RAG Systems & Agents with LangChain

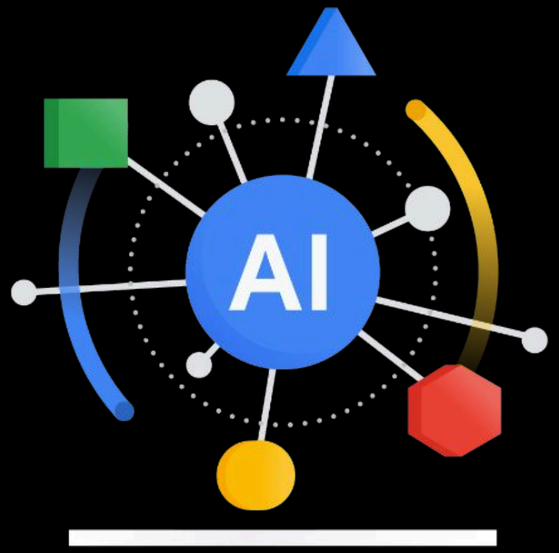
/rumeysskara

/rumeysskara

/rumeysakara

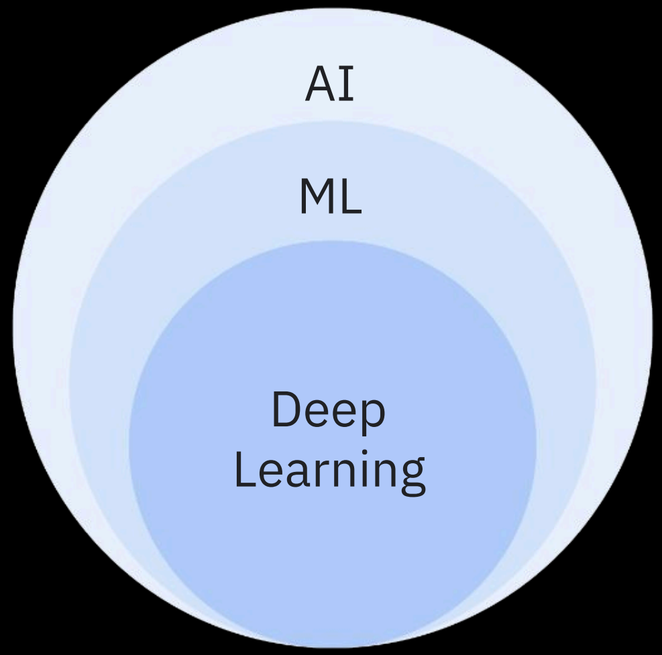
Rümeysa Kara

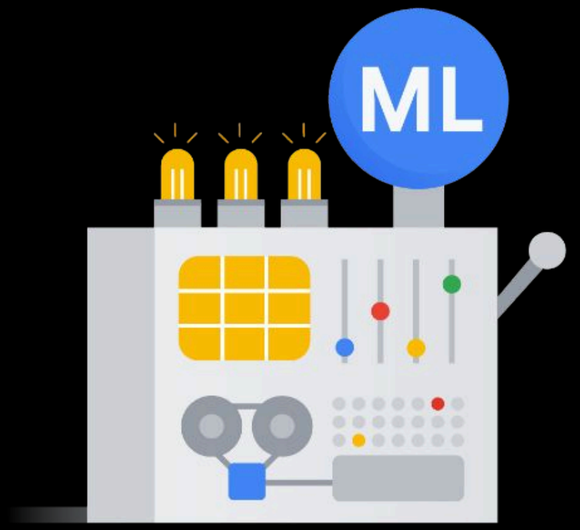
Data Scientist
Google Developer Expert



Artificial Intelligence

is a discipline

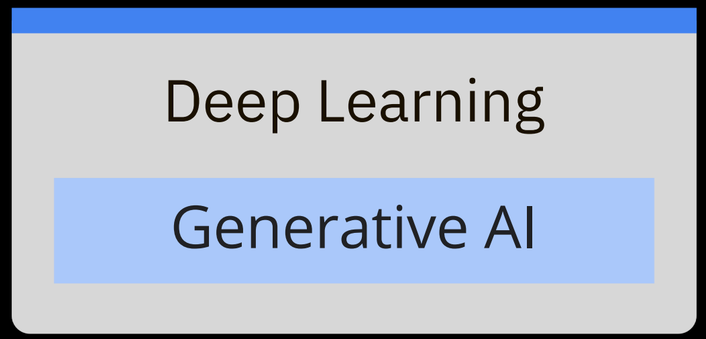




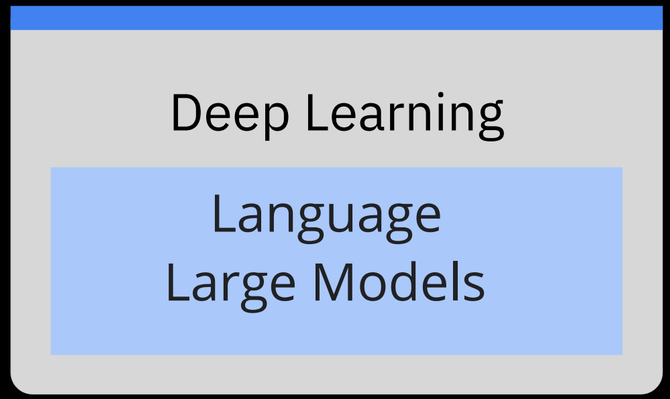
Machine Learning

is a subfield

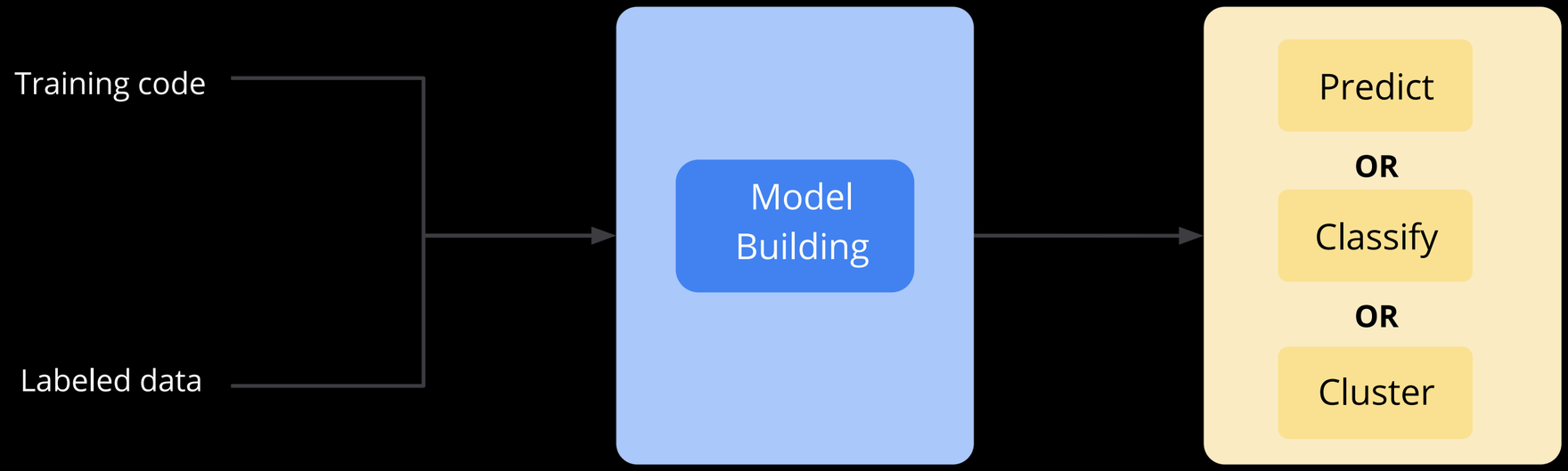
Generative AI is a subset of Deep Learning

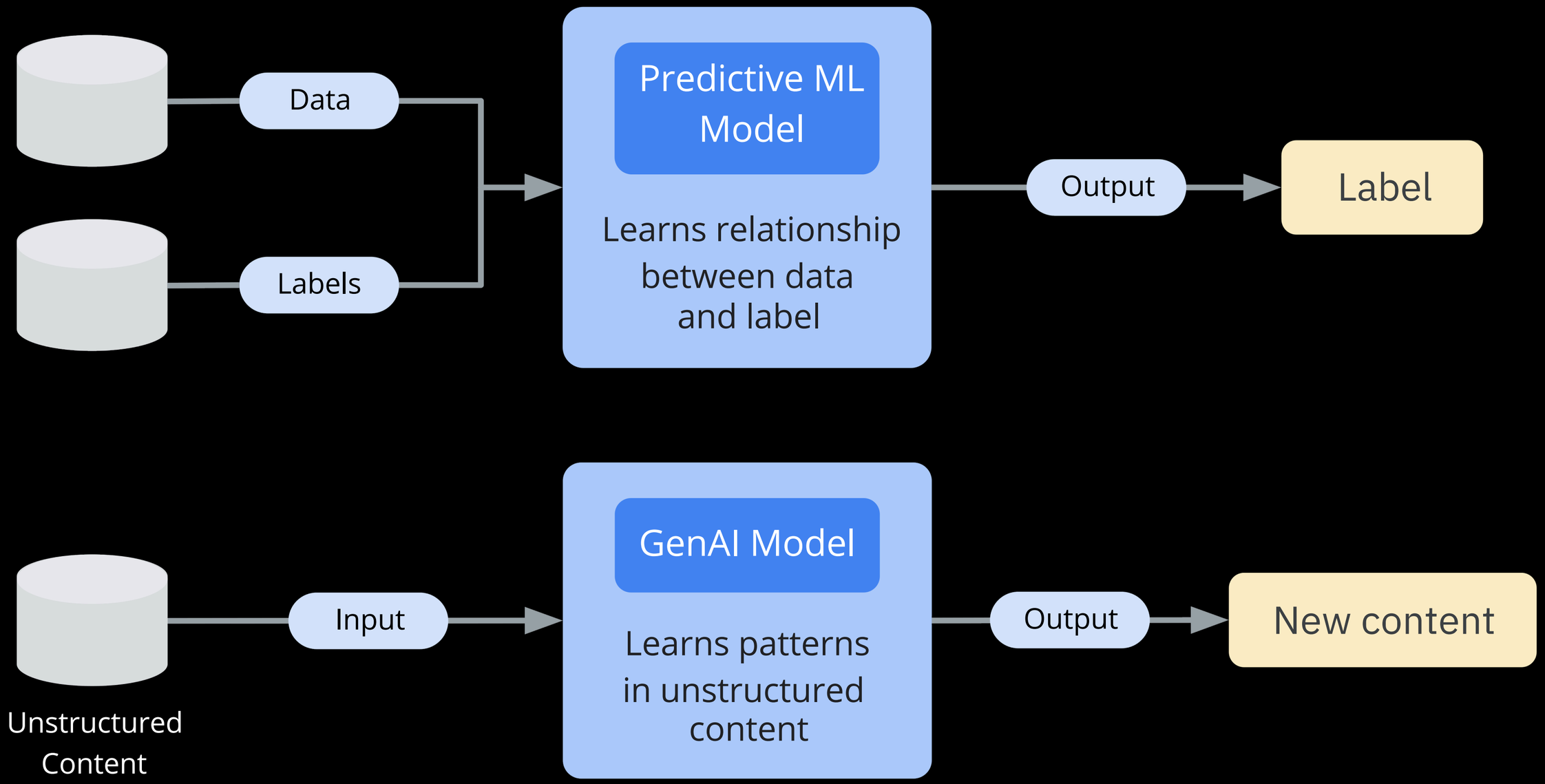


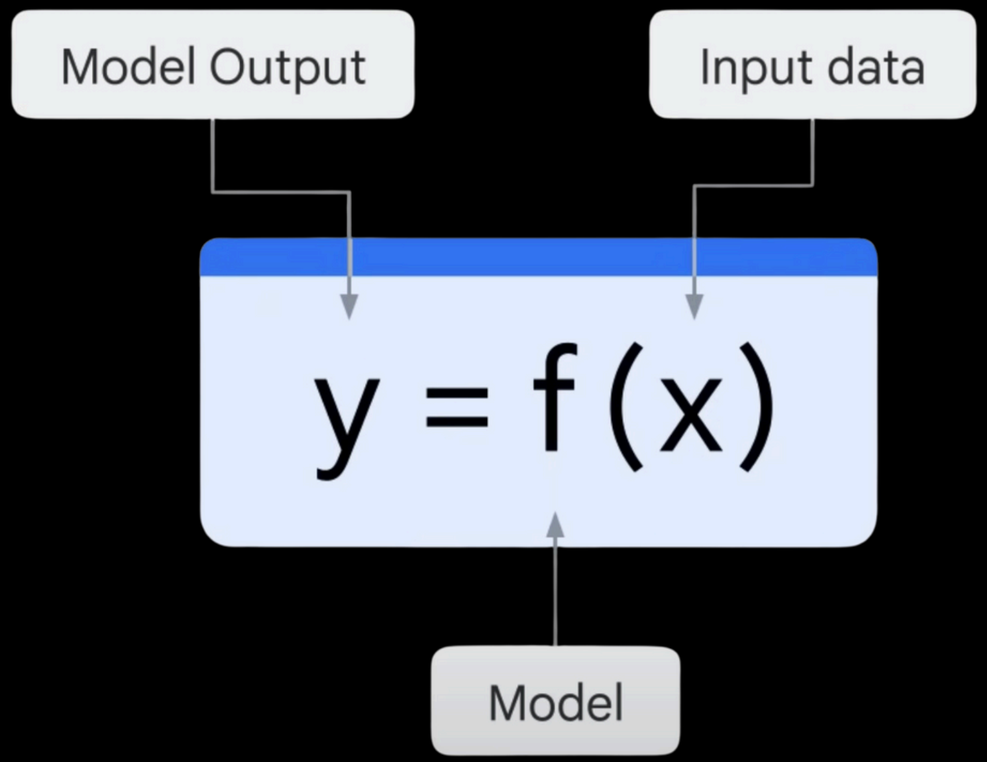
Large Language Models (LLMs) are also a subset of Deep Learning



Classical Supervised & Unsupervised Learning




Data
Labels
Predictive ML Model
Learns relationship between data and label
Output
Label
Input
GenAI Model
Learns patterns in unstructured content
Output
New content
Unstructured Content

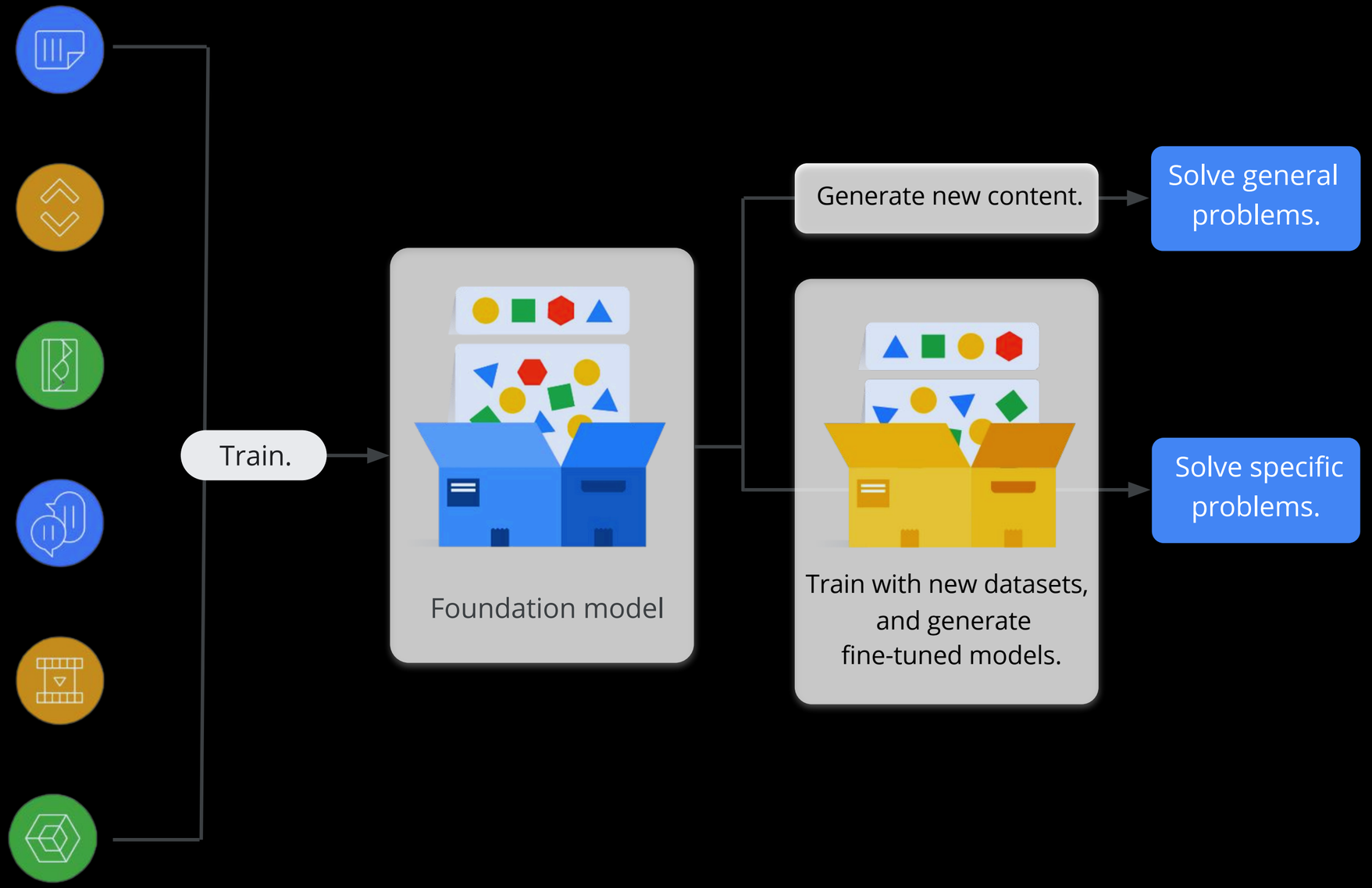


Not GenAI when y is a:

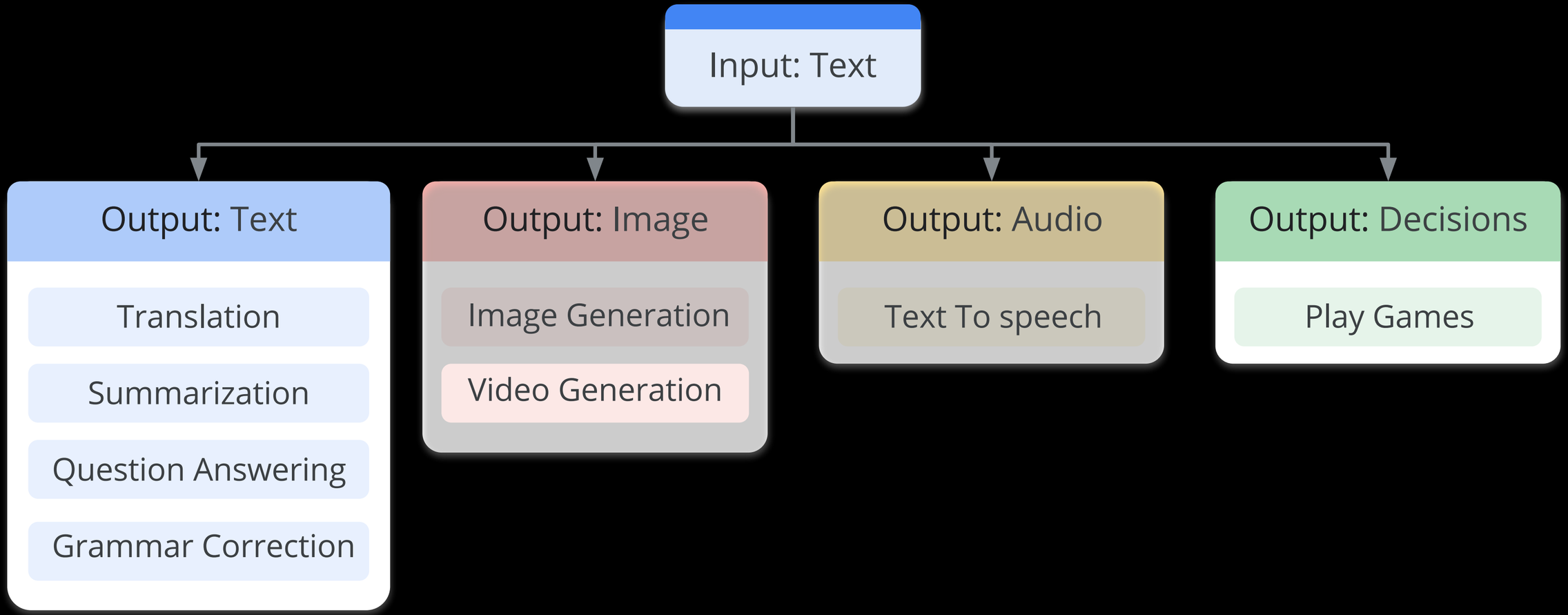
- Number
- Class

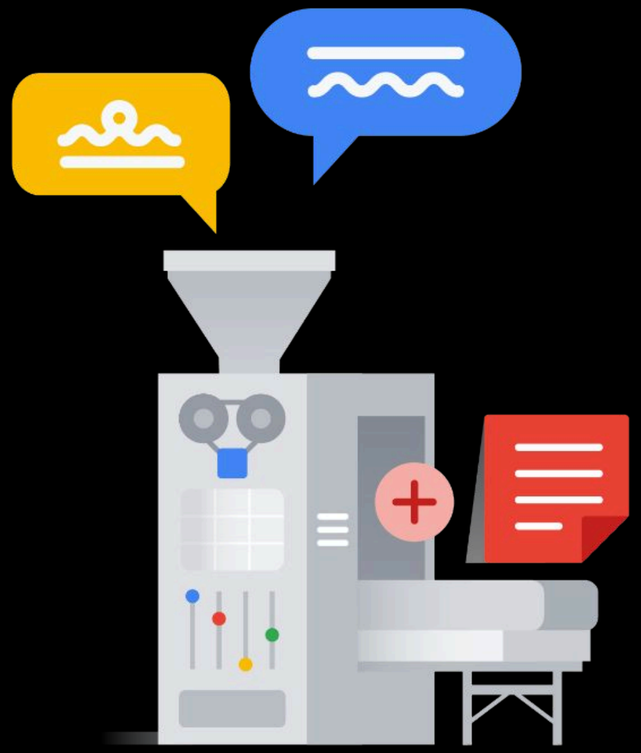
Is GenAI when y is:

- Natural language
- Image
- Audio


Train.
Foundation model
Generate new content.
Solve general problems.
Train with new datasets, and generate fine-tuned models.
Solve specific problems.

Types of Generative AI Based on Data

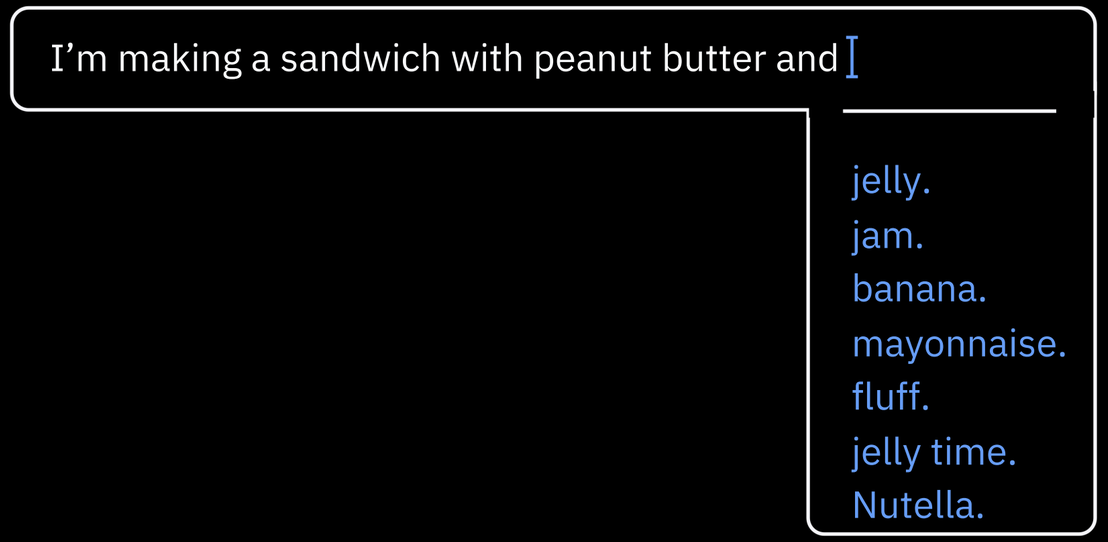




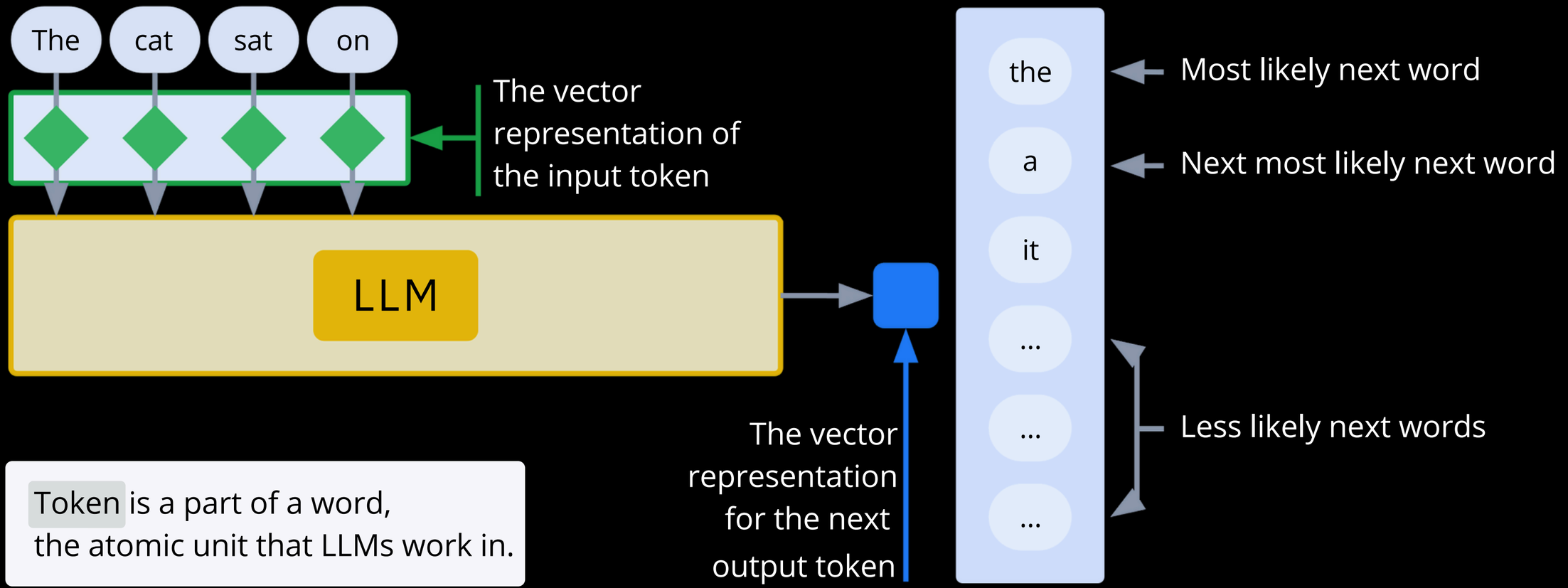
Generative language models learn about patterns in language through training data.

Then, given some text, they predict what comes next.

I’m making a sandwich with peanut butter and


I’m making a sandwich with peanut butter and
jelly.
jam.
banana.
mayonnaise.
fluff.
jelly time.
Nutella.

Generic language model - A next word predictor...

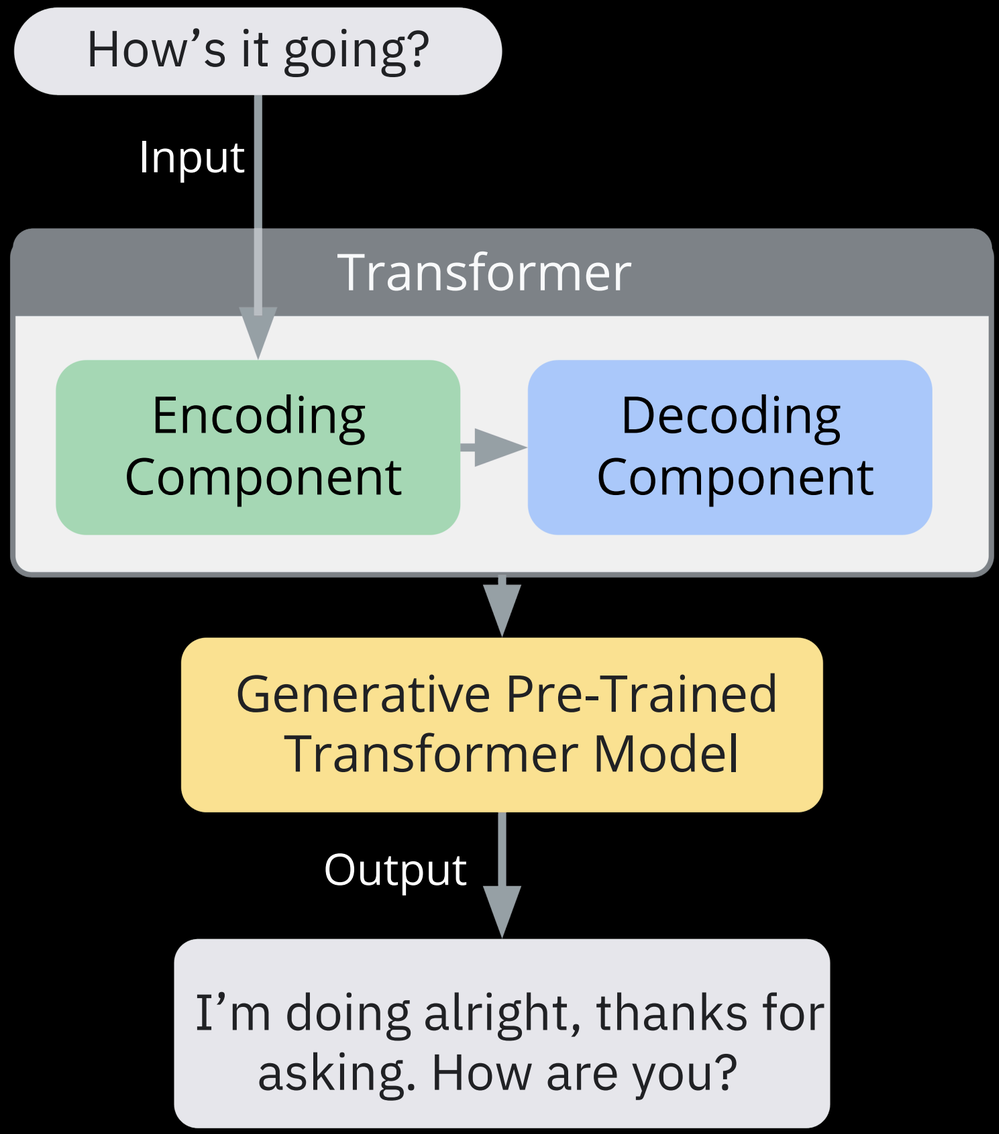


Token is a part of a word,
the atomic unit that LLMs work in.

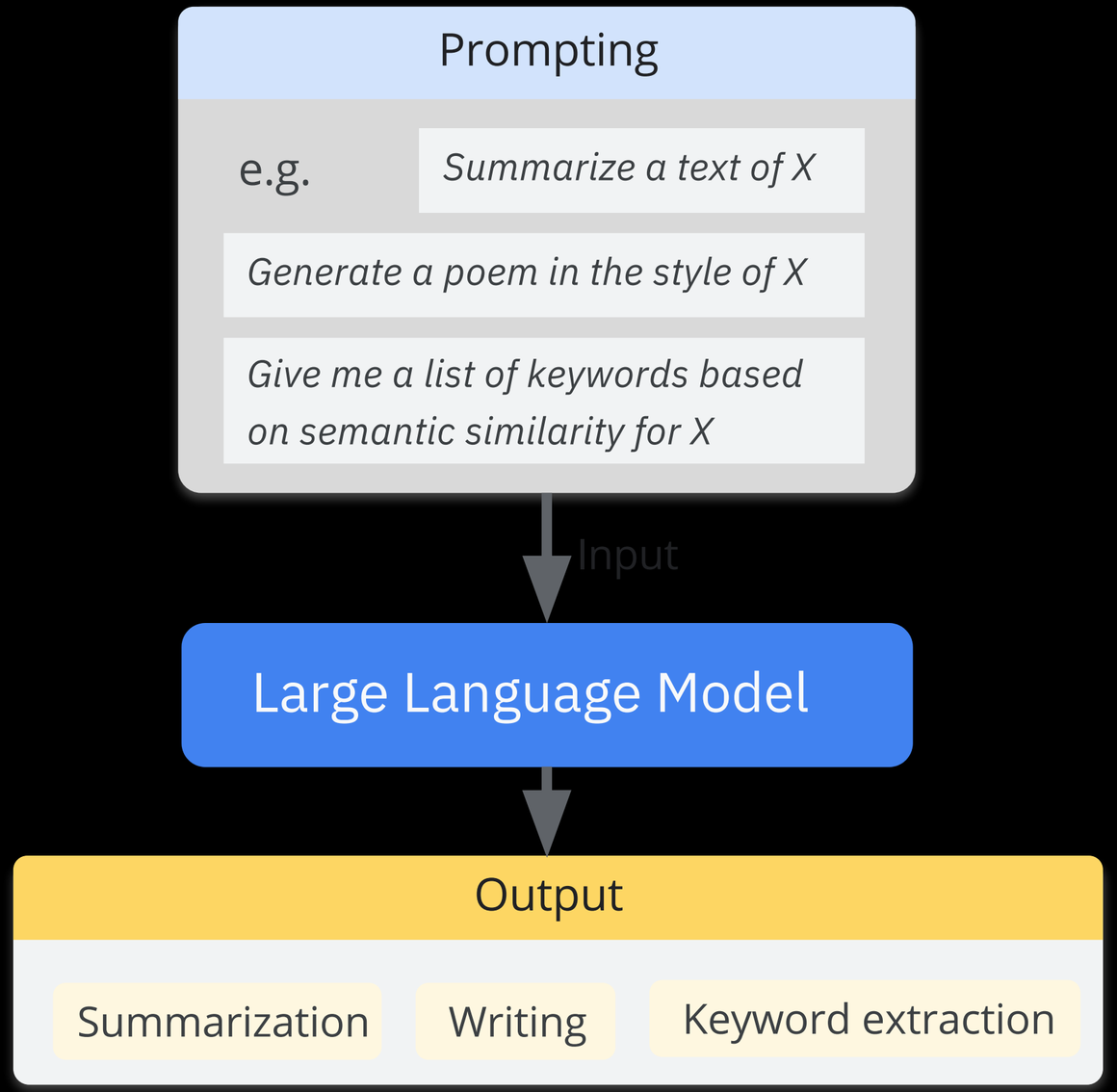
How it Works

Pre-Training:

- Large amount of Data
- Billions of parameters
- Unsupervised learning



Prompt Design: the quality of the input determines the quality of the output.



The **response** from the model

The answers you get depend on the questions you ask.

The **prompt** you designed

LangChain

langchain-core

langchain-community

langchain

Provider	Tool calling	Structured output	JSON mode	Local	Multimodal	Package
ChatOpenAI	✅	✅	✅	❌	✅	langchain-openai
ChatTogether	✅	✅	✅	❌	❌	langchain-together
ChatVertexAI	✅	✅	❌	❌	✅	langchain-google-vertexai
ChatGoogleGenerativeAI	✅	✅	❌	❌	✅	langchain-google-genai
ChatGroq	✅	✅	✅	❌	❌	langchain-groq
ChatCohere	✅	✅	❌	❌	❌	langchain-cohere
ChatBedrock	✅	✅	❌	❌	❌	langchain-aws
ChatHuggingFace	✅	✅	❌	✅	❌	langchain-huggingface

```
%pip install langchain
%pip install langchain-google-genai
```

✓ 1.8s

```python
import os

os.getenv("GOOGLE_API_KEY")
```

✓ 0.0s

```python
os.environ["GOOGLE_API_KEY"]="Your-GOOGLE-API-Key"
```

✓ 0.0s

AI Studio

aistudio.google.com

- Generative API Keys
- Create, test and save prompts
- Customize model in minutes
- Generate starter code

Google AI Studio

Get API key

Create new prompt

New tuned model

My library

Allow Drive access

Prompt Gallery

Developer documentation

Developer forum

Gemini API for Enterprise

Gemini makes mistakes, so double-check it.

Settings

rumeysskara@gmail.com

Untitled prompt

Enable Autosave

Compare

Get code

System Instructions

Optional tone and style instructions for the model

Get started

Try a sample prompt or add your own input below

Which shape comes next?

Given a series of shapes, guess which shape comes next.

Blog post creator

Generate a unique blog post from a single image.

Sentiment analysis

Analyze the sentiment of text messages.

Type something

Run

Run settings

Reset

Model

Gemini 1.5 Flash

Token Count

0 / 1,000,000

Temperature

1

Tools

JSON mode

Edit schema

Code execution

Function calling

Edit functions

Advanced settings

Get API key
Create new prompt
New tuned model
My library
Allow Drive access
Prompt Gallery

Developer documentation
Developer forum
Gemini API for Enterprise

Settings

rumeysskara@gmail.com

API keys

Cloud projects are subject to the Google Cloud Platform Terms of Service, and use of Gemini API and Google AI Studio is subject to the Gemini API Additional Terms of Service.

Remember to use API keys securely. Don't share or embed them in public code. Use of Gemini API from a billing-enabled project is subject to pay-as-you-go pricing.

Quickly test the API by running a cURL command

API quickstart guide

```
curl \
  -H 'Content-Type: application/json' \
  -d '{"contents":[{"parts":[{"text":"Explain how AI works"}]}]}' \
  -X POST 'https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash-latest:generateContent?key=YOUR_API_KEY'
```

Use code with caution.

Create API key

Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.

Project number	Project name	API key	Created	Plan
...0757	Generative Language Client	...XAgc	Sep 30, 2024	Free of charge Set up Billing
		...xqQw	May 13, 2024	View usage data

Access GoogleAI Gemini models such as gemini-pro and gemini-pro-vision through the ChatGoogleGenerativeAI class.

```python
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(model="gemini-pro",
                             temperature = 0.6)
```

```
text = "What is the capital of Azerbaijan?"

response = llm.invoke(text)
response.content
```

```
'Baku'
```

Prompt Template

```python
from langchain.prompts import PromptTemplate

prompt = PromptTemplate.from_template(
    "Tell me a joke about {topic}"
)

prompt.format(topic="Bill Gates")
```

```
'Tell me a joke about Bill Gates'
```

```
from langchain_google_genai import ChatGoogleGenerativeAI

chat_model=ChatGoogleGenerativeAI(model="gemini-pro")

chain = prompt | chat_model

chain.invoke({"topic":"Bill Gates"})
```

```
AIMessage(content="Why did Bill Gates get lost in the desert?\n\nBecause he couldn't find his Windows.",
```

```python
from langchain.prompts.chat import ChatPromptTemplate

template = "You are a helpful assistant that traslates\
 {input_language} to {output_language}."

human_template = "{text}"

chat_prompt = ChatPromptTemplate.from_messages([
    ("system", template),
    ("human", human_template),
])
```

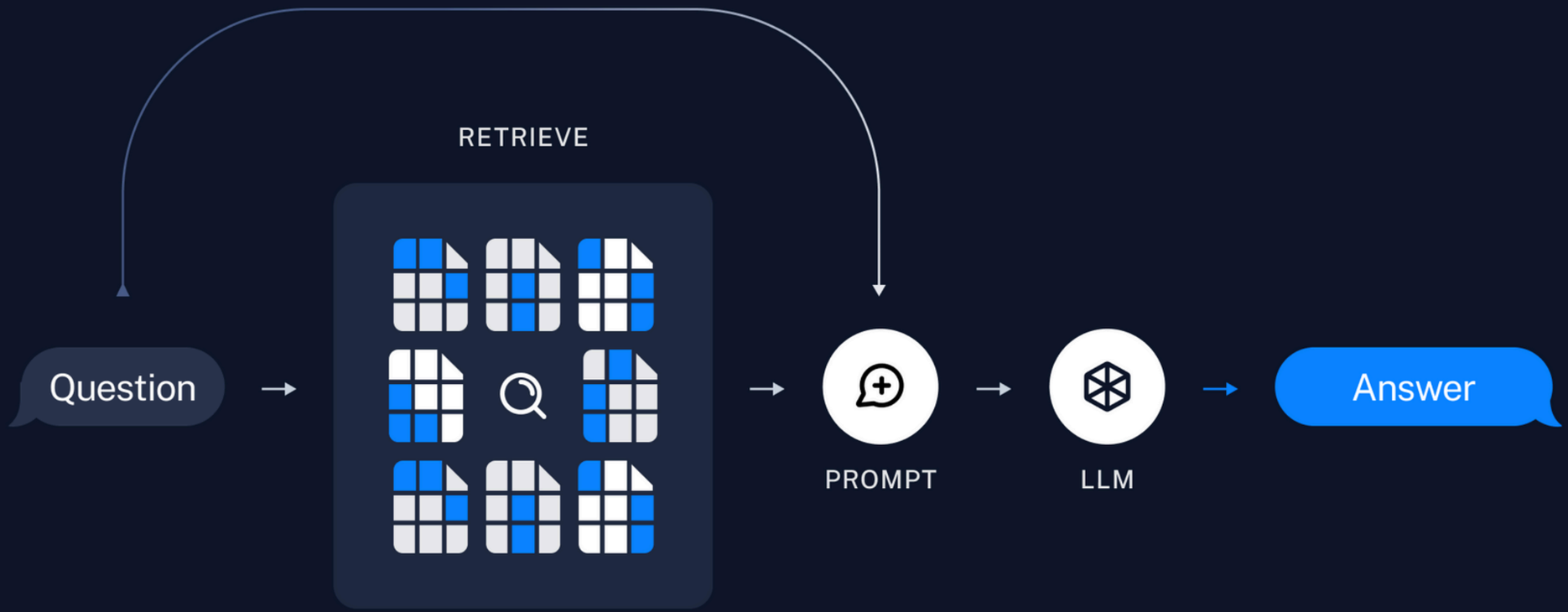
```
chat_prompt.format_messages(
    input_language = "English",
    output_language ="Turkish",
    text = "I love programming."
)
```

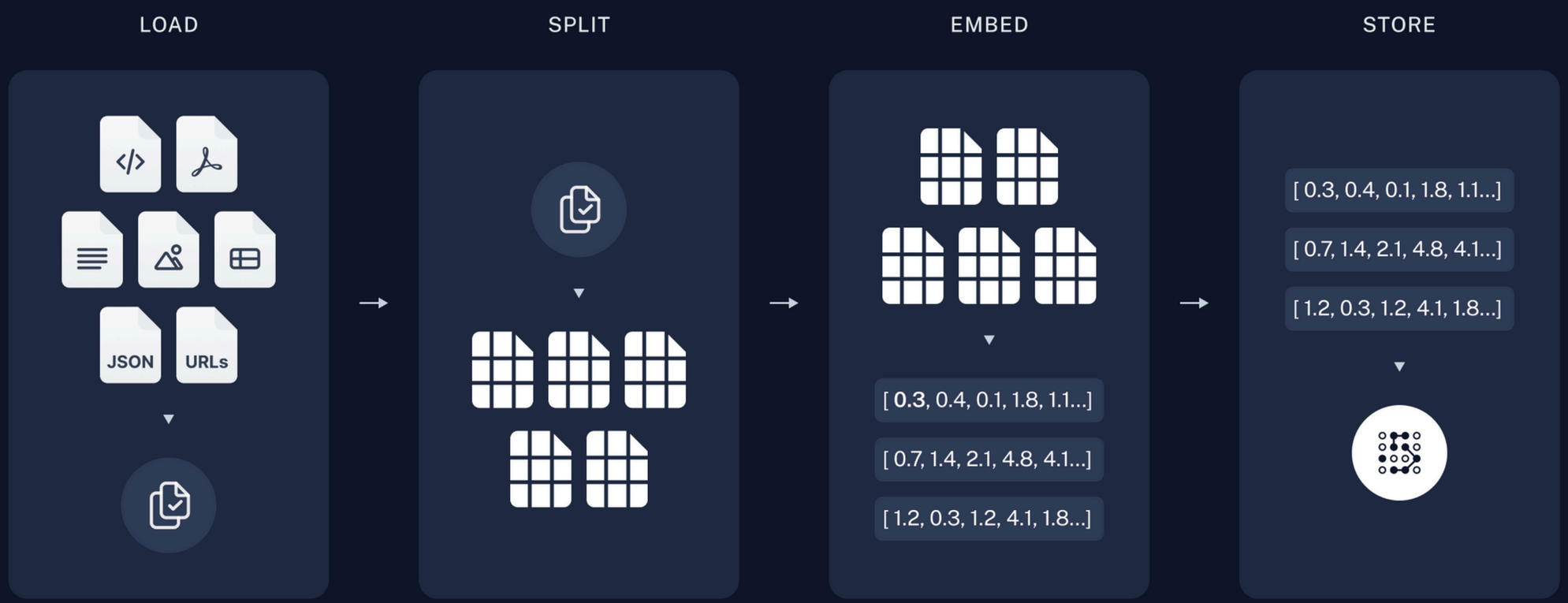
```
[SystemMessage(content='You are a helpful assistant that traslates English to Turkish.', additional_kwargs={}, response_metadata={}),
 HumanMessage(content='I love programming.', additional_kwargs={}, response_metadata={})]
```

```
chain = chat_prompt | llm | output_parser

chain.invoke({
    "input_language" : "English",
    "output_language" : "Turkish",
    "text" : "I love programming."
    })
```

```
'Programlamayı çok seviyorum.'
```

RAG
(Retrieval Augmented Generation)


RETRIEVE
Question
PROMPT
LLM
Answer


LOAD
SPLIT
EMBED
STORE
JSON
URLs
[0.3, 0.4, 0.1, 1.8, 1.1...]
[0.7, 1.4, 2.1, 4.8, 4.1...]
[1.2, 0.3, 1.2, 4.1, 1.8...]
[0.3, 0.4, 0.1, 1.8, 1.1...]
[0.7, 1.4, 2.1, 4.8, 4.1...]
[1.2, 0.3, 1.2, 4.1, 1.8...]

Text Split

```python
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter()

documents = text_splitter.split_documents(docs)
```

Embeddings

```python
from langchain_google_genai import GoogleGenerativeAIEmbeddings
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
```

Vector Databases

FAISS, Chroma, ElasticsearchStore etc.

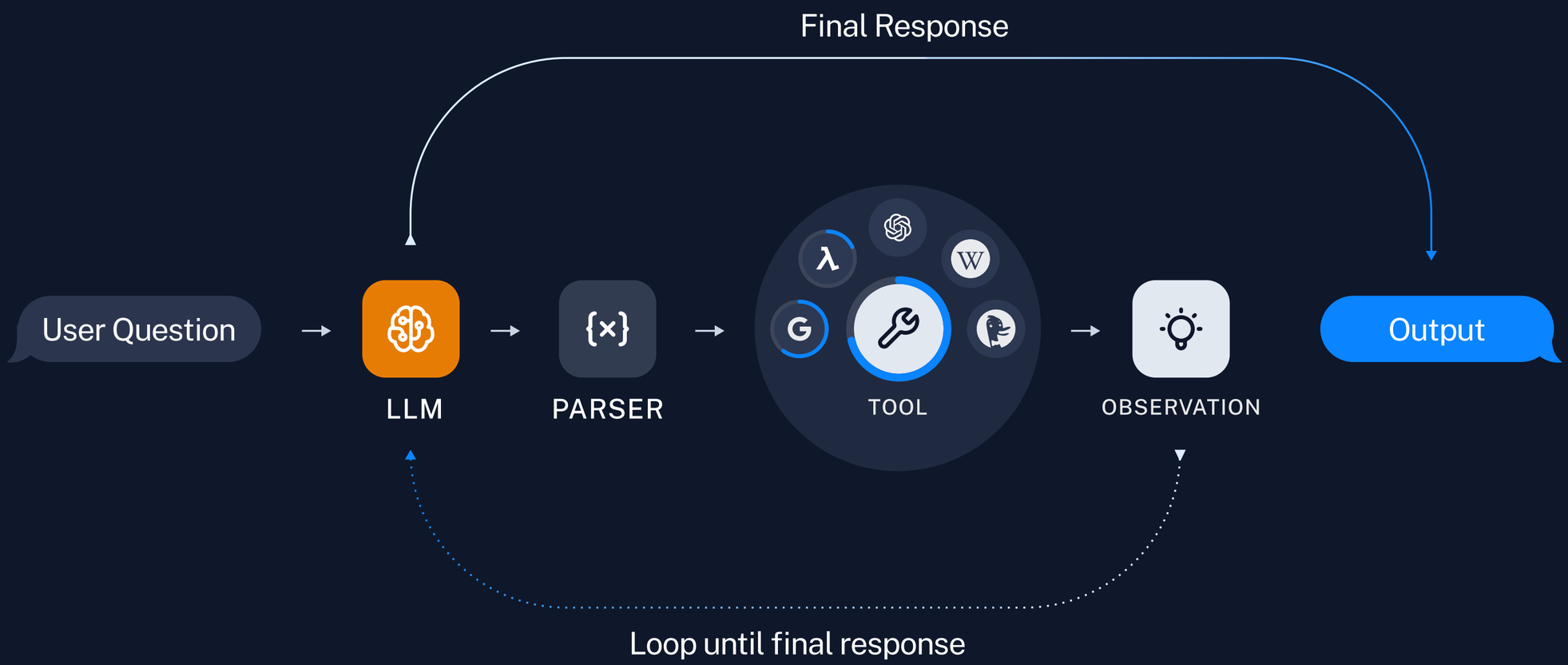
```
from langchain_community.vectorstores import FAISS

vector = FAISS.from_documents(documents, embeddings)

retriever = vector.as_retriever()
```

Conversation Chain

```python
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.prompts import MessagesPlaceholder

prompt = ChatPromptTemplate.from_messages(
    [
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
        ("user", "Given the above conversation, generate a search\
 query to look up in order to get information relevant to the conversaion")
    ]
)
```

```python
from langchain.chains import create_history_aware_retriever

retriver_chain=create_history_aware_retriever(
    model, retriever, prompt
)
```

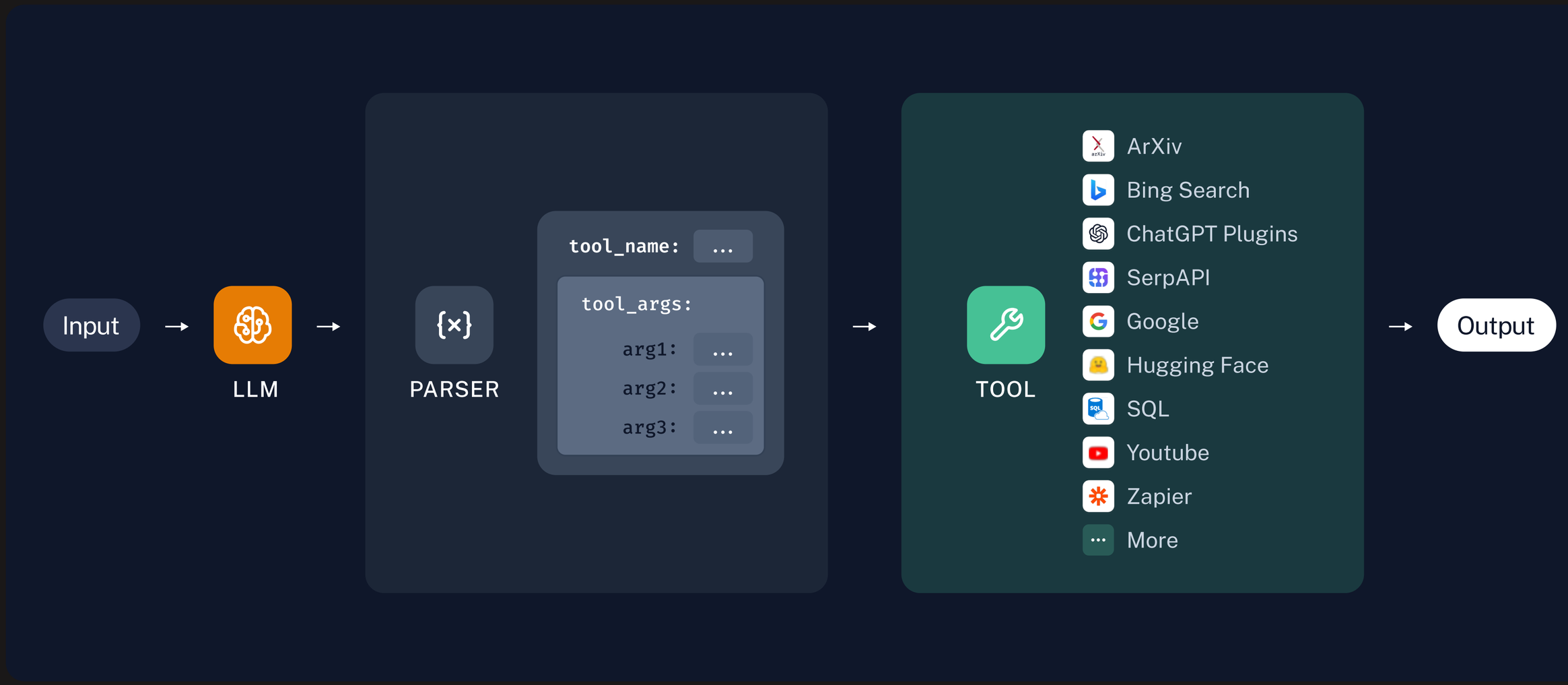
Agents


Final Response
User Question
LLM
PARSER
TOOL
OBSERVATION
Output
Loop until final response

```python
from langchain_google_community import GoogleSearchAPIWrapper

search = GoogleSearchAPIWrapper()

tools = [search]
```

```python
from langchain.agents import create_gemini_functions_agent

agent=create_gemini_functions_agent(model, tools, prompt)
```

```python
from langchain.agents import AgentExecutor

agent_executor = AgentExecutor(
    agent=agent, tools=tools, verbose =True
)
```


Input
LLM
PARSER
tool_name: ...
tool_args:
arg1: ...
arg2: ...
arg3: ...
TOOL
ArXiv
Bing Search
ChatGPT Plugins
SerpAPI
Google
Hugging Face
SQL
Youtube
Zapier
More
Output

Thank you :)

/rumeysskara

/rumeysskara

/rumeysakara

Rümeysa Kara

Data Scientist
Google Developer Expert