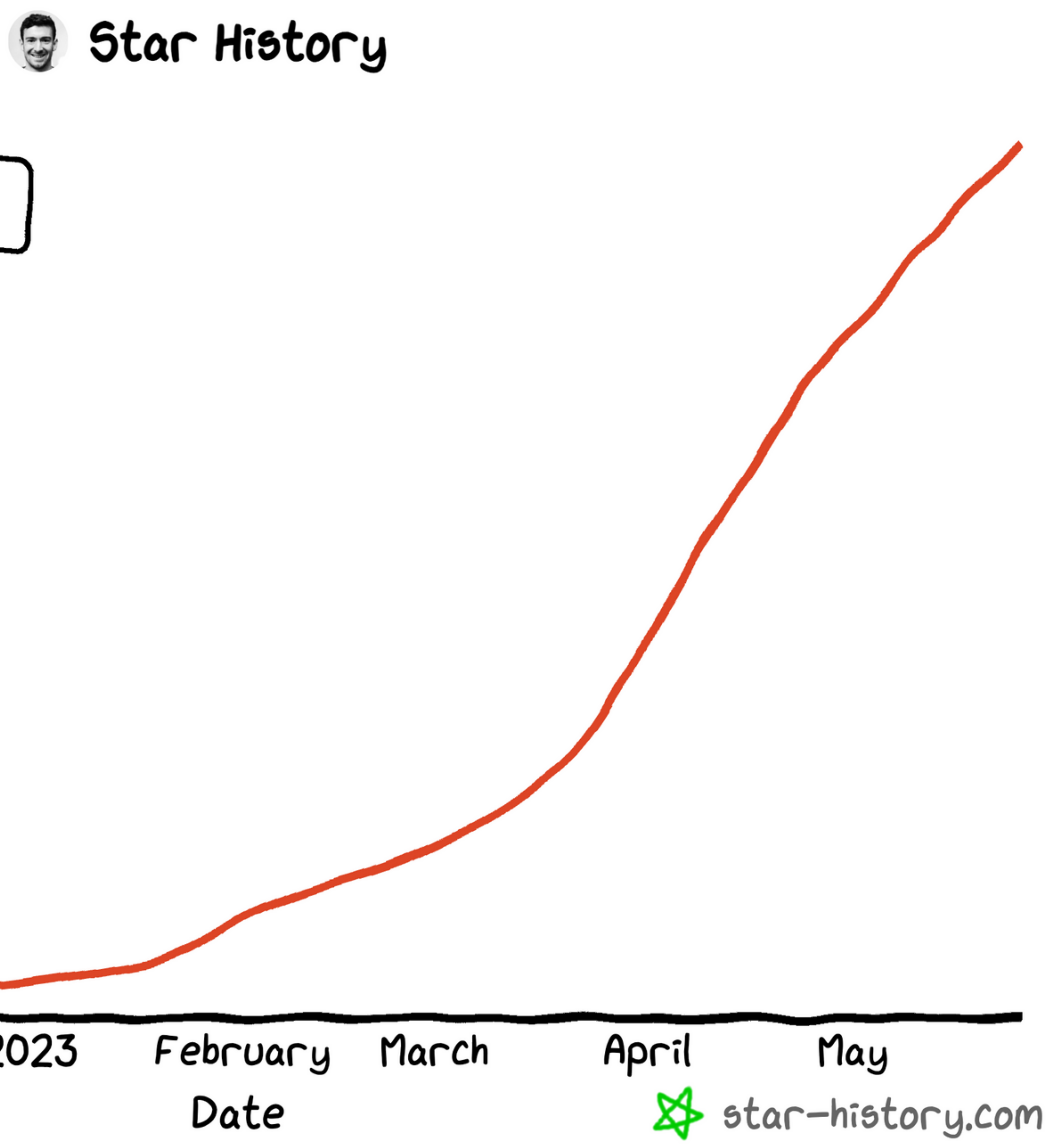
Introduction to Langchain🦜🔗

Connect LLMs to external* data/compute

Mohammad Hossein Biniaz | 09.03.2024 | GWDG

Langchain

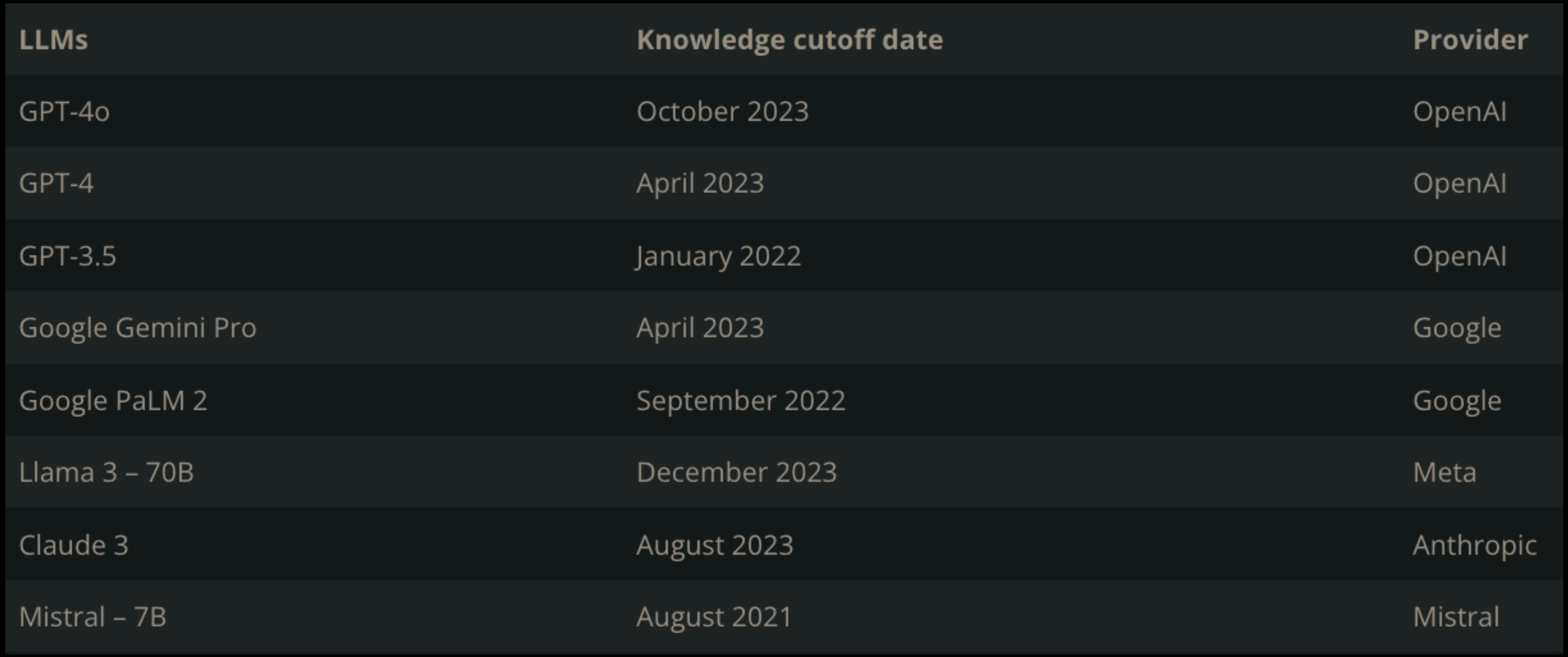
An open source framework that allows AI developers to combine LLMs like GPT-4 with external sources of computation and data.



https://discuss.streamlit.io/t/langchain-tutorial-1-build-an-llm-powered-app-in-18-lin

What is the issue we are trying to address?

Background: knowledge cutoff



LLMs	Knowledge cutoff date	Provider
GPT-4o	October 2023	OpenAI
GPT-4	April 2023	OpenAI
GPT-3.5	January 2022	OpenAI
Google Gemini Pro	April 2023	Google
Google PaLM 2	September 2022	Google
Llama 3 – 70B	December 2023	Meta
Claude 3	August 2023	Anthropic
Mistral – 7B	August 2021	Mistral

- LLM has good general knowledge up to a certain date

- No idea about things that happened after it

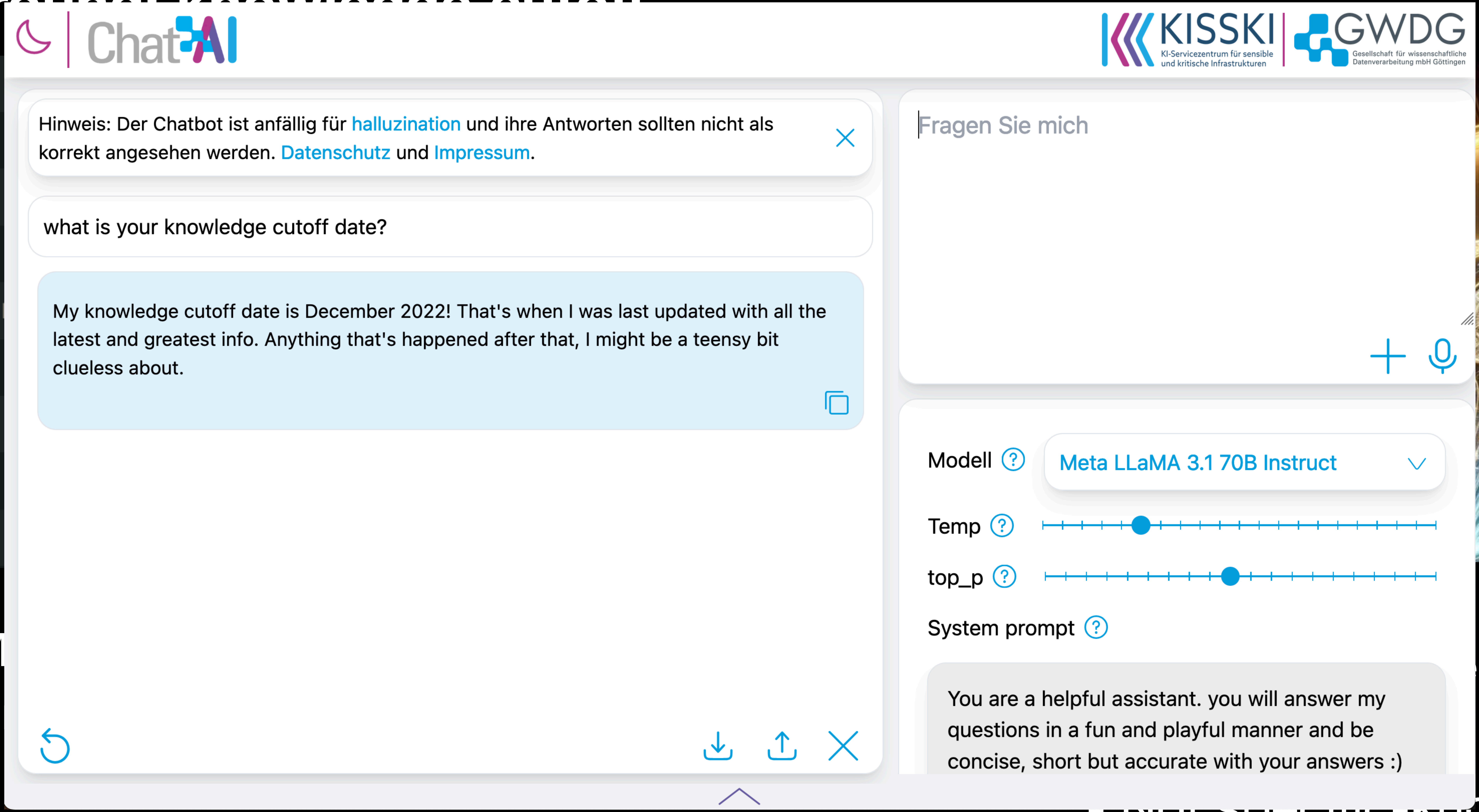
- Very good general knowledge

- Not specific-knowledge

What is the issue we are trying to address?

Backgr knowledge cutoff

LLMs
GPT-4o
GPT-4
GPT-3.5
Google Gemini
Google PaLM 2
Llama 3 – 70B
Claude 3
Mistral – 7B


Chat AI
KISSKI
KI-Servicezentrum für sensible und kritische Infrastrukturen
GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Hinweis: Der Chatbot ist anfällig für halluzination und ihre Antworten sollten nicht als korrekt angesehen werden. Datenschutz und Impressum.

what is your knowledge cutoff date?

My knowledge cutoff date is December 2022! That's when I was last updated with all the latest and greatest info. Anything that's happened after that, I might be a teensy bit clueless about.

Fragen Sie mich

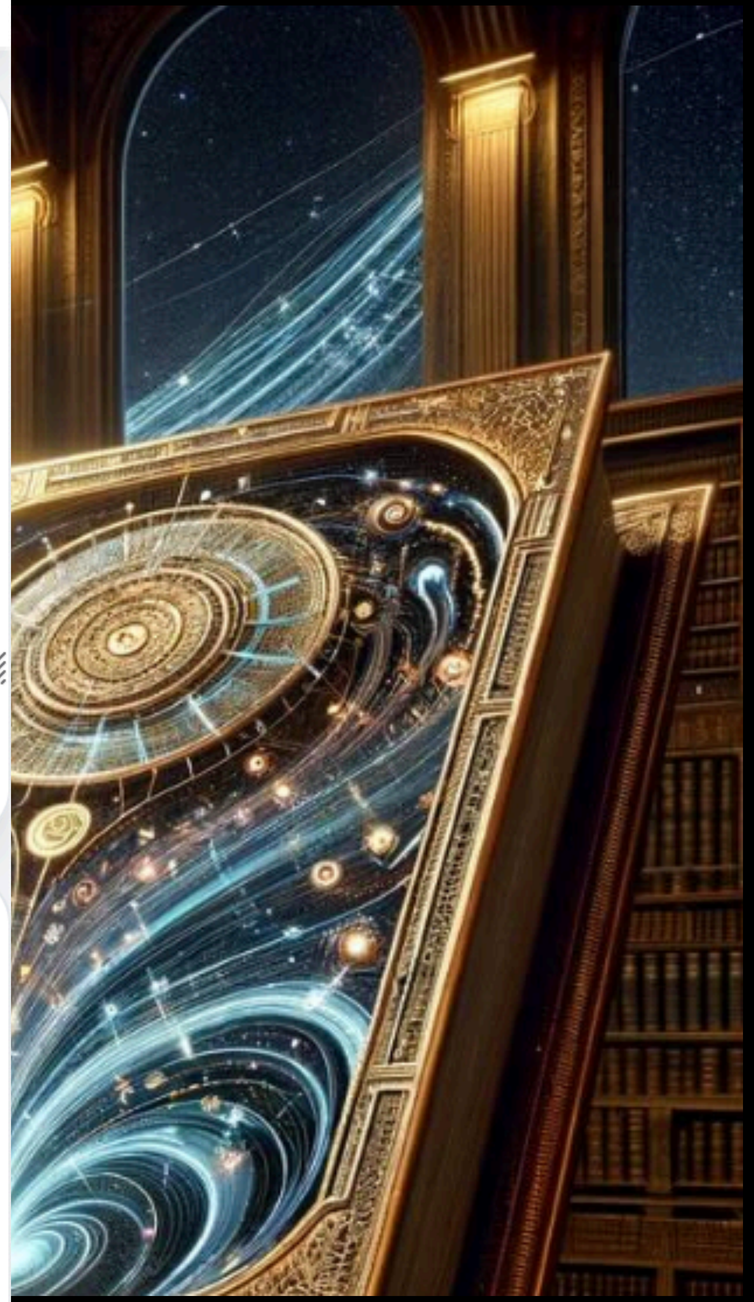
Modell: Meta LLaMA 3.1 70B Instruct

Temp

top_p

System prompt

You are a helpful assistant. you will answer my questions in a fun and playful manner and be concise, short but accurate with your answers :)



- LLM
certai

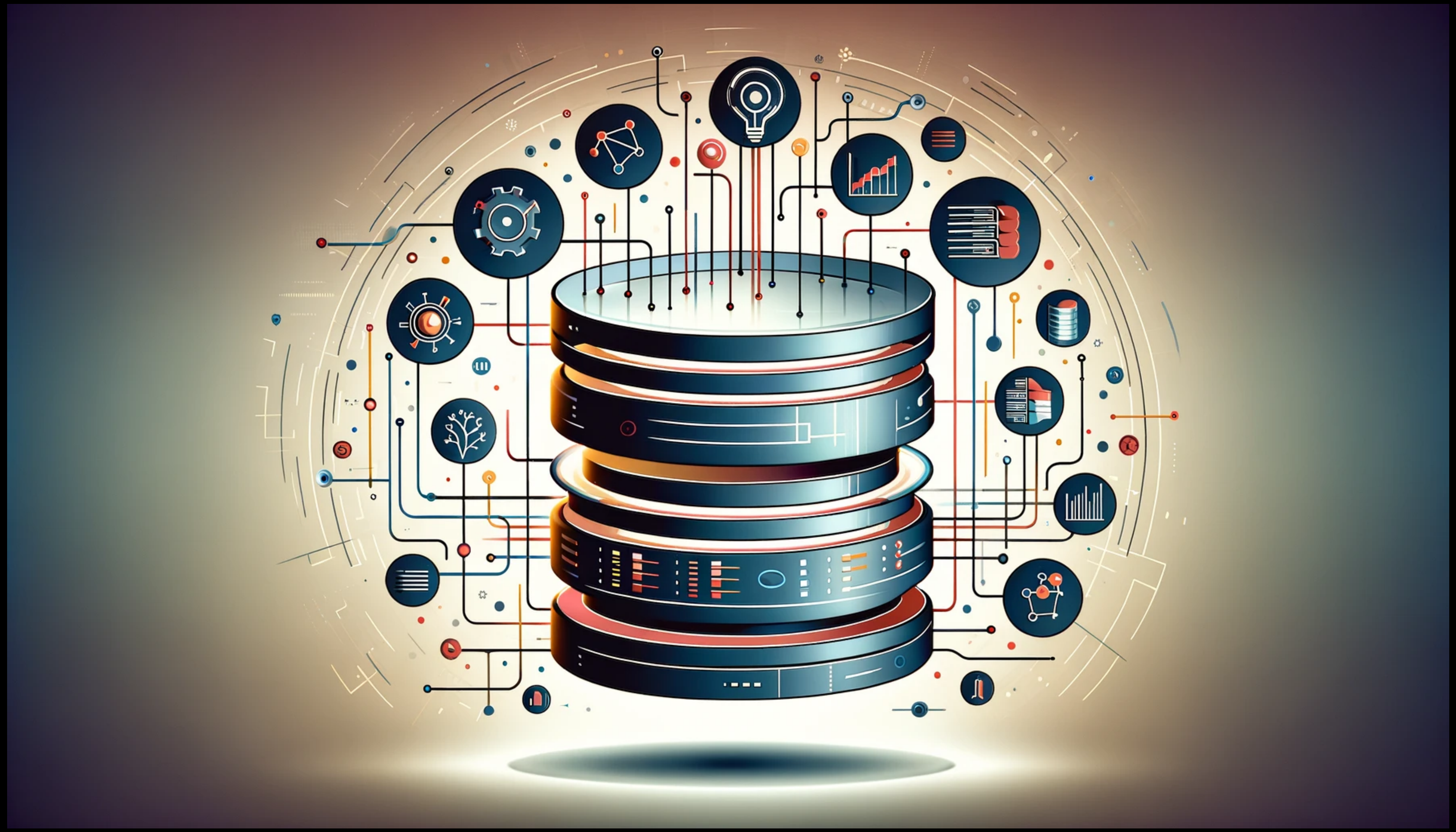
ral knowledge

- Not specific knowledge

- No idea about things that happened after it

What is the issue we are trying to address?

Background: specialized knowledge



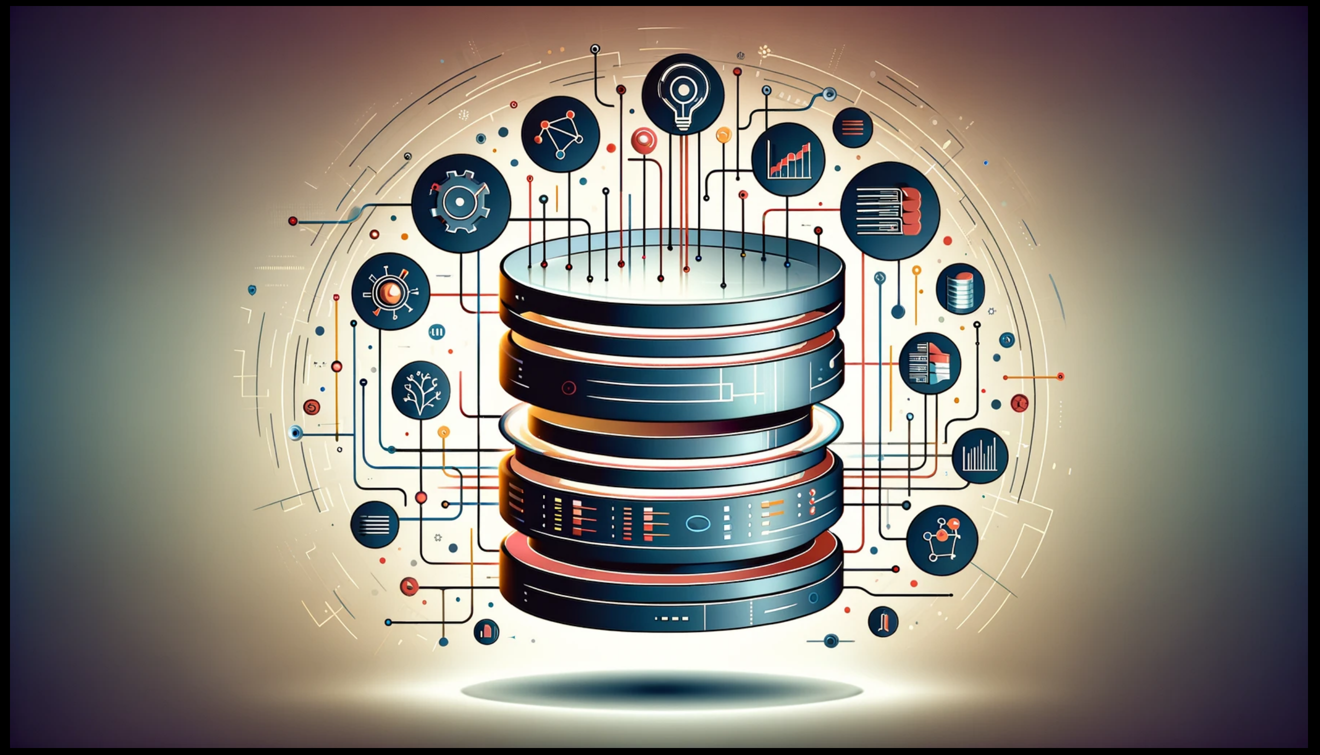
- User confidential documents (private model)?
- Finetuning? RAG? Prompt-engineering?

- Database with proprietary information

What is the issue we are trying to address?

Background: specialized knowledge

Agentic



External data

- Take action!
- Send email

Finetuning? Prompt-engineering? RAG?

Vs

🦜 🔗

Components of lanchain

Embeddings & indexing

	battle	horse	king	man	queen	..	woman
authority	0	0.01	1	0.2	1	...	0.2
event	1	0	0	0	0	...	0
has tail?	0	1	0	0	0	...	0
rich	0	0.1	1	0.3	1	...	0.2
gender	0	1	-1	-1	1	...	1

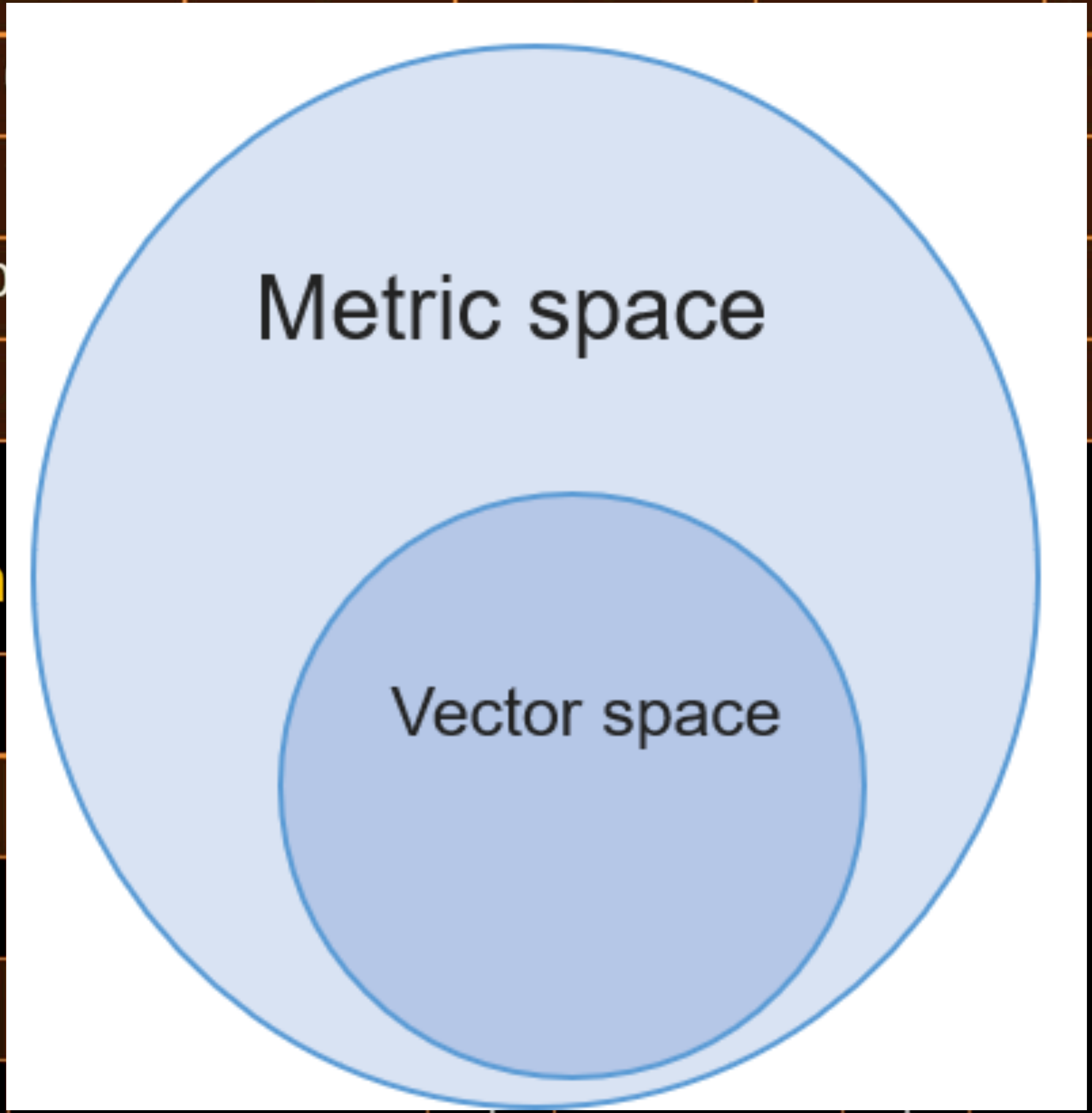
$$\overset{\text{King}}{\begin{bmatrix}1\\0\\0\\1\\-1\end{bmatrix}} \quad \overset{\text{- man}}{\begin{bmatrix}0.2\\0\\0\\0.3\\-1\end{bmatrix}} \quad \overset{\text{+ woman}}{\begin{bmatrix}0.2\\0\\0\\0.2\\1\end{bmatrix}} = \begin{bmatrix}1\\0\\0\\0.9\\1\end{bmatrix} \sim \overset{\text{Queen}}{\begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}}$$

Components of lanchain

Embeddings & indexing

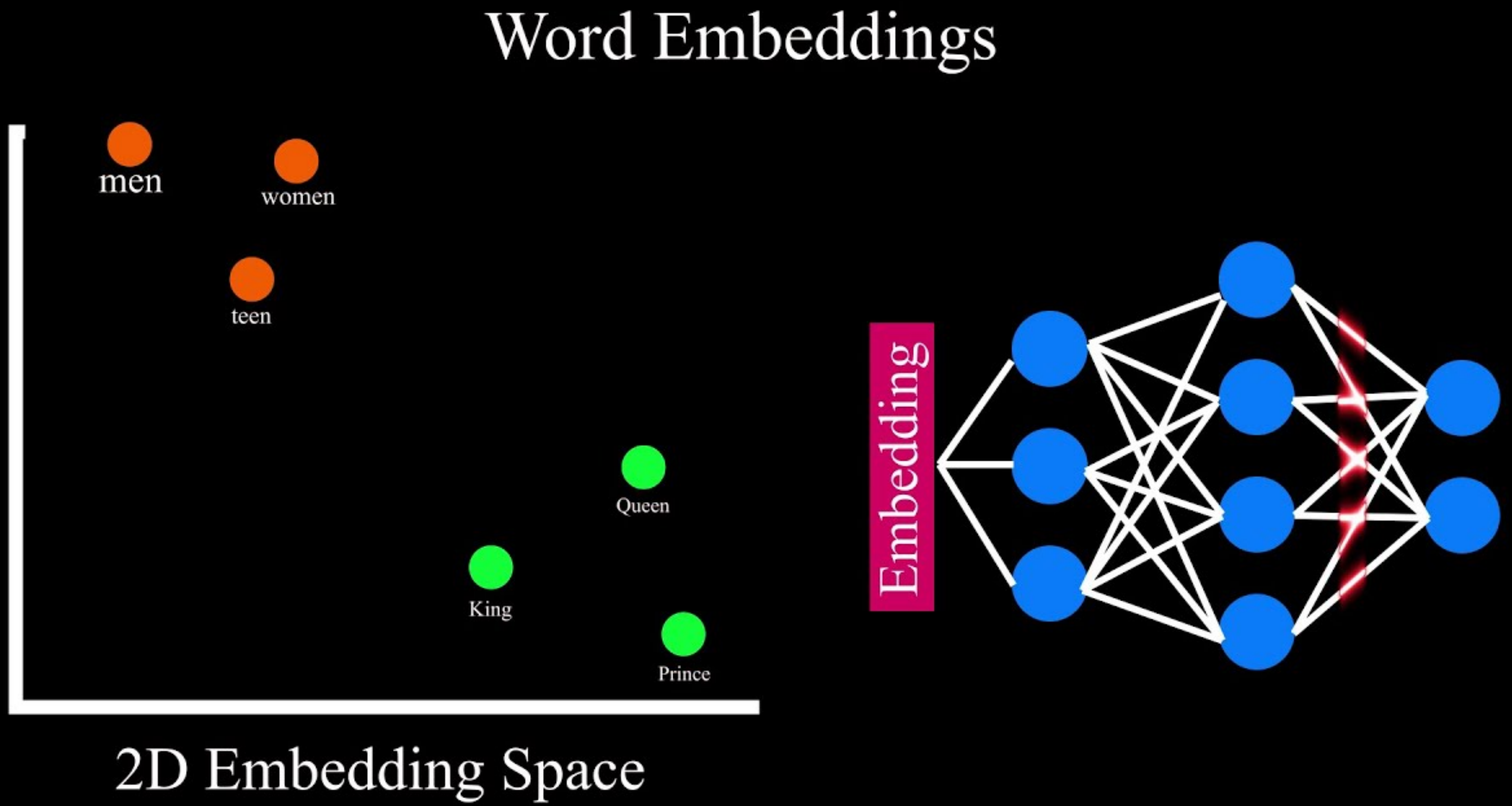
	battle	horse	king	man	queen	..	woman
authority	0	0.01	1	0.2	1	...	0.2
event	1					...	0
has tail?	0					...	0
rich	0	0				...	0.2
gender	0					...	1

King	- man	Queen
1	0.2	1
0	0	0
0	0	0
1	0.3	1
-1	-1	1



Components of lanchain

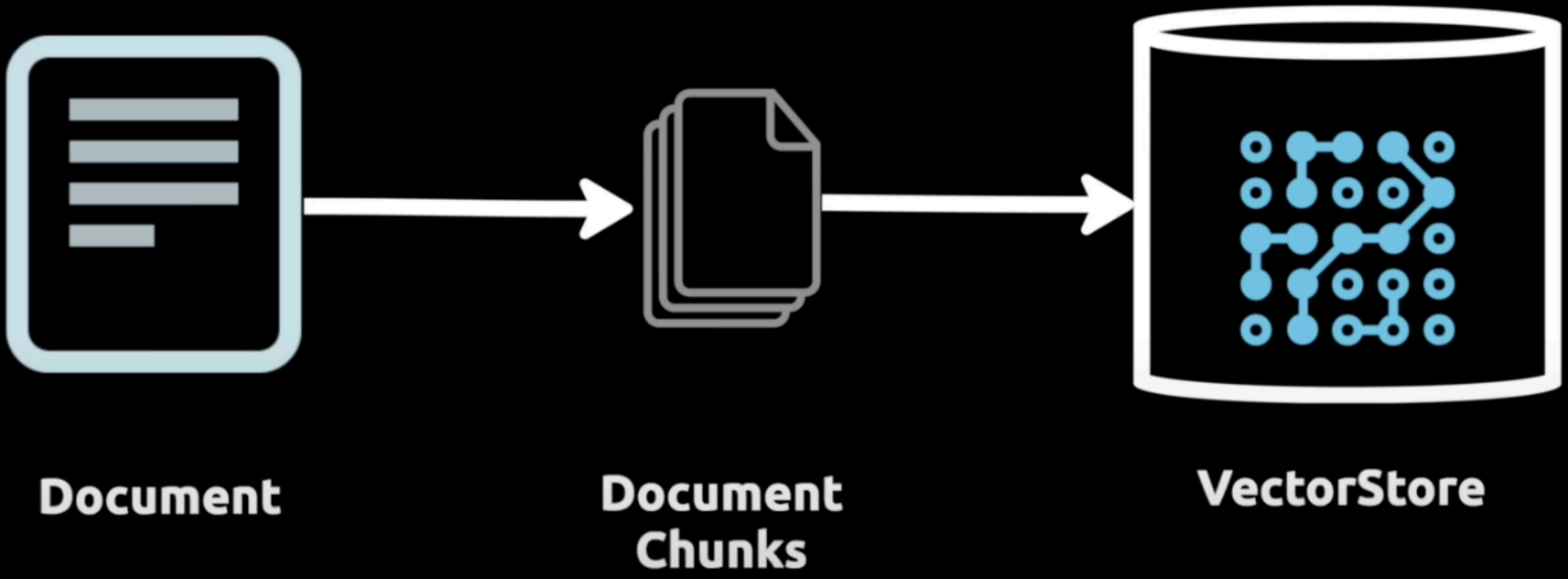
Embeddings & indexing



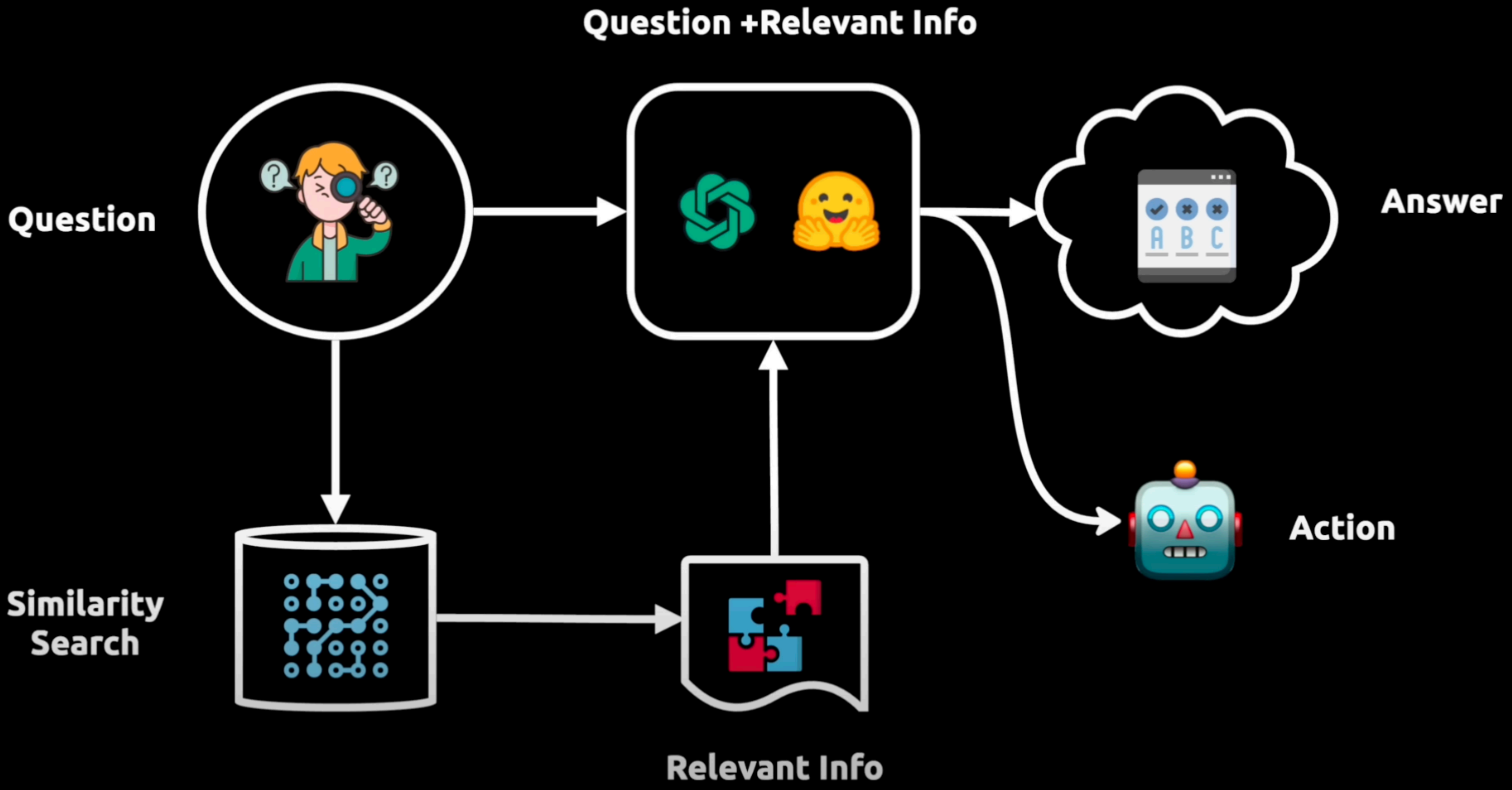
Components of lanchain

Embeddings & indexing

Storing vector representations for text segments



General pipeline 🦜 🔗



🦜 🔗 Data aware + Agentic!

Components

-LLM Wrappers
-RAG Prompts
 (external data)
-Templated prompts

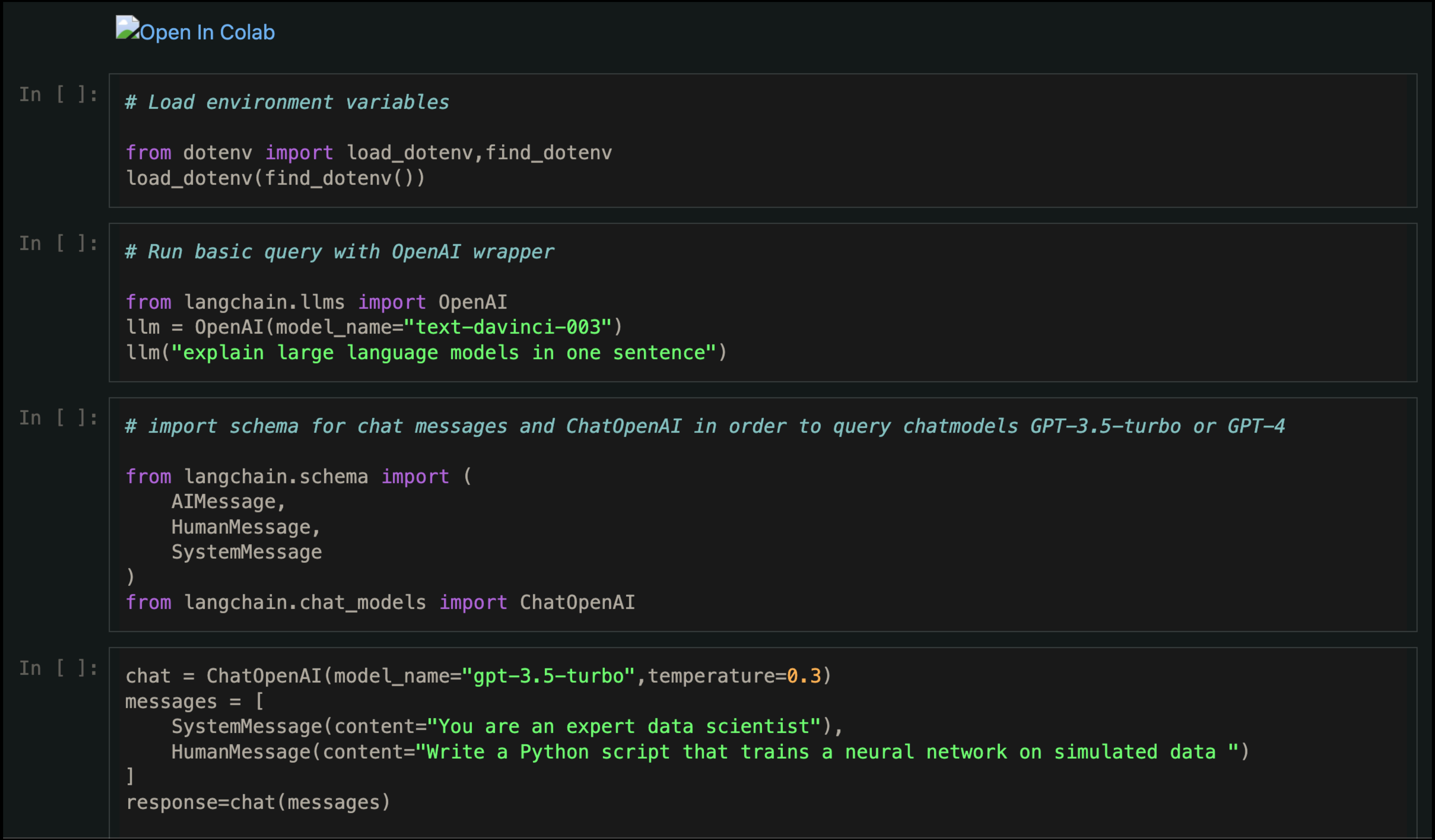
Chains

If loops:
When to make
decision?

Agents

Agent interaction:
-API request
-Consultation

https://github.com/rabbitmetrics/langchain-13-min



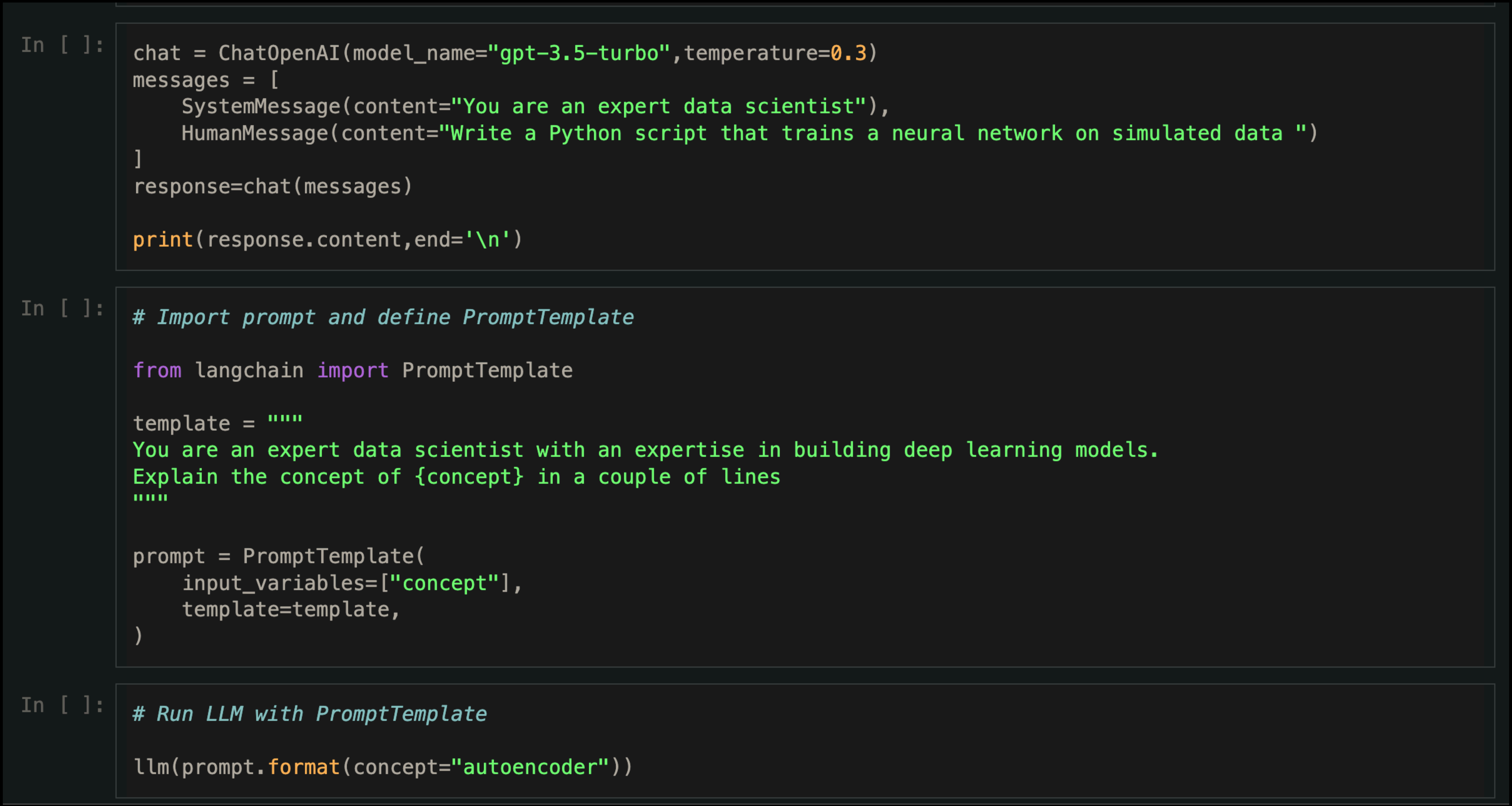
Open In Colab

In [ ]:
```python
# Load environment variables

from dotenv import load_dotenv,find_dotenv
load_dotenv(find_dotenv())
```

In [ ]:
```python
# Run basic query with OpenAI wrapper

from langchain.llms import OpenAI
llm = OpenAI(model_name="text-davinci-003")
llm("explain large language models in one sentence")
```

In [ ]:
```python
# import schema for chat messages and ChatOpenAI in order to query chatmodels GPT-3.5-turbo or GPT-4

from langchain.schema import (
    AIMessage,
    HumanMessage,
    SystemMessage
)
from langchain.chat_models import ChatOpenAI
```

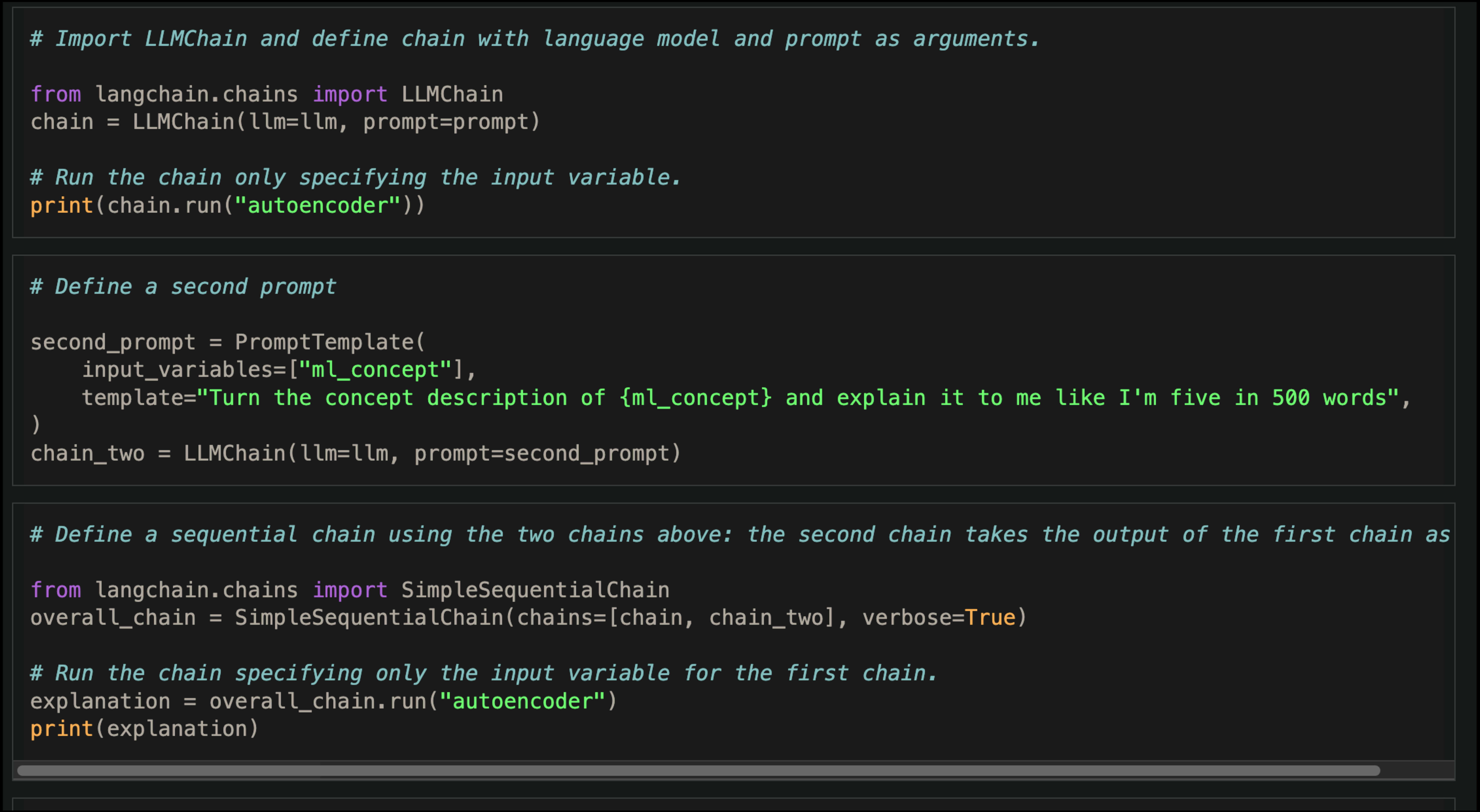
In [ ]:
```python
chat = ChatOpenAI(model_name="gpt-3.5-turbo",temperature=0.3)
messages = [
    SystemMessage(content="You are an expert data scientist"),
    HumanMessage(content="Write a Python script that trains a neural network on simulated data ")
]
response=chat(messages)
```



```
In [ ]:
chat = ChatOpenAI(model_name="gpt-3.5-turbo",temperature=0.3)
messages = [
    SystemMessage(content="You are an expert data scientist"),
    HumanMessage(content="Write a Python script that trains a neural network on simulated data ")
]
response=chat(messages)

print(response.content,end='\n')
```

```
In [ ]:
# Import prompt and define PromptTemplate

from langchain import PromptTemplate

template = """
You are an expert data scientist with an expertise in building deep learning models.
Explain the concept of {concept} in a couple of lines
"""

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

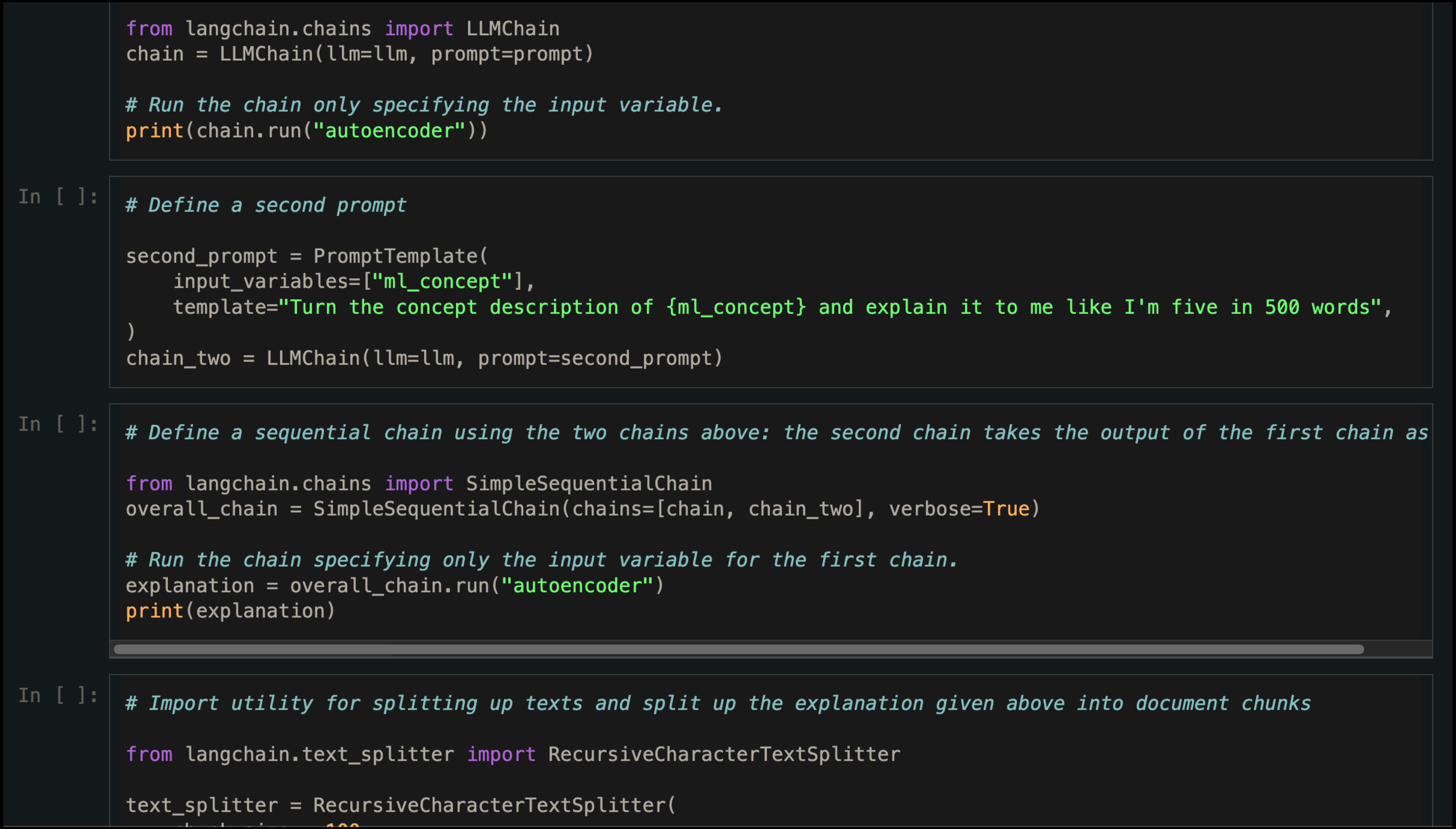
```
In [ ]:
# Run LLM with PromptTemplate

llm(prompt.format(concept="autoencoder"))
```



```python
# Import LLMChain and define chain with language model and prompt as arguments.

from langchain.chains import LLMChain
chain = LLMChain(llm=llm, prompt=prompt)

# Run the chain only specifying the input variable.
print(chain.run("autoencoder"))
```

```python
# Define a second prompt

second_prompt = PromptTemplate(
    input_variables=["ml_concept"],
    template="Turn the concept description of {ml_concept} and explain it to me like I'm five in 500 words",
)
chain_two = LLMChain(llm=llm, prompt=second_prompt)
```

```python
# Define a sequential chain using the two chains above: the second chain takes the output of the first chain as

from langchain.chains import SimpleSequentialChain
overall_chain = SimpleSequentialChain(chains=[chain, chain_two], verbose=True)

# Run the chain specifying only the input variable for the first chain.
explanation = overall_chain.run("autoencoder")
print(explanation)
```

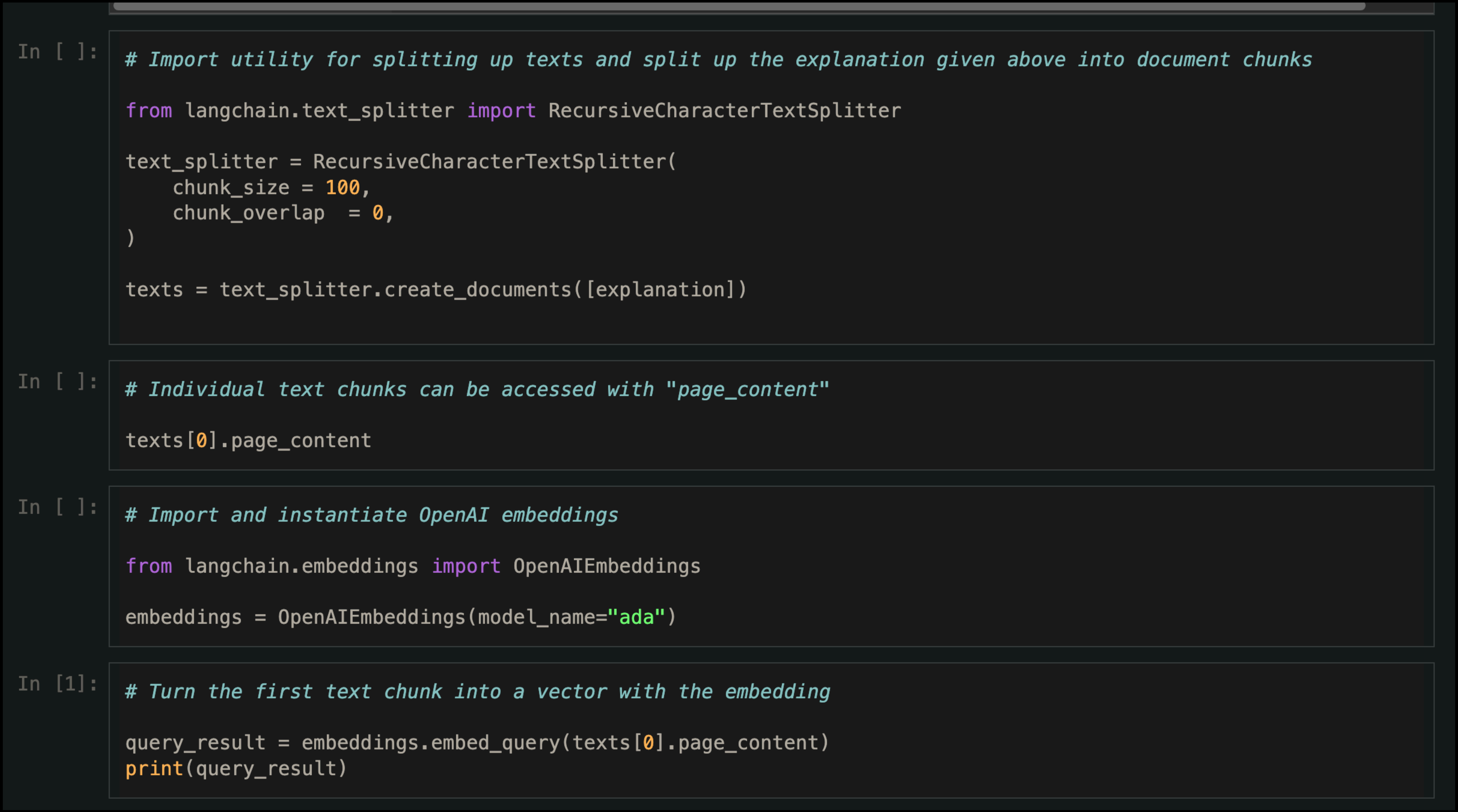


```
from langchain.chains import LLMChain
chain = LLMChain(llm=llm, prompt=prompt)

# Run the chain only specifying the input variable.
print(chain.run("autoencoder"))
```

In [ ]:
```
# Define a second prompt

second_prompt = PromptTemplate(
    input_variables=["ml_concept"],
    template="Turn the concept description of {ml_concept} and explain it to me like I'm five in 500 words",
)
chain_two = LLMChain(llm=llm, prompt=second_prompt)
```

In [ ]:
```
# Define a sequential chain using the two chains above: the second chain takes the output of the first chain as

from langchain.chains import SimpleSequentialChain
overall_chain = SimpleSequentialChain(chains=[chain, chain_two], verbose=True)

# Run the chain specifying only the input variable for the first chain.
explanation = overall_chain.run("autoencoder")
print(explanation)
```

In [ ]:
```
# Import utility for splitting up texts and split up the explanation given above into document chunks

from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
```



In [ ]:

```python
# Import utility for splitting up texts and split up the explanation given above into document chunks

from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 100,
    chunk_overlap  = 0,
)

texts = text_splitter.create_documents([explanation])
```

In [ ]:

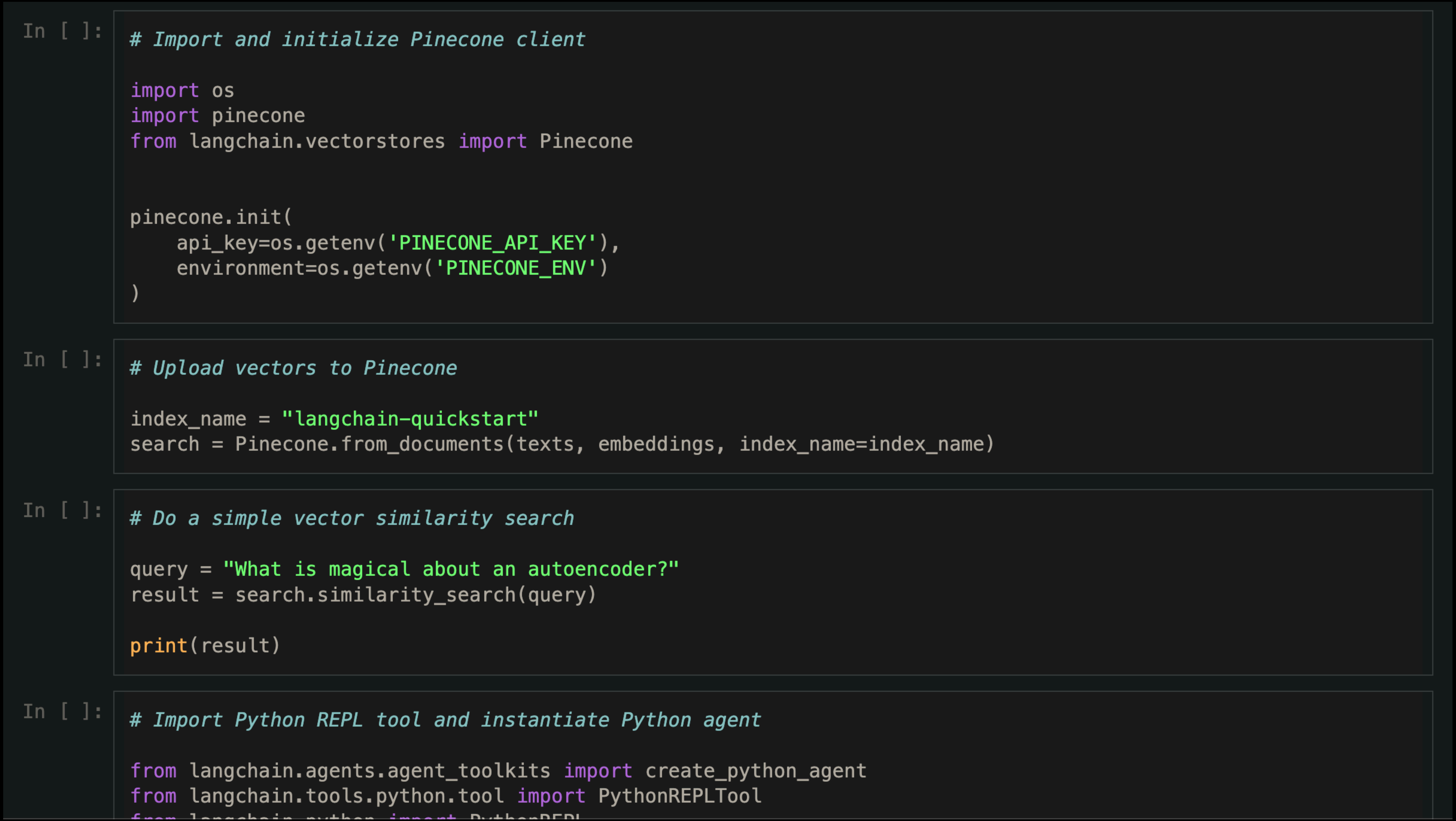
```python
# Individual text chunks can be accessed with "page_content"

texts[0].page_content
```

In [ ]:

```python
# Import and instantiate OpenAI embeddings

from langchain.embeddings import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(model_name="ada")
```

In [1]:

```python
# Turn the first text chunk into a vector with the embedding

query_result = embeddings.embed_query(texts[0].page_content)
print(query_result)
```



```
In [ ]: # Import and initialize Pinecone client

        import os
        import pinecone
        from langchain.vectorstores import Pinecone


        pinecone.init(
            api_key=os.getenv('PINECONE_API_KEY'),
            environment=os.getenv('PINECONE_ENV')
        )
```

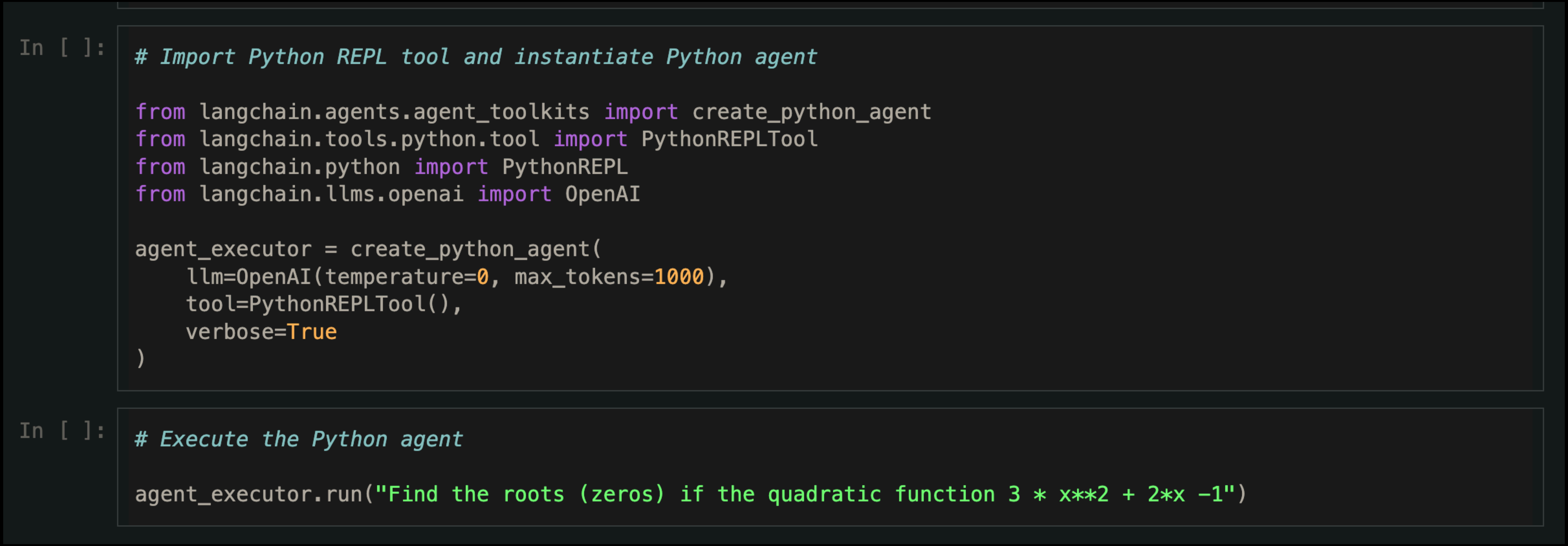
```
In [ ]: # Upload vectors to Pinecone

        index_name = "langchain-quickstart"
        search = Pinecone.from_documents(texts, embeddings, index_name=index_name)
```

```
In [ ]: # Do a simple vector similarity search

        query = "What is magical about an autoencoder?"
        result = search.similarity_search(query)

        print(result)
```

```
In [ ]: # Import Python REPL tool and instantiate Python agent

        from langchain.agents.agent_toolkits import create_python_agent
        from langchain.tools.python.tool import PythonREPLTool
        from langchain.python import PythonREPL
```

Agent execution

Python code interpretor



```
In [ ]: # Import Python REPL tool and instantiate Python agent

        from langchain.agents.agent_toolkits import create_python_agent
        from langchain.tools.python.tool import PythonREPLTool
        from langchain.python import PythonREPL
        from langchain.llms.openai import OpenAI

        agent_executor = create_python_agent(
            llm=OpenAI(temperature=0, max_tokens=1000),
            tool=PythonREPLTool(),
            verbose=True
        )
```

```
In [ ]: # Execute the Python agent

        agent_executor.run("Find the roots (zeros) if the quadratic function 3 * x**2 + 2*x -1")
```

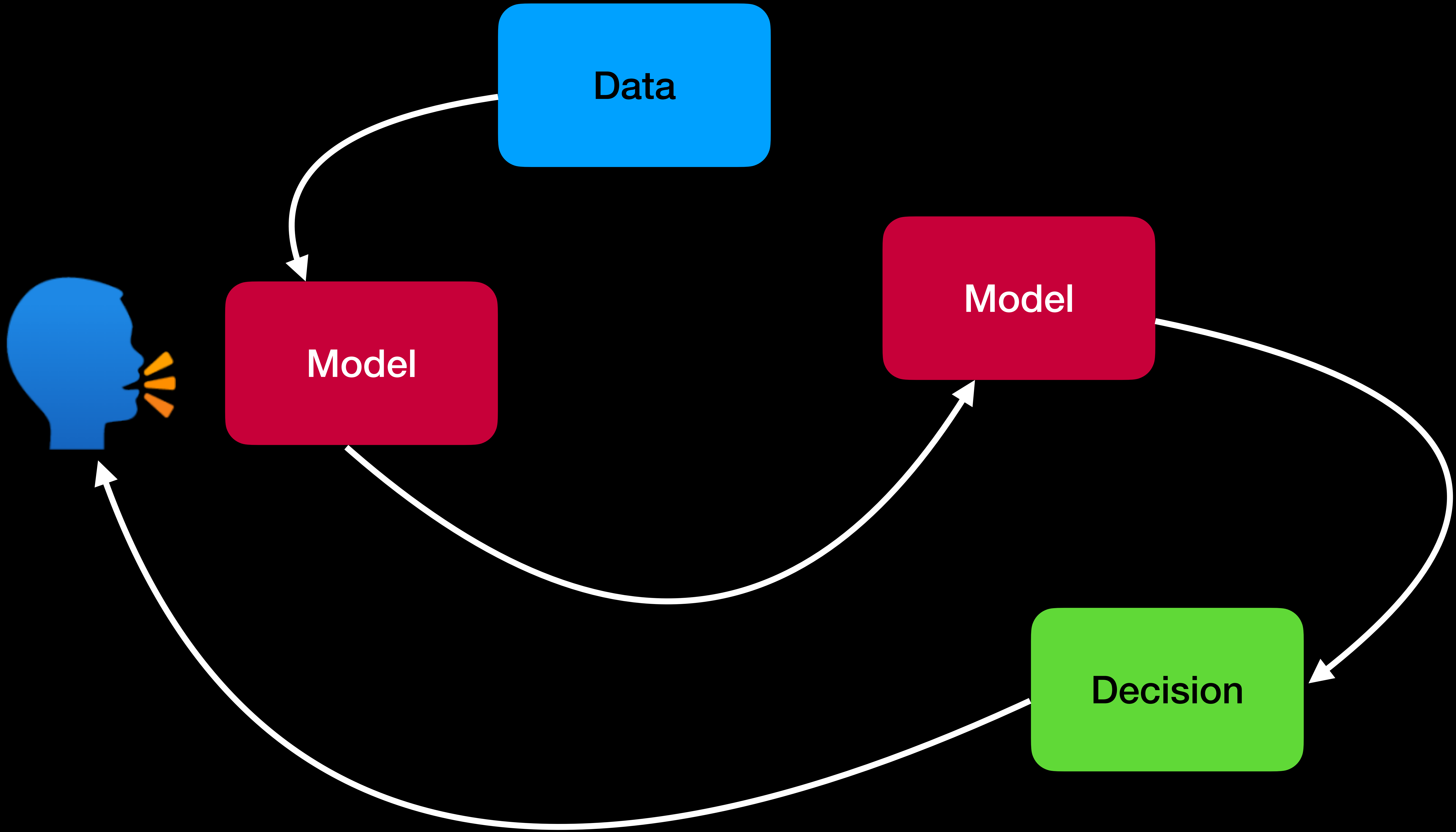
Vectorstores
Document Loaders
Prompts
LangChain
Agents
LLMs
Chains
ProjectPro

What is the issue we are trying to address?

- Starting off with a chat model… make it work with any model…
- Use it as a tutor (how I like to use it as well 😁), talk to it, ‘upgrade our understanding of a topic’… good but not scalable/automated :)
- It’s better to be able to automate tasks! Like learning => leads to => decision making
- Chat model has limited knowledge up to the creation date (give some examples) if you ask it something +1 day after this date, it doesnt know it… Requirement: feed it new data!
- What about private data? (Besides the NY times article retrieved by chatgpt - cite the date) feed it this data which you have access/rights to, as well :)
- Finally, you would like to make ‘decisions’ based on this newly developed understanding 🧠… Requirement: some sort of loop/graph structure!
- For this, we need to connect this ‘source of compution’ to other sources!

What is the issue we are trying to address?

In summary



Further research

- How to take the feedback of the models and perform in-place training :)