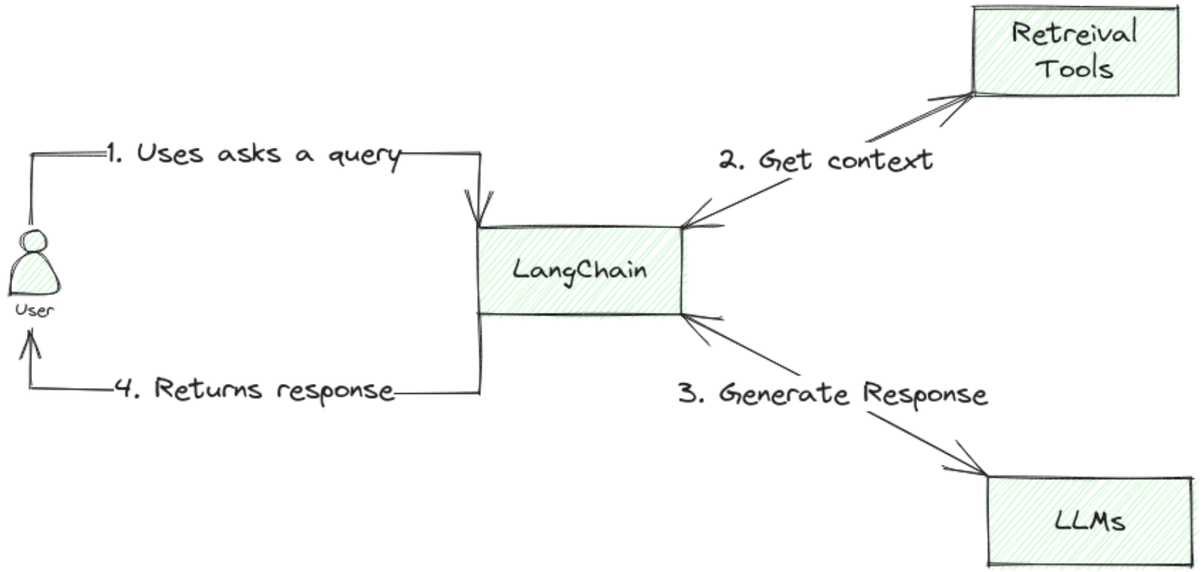
```
RAG, or Retrieval Augmented Generation, is a method
introduced by Meta AI researchers to enhance the
performance of language models in knowledge-intensive
tasks that require access to external knowledge sources.
It is designed to address the limitations of general-
purpose language models (LLMs) ...
```

# General RAG Workflow

Indexing Workflow




Generation Workflow




Source

# Retrieval Augmented Generation

***Query:***
How can we load csv data in LangChain?

***Embedding:***
<1,2,3,4>

| | |
|---|---|
| ***Embedding***<br><1,1,2,4><br><br>***Similarity***<br>0.8 | ***CSV***<br>Load CSV data with a single row per document.<br><br>loader = CSVLoader(file_path='/./.csv')<br>data = loader.load() |
| ***Embedding***<br><3,5,2,4><br><br>***Similarity***<br>0.4 | ***File Directory***<br>This covers how to load all documents in a directory.<br><br>loader = DirectoryLoader('../', glob="**/*.md")<br>docs = loader.load() |
| ***Embedding***<br><6,3,5,7><br><br>***Similarity***<br>0.2 | ***File Directory***<br>This covers how to load all documents in a directory.<br><br>loader = UnstructuredHTMLLoader("example.html")<br>docs = loader.load() |

***Prompt:***
How can we load csv data in LangChain?

Context:
*CSV*
*Load CSV data with a single row per document.*

*loader = CSVLoader(file_path='/./.csv')*
*data = loader.load()*

[LangChain Document Loaders]

# LangChain Implementation

```
embeddings = 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)
```

Types of vectorstores and retrievers:

- local: Chroma, Weaviate
- hosted: Pinecone
- large quantities: FAISS, ANNOY, ScaNN

# Improving RAG (Splitting)

***Querying Tabular Data***

Lots of data and information is stored in tabular data, whether it be csvs, excel sheets, or SQL tables. This ...

***Document Loading***

If you have text data stored in a tabular format, you may want to load the data into a Document and then index it as you ...

***Querying***

If you have more numeric tabular data, or have a large amount of data and don't want to index it, you can also use a ...

***Chains***

If you are just getting started, and you have relatively small/simple tabular data, you should get started with ...

***Agents***

Agents are more complex, and involve multiple queries to the LLM to understand what to do. The downside of agents are ...

***Querying Tabular Data***

Lots of data and information is stored in tabular data, whether it be csvs, excel sheets, or SQL tables. This ...

***Document Loading***

If you have text data stored in a tabular format, you may want to load the data into a Document and then index it as you ...

***Querying***

If you have more numeric tabular data, or have a large amount of data and don't want to index it, you can also use a ...

***Chains***

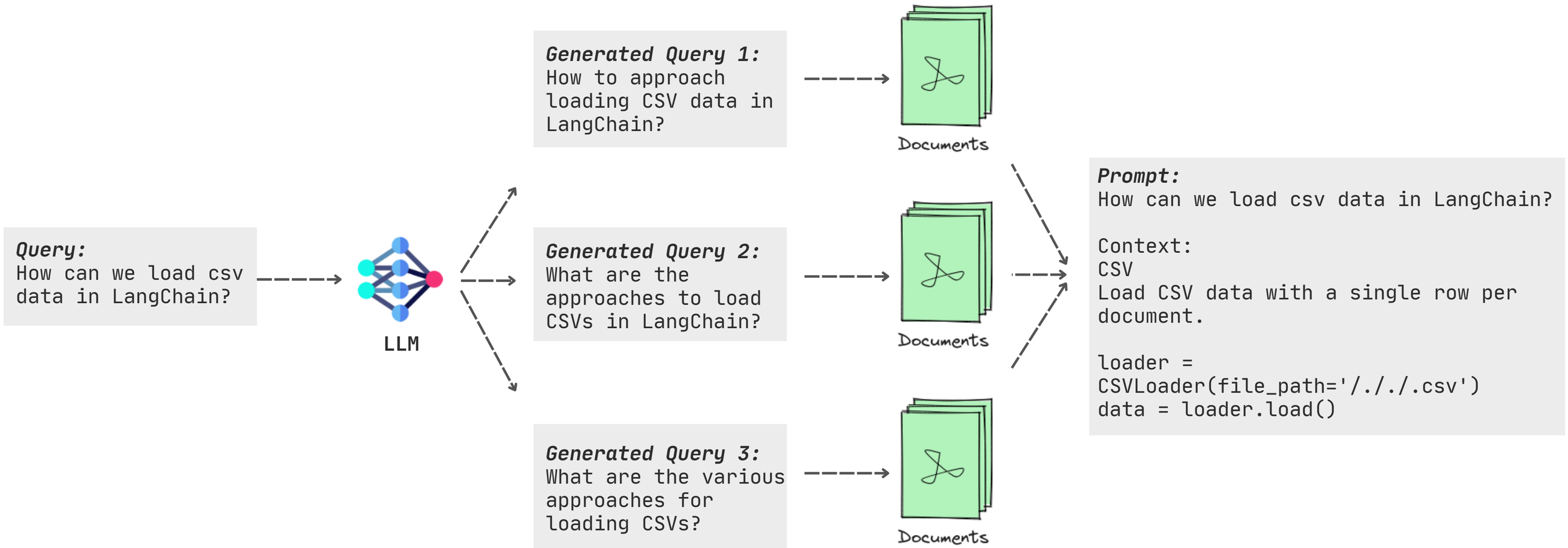
If you are just getting started, and you have relatively small/simple tabular data, you should get started with ...

***Agents***

Agents are more complex, and involve multiple queries to the LLM to understand what to do. The downside of agents are ...

*Count based splitting vs Context based splitting [Source]*

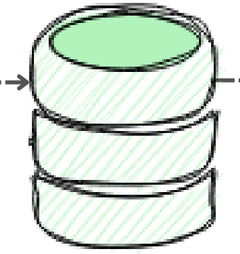
# Improving RAG (Retreival)




Multi query retriever [Source]

# Improving RAG (Post-retreival)

***Query:***
How can we load csv data in LangChain?



Retriever

***Document:*** 1

A CSV file is a delimited text file that uses a comma to separate values. Each line of the file is a data record.

**Load CSV data with a single row per document.**

```
loader = CSVLoader(file_path='a.csv')
data = loader.load()
```

***Compressed Document:*** 2

Load CSV data with a single row per document.

```
loader = CSVLoader(file_path='a.csv')
data = loader.load()
```

***Prompt:*** 3
How can we load csv data in LangChain?

Context:
Load CSV data with a single row per document.

```
loader = CSVLoader(file_path='a.csv')
data = loader.load()
```

***Contextual Compression***

# Improving RAG

infocusp Innovations

## Splitting Techniques

- Based on count
  - Characters/Tokens
- Based on context
  - HTML/Code/Markdown
- LangChain TextSplitters

## Retrieval Techniques

- Multiquery retriever
- Ensemble retriever
- MultiVector retriever
- LangChain Retrievers

## Post-retrieval Techniques

- Reranking documents
- Refining documents
- Prompt Engineering

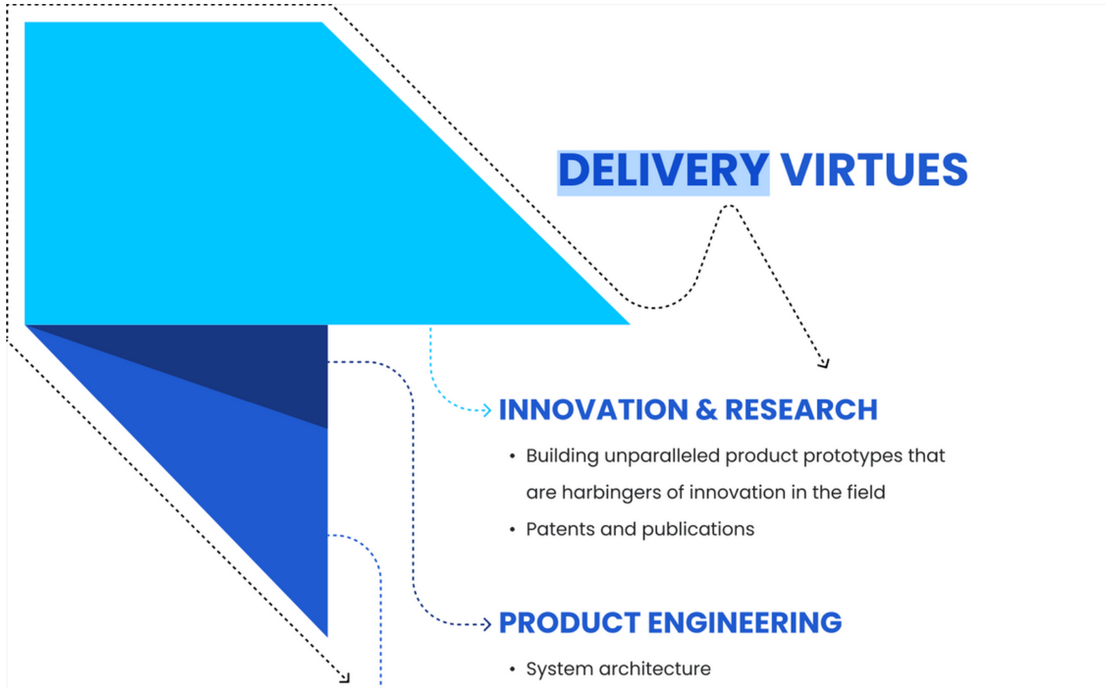
# Material Links

Github repo: bit.ly/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/

infocusp
Innovations



## DELIVERY VIRTUES

### INNOVATION & RESEARCH

- Building unparalleled product prototypes that are harbingers of innovation in the field
- Patents and publications

### PRODUCT ENGINEERING

- System architecture
- App development
- UI / UX

### GROWTH, SCALE, AND SUSTAINABILITY

- Platform scaling and process orchestration
- Maximum efficiency, optimized efficacy with minimum delays
- DevOps

# WHAT WE OFFER

# DOMAINS WE WORK IN

- Healthcare
- FinTech, Banking, and Insurance
- Wearable Technologies
- IoT
- Legal
- Agriculture and Horticulture
- Sciences
- Leisure and Gaming
- Energy
- Molecular Chemistry
- Logistics Supply Chain
- Geophysics