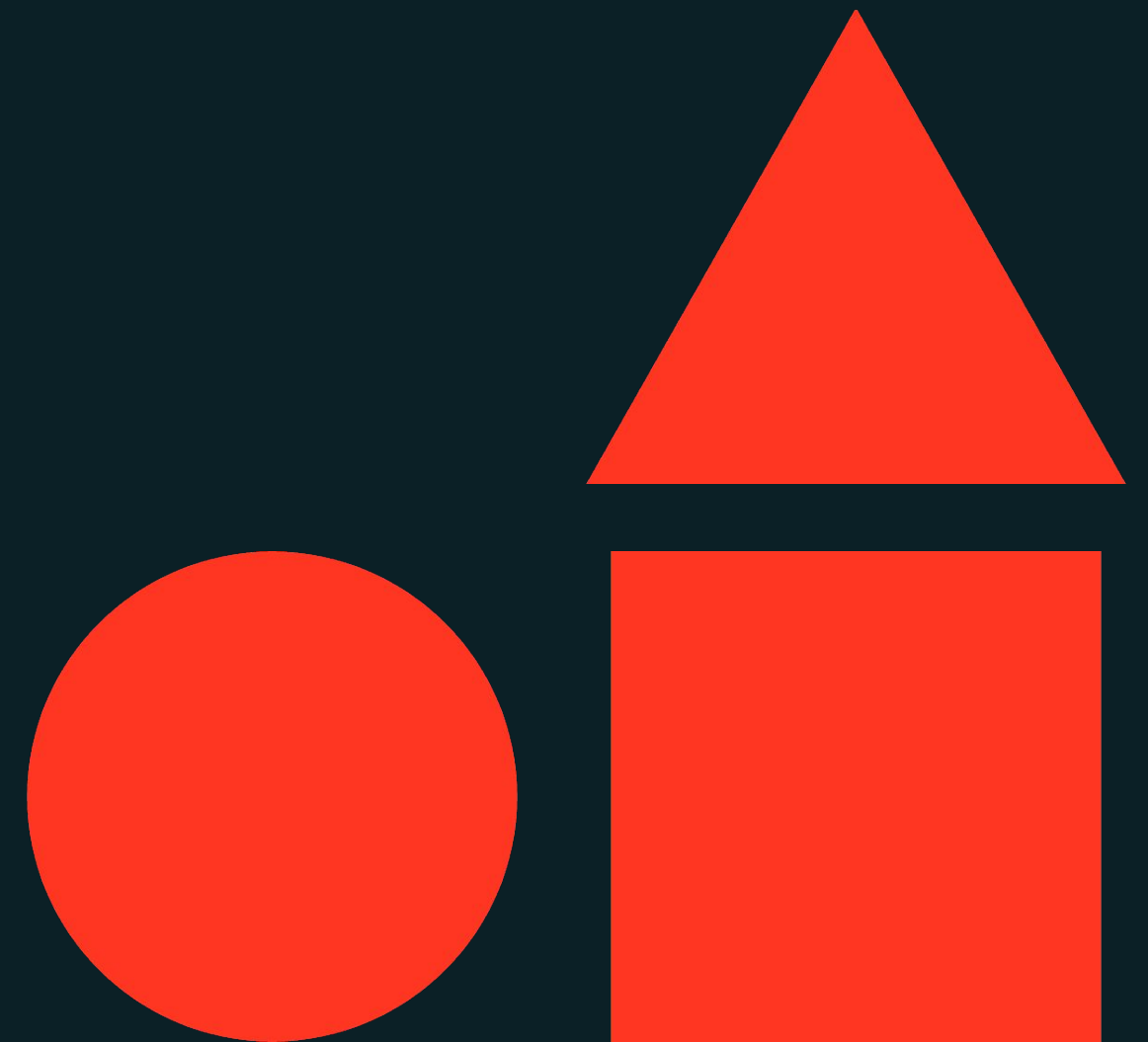




Generative AI Application Development

Databricks Academy

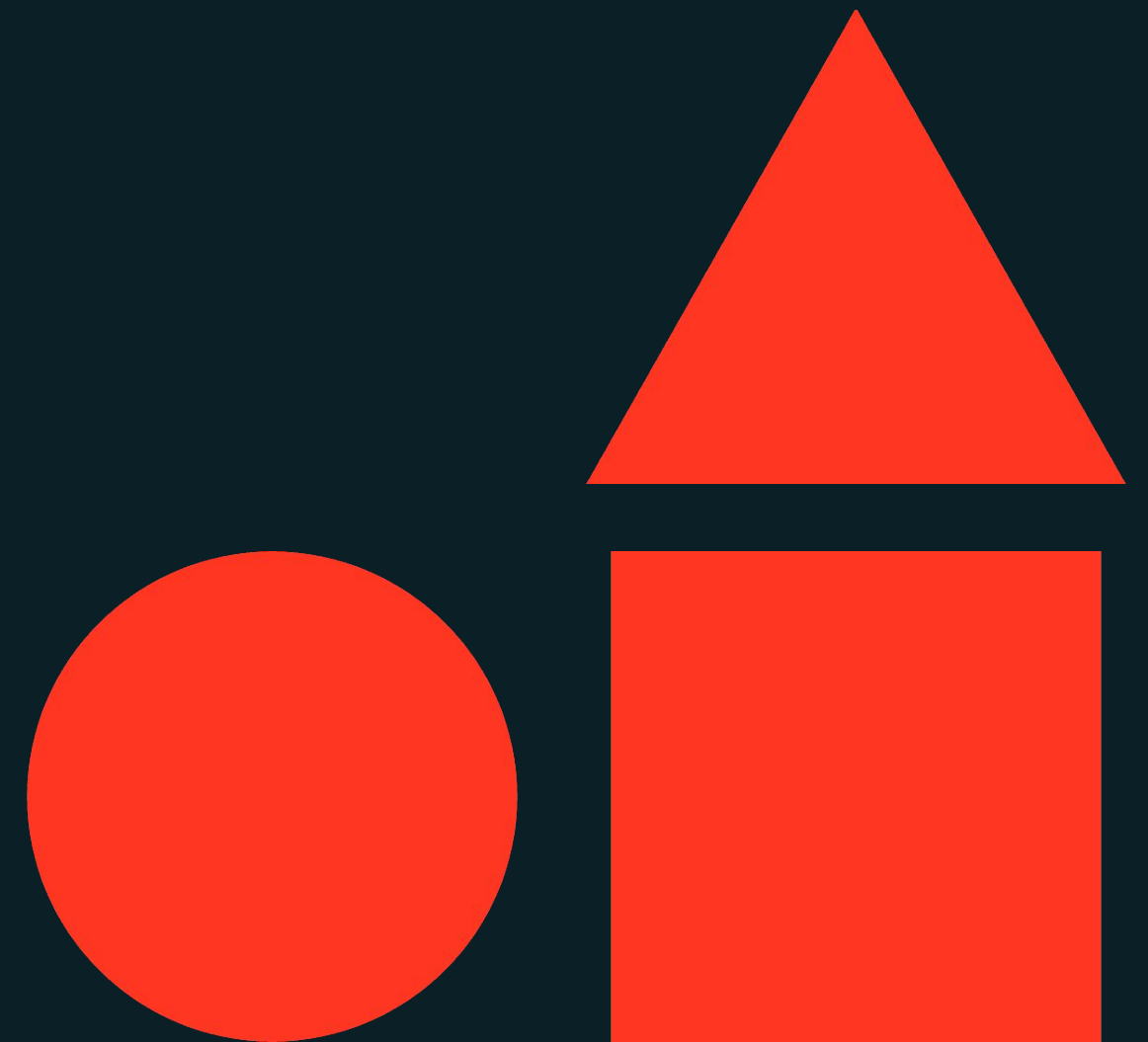


© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark Logo, Apache Iceberg, Iceberg, and the Apache Iceberg logo are trademarks of the [Apache Software Foundation](https://www.apache.org/).



Foundations of Compound AI Systems

Generative AI Application Development



Learning Objectives

- Explain the shift from models to compound AI systems.
- Describe various types and components of compound AI systems.
- Define main concepts such as intent, task, and pipeline.
- Discuss intent classification and chain-building steps.
- Describe the intent behind each prompt in a chain.
- Distinguish between an LLM task and an LLM-based chain.





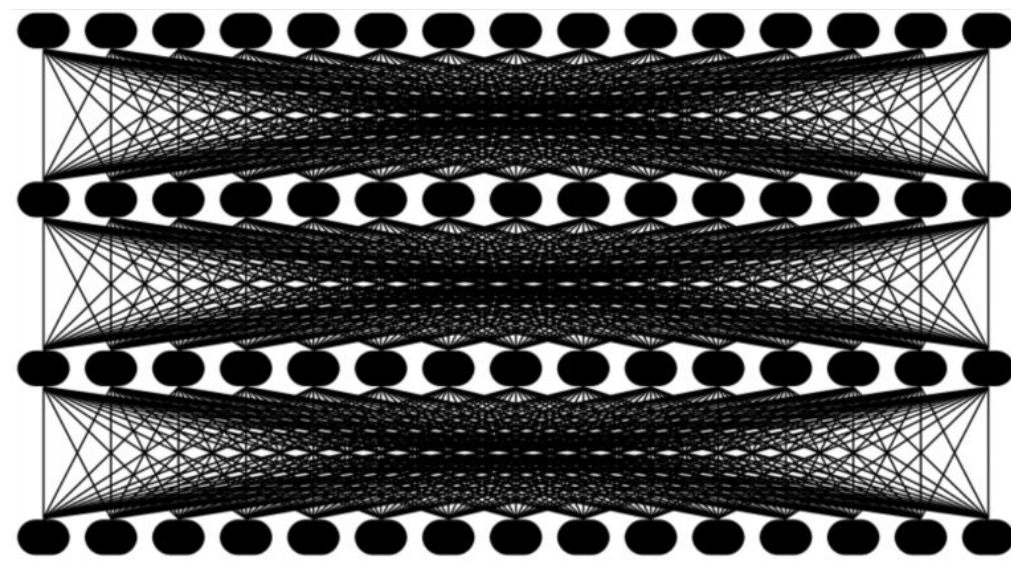
LECTURE

Defining Compound AI Systems

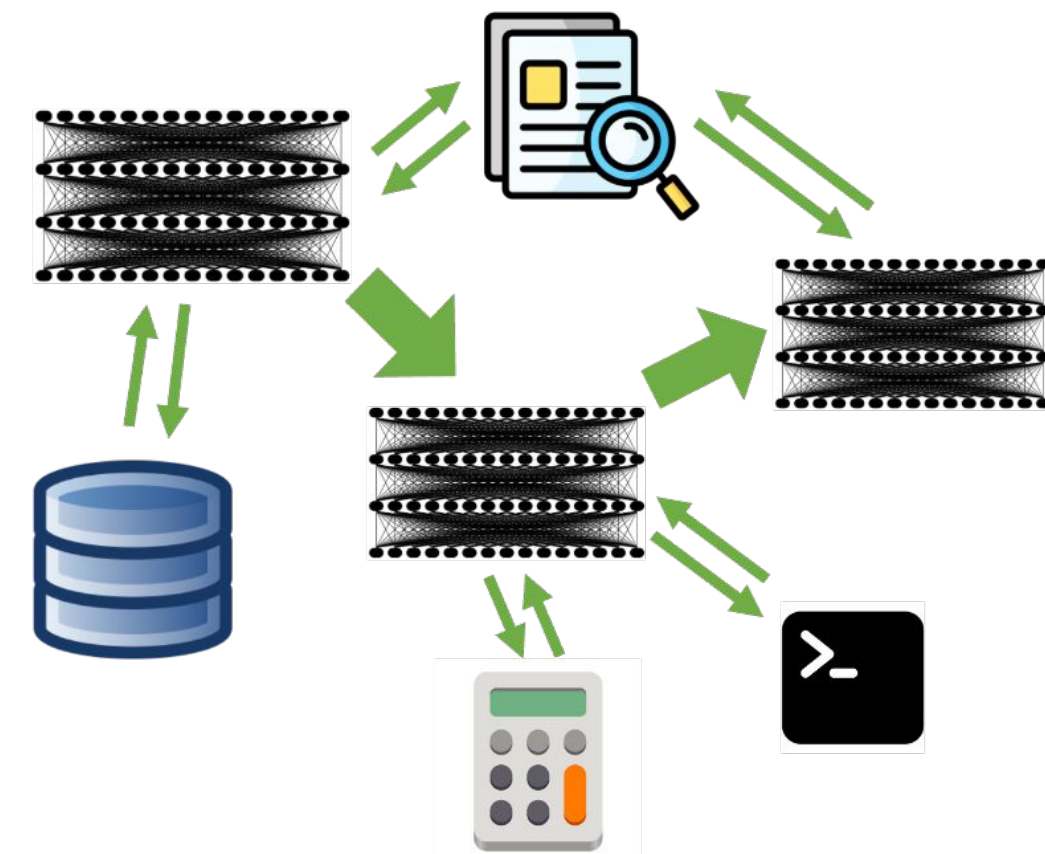
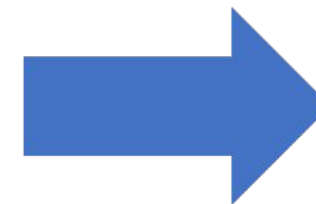


Shift from Models to Compound AI Systems

Compound AI System: A system that tackles AI tasks using **multiple interacting components**, including multiple calls to models, retrievers, or external tools.



Model



System (Chain)



Compound AI Systems

There are various AI systems

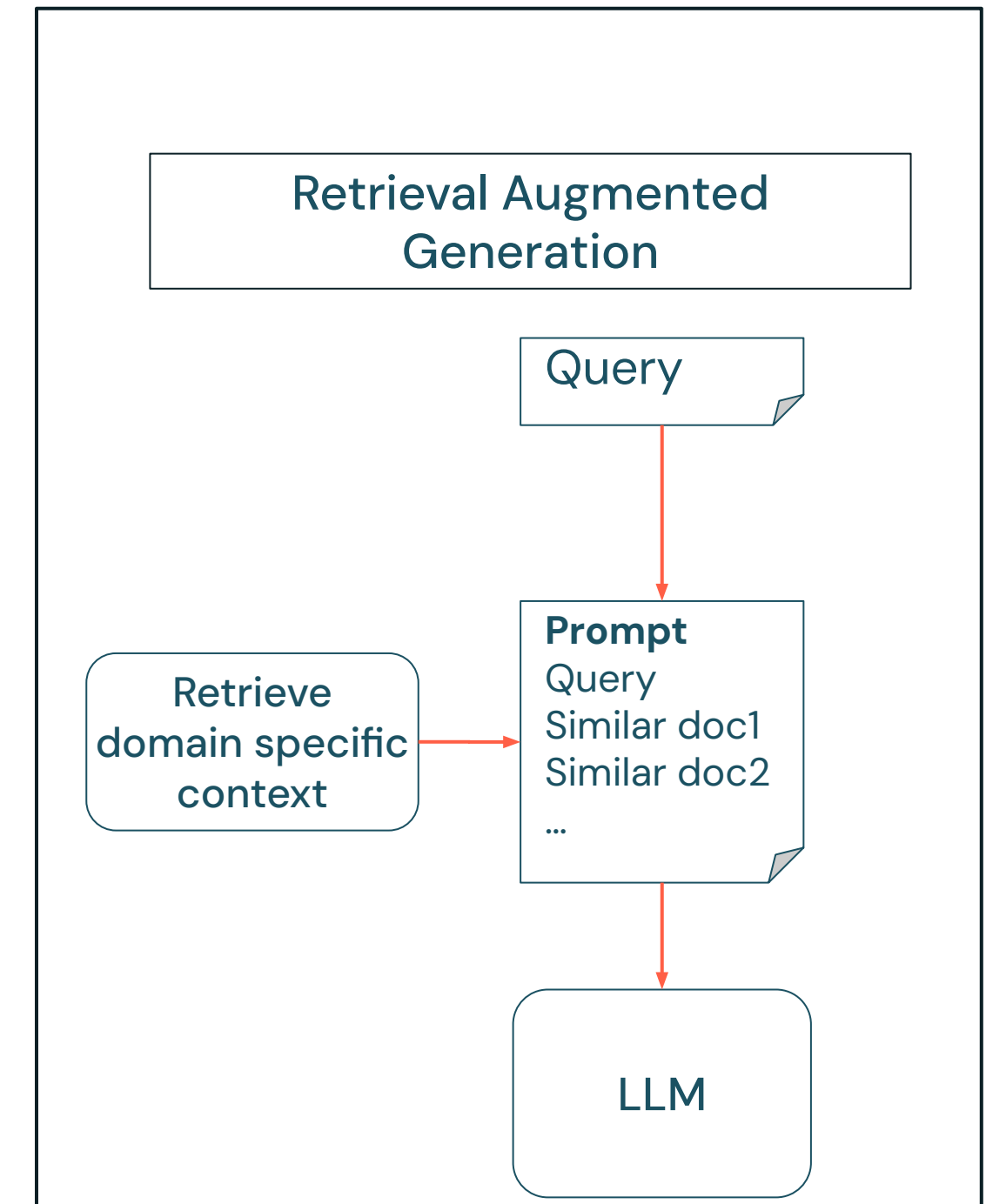
Type of system	Components
Prompt engineering	<ul style="list-style-type: none">● Prompt generation logic● LLM
Unstructured docs RAG	<ul style="list-style-type: none">● LLM● Retrieval system
Structured data RAG	<ul style="list-style-type: none">● LLM● Data API e.g., Databricks Online Table● Text-to-SQL engine e.g., Genie ← <i>more complicated in reality</i>
Agent-based Chain	<ul style="list-style-type: none">● Function-calling capable LLM● RAG chain● Orchestration chain
Orchestration Chain	<ul style="list-style-type: none">● Function-calling capable LLM● API services



RAG is just a Simple AI System

Compound AI system and RAG:

- Multiple AI systems, including RAG, can be combined to tackle AI tasks.
- RAG is an AI system composed of a **single task**.
- RAG pipeline includes various steps;
 - Retrieve documents for additional context
 - Building prompt with additional context
 - Generating response



Real-word Prompts Have Multiple Tasks

Objective: Analyze sentiment for customer reviews in a foreign language.

Problem:

- Single objective with **multiple tasks**.
- A task consists of one to many **sub-tasks**.
- Some tasks typically depend on each other (Translation → Summarize)

Tasks:

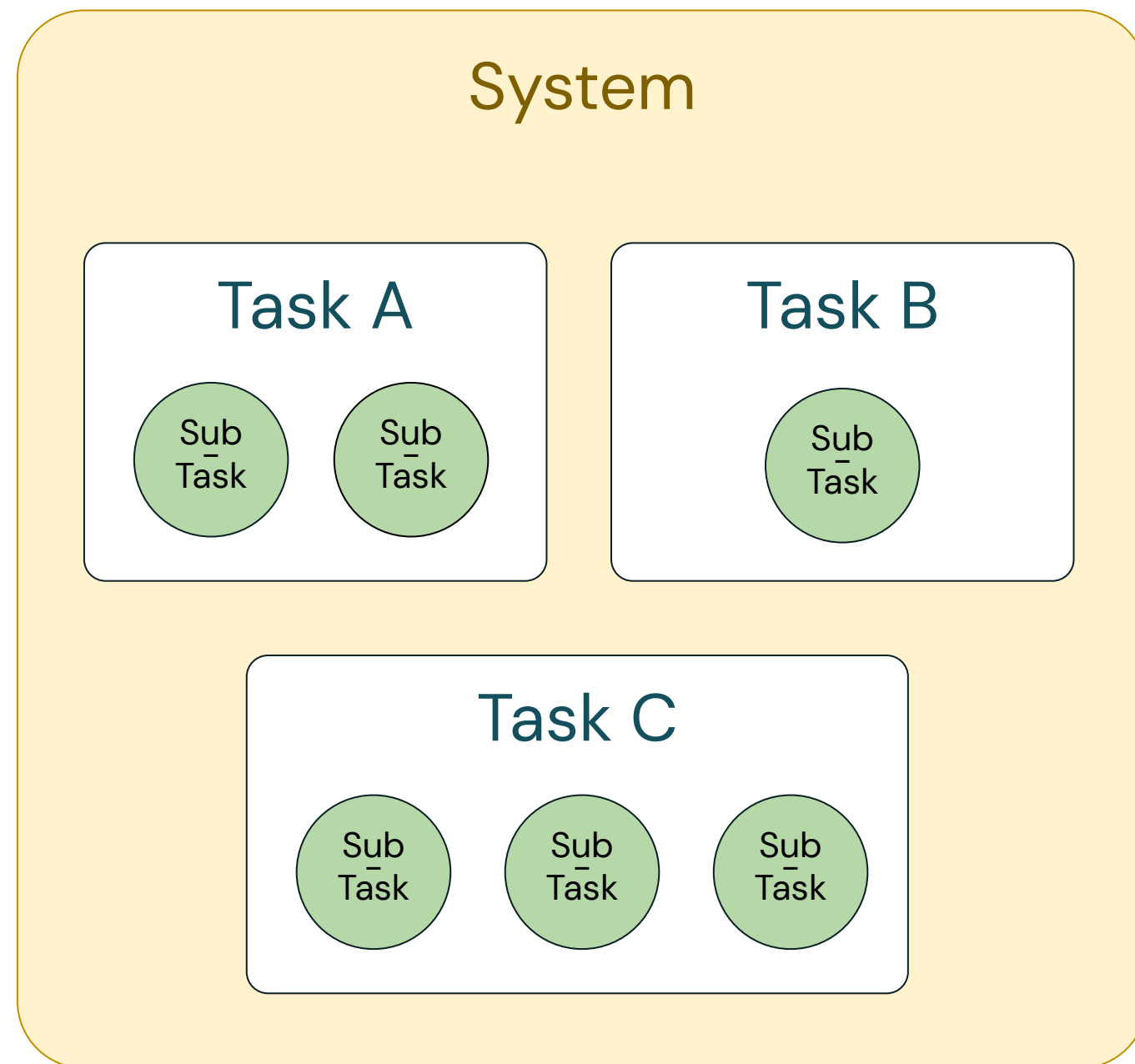
- **Translate** an Japanese customer review into English
- **Summarize** the translated text
- Analyze its **sentiment**
- **Classify** the type of feedback provided

 **We need a structured **pipeline** to complete these intents.**



Concepts

Tasks, sub-tasks, pipelines



System: Create a personalized response to a customer question.

Tasks:

- Identify the question type (e.g. complaint)
- Identify the language of the message
- ...

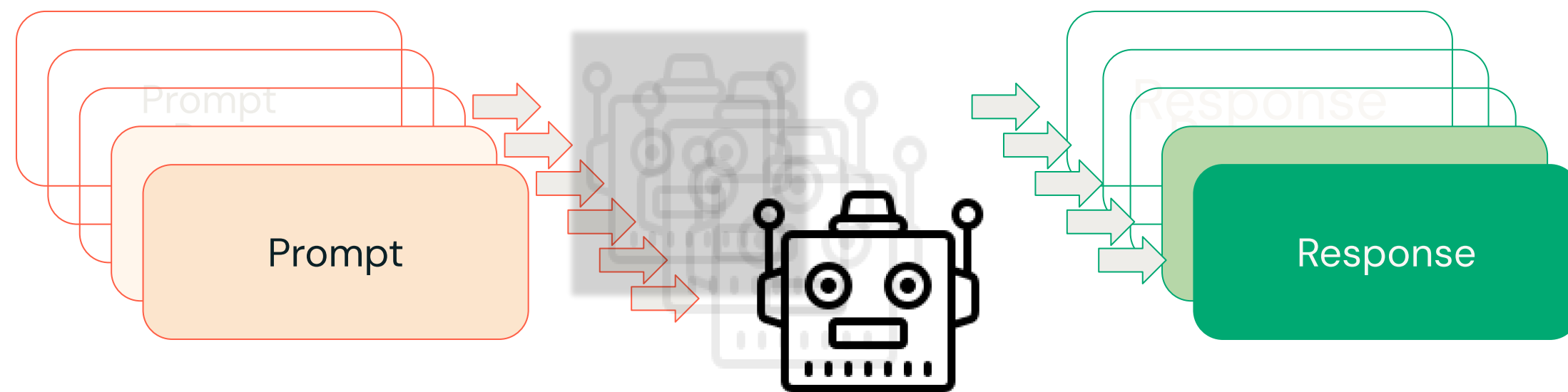
Sub-tasks:

- Retrieve customer purchase history
- Create a message
- ...



Tasks in Compound AI Systems

LLMs can complete a huge array of challenging tasks.



Summarization

Sentiment analysis

Translation

Zero-shot classification

Few-shot learning

Conversation/chat

Question-answering

Token classification

Text classification

Text generation

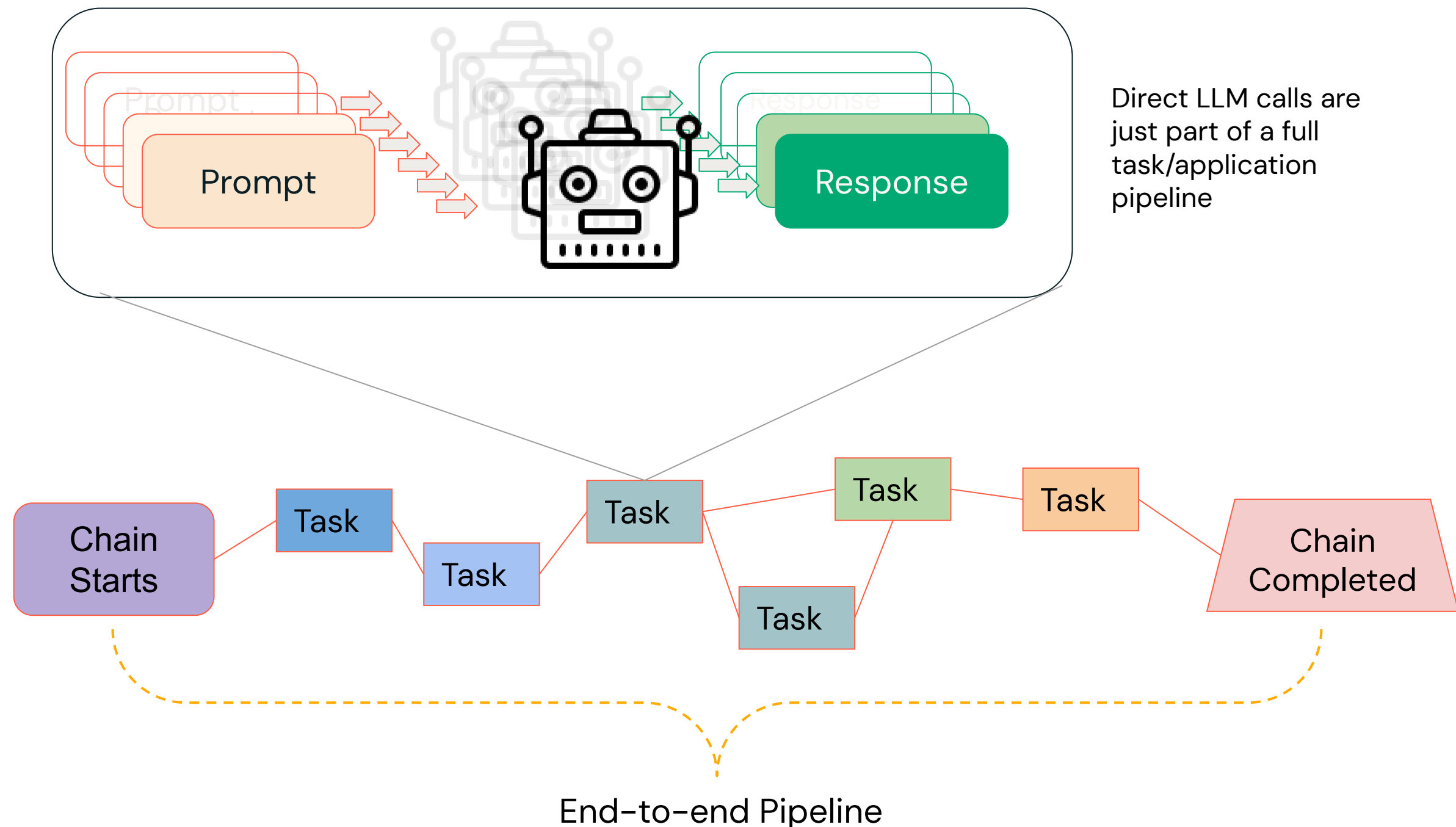
...



Tasks in Compound AI Systems

Typical applications are more than just a prompt-response system.

Example Task: Single interaction with an LLM (e.g. query a SQL database, search web for _)

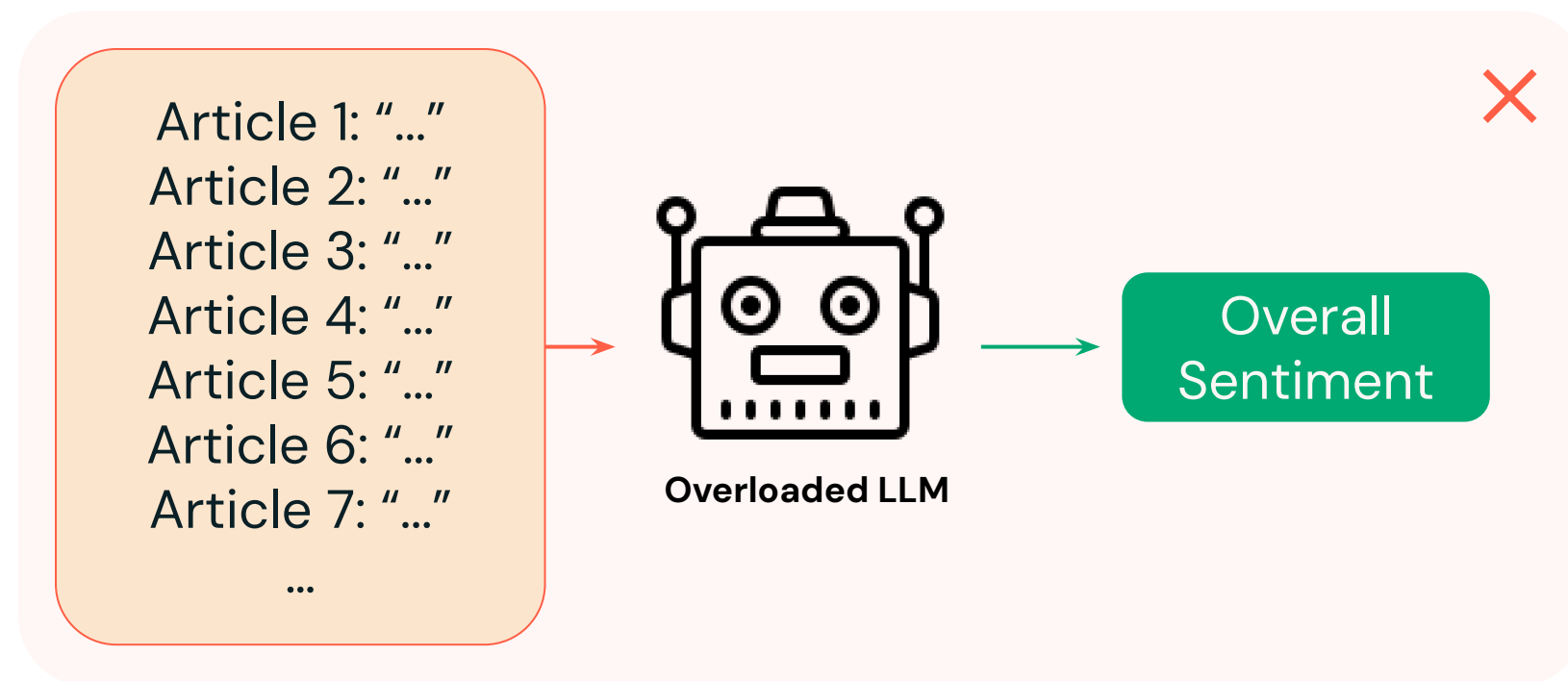


Chain: Applications with more than a single interaction



Summarize and Sentiment

Example multi-LLM problem: get the sentiment of many articles on a topic

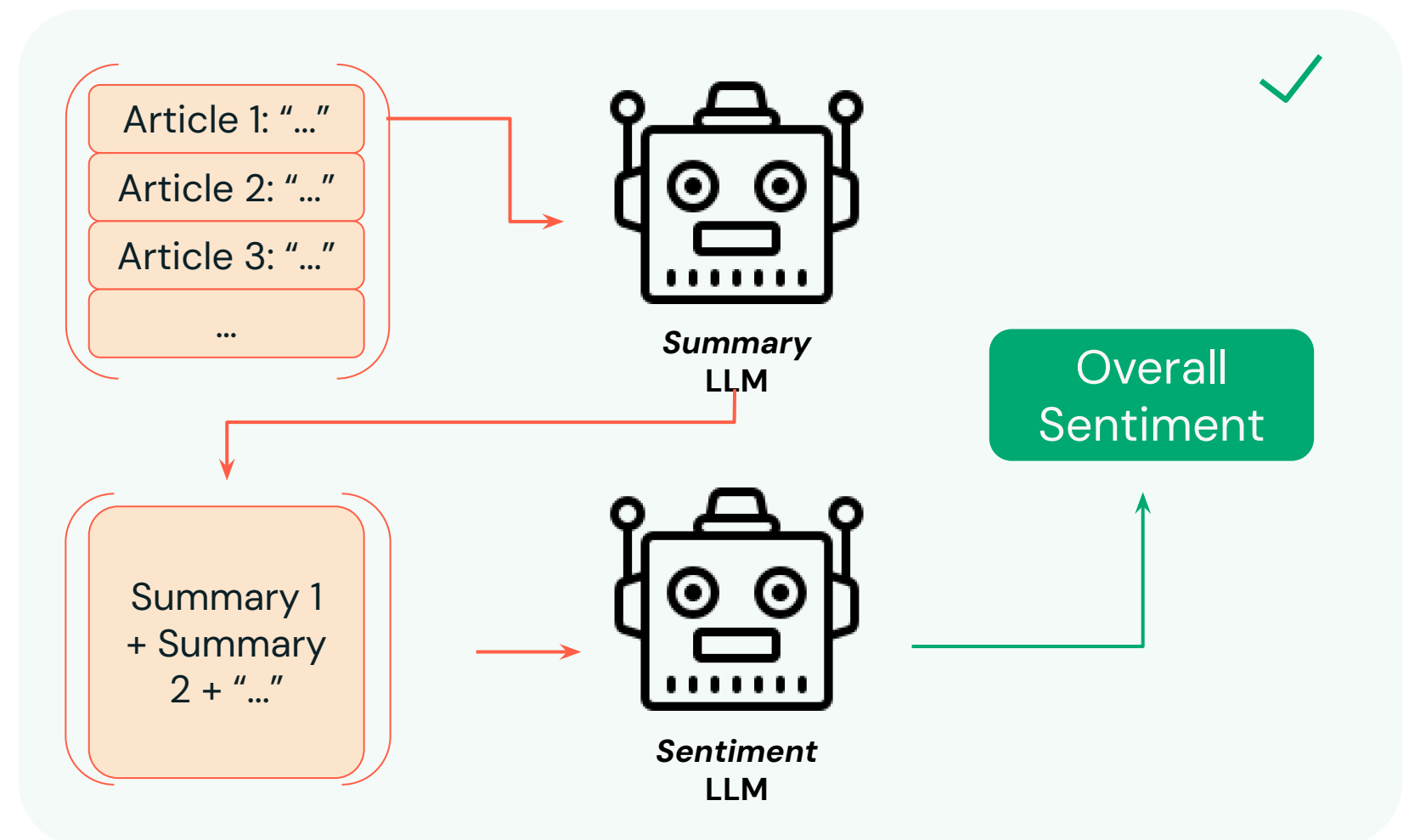


Initial solution

Put all the articles together and have the LLM parse it all

Issue

Can quickly overwhelm the model input length



Better solution

A two-stage process to first summarize, then perform sentiment analysis.



Summarize and Sentiment

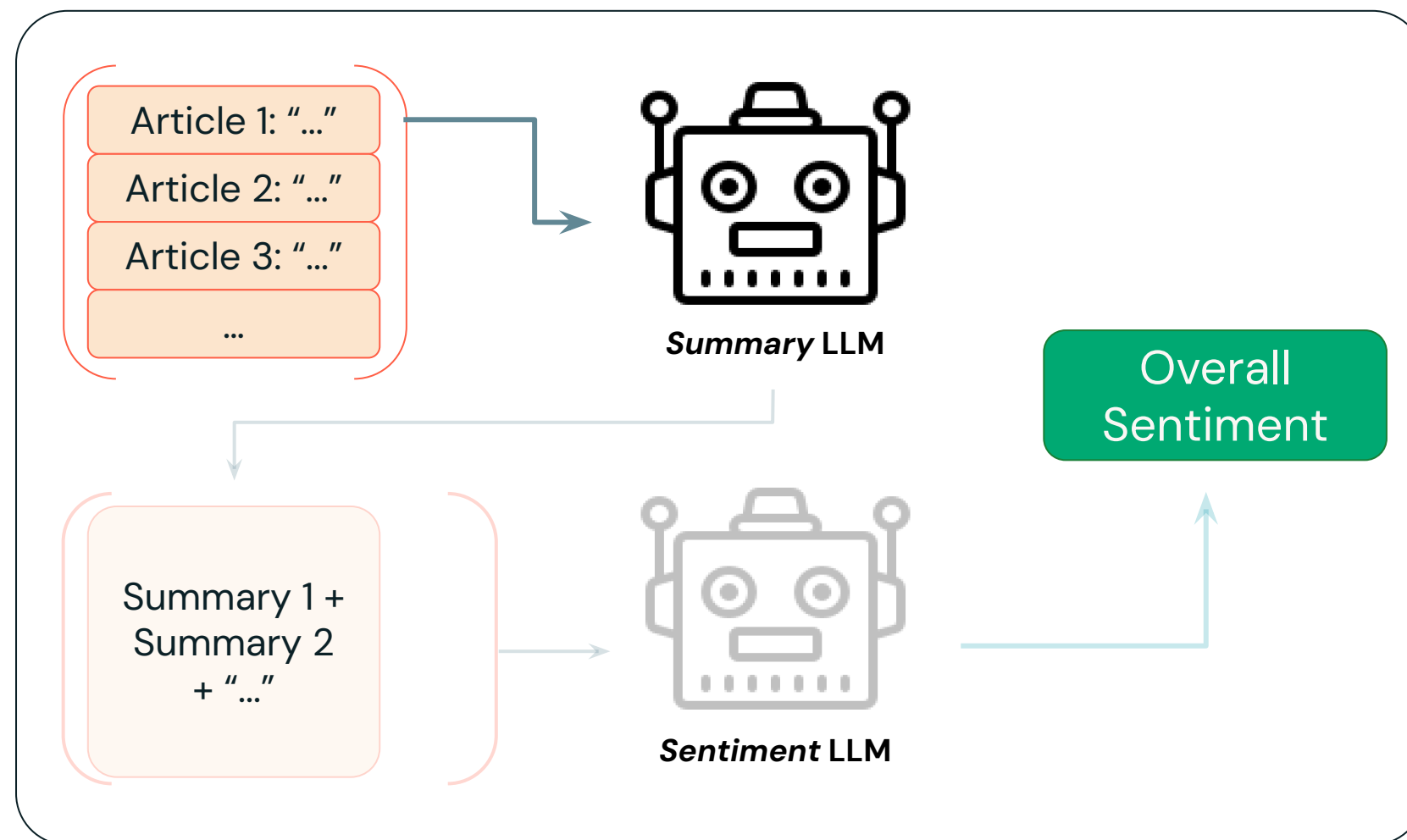
Step 1: Let's see how we can build this example.

Goal:

Create a reusable workflow for multiple articles.

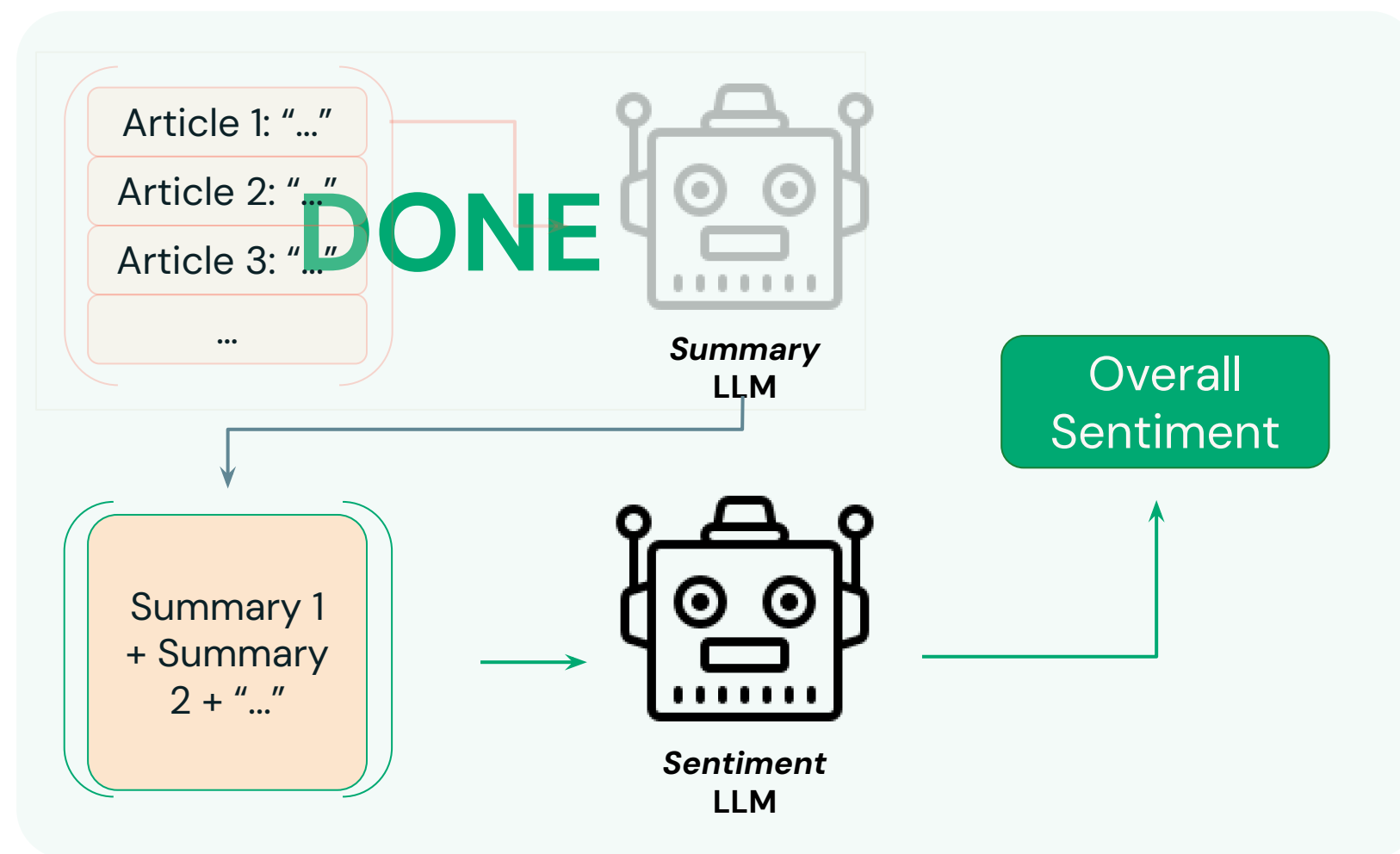
For this, we begin with the first task.

How do we make this process systematic?



Multiple LLM Interactions in a System

Chain prompt outputs as input to LLM



Now we need the **output** from our new engineered prompts to be the **input** to the sentiment analysis LLM.

For this, we're going to **chain** together these LLMs.





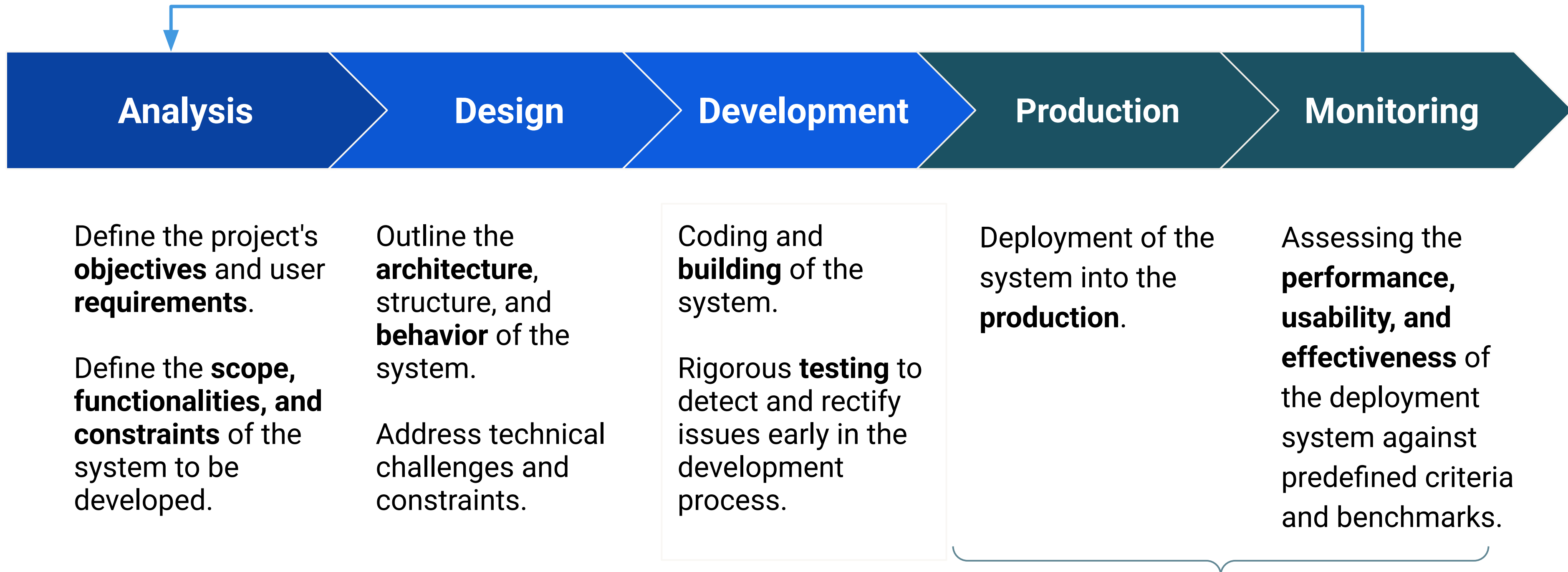
LECTURE

Designing Compound AI Systems



Application Development Life-cycle

A systematic approach to designing compound AI system



"Gen AI Deployment and Monitoring",
"Gen AI Evaluation and Governance"



Intent Classification and Chain Building

1. Identify Intents

- Define possible intents based on sample user queries
- Define tasks dependencies; are they sequential or parallel subtasks

02. Identify Tools

- What tools do you need to accomplish a task?
 - Web search
 - API interaction
 - Code execution
 - Text-to-speech, speech-to-text service
 - etc.

03. Build the Chain

- Build a workflow based on identified tasks
- What architecture is needed to solve the problem: RAG, Text2SQL etc.
- Iterate the process as needed

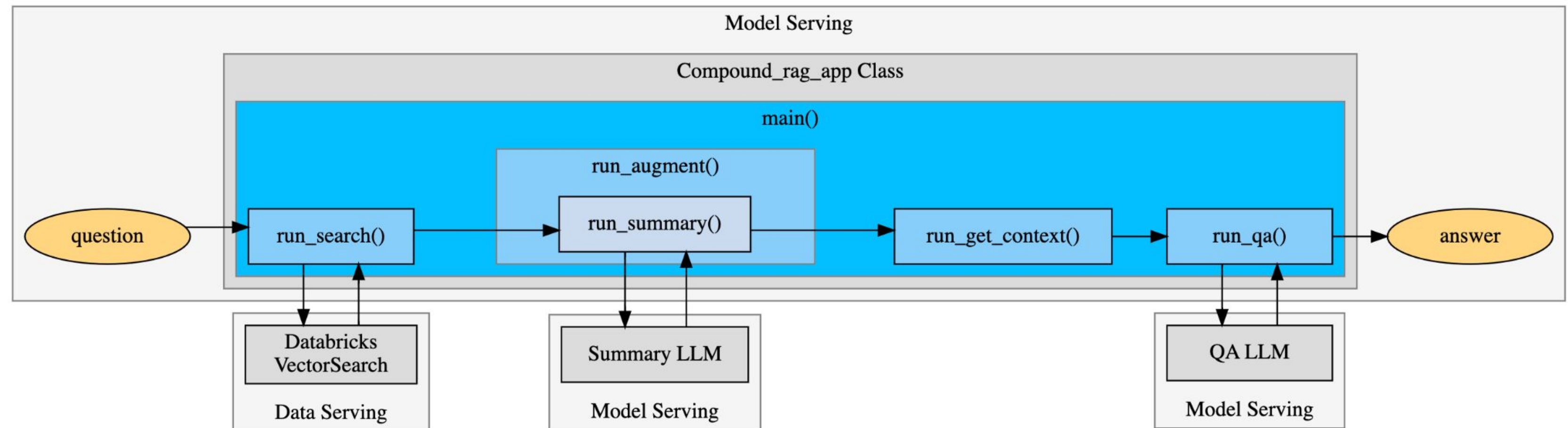


Designing A Compound AI System

A sample compound AI system architecture

A **sample architectural design** of a compound AI system.

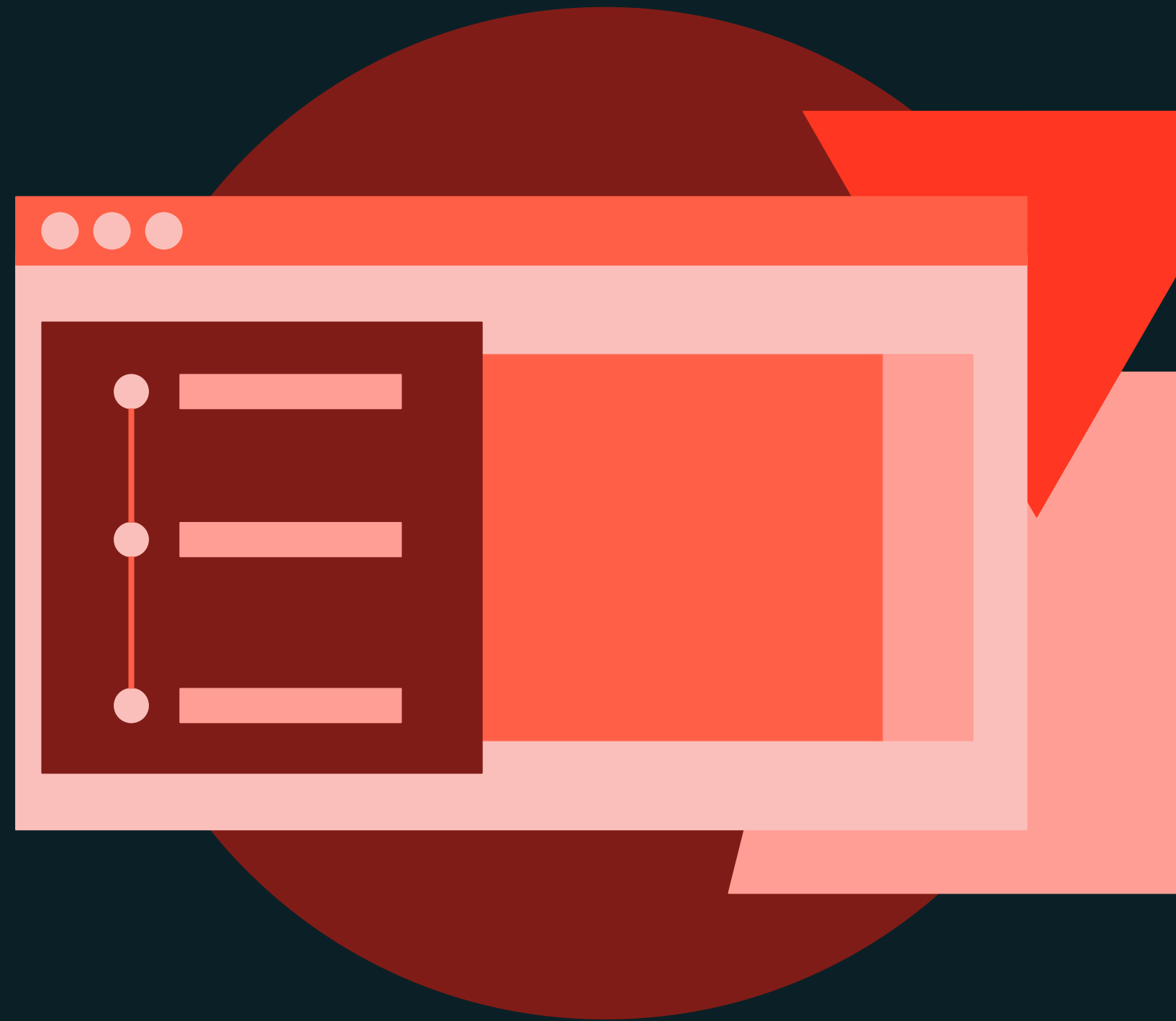
This design is independent of the framework or language model to be used.





DEMONSTRATION

Deconstruct and Plan a Use Case



Demo Outline

What we'll cover:

- Application Planning
 - Run_search stage
 - Run_summary stage
 - Run_get_context stage
 - Run_qa stage
- Full multi-endpoint architecture





LAB EXERCISE

Planning an AI System for Product Quality Complaints



Lab Outline

What you'll do:

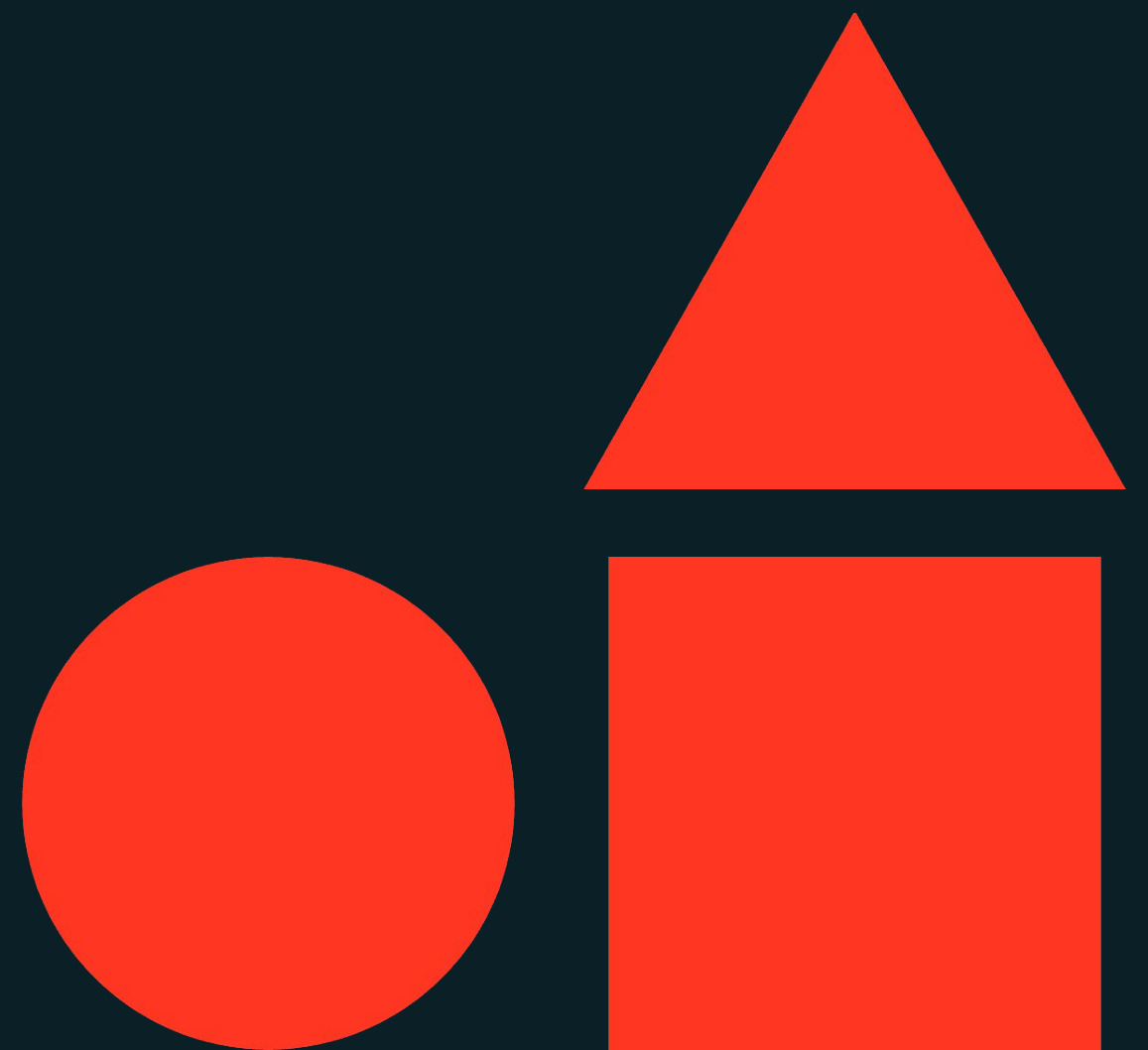
- **Task 1:** Define system components
- **Task 2:** Draw architectural diagram
- **Task 3:** Define possible input and output parameters for each component
- **Task 4:** Define libraries or frameworks for each component





Building Multi-stage Reasoning Chains

Generative AI Application Development



Learning Objectives

- Define main concepts of multi-stage reasoning systems.
- Describe LangChain and its main components.
- Describe Databricks products and features for building multi-stage reasoning systems.
- Describe the benefits of using composition frameworks in AI system development.
- Explain how multi-stage reasoning is a more accurate representation of how the human mind handles problems.





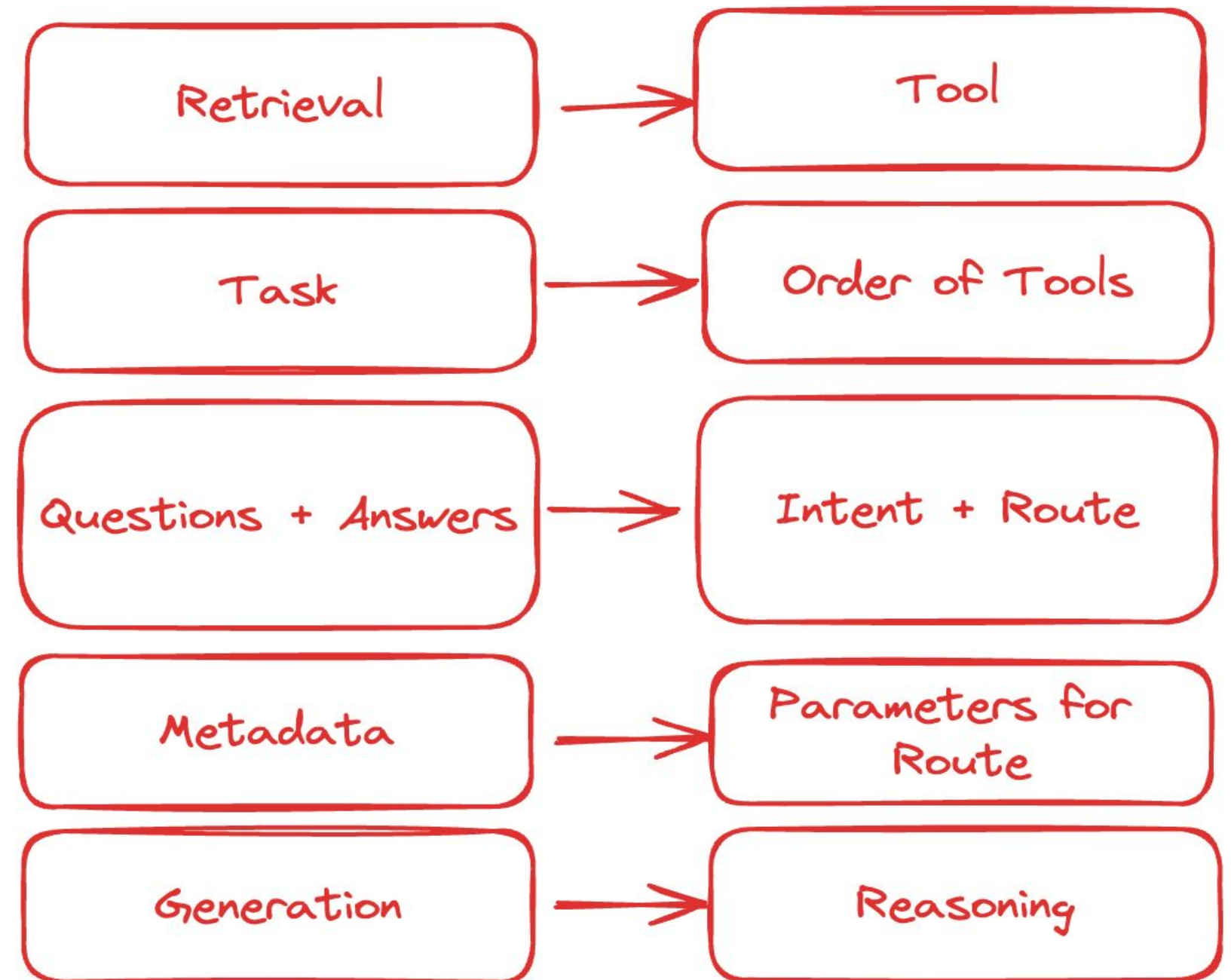
LECTURE

Introduction to Multi-stage Reasoning Chains



Map your Mindset

- While building a multi-stage chaining system, we need to map the concepts with the technical terms.
- **Composition frameworks** help to manage multi-stage reasoning systems.
- LangChain is one of the most popular framework and has **Databricks integrations**.



Designing of AI Systems

Emergence of composition frameworks

- Building compound AI systems requires new tools
- Composition frameworks are becoming popular
- Popular composition libraries
 - [LangChain](#)
 - [LLamaIndex](#)
 - [Haystack](#)
 - [DSPy](#)
- Popular agent libraries
 - LangChain – agents
 - AutoGPT



LangChain

Introduction and concepts



LangChain:

- A software framework designed to help **create Gen AI applications** that utilize large language models.
- Enables applications to be **context-aware, reason, and interact dynamically** with various data sources and environments.
- Includes **components** for building chains and agents, integrations with other tools, and off-the-shelf implementations for common tasks



LangChain

Main components



Prompt:

A structured text input designed to **communicate a specific task or query to a language model**, guiding it to produce the desired output.

Chain:

A **sequence of automated actions or components** that process a user's query and produce a model's output.

Retriever:

An interface that returns **relevant documents or information based** on an unstructured query, often used in conjunction with indexed data to enhance search and retrieval capabilities.

Tool:

A **functionality or resource** that an agent can activate, such as APIs, databases, or custom functions, to perform specific tasks.



LLama Index

Introduction and concepts



Data framework that enhances the capabilities of LLMs by structuring and indexing data to make it easily consumable.

Components:

- Models
- Prompts
- Indexing and storing
- Querying
- Agents



Haystack

Introduction and concepts



Haystack is an open-source Python framework for building custom applications with LLMs, focusing on document retrieval, text generation, and summarization.

Components:

- Generators
- Retrievers
- Document Stores
- Pipelines





Introduction and concepts

Framework for programming with LLMs and retrieval models (RMs). It provides a structured and composable approach to tasks involving LMs, going **beyond manual prompting** techniques.

Components:

- Signatures: declarative modules that guide LLMs, following a Pythonic structure. Examples include *ChainOfThought*, *Retrieve*, and *ReAct*.
- Teleprompters: optimizers that "compile" a program into instructions, few-shot prompts, or weight updates (fine-tuning) tailored for a specific LM.
- Automatic Compiler: an automatic compiler that traces the execution of a program and generates high-quality prompts or automatically fine-tunes LMs to internalize the task's procedural details.



Choosing a Library

Factors to consider

Library Features

- Research if the library supports your **use case requirements**.
- Check if the library has support for a wide array of LLMs and LLM interfaces.
- Evaluate the ease of integrating with **external data sources** and knowledge bases.

Performance and Scalability

- Evaluate the **performance and scalability of each library**, especially if your application will handle large volumes of data or require high throughput

Stability and Complexity

- These libraries are evolving quickly, and **instability in APIs is a major issue**.
- Users might find some libraries challenging to understand and use.



Databricks Products for Building Multi-stage Reasoning Systems



Foundation Model API

Access and query state-of-the-art open generative AI models

Features:

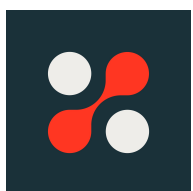
- **Instant access** to popular foundation models.
- Pricing model: **pay-per-token** for rapidly iterating and prototyping applications and **provisioned throughput** for high-throughput applications.
- Users can integrate **external models**, such as Azure OpenAI GPT models or AWS Bedrock Models
- **Unified interface** for deploying, governing, and querying AI models

Some of the supported models and tasks:

Model	Task type
DBRX Instruct	Chat
Meta Llama 3.1 70B/405B Chat	Chat
Mixtral-8x7B Instruct	Chat
GTE Large (English)	Embedding

[Up-to-date list of supported models](#)





DBRX

A new open LLM by Databricks

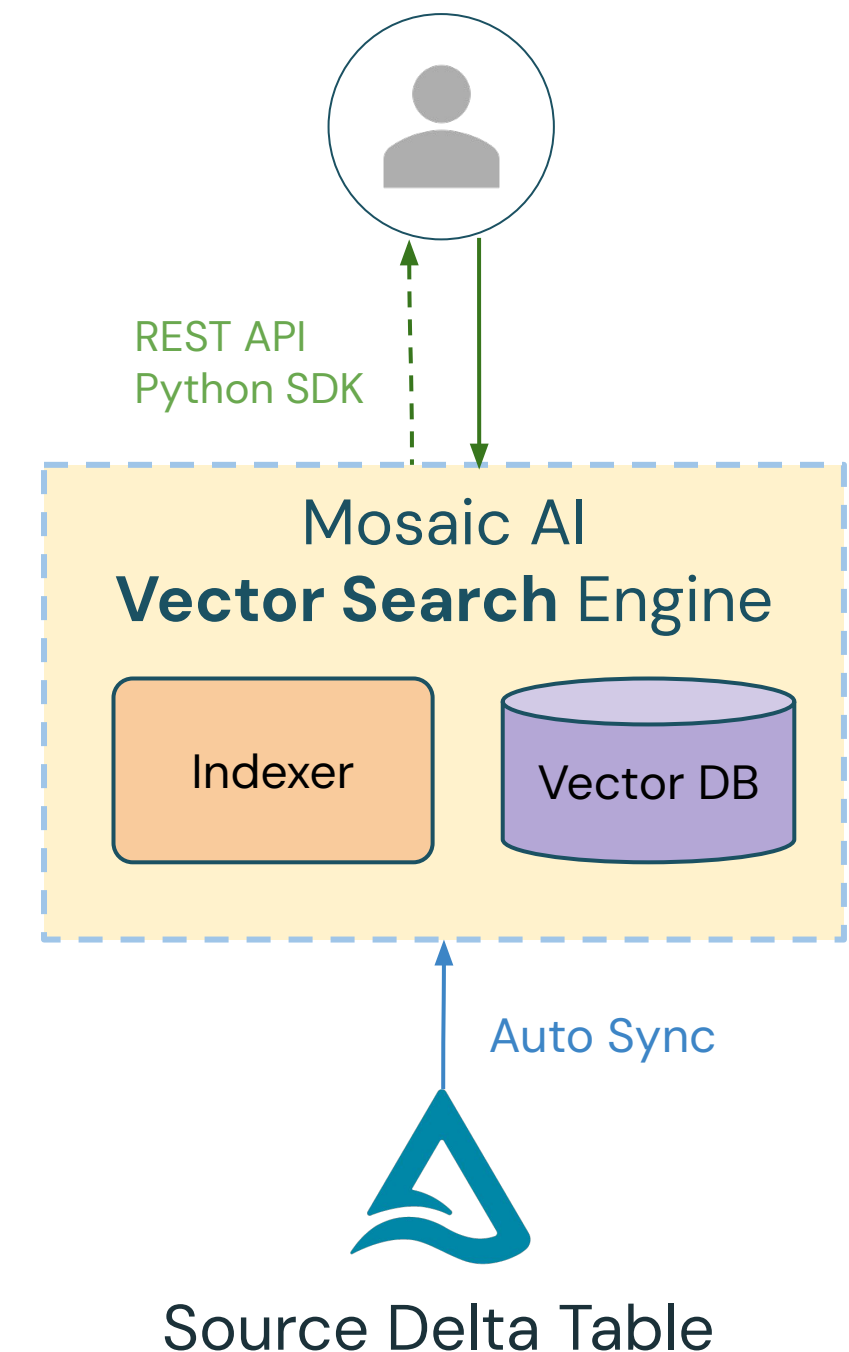
- DBRX is Databricks' very own **open-source LLM**.
- **DBRX Base** pretrained model
 - It functions like a smart autocomplete – it will continue whatever you say to it.
 - Useful for further fine-tuning your data
- **DBRX Instruct** fine-tuned model
 - Designed to answer questions and follow instructions.
 - Built on top of DBRX by performing further training on domain-specific data and fine-tuning for instruction-following.



Vector Search

A vector database that is built into the Databricks Intelligence Platform

- Stores vector representation of your data, plus metadata
- Tightly integrated with your Lakehouse
- Scalable, low latency production service with zero operational overhead
- Supports ACLs using Unity Catalog integration
- API for real-time similarity search
 - Query can include filters on metadata
 - REST API and Python client



Mosaic AI Agent Framework

A suite of tooling designed to help developers build and deploy high-quality GenAI applications

Mosaic AI Agent Framework **makes it easy** for developers;

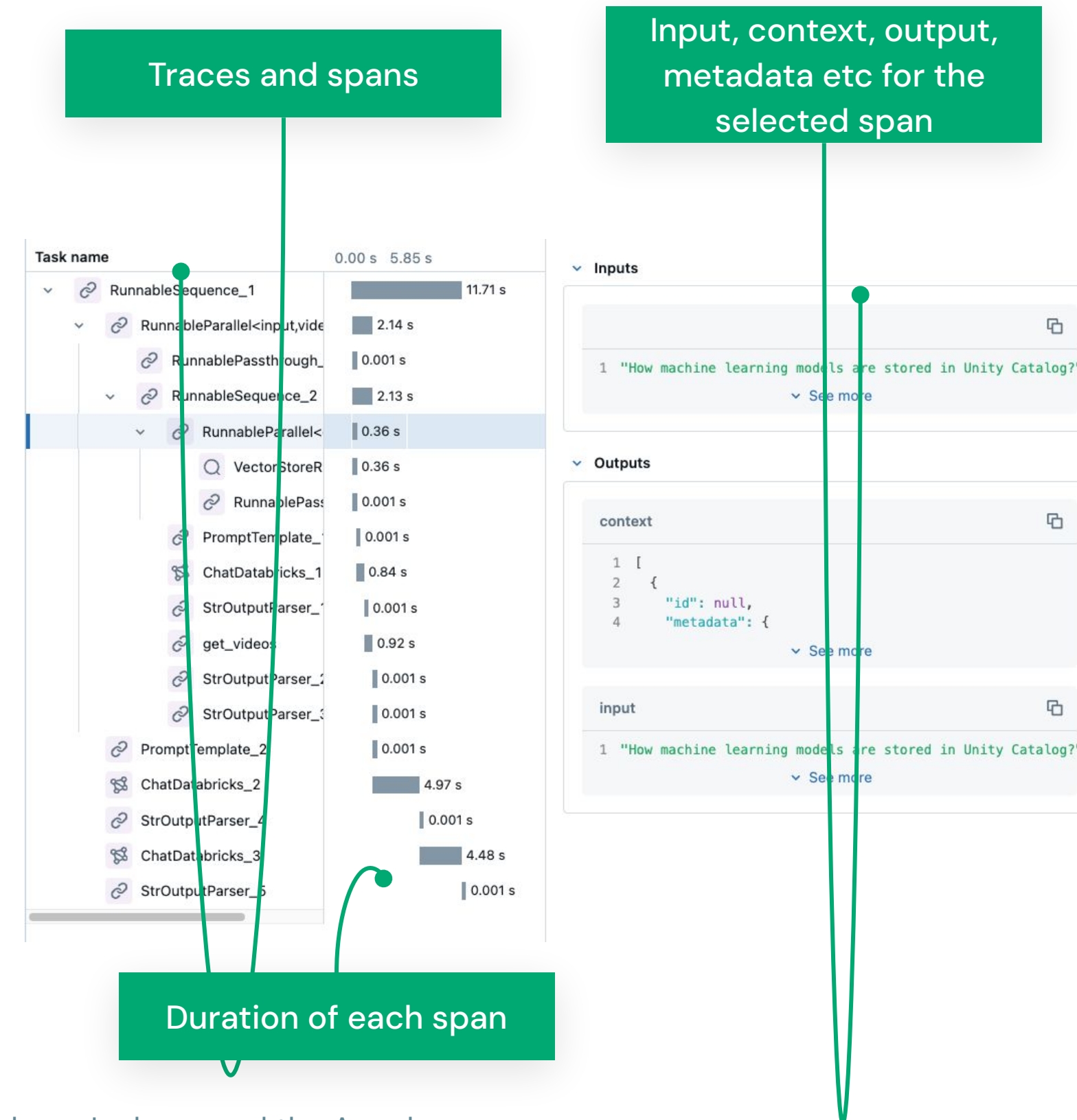
- To evaluate the quality of their RAG application
- Iterate quickly with the ability to test their hypothesis and redeploy their application easily
- Have the appropriate governance and guardrails to ensure quality continuously



Mosaic AI Agent Framework

Tracing with MLflow Tracing

- MLflow Tracing allows **logging, analyzing, and comparing traces** across different versions of Gen AI applications.
- It allows **debugging apps** and keeping track of inputs and responses of the Gen AI applications.
- **Benefits;**
 - Interactive trace visualizations.
 - Track the latency impact of different frameworks, models, chunk sizes, etc.
 - Measure cost by tracking token usage.



Mosaic AI Agent Framework

Tracking and Logging with MLflow Tracking

- Logging captures a **“point in time”** of the agent’s code and configuration so you can evaluate the quality of the configuration.
- Databricks recommends that you use **code-based logging** instead of **serialization-based logging**.
- MLflow tracking can be integrated with popular Gen AI libraries, enabling automatic logging of parameters and metrics.

```
1 with mlflow.start_run(run_name="multi_stage_demo") as run:
2     signature = infer_signature(query, response)
3     model_info = mlflow.langchain.log_model(
4         multi_chain,
5         loader_fn=get_retriever,
6         artifact_path="chain",
7         registered_model_name=model_name,
8         input_example=query,
9         signature=signature
10    )
```



Other Databricks Products

Other products for AI system logging, deployment and monitoring

List of products that will be covered in other courses:

- **Mosaic AI Agent Evaluation^(*)**: For evaluating the quality, cost, and latency of Gen AI applications, including RAG applications and chains
- **MLflow Evaluation^(*)**: For computing and managing evaluation metrics
- **Model Serving^(**)**: For serving custom chains and pipelines
- **Lakehouse Monitoring^(**)**: For monitoring the performance of deployed chains and pipelines

Related course:

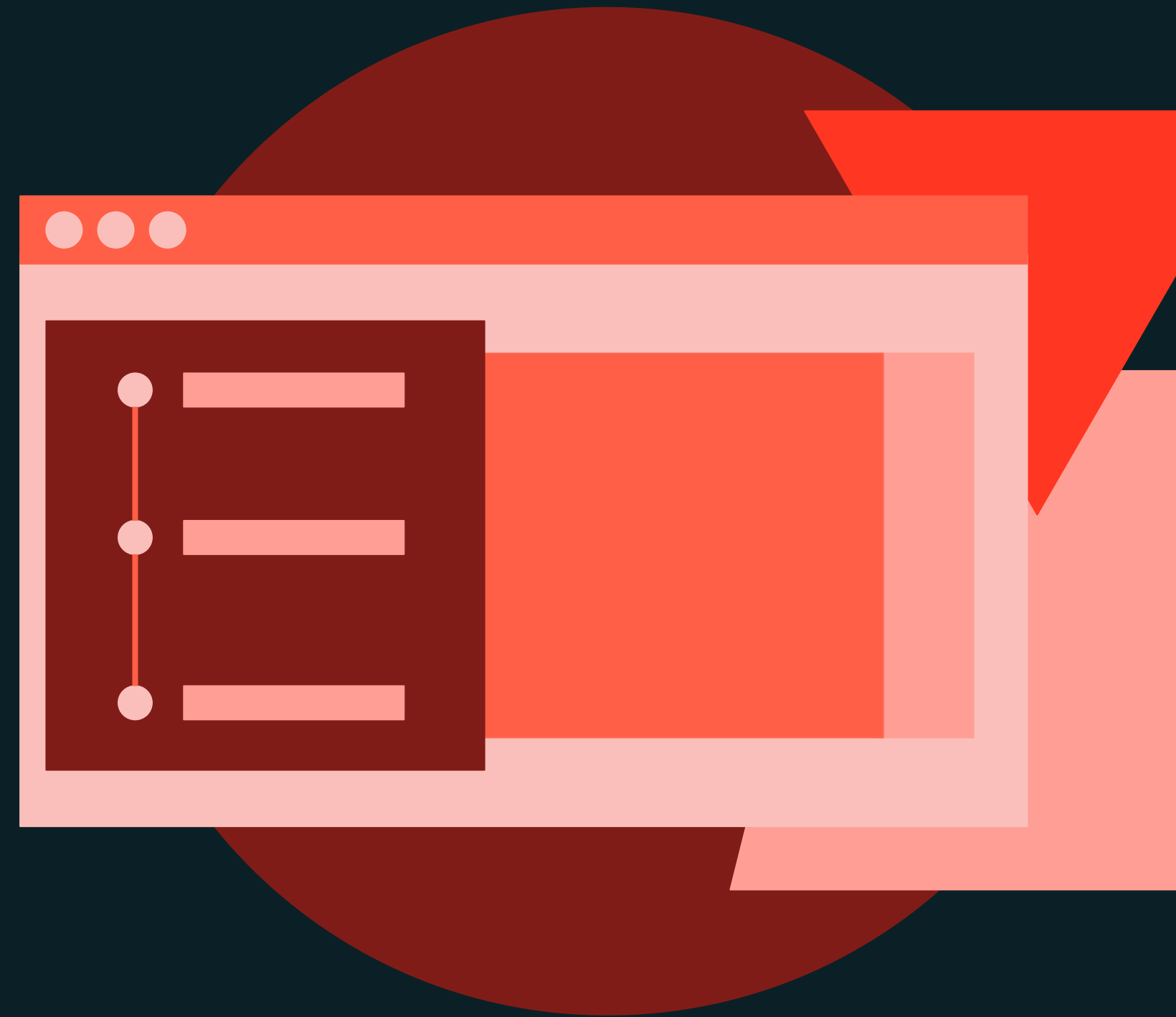
- ^(*) Generative AI Evaluation and Governance
- ^(**) Generative AI Deployment and Monitoring





DEMONSTRATION

Building Multi-stage Reasoning Chain in Databricks



Demo Outline

What we'll cover:

- Components
 - Prompt, LLMs, Retriever, Tools, Chaining
- Build a multi-stage chain
 - Create a vector store
 - Build first chain
 - Build second chain
 - Chaining chains





LAB EXERCISE

Building Multi-stage AI System



Lab Outline

What you'll do:

- **Task 1:** Create a vector index and store embeddings
- **Task 2:** Build a retriever-based chain
- **Task 3:** Build an image generation chain
- **Task 4:** Combine chains into a Multi-chain System





Agents and Cognitive Architectures

Generative AI Application Development



© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark, the Spark logo, Apache Iceberg, Iceberg, and the Iceberg logo are trademarks of the [Apache Software Foundation](https://www.apache.org/).

Learning Objectives

- Describe agents as a major component of more advanced Generative AI applications.
- Define an LLM agent as a centralized reasoning unit to solve complicated tasks using other tools.
- Describe multi-agent and multi-model agents.
- List the components of an LLM agent: task assigned, LLM for reasoning, and a set of tools that it can use.
- Explain the architecture of a common agent-based LLM workflow for self-monitoring/autonomous Generative AI application.
- Identify LLM agent plugins as tools that can simplify this process.





LAB EXERCISE

Introduction to Agents

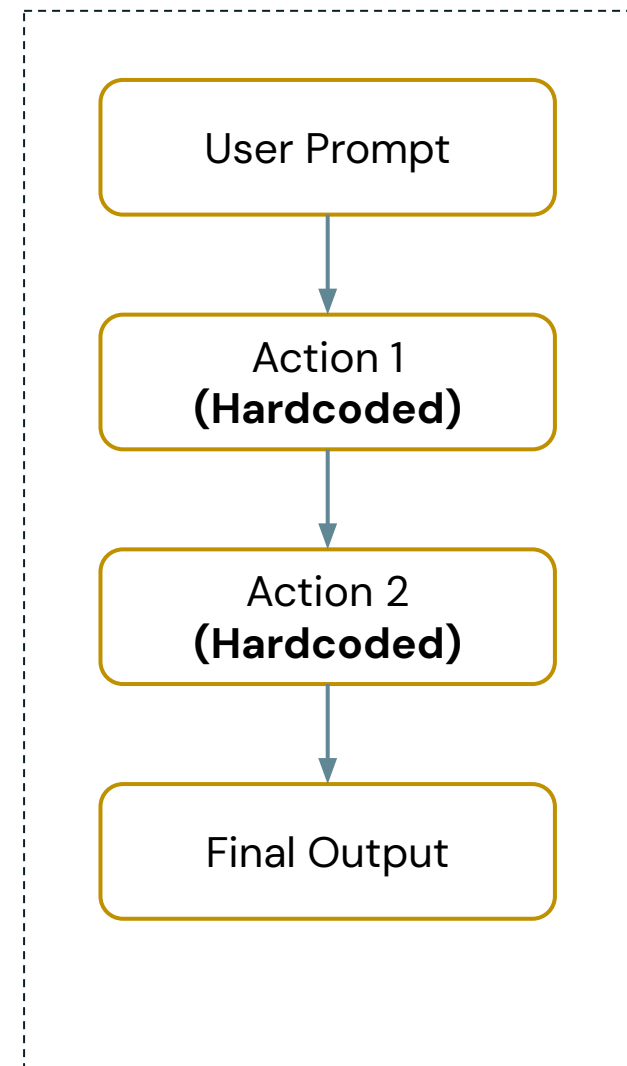


What is an Agent?

Non-agentic vs. Agentic workflow

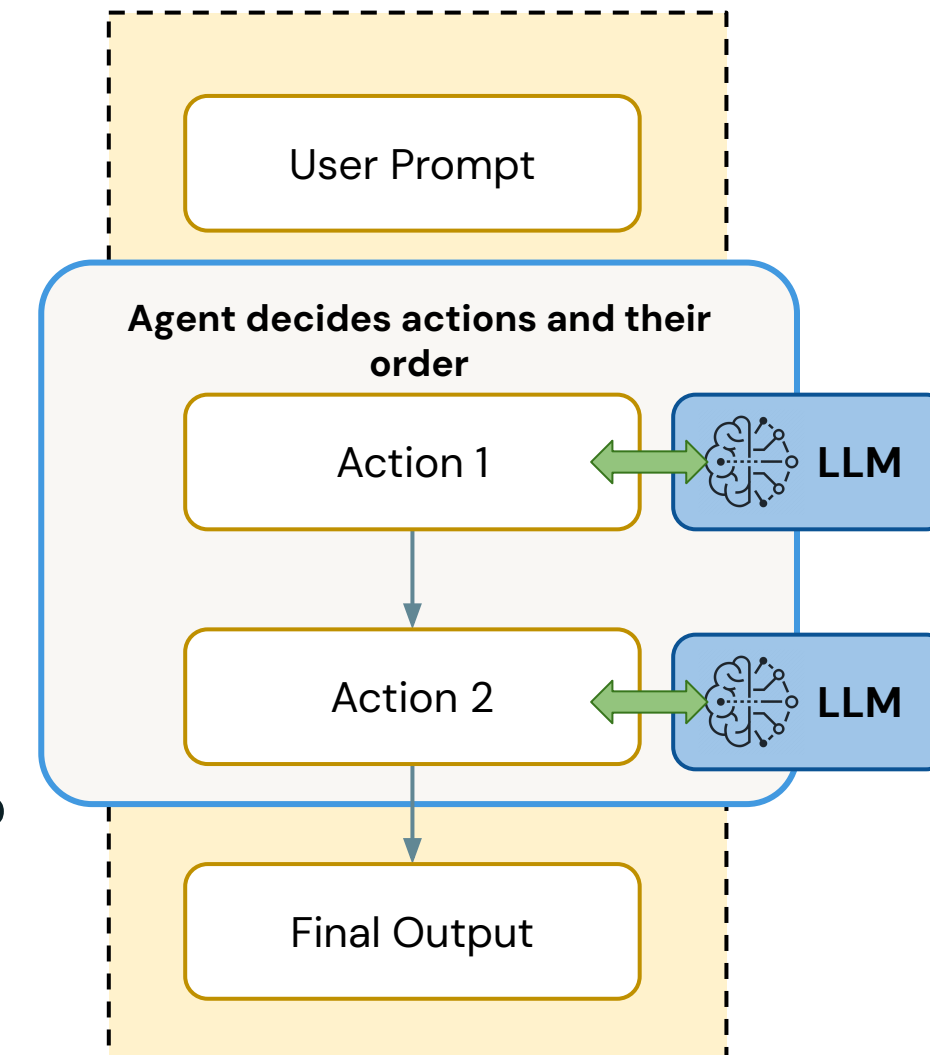
Non-Agentic Workflow

- The LLM generates an answer based on actions defined.
- Actions are **deterministic**.
- **Example:** Should I invest in NVIDIA stock?
 - Action 1 and Action 2 are hardcoded.



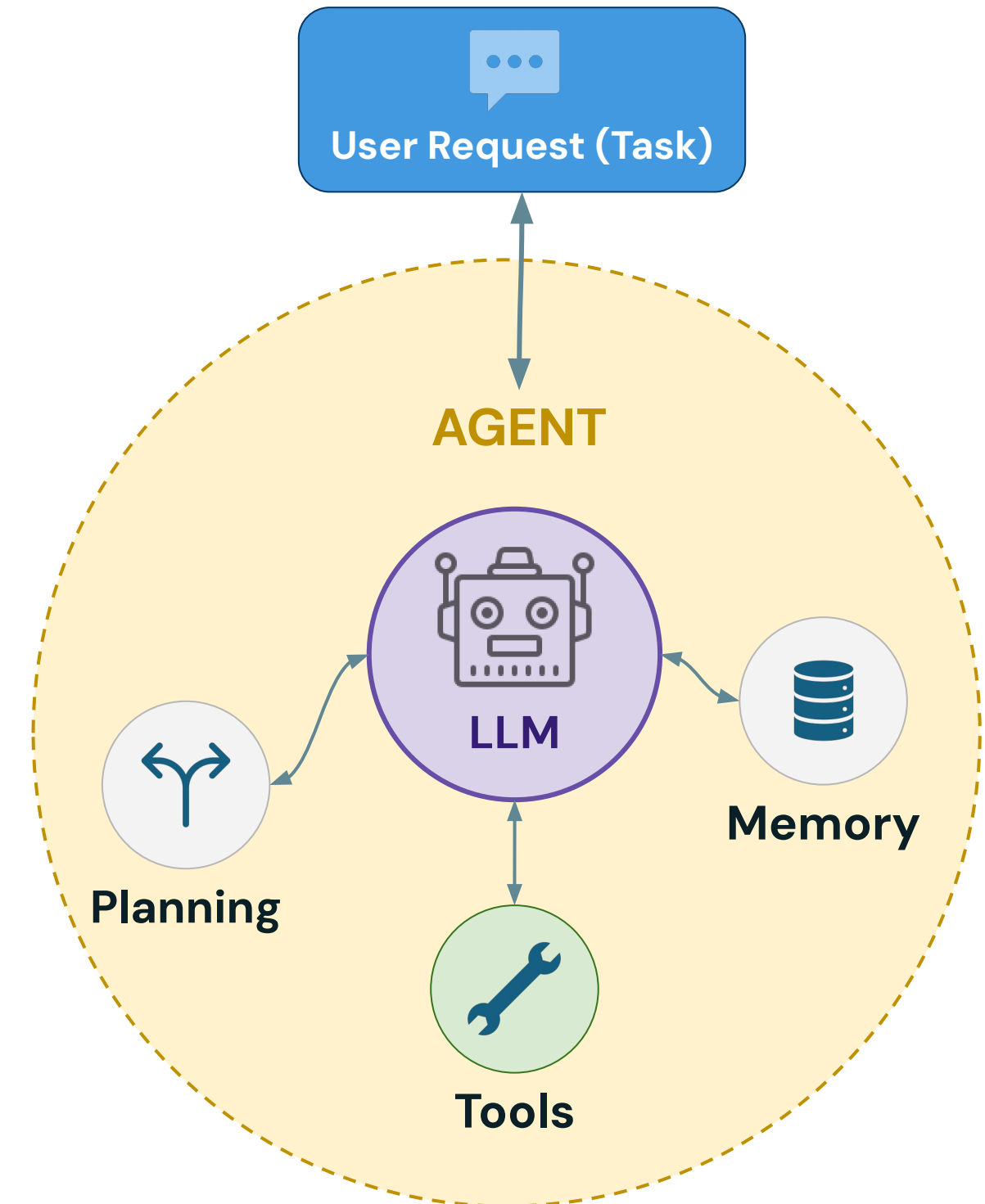
Agentic Workflow

- Workflows are **iterative**: the number of times the applications does the draft/review is **non-deterministic**
- **Example:** Should I invest in NVIDIA stock?
 - Agent does research, writes first draft, another model checks the daft.



What is an Agent?

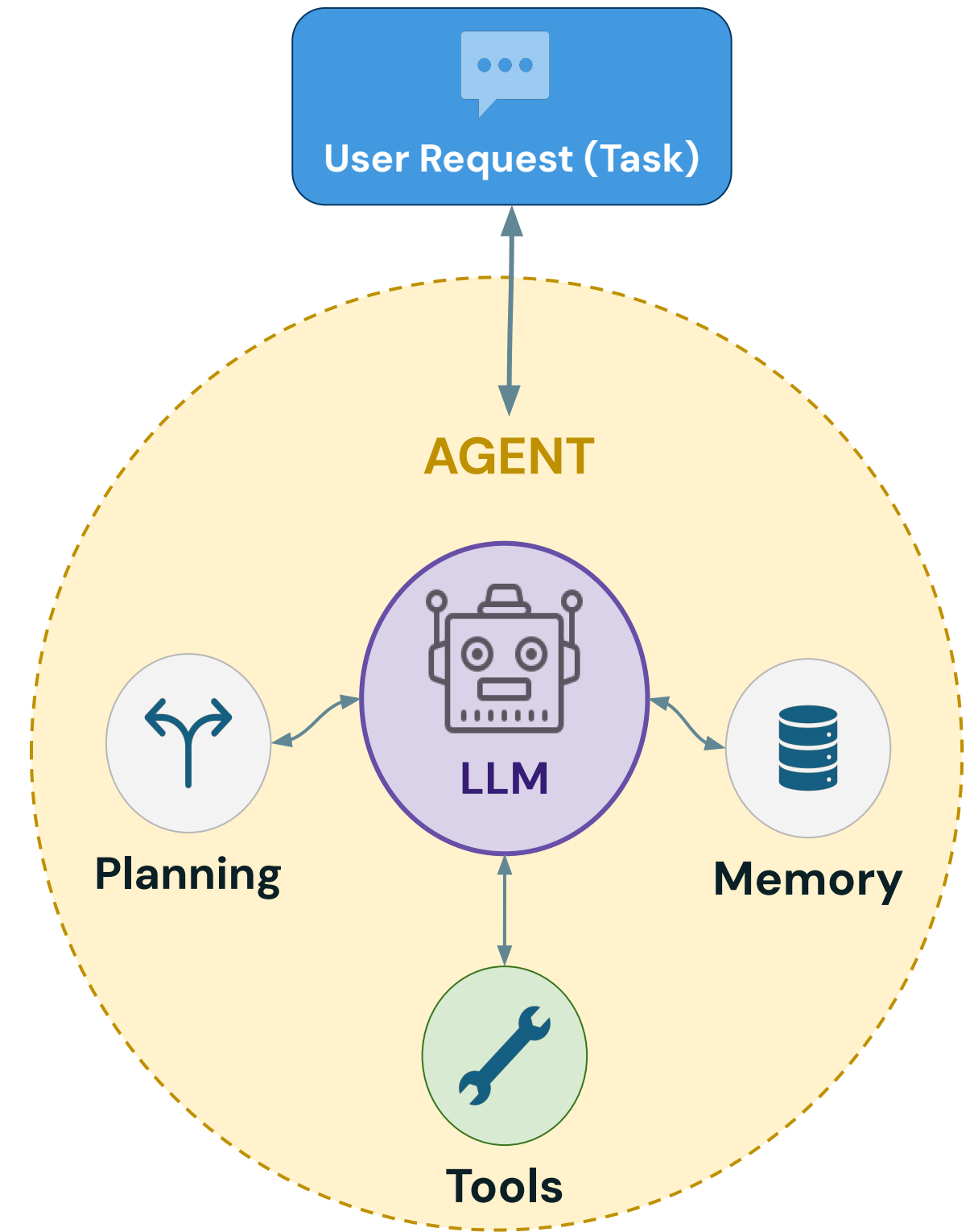
- An Agent is an application that can **execute complex tasks** by using a language model to define a **sequence of actions** to take.
- So far, we have discussed Compound AI Systems with **hard-coded calls to external tools and services**. Agents replace hard-coded sequences of actions with **query-dependent sequences chosen dynamically by LLMs**.



What is an Agent?

Core components of a typical agent system

- **Task**: User request through prompt to be solved.
- **LLM (Brain)**: Central coordination module that manages the core logic and behavioral characteristics of an agent. It is the "**brain**" of the agent.
- **Tools**: External resources that the agent uses to accomplish tasks.
- **Memory** and **planning** components for planning and executing the future actions.



An Example Agent

A hypothetical financial-advisor agent

Task: Is it the right time to invest in NVIDIA stock?

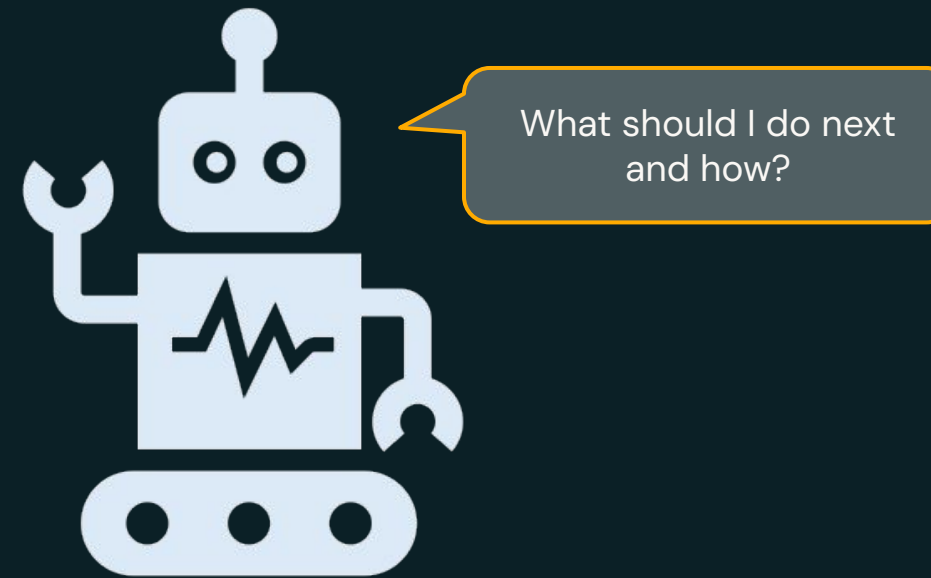
Sub-tasks include:

- Decide which actions are needed? In which order these actions should be done?
- What is the current stock price?
- What are the latest financials?
- What is the latest news/announcement?
- What is current sentiment score?

Agent Workflow:

- **Process task:** Agent uses an LLM to understand the request and determine actions to fulfill the task.
- **Collect data:** financials, stock price, latest news articles, etc.
- **Data Analysis:** Use an LLM to analyze data and sentiment analysis.
- **Output generation:** Use an LLM to synthesize the information into a coherent report.





How does an agent decide which actions to take?



Agent Reasoning

- The cognitive process by which artificial agents, operating within the domain of AI, analyze information, draw logical conclusions, and **make decisions autonomously**, mirroring aspects of human cognitive abilities.
- Independent from the language and framework, there are **design patterns** for agent reasoning;
 - ReAct
 - Tool Use
 - Planning
 - Multi-agent Collaboration



Pattern: ReAct (Reason + Act)

Agent reasoning patterns

- Enables models to generate **verbal reasoning** traces and **actions**.
- Main states used in a ReAct agent are:
 - **Thought:** Reflect on the problem given and previous actions taken
 - **Act:** Choose the correct tool and input format to use.
 - **Observe:** Evaluate the result of the action and generate the next thought.

Source: Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. ArXiv. /abs/2210.03629

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

Standard

Answer: iPod

Reason only

Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

Answer: iPhone, iPad, iPod Touch

Act only

Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced...
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ...
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software ...
Act 4: Finish[yes]

ReAct

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys]



Pattern: Tool Use / Function Calling

Agent reasoning patterns

- Agents interact with external tools and APIs to perform specific tasks.
- Agent's reasoning skills decide which tools to use and when/how to use them.

Example Tools

Research/Search Tools:

- Web browsing
- Search engines
- Wikipedia

Image:

- Image generation
- Object detection
- Image Classification

Document Retrieval:

- Database retriever
- Vector db retriever
- Document loader

Coding:

- Code execution
- Documentation generator
- Debugging/Testing





Pattern: Planning








Agent reasoning patterns

- Agents must be able to dynamically adjust their goals and plans based on changing conditions.
- In real-world scenarios, tasks have multiple sub-tasks that need to be orchestrated carefully.
- Tasks can have:
 - A single sub-task
 - Sequential sub-tasks
 - Graph sub-tasks

Source: Shen, Y., Song, K., Tan, X., Li, D., Lu, W., & Zhuang, Y. (2023). HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. ArXiv. /abs/2303.17580


Query: based on the pose image of example1.jpg and the description of example2.jpg, please generate a new one.



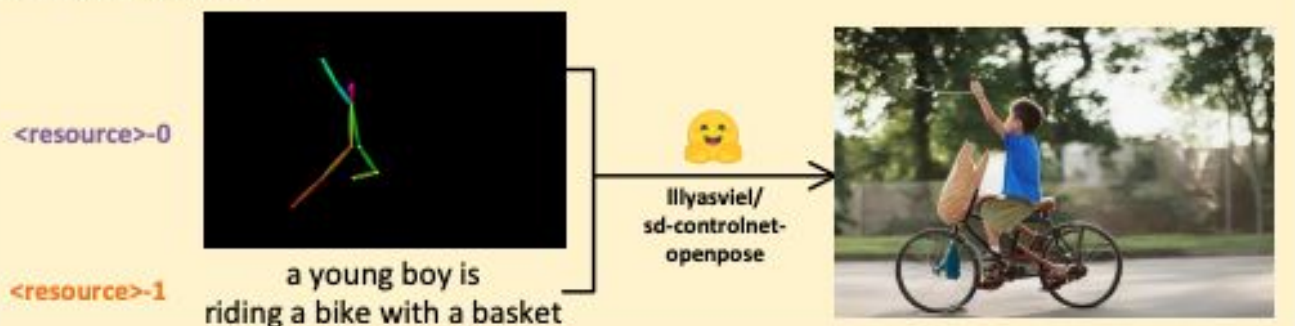
Response: Based on the pose image of  example1.jpg and the description of  example2.jpg, I have generated a new image  s5f8.jpg for you. The process of generating this image involved using the  openpose-control model to extract the pose information from the basketball image, and then using the  nlpconnect/vit-gpt2-image-captioning to generate a description of the boy in the image. Finally, I used the  iliasviel/td-controlnet-openpose model to combine the pose information and the description to generate the final image. The complete path of the generated image is  s5f8.jpg. If you have any other questions, please let me know.

Task Planning:

1. {'task': 'pose-detection', 'id': 0, 'dep': [-1], 'args': {'image': 'example1.jpg'}}
2. {'task': 'image-to-text', 'id': 1, 'dep': [-1], 'args': {'image': 'example2.jpg'}}
3. {'task': 'pose-text-to-image', 'id': 2, 'dep': [0, 1], 'args': {'text': "<resource>-1", "image": "<resource>-0"}}



Execution Results:



<resource>-0

<resource>-1

a young boy is riding a bike with a basket

iliasviel/td-controlnet-openpose



Pattern: Multi-agent Collaboration

Agent reasoning patterns

- With complex tasks, it is difficult to scale the behaviour of a single agent.
- This pattern involves several agents **working collaboratively**, each handling different aspects of the task.
- Separating responsibilities to agents allows **modularization**.
- We can define agents **specialized** in solving specific problems.
- Each agent can **use the same or different (e.g. fine-tuned) LLMs** for their expert task.

Source: Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). Communicative Agents for Software Development. ArXiv. /abs/2307.07924



Tools for building Agents



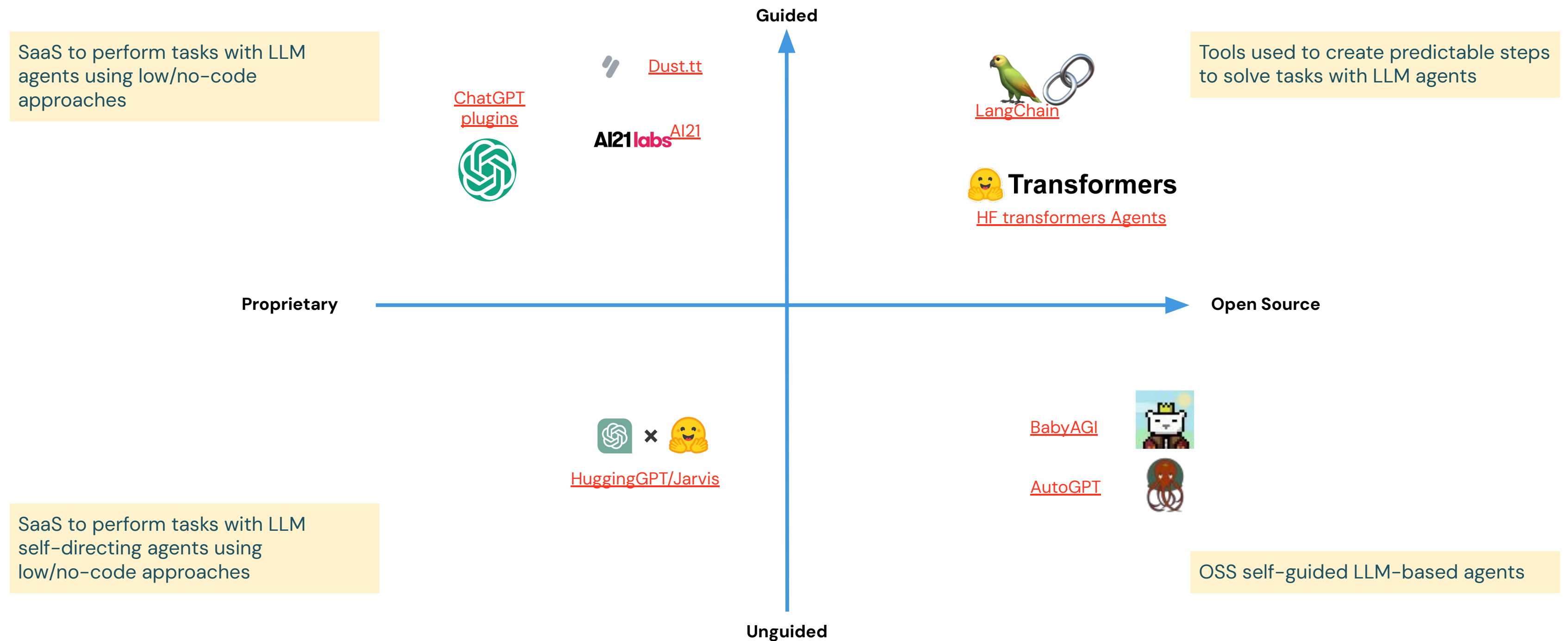
Tools to Build Agents

Notable tools and frameworks for building agents

- **LangChain Agents:** Provides a structure for building agents that can use tools to interact with the world
- **AutoGPT:** Provides tools to build AI agents.
- **OpenAI Function Calling:** Enables calling a set of functions defined in user query. The model can choose to call one or more functions.
- **Crew AI:** Enables collaborative intelligence by orchestrating autonomous AI agents to work together seamlessly on complex tasks.
- **Transformers Agents:** Provides a natural language API for interacting with transformers.



The Landscape of Tools/Frameworks



A Simple Agent with LangChain

Define Tools

```
from langchain.tools import BaseTool
from math import pi
from typing import Union

# Define tool1: A simple calculator for circle circumference
class CircumferenceTool(BaseTool):
    name = "CircumferenceCalculator"
    description = "Calculates the circumference of a circle given its radius"

    def _run(self, radius: Union[int, float]) -> float:
        return float(radius) * 2.0 * pi

# Define tool2: A hypothetical tool for retrieving weather information (pseudo
class WeatherRetrieverTool(BaseTool):
    name = "WeatherRetriever"
    description = "Retrieves current weather information for a given location"

    def _run(self, location: str) -> str:

        return f"Current weather in {location}: Sunny, 25°C"
```

Define Agent

```
from langchain.agents import initialize_agent
from langchain.llms import OpenAI
from langchain.tools import load_tools

# Assuming the LLM and API key setup from the previous snippet
openai_api_key = "your-api-key"
llm = OpenAI(openai_api_key=openai_api_key)

# Initialize custom tools
circumference_tool = CircumferenceTool()
weather_retriever_tool = WeatherRetrieverTool()

# Load tools into the agent
tools = [circumference_tool, weather_retriever_tool]

# Initialize the agent with the LLM and custom tools
agent = initialize_agent(llm=llm, tools=tools, agent_kwargs={})

agent.run("...")
```



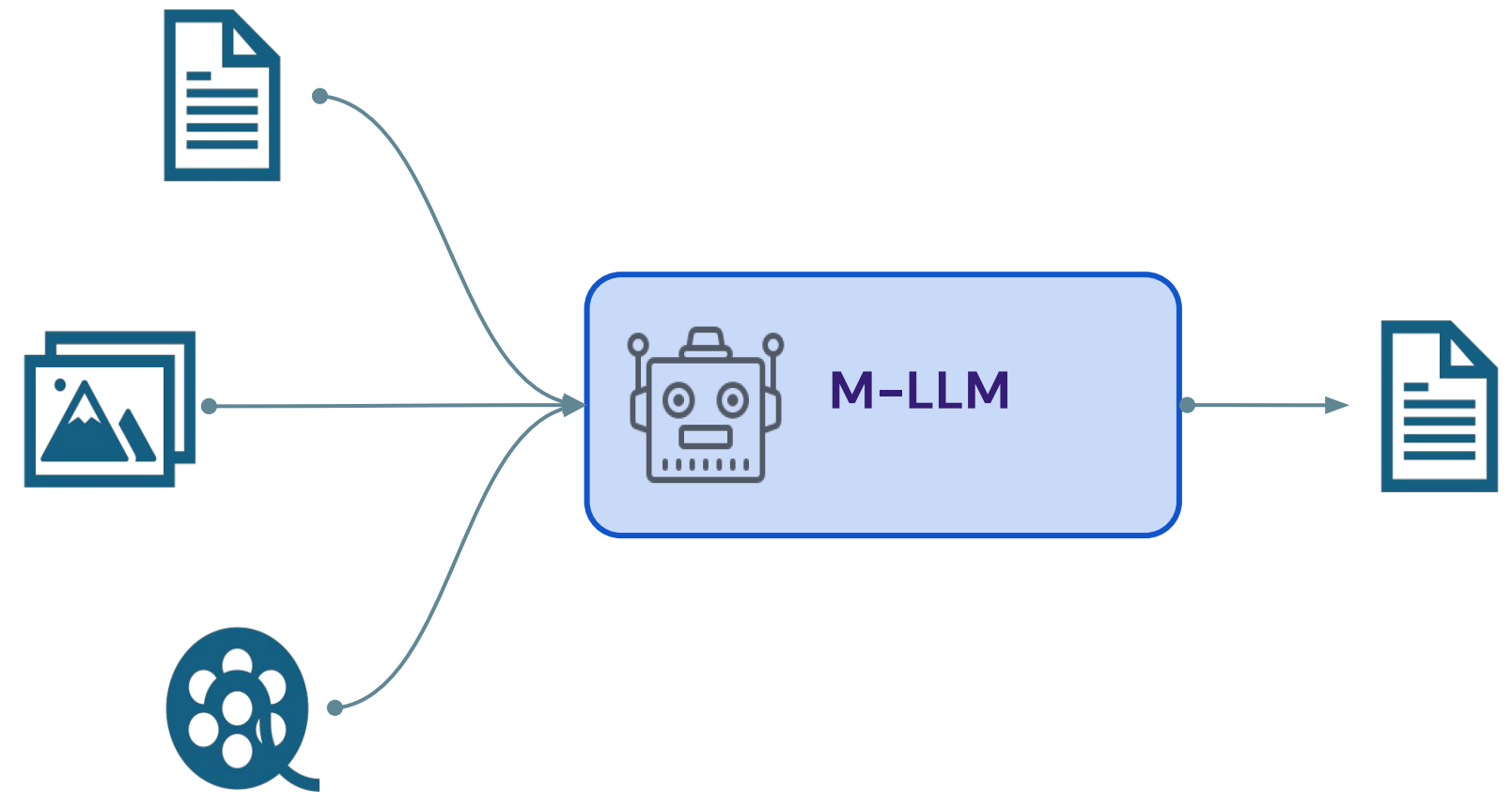
Multi-Modal AI



Multi-modal AI Applications

Multi-modal Models:

- Models with inputs or outputs that include data types beyond text.
- Common data types include image, audio, and video.



Example Application: Smart home assistant

A user can ask the assistant to identify an object in a room, which it does by analyzing the camera feed and providing a description, while also enabling follow-up text queries for more details.



Multi-modal Architectures

This area is actively being explored.

Multi-Modal Retrieval

- Each modality has its own challenges
- Approaches to deal with storing different modalities
 - Method 1: Embed all modalities in the **same vector space** (e.g. [CLIP](#)).
 - Method 2: **Select a main modality** based on the application's focus and **ground all other modalities to this primary one.**
 - Method 3: Embed each modality **separately.**

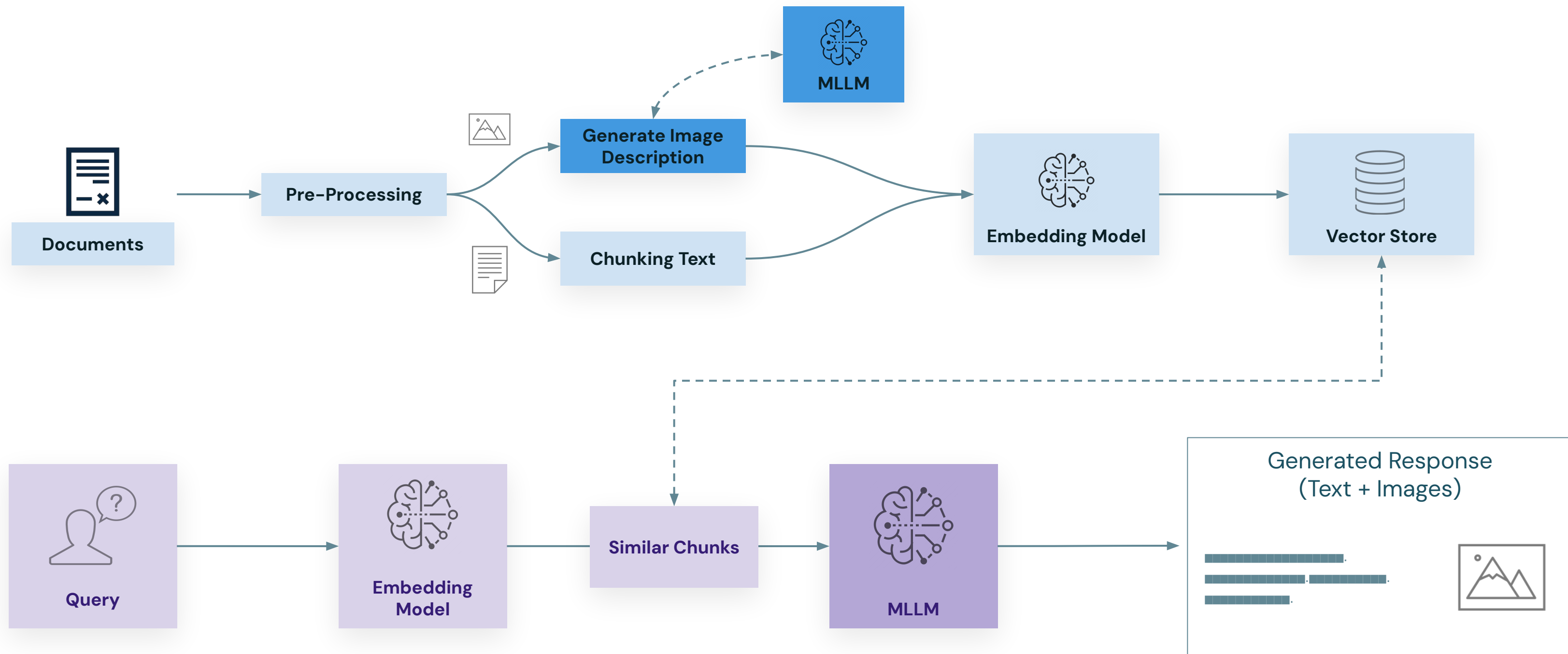
Multi-Modal Generator

- Emerging **Multi-modal LLMs (MLLMs)** such as GPT-4V(ision) enable generating responses in multiple formats.
- Example: Generating a story with images.



Multi-modal Architectures

A sample multi-modal architecture



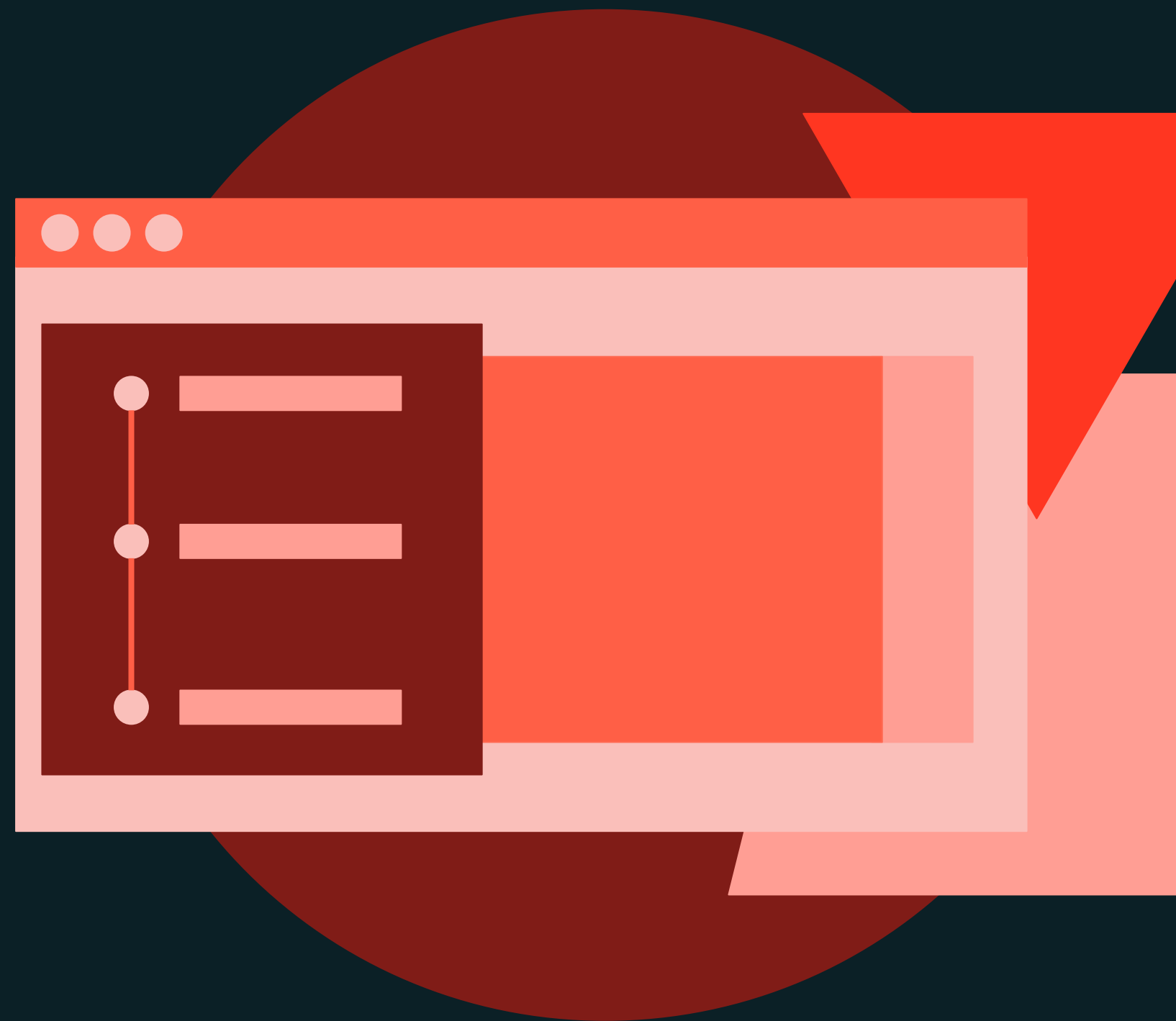
Sample Research Paper: Retrieval-Augmented Multimodal Language Modeling. [ArXiv: /abs/2211.12561](https://arxiv.org/abs/2211.12561)





DEMONSTRATION

Agent Design in Databricks



Demo Outline

What we'll cover:

- Create an autonomous agent 1
 - Define the brain of the agent
 - Define tools that the agent can use
 - Define planning logic
- Create an autonomous agent 2
 - Prepare dataset
 - Define the the brain and tools to use
 - Talk with the agent





LAB EXERCISE

Create a ReAct Agent



Lab Outline

What you'll do:

- **Task 1:** Define the agent brain
- **Task 2:** Define the agent tools
- **Task 3:** Define an agent logic
- **Task 4:** Create the agent
- **Task 5:** Run the agent



Summary and Next Steps

Generative AI Application Development

