lyzr

Agent Architect Cohort 1

Basic Concepts

Agent Architect Cohort

Day 1 : Basic Concepts

- Understanding Large Language Model
- Different Types of LLMs
- Vector Databases
- Retrieval-Augmented Generation (RAG)
- Writing Effective Prompts
- How Agents actually evolved?
- LLM v/s Agents v/s Workflows
- Agent Orchestrations

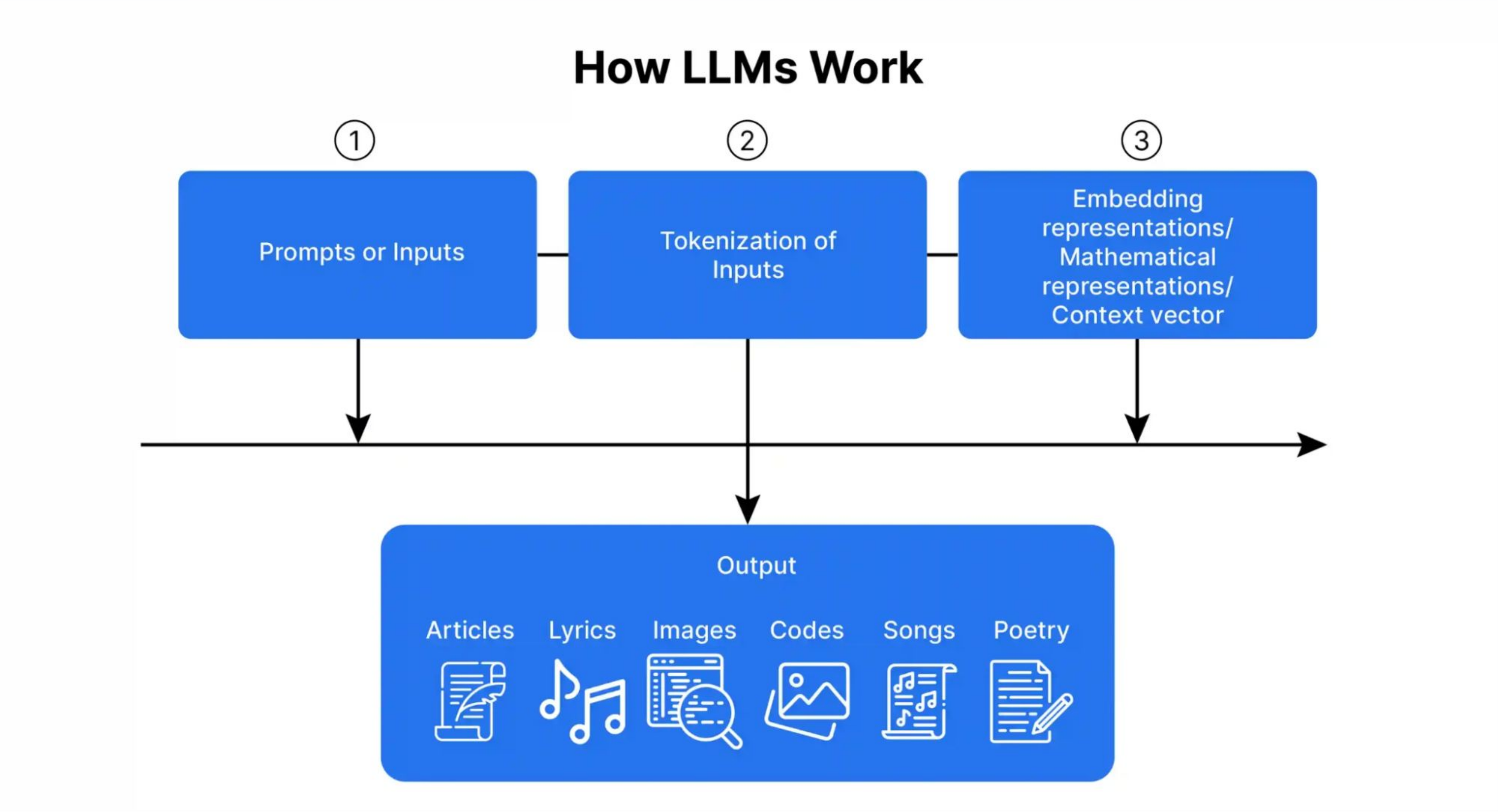
Understanding Large Language Model

What are LLMs?
Large Language Models are AI systems trained on massive text data that can understand and produce natural-sounding language.

How they work:
They predict one word at a time, which lets them generate text, summarize content, translate languages, and answer questions.

Why they matter for agents:
Their growing ability to reason through multi-step problems makes them the “brain” of modern AI agents.

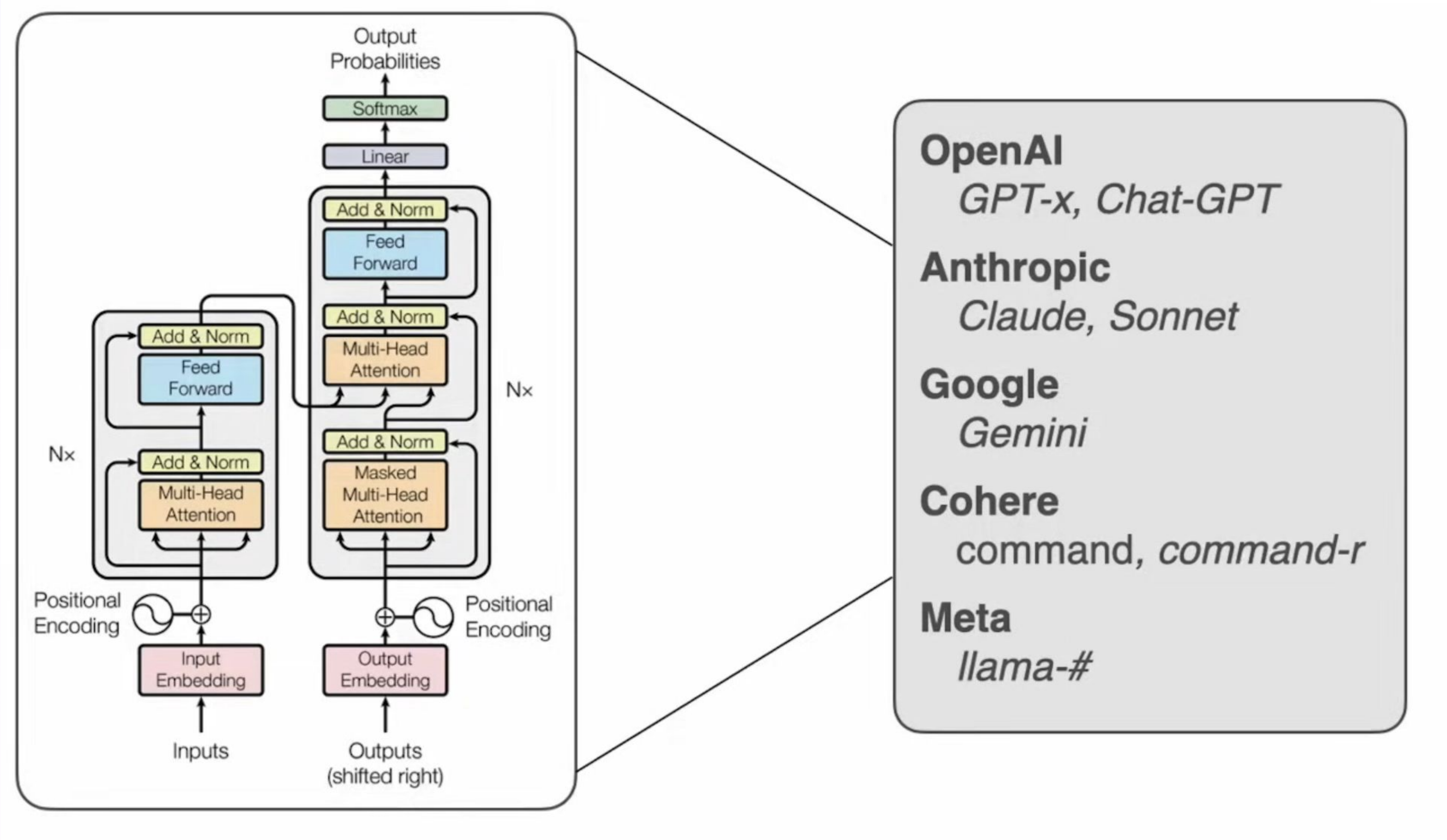
Recent breakthroughs:
Better reasoning, multimodality, and built-in “tool use” have paved the way for richer, more capable agentic systems.



Understanding Large Language Model

The Transformer architecture revolutionized natural language processing by enabling models to process entire sequences simultaneously using self-attention mechanisms. This architecture comprises:

- **Encoder**: Processes the input sequence and generates a contextual representation.
- **Decoder**: Generates the output sequence by attending to the encoder's output and previously generated tokens.



lyzr

Different Types of LLMs

🔍 These models simulate intelligence by compressing the distribution of human knowledge. Reasoning is emergent — not designed, but learned as a side-effect of scale + data richness.

Frontier LLMs (GPT, Claude, Gemini)

🔹 Goal: General-purpose cognition at scale
🔹 Optimized for: Conversational fluency, instruction-following, multimodal tasks

ChatGPT Claude Gemini

Open-Source LLMs (Mistral, LLaMA, NVIDIA)

🔹 Goal: Democratized access to performant LLMs
🔹 Optimized for: Customization, fine-tuning, modularity

Mistral AI LLaMA by Meta NVIDIA

🔍 These aren't trained to reason, but give builders the raw foundation to scaffold their own logic, rules, and behaviors — making them ideal for fine-tuned workflows.

🔍 These models don't reason — they recall and summarize well within narrow bounds. Think autocomplete with a memory, not a brain.

SLMs (Small Language Models - Phi, TinyLLaMA, Gemma)

🔹 Goal: Efficiency over raw intelligence
🔹 Optimized for: Edge devices, mobile inference, ultra-low latency

Microsoft Phi-3 Gemma

Reasoning LLMs (DeepSeek, Qwen, GPT-o3)

🔹 Goal: Structured, step-wise logical inference
🔹 Optimized for: Math, coding, planning, chain-of-thought tasks

deepseek Qwen

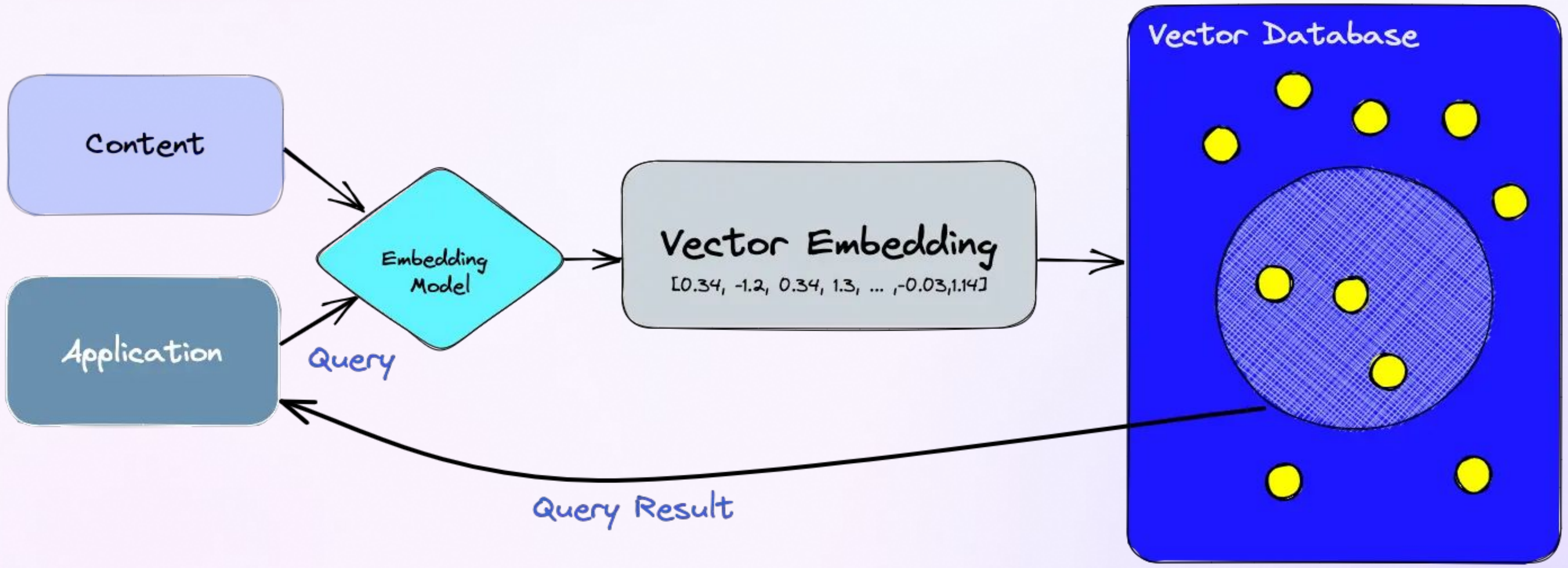
🔍 Reasoning LLMs are optimized not just to predict next tokens, but to simulate cognitive steps — breaking problems into sub-tasks, verifying answers, and self-correcting. This is where the line between "language model" and "thinking machine" starts to blur.

Vector Databases

Vector databases are built to handle high-dimensional **vector embeddings** - mathematical representations of data like text, images, or audio.

- They're essential for **semantic search**, which finds results based on meaning and context, not just exact keywords.
- In AI agents and **RAG (Retrieval-Augmented Generation)** systems, these databases store embeddings of documents or knowledge, enabling agents to pull in the most relevant information during a task.
- The workflow:
  1. Text is converted into vector embeddings using an **embedding model**.
  2. These embeddings are stored and indexed in the vector database.
  3. When a query comes in, the system searches for the closest matching vectors using efficient algorithms like **SCaNN**. (Scalable Nearest Neighbors)

This capability powers smart, context-aware retrieval—critical for building intelligent, responsive AI systems.



lyzr

Vector Databases

1. Weaviate

- Vector-native DB with built-in **hybrid search** (vector + keyword).
- **Modular integrations** for auto-embedding (OpenAI, Cohere, etc.).
- Ideal for **production-grade RAG** with strong filtering and metadata support.

Feature	Weaviate	PgVector	Qdrant	MongoDB Vector	Chroma
Infra Type	Native vector DB	Postgres plugin	Rust-native DB	Document DB	Local/embedded
Hybrid (Text + Vector)	✅ Yes	❌ No	✅ Basic	✅ Yes	✅ Basic
Metadata Filtering	✅ Advanced	✅ via SQL	✅ Fast	✅ Good	✅ Simple
LLM Integration	✅ Built-in	❌ Manual	❌ External	❌ Manual	✅ Built-in
Scaling	✅ Cloud-native	✅ RDBMS-scale	✅ High	✅ Mongo-native	❌ Not scalable
Ideal For	Production RAG	Relational + vec	High-perf RAG	Doc-based RAG	Agent memory/dev

2. PgVector

- Adds vector search to **PostgreSQL** via extension.
- Supports **exact and ANN search** with SQL compatibility.
- Best for teams already using relational data needing light vector ops.

3. Qdrant

- High-performance, **Rust-based** vector engine.
- Excellent **payload filtering** and fast search at scale.
- Great for scalable **LLM apps with structured metadata**.

4. MongoDB Atlas Vector Search

- Embeds vectors directly in **BSON documents**.
- Combines **vector + document queries** in one platform.
- Best when working with **chat logs, JSON, or schema-less data**.

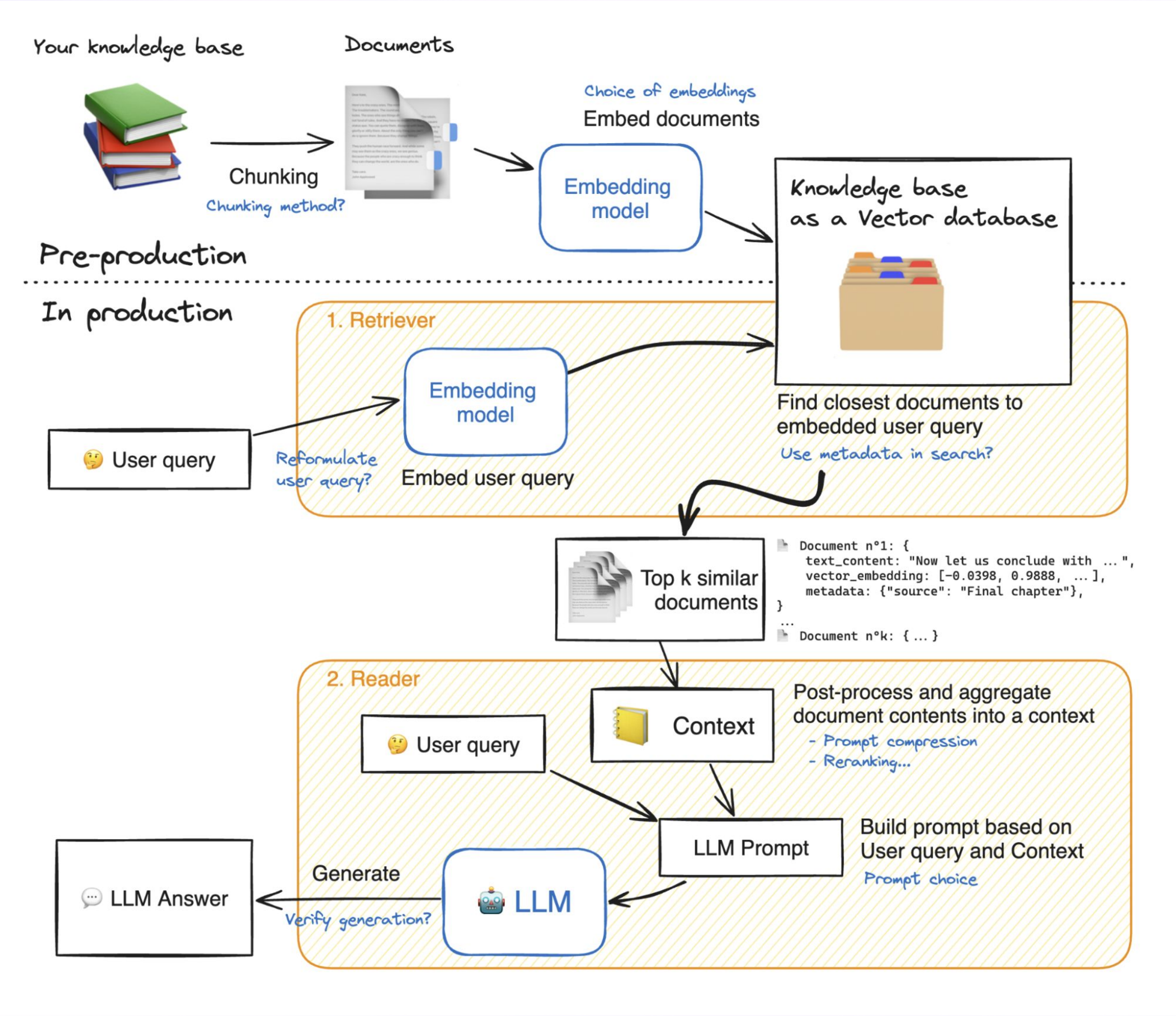
5. Chroma

- Lightweight, **LLM-first local vector store**.
- Built-in memory-style ingestion with simple API.
- Ideal for prototyping **agent memory** or local RAG workflows.

Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a technique that improves the responses of Large Language Models by pulling in relevant information from external knowledge sources.

- It works by retrieving contextually relevant data and supplying it to the LLM during generation.
- This helps **ground responses in factual, current information**, reducing hallucinations and boosting accuracy - especially for topics not well covered in the model's original training.
- Traditional RAG pipelines follow a **static process**:
    1. Retrieve relevant documents from a vector database
    2. Feed the retrieved content to the LLM
    3. The LLM synthesizes the final response based on this context.



Retrieval-Augmented Generation (RAG)

Naive RAG - *Straightforward Search & Retrieve*

- Basic RAG approach that retrieves top-matching text chunks using simple semantic similarity and feeds them directly to the LLM.

Advanced RAG - *Smarter Retrieval with Relevance Optimization*

- Uses enhanced retrieval methods (e.g. hybrid search, reranking) and better integration with LLMs to handle complex or nuanced queries.

Multimodal RAG - *Text + Images (or More)*

- Extends RAG beyond text to include multiple data types like images, videos, or audio, enabling retrieval and generation across modalities.

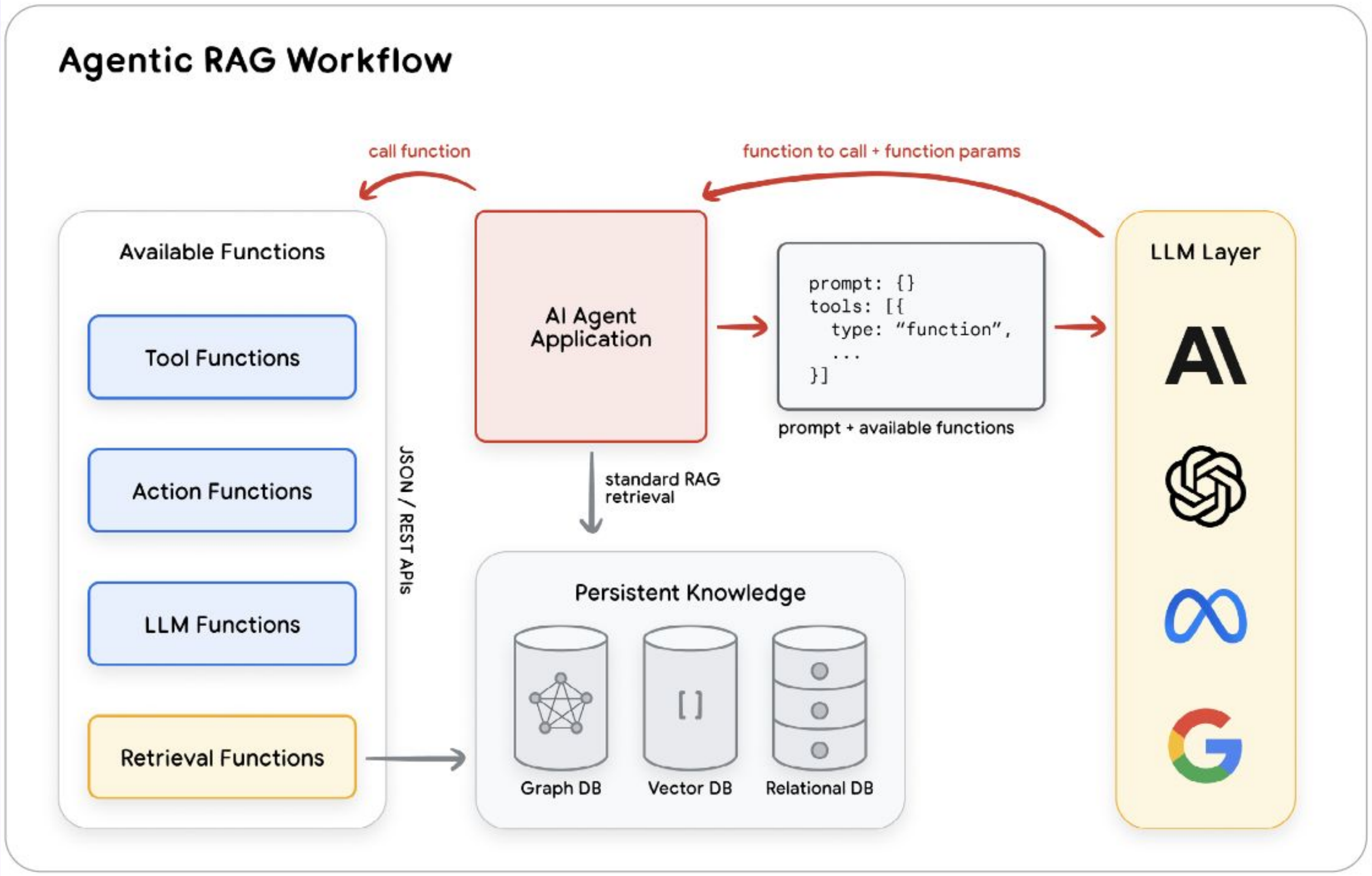
RAG Approaches: When to Use What

	Naive RAG	Advanced RAG	Agentic RAG	Multimodal RAG
When to Use:	• Simple QA Systems • Single-hop Queries	• Complex Queries • Context-Heavy Tasks	• Multi-step Tasks • Complex Reasoning	• Mixed Media Data • Visual + Text Tasks
Scenarios:	• HR Pollcy Lookup • Product FAQs • Basic Support • Knowledge Bearch • Knowledge Base	• Legal Research • Medical Literature • Technical Docs • Research Analysis • Expert Systems	• Strategy Planning • Market Analysis • Code Generation • Research • Problem Solving	• Medical Imaging • Visual Search • E-commerce • Content Analysis • Visual QA
Requirements:	• Basic Vector DB • Simple Embeddings • Basic LLM • Fast Response Time	• Hybrid Search • Query Expansion • Reranking • Advanced LLM	• Task Planning • Chain-of-Thought • Multiple Agents • Strong LLM	• Image Processing • Multi-Embeddings • Cross-Attention • Vision Models

lyzr

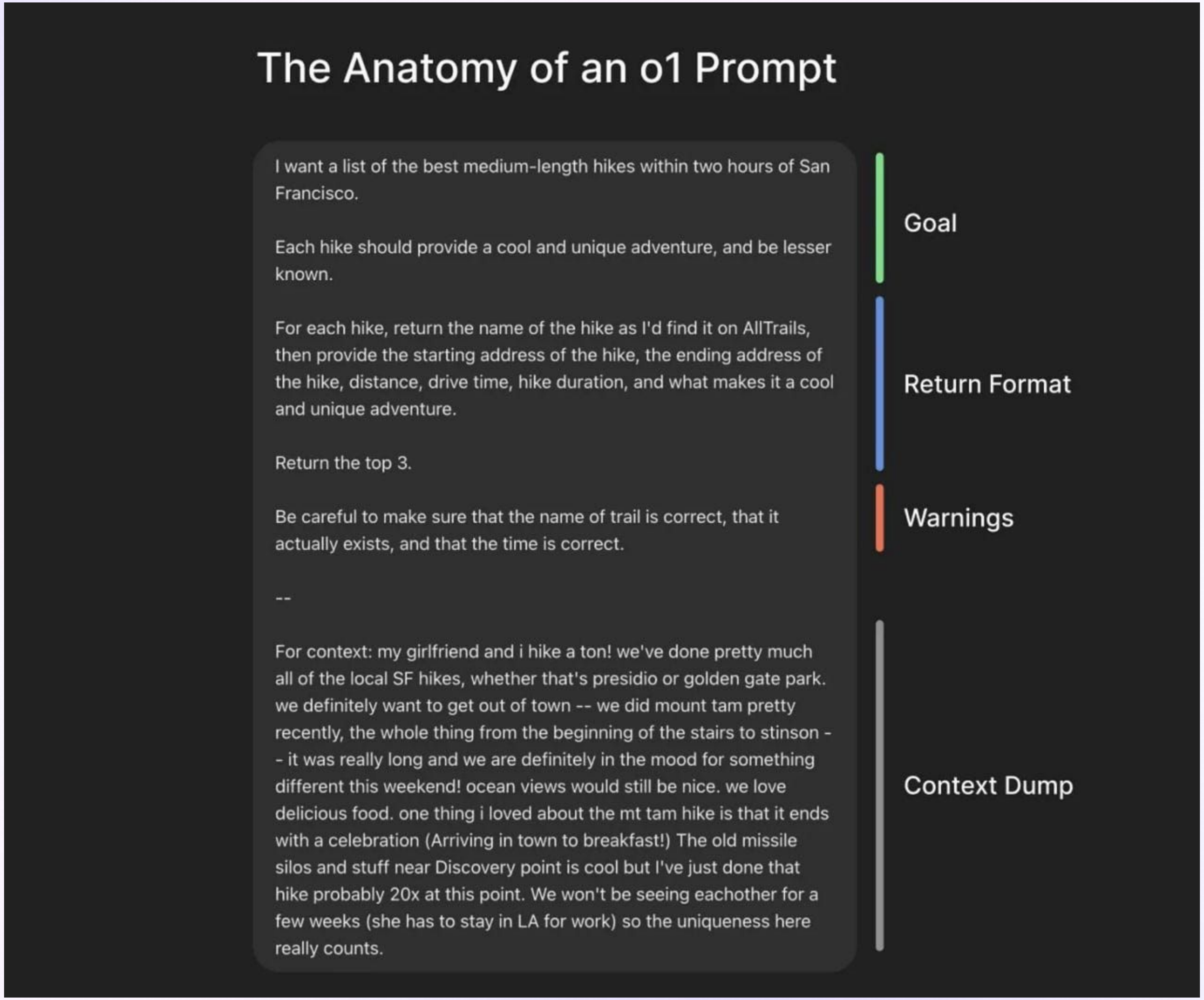
Agentic RAG

- **What it is:** An evolution of RAG that embeds autonomous AI agents into the retrieval loop.
- **How it works:** Agents orchestrate the end-to-end flow - selecting sources, judging quality, and deciding how each retrieved chunk should influence the LLM's answer.
- **Accuracy boost:** By vetting and ranking sources in real time, agents discard low-quality evidence and elevate trustworthy data, yielding more reliable outputs.
- **Deeper context:** Agents reason over both the user's intent and the retrieved material, enabling responses that are richer, more relevant, and better aligned with the query's nuance.
- **Adaptive retrieval:** Agents continuously refine their search strategies as new information appears, keeping answers current in fast-moving fields such as healthcare, finance, and law.



Prompt Engineering

- Prompt engineering is the skill of crafting **effective inputs (prompts)** to guide LLMs toward desired outputs.
- It's crucial for **controlling LLM behavior, improving response quality, and achieving specific tasks** with agents.
- Good prompts are **typically specific, provide sufficient context, and clearly define the desired task** and output format.
- Iteration and refinement are key; one might need to try several prompts to achieve the best results.



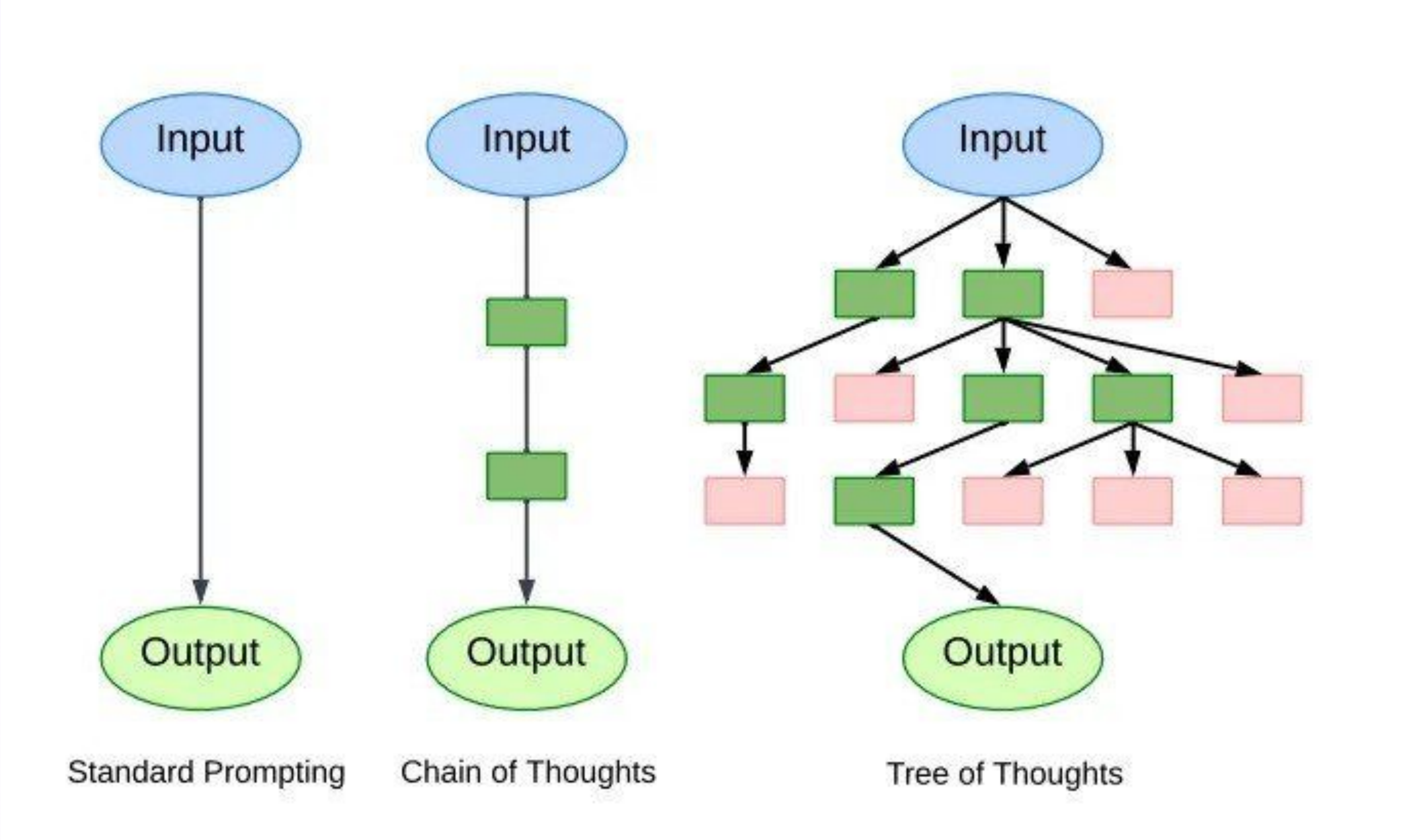
Writing Effective Prompts

- **Structure**: A well-structured prompt often includes elements like role, task, context, background data/documents, detailed rules, conversation history, immediate request, and output formatting instructions.
- **Clarity and Specificity**: Clearly define the task the LLM should perform (e.g., summarize, write, classify, translate). Be explicit about the desired output and any constraints.
- **Context is Key**: Provide relevant background information, documents (e.g., by tagging files in Gemini for Workspace), or examples to guide the LLM.
- **Iterative Refinement**: Treat prompting as a conversation; fine-tune prompts based on initial outputs if they don't meet expectations. Start simple and add complexity as needed.

Agent Persona	You are an expert legal contract document writer
Agent Goal	Your goal is to write high quality professional contract documents
Agent Context	You work for an IT Services firm. The firm signs contracts with its enterprise customers. The firm provides professional services.
Agent Task	Understand the input request from the user. Ask clarifying questions if required. Write a draft contract document by referring to the example provided. Review the document step by step to ensure that you have written the document as per the context.
Agent Output	Write the output in contract agreement style.
Agent Example	'Sample Contract.pdf'

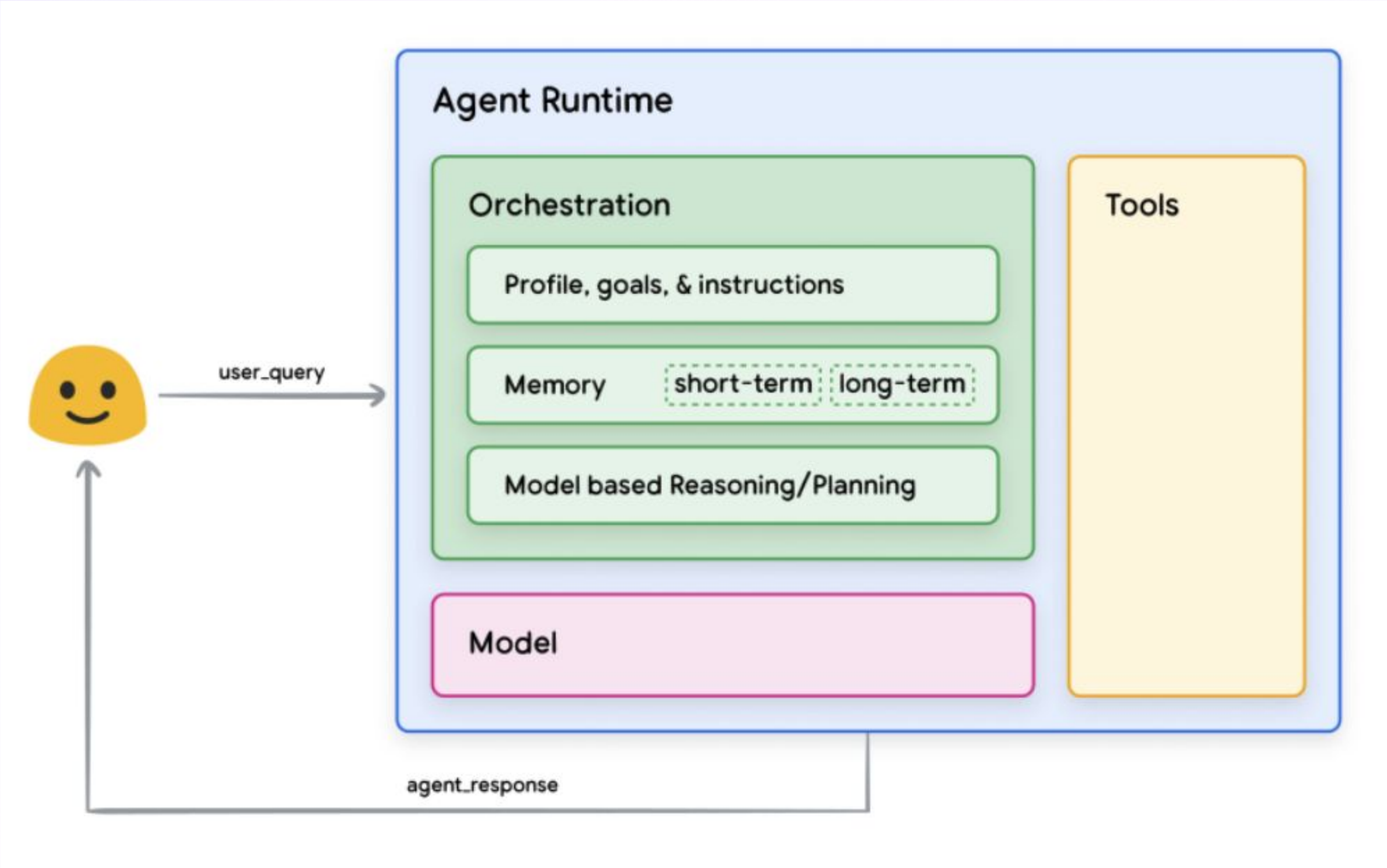
Best Practices for Agent Instructions

- Prompt agents to **break down complex tasks into smaller,** manageable steps to minimize ambiguity.
- Ensure every step in a routine corresponds to a specific, **clearly defined action** or output.
- Anticipate and include instructions for **handling common variations, edge cases, and conditional logic.**
- Advanced models can assist in automatically generating instructions from existing help center documents or policies.



What is an Agent?

- An application that acts autonomously to achieve goals using available tools.
- Operates independently, reasoning proactively without constant human input.
- Moves beyond automation by performing workflows on behalf of users.



General agent architecture and components

'Agents' by Google : https://www.kaggle.com/whitepaper-agents

LLM v/s Agents

- **LLMs are flexible but unstructured** - like water - while agents act as containers, giving them structure, memory, and purpose.
- Agents go beyond basic function-calling by adding **memory, event triggers, tool use, and orchestration**, enabling automation without repeated instructions.
- Unlike ChatGPT, agents persist context and workflows, making interactions **consistent, scalable, and production-ready**.
- Lyzr enhances agents with **enterprise-grade features** like hallucination control, responsible AI modules, and central governance—ensuring trust and reliability at scale.

How Agents actually evolved?

How it started?

This is the JSON of OpenAI's core function calling - including system prompt, temperature, model and top_p only.

System prompt + model + temperature + top_p + one function — that's it.

json

Copy Edit

```json
{
  "model": "gpt-3.5-turbo",
  "temperature": 0.7,
  "top_p": 1,
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful assistant."
    },
    {
      "role": "user",
      "content": "Hello! How are you today?"
    }
  ]
}
```

How Agents actually evolved?

How it's going - from single prompts to intelligent systems.
Lyzr Agent API :

json

```json
{
  "name": "",
  "description": "",
  "agent_role": "",
  "agent_instructions": "",
  "features": [
    { "type": "KNOWLEDGE_BASE", ... },
    { "type": "SHORT_TERM_MEMORY", ... },
    { "type": "LONG_TERM_MEMORY", ... },
    { "type": "HUMANIZER", ... },
    { "type": "RAI", ... },
    { "type": "SRS", ... },
    { "type": "GROUNDEDNESS", ... },
    { "type": "CONTEXT_RELEVANCE", ... }
  ],
  "tools": [],
  "provider_id": "OpenAI",
  "temperature": "0.7",
  "top_p": "0.9",
  "response_format": { "type": "text" }
}
```

🧠 Knowledge Base (Lyzr RAG)

json Copy Edit

```json
{
  "type": "KNOWLEDGE_BASE",
  "priority": 0,
  "config": {
    "lyzr_rag": {
      "base_url": "https://rag-prod.studio.lyzr.ai",
      "rag_id": "682810fc9e04ce9a597220e7",
      "rag_name": "prd_knowledge_base86mz",
      "params": {
        "top_k": 10,
        "retrieval_type": "basic",
        "score_threshold": 0
      }
    }
  }
}
```

🧠 Short-Term Memory

json

```json
{
  "type": "SHORT_TERM_MEMORY",
  "priority": 0,
  "config": {}
}
```

🧠 Long-Term Memory

json

```json
{
  "type": "LONG_TERM_MEMORY",
  "priority": 0,
  "config": {}
}
```

🛡️ Responsible AI (RAI)

json

```json
{
  "type": "RAI",
  "priority": 0,
  "config": {
    "endpoint": "https://rai-prod.studio.lyzr.aiv1/rai/inference",
    "policy_id": "683444af82e81ef927d4190a",
    "policy_name": "RAI_demo"
  }
}
```

Agents vs. models

To gain a clearer understanding of the distinction between agents and models, consider the following chart:

Models	Agents
Knowledge is limited to what is available in their training data	Knowledge is extended through the connection with external systems via tools
Single inference / prediction based on the user query. Unless explicitly implemented for the model, there is no management of session history or continuous context. (i.e. chat history)	Managed session history (i.e. chat history) to allow for multi turn inference / prediction based on user queries and decisions made in the orchestration layer. In this context, a 'turn' is defined as an interaction between the interacting system and the agent. (i.e. 1 incoming event/ query and 1 agent response)
No native tool implementation.	Tools are natively implemented in agent architecture.
No native logic layer implemented. Users can form prompts as simple questions or use reasoning frameworks (CoT, ReAct, etc.) to form complex prompts to guide the model in prediction.	Native cognitive architecture that uses reasoning frameworks like CoT, ReAct, or other pre-built agent frameworks like LangChain.

lyzr

Conventional Software vs. Agents:

- Conventional software automates by following predefined user-driven instructions to streamline tasks.
- Agents perform workflows on behalf of users with high independence.

Simple LLM Applications vs. Agents:

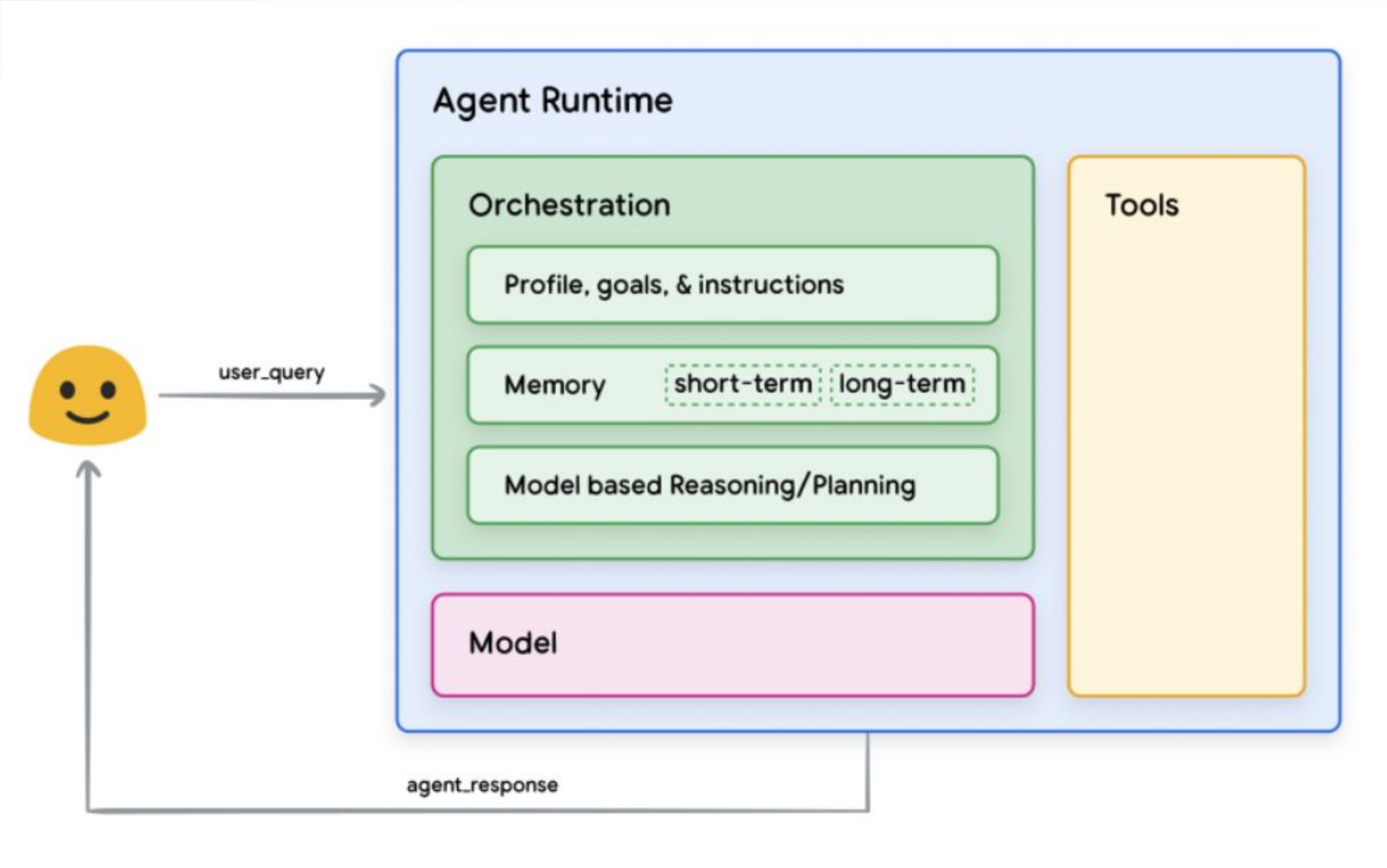
- Simple LLM apps (e.g., basic chatbots, sentiment classifiers) integrate LLMs but don't use them to control workflow execution and are thus not agents.
- Agents leverage an LLM to manage workflow execution and make decisions.

What is Agent Orchestration?

Definition: Agent orchestration describes the cyclical process that

- **governs how an agent ingests information,**
- **performs internal reasoning, and**
- **uses that reasoning to inform its next action or decision.**

This loop typically continues until the agent achieves its goal or reaches a designated stopping point. The orchestration layer is responsible for maintaining memory, state, reasoning, and planning.



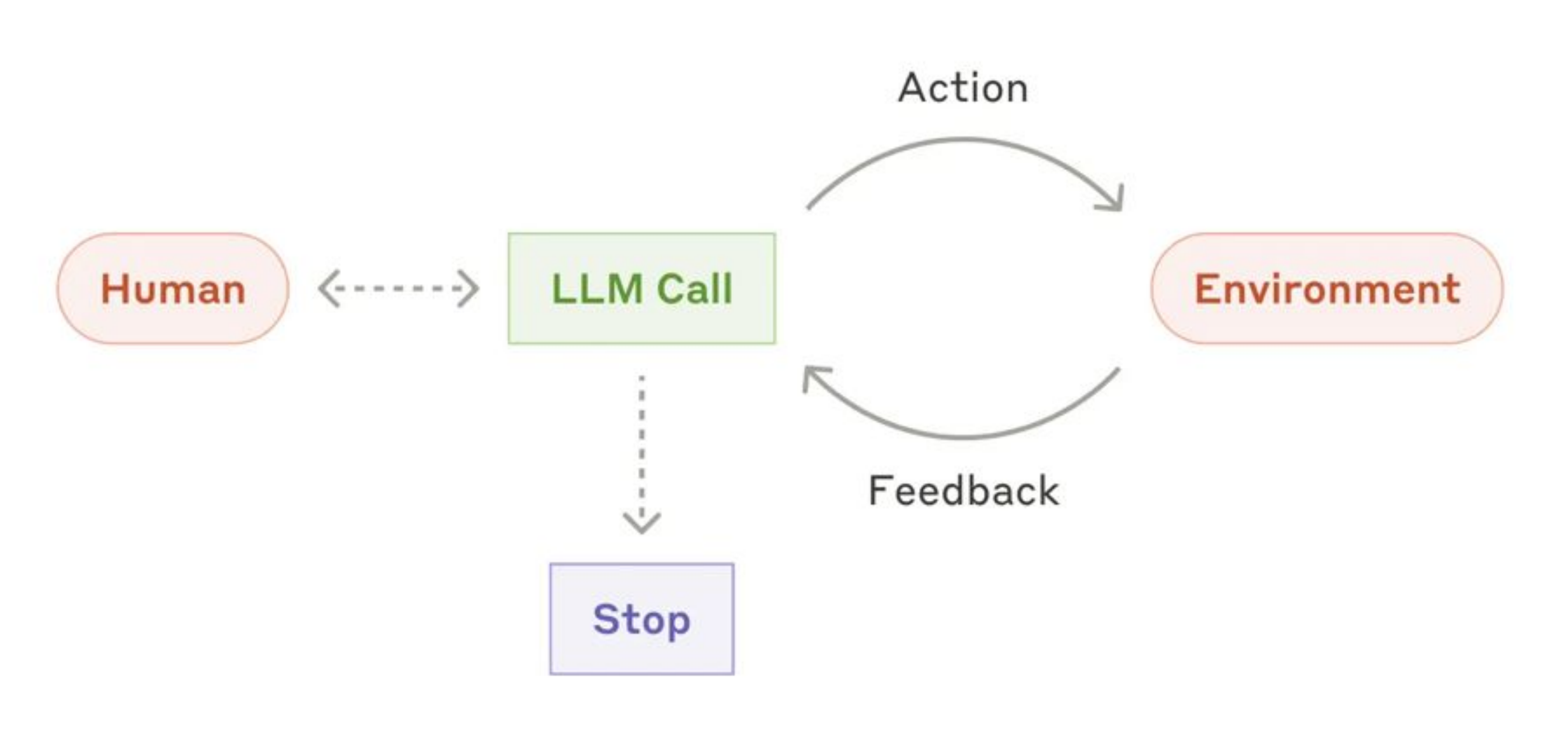
General agent architecture and components

'Agents' by Google : https://www.kaggle.com/whitepaper-agents

Agents v/s Workflows

Anthropic categorizes agentic systems with an important architectural distinction between Workflows and Agents.

- **Workflows** are systems where Large Language Models (LLMs) and tools operate through predefined code paths.
- **Agents,** in contrast, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.



Building Effective Agents by Anthropic :
https://www.anthropic.com/engineering/building-effective-agents

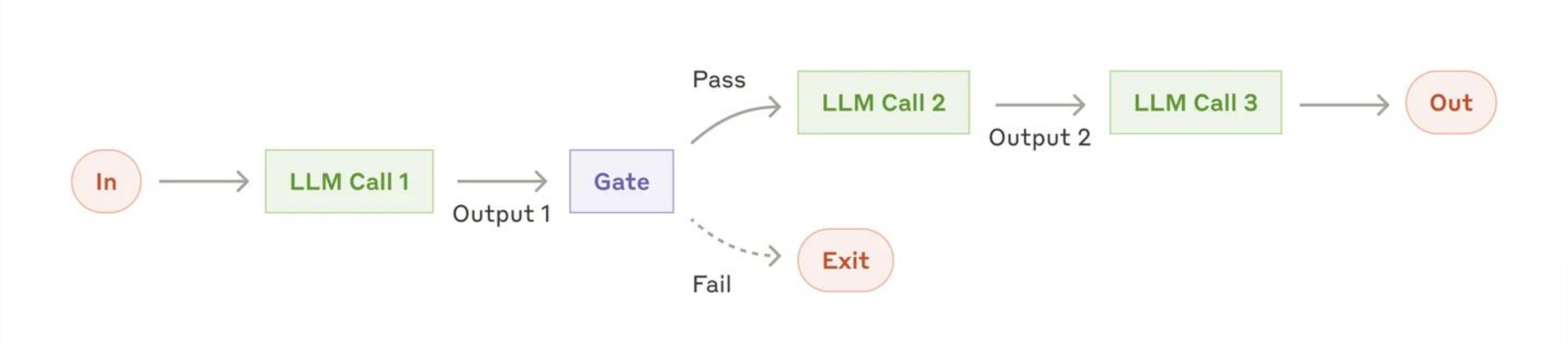
Types of Agent Orchestrations

Orchestration patterns can be categorized into single-agent and multi-agent systems.

Single-Agent Systems:

A single model equipped with tools and instructions executes workflows in a loop.

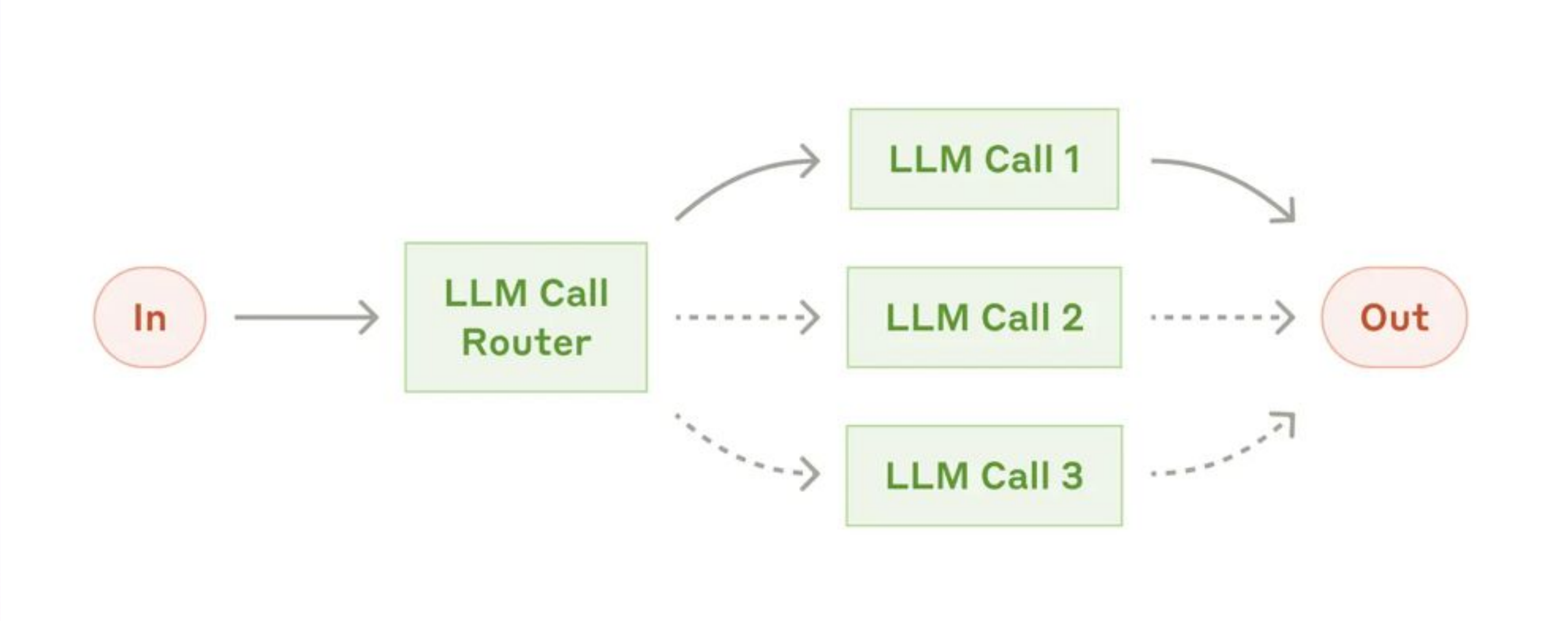
- **Workflow Patterns (often single or sequential agent setups):**
    - **Prompt Chaining:** Decomposes a task into a sequence of steps, where each LLM call processes the output of the previous one. Programmatic checks can be added at intermediate steps.



Types of Agent Orchestrations

Single-Agent Systems:

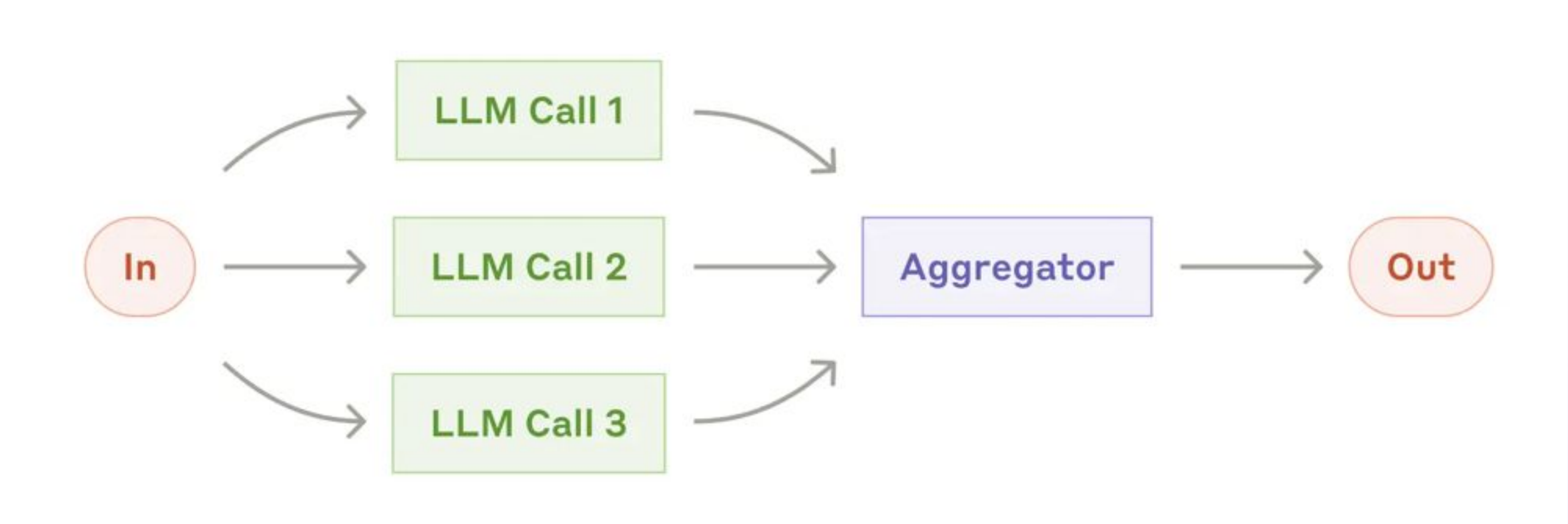
- **Routing:** Classifies an input and directs it to a specialized follow-up task or LLM.



Types of Agent Orchestrations

Single-Agent Systems:

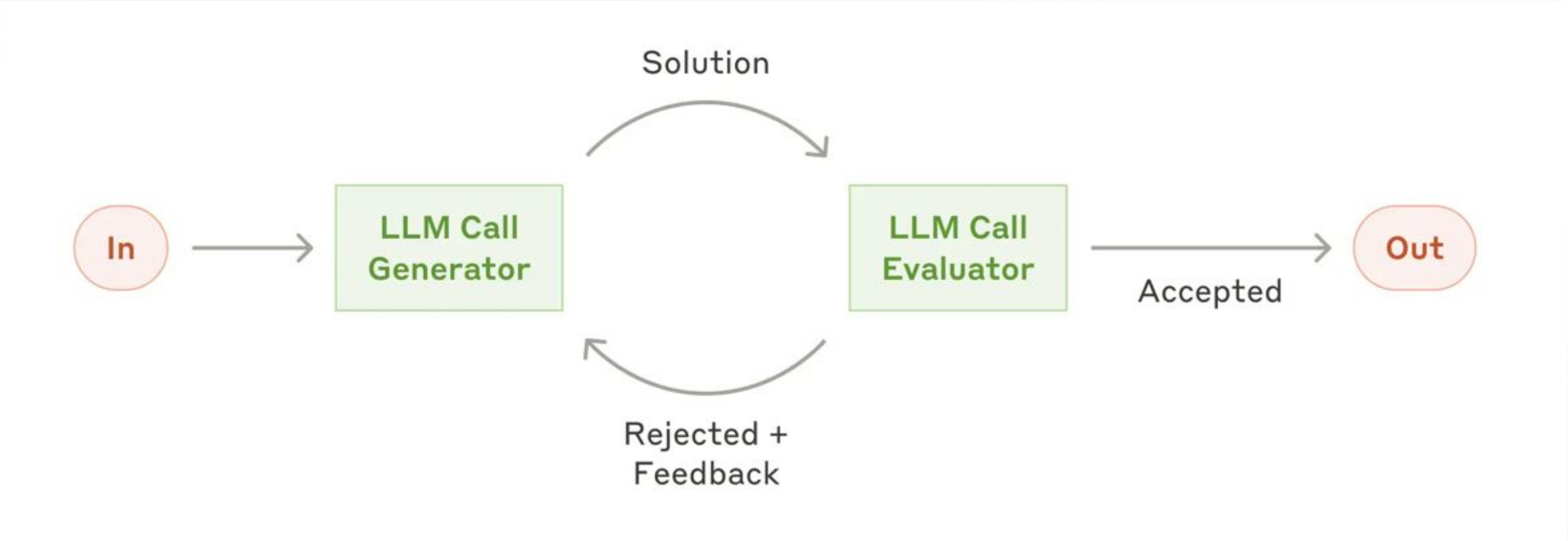
- **Parallelization:** LLMs work simultaneously on a task (either sectioning into subtasks or voting on multiple attempts), and outputs are aggregated.



Types of Agent Orchestrations

Single-Agent Systems:

- **Evaluator-Optimizer:** One LLM generates a response, while another provides evaluation and feedback in a loop for iterative refinement.

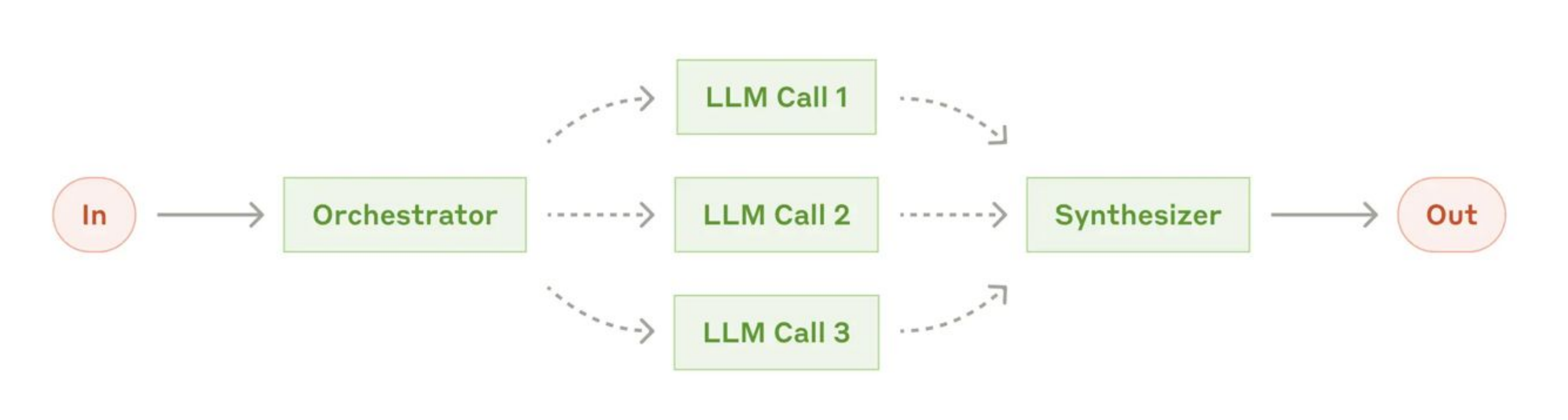


Types of Agent Orchestrations

Multi-Agent Systems:

Workflow execution is distributed across multiple coordinated agents.

- **Orchestrator-Workers (a type of multi-agent system):** A central LLM dynamically breaks down tasks, delegates them to worker LLMs, and synthesizes their results.

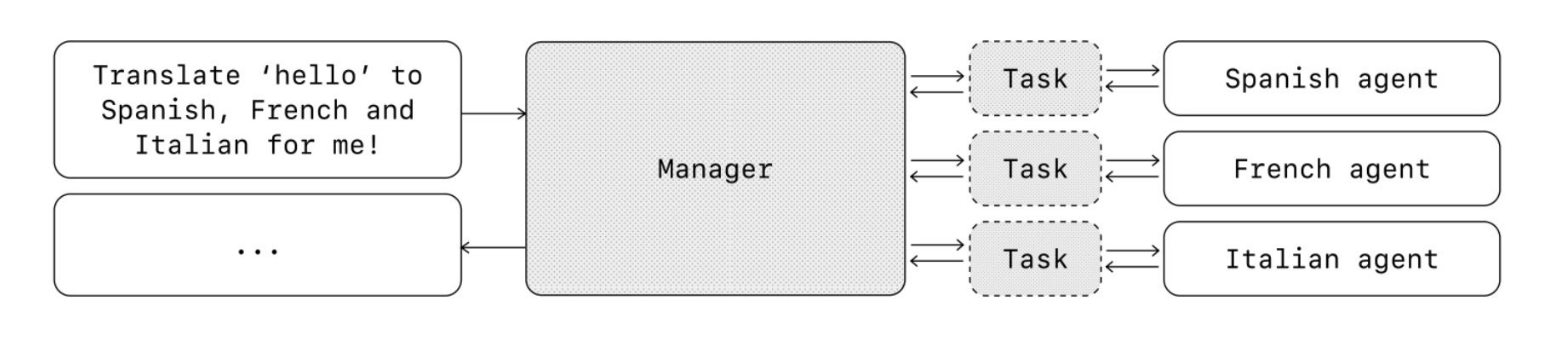


Types of Agent Orchestrations

Multi-Agent Systems:

Workflow execution is distributed across multiple coordinated agents.

- **Manager (Agents as Tools):** A central "manager" agent coordinates multiple specialized agents via tool calls. The manager delegates tasks and synthesizes results.



A Practical Guide to Building Agents by OpenAI :
https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf

Types of Agent Orchestrations

Multi-Agent Systems:

- Multi-Agent Design Patterns:
  - **Sequential:** Agents work sequentially, each completing its task before passing output to the next.
  - **Hierarchical:** A "manager" agent coordinates and delegates tasks to "worker" agents. (Similar to OpenAI's Manager pattern).
  - **Collaborative:** Agents work together, sharing information and resources to achieve a common goal.
  - **Competitive:** Agents may compete to achieve the best outcome or contribute to a shared goal where resources might be contended.

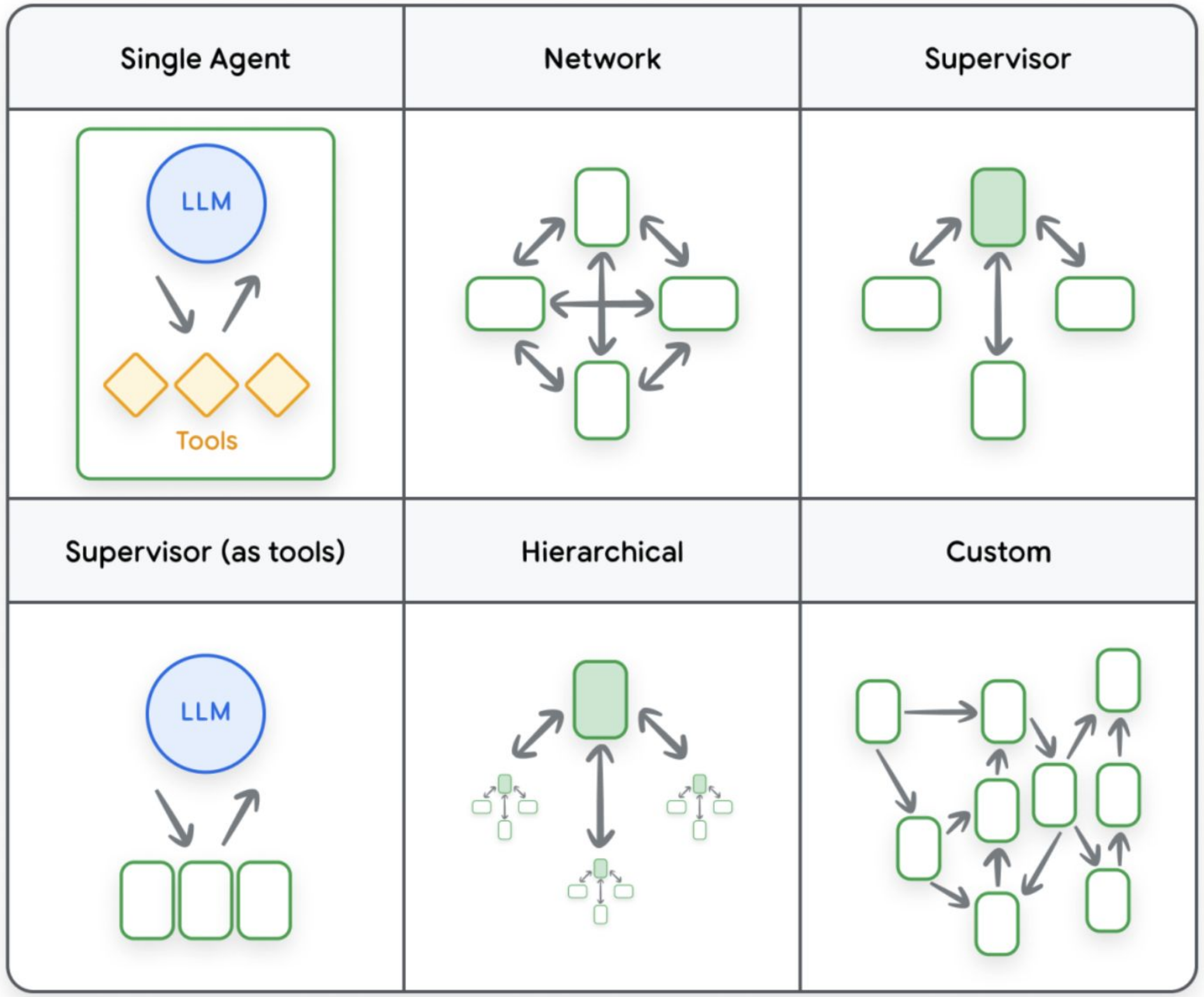


Figure 7: An image depicting different multi-agent topologies, from LangGraph documentation.[23]

Agents Companion by Google : https://www.kaggle.com/whitepaper-agent-companion

Why is Agent Orchestration powerful?

Agent orchestration allows for dynamic, flexible, and adaptive problem-solving that goes beyond simpler, more rigid automation techniques.

Differences from prompt chaining:

- Prompt chaining involves a predefined sequence of LLM calls. While useful for decomposing tasks, it **lacks the dynamic decision-making** and tool use capabilities inherent in more complex agentic systems where an LLM actively directs the process.
- Agent orchestration allows the agent to choose its next steps, tools, and even invoke other agents based on the evolving context.

Differences from "vibe coding" (Interpreted as intuitive, less structured coding):

- Agentic systems, while leveraging the natural language strengths of LLMs, aim for more structured and reliable execution through defined tools, instructions, and orchestration logic.
- The goal is to create robust and predictable systems, whereas "vibe coding" might imply less formal or tested approaches.

Differences from workflow builders like n8n:

- Workflow builders typically define deterministic, rule-based automation flows. While powerful for many tasks, agent orchestration introduces a layer of AI-driven reasoning and decision-making, allowing the system to handle more ambiguity, adapt to unforeseen situations, and learn from interactions in a way that hardcoded workflows cannot.
- Agents can dynamically select tools and paths rather than strictly following a predefined graph.

Agent Architect Cohort 1

Basic Concepts

Questions?

See You Tomorrow!

Day 2 Focus: Agent Architecting.

- Business Requirements & AI Agents
- Core Components of an AI Agent (Tools, Functions, Extensions etc.)
- Deeper into Different Orchestrations
- Agent Communication
- Safe & Responsible AI
- Model Fine Tuning
- Secure Deployment of AI Agents
- Improving an AI Agent