

# Vizuara AI Agents Bootcamp Day 3

How Agents Actually Work and the ReAct framework



VIZUARA AI  
JUN 25, 2025



5

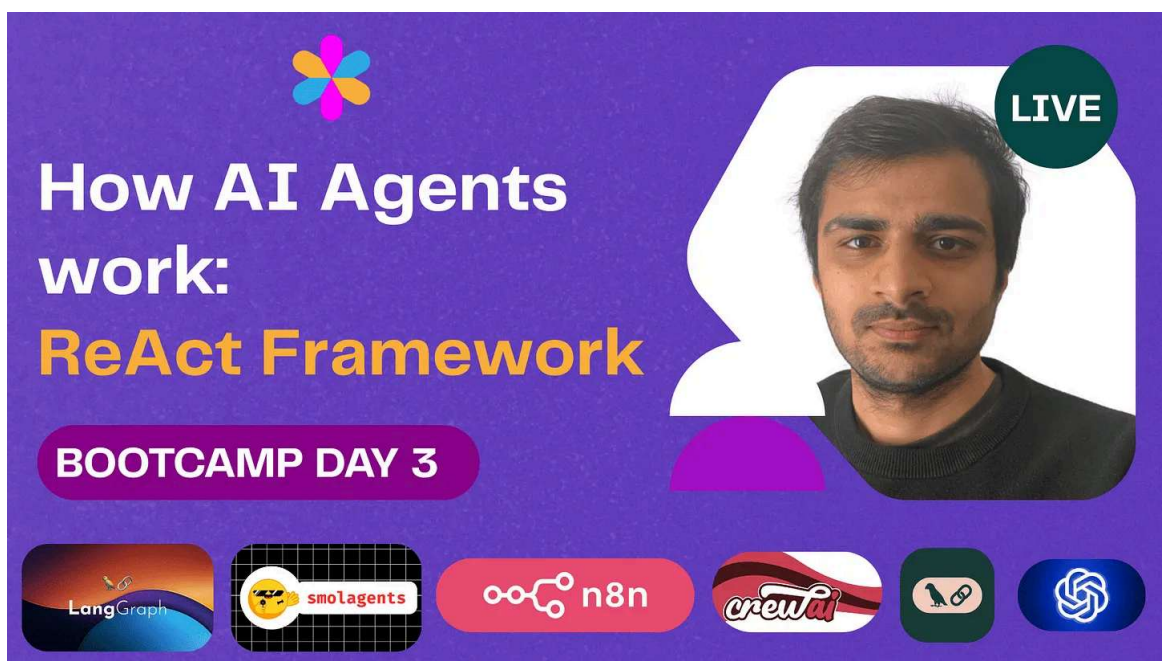


Share

Day 3 session was all about **how AI agents work under the hood** – equipping language models with tools, guiding them through reasoning+action loops, and giving them memory.

We took a deep dive into what transforms a plain LLM into a smart, autonomous agent.

In this blog post (in Substack style), we'll break down the key concepts from Day 3 in a conversational way. Get ready for analogies, intuitive examples, and even some diagrams to illustrate the ideas clearly. Let's dive in!

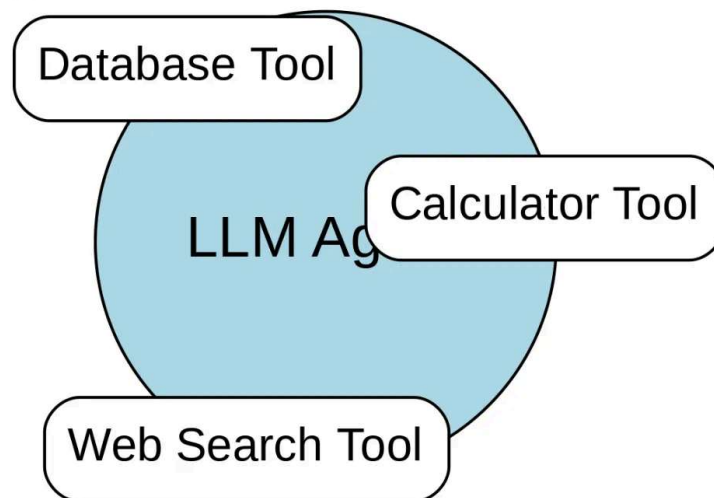
A live stream banner for the Vizuara AI Agents Bootcamp Day 3 session. The banner has a purple background. At the top center is the Vizuara AI logo. Below it, the text 'How AI Agents work: ReAct Framework' is displayed in white and yellow. To the right is a circular profile picture of a man with a beard, with a green 'LIVE' badge in the top right corner. Below the text, a purple button reads 'BOOTCAMP DAY 3'. At the bottom, there is a row of logos for various AI tools and frameworks: LangGraph, smolagents, n8n, crewAI, and OpenAI.

# Table of contents

1. *Tools: The Agent's Arsenal of "Weapons"*
2. *Prompt Engineering: Opening the Toolbox for the LLM*
3. *From Chain-of-Thought to ReAct: Reason + Action*
4. *The TAO Loop: Thought → Action → Observation (Repeat)*
5. *Don't Forget: Memory in Agent Workflows*
6. *Wrapping Up Day 3*

## (1) Tools: The Agent's Arsenal of "Weapons"

### Agent with Arsenal of Tools



*Illustration: An LLM agent equipped with an arsenal of tools (e.g. a calculator, web search, database access). Tools extend what the LLM can do on its own.*

In the world of AI agents, **tools are like weapons** that extend an agent's capabilities. A large language model (LLM) on its own can only **read and generate text** – it has no direct access to the internet, calculators, databases, or other external systems.

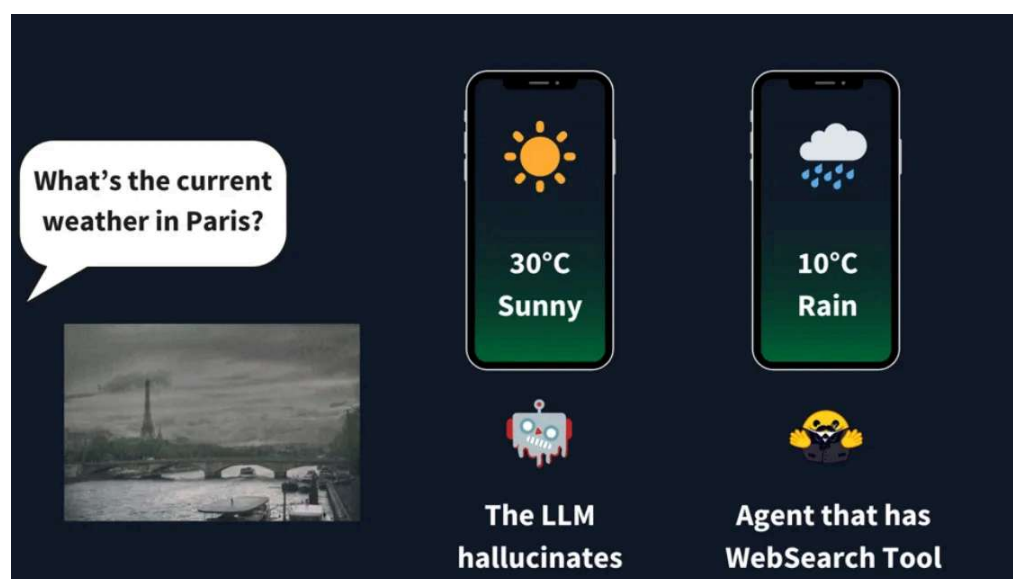
Tools give the LLM agent the power to **act on the world** or fetch information beyor its built-in knowledge.

“Tools are like weapons, which allow the assistant to do additional tasks”.

An AI agent can have tools like a **web search** (for up-to-date info), a **calculator** (for precise math), or an **API interface** (to query databases or services). These tools dramatically **expand what the agent can accomplish**. Instead of being limited to w the model “knows” (or guesses) from training data, the agent can **fetch real data and take actions**.

A good rule of thumb is that a tool should **complement the LLM’s strengths and co its weaknesses**. For example, LLMs are not great at long arithmetic or exact calculations, so giving an agent a calculator tool will yield more accurate results for math than relying on the LLM’s own guess.

Likewise, LLMs can’t know events that happened after their training cutoff. If you : a vanilla LLM, “What’s the weather in Paris right now?”, it will likely **hallucinate** an answer based on older data or pure guesswork. An agent equipped with a **live weatl API or web search tool**, however, can look up the actual current weather and give y a truthful answer.



*Asking “What’s the current weather in Paris?” to a plain LLM vs an agent with a web search tool. The LLM (left) guesses a sunny 30°C (hallucination), while the tool-augmented agent (right) correctly finds that it’s 10°C and raining.*

As shown above, tools help **ground the agent in reality**. The plain LLM confidently answered with a sunny forecast (completely made up!), whereas the tool-using agent performed a live search and discovered it’s actually rainy and 10°C in Paris. This highlights why tools are so important: they let the agent **retrieve up-to-date, factual information**, reducing the nonsense that can come from the model’s imagination.

In summary, **tools turn an LLM into an agent** by giving it the ability to *do things*: call APIs, run computations, fetch knowledge, and more. An LLM agent armed with the right tools is like a superhero with the right weapons or gadgets – far more effective at tackling challenges than an unarmed model.

## **(2) Prompt Engineering: Opening the Toolbox for the LLM**

Okay, so tools are great – but **how does the LLM actually use a tool?** After all, the model can’t physically tap a calculator or click a web browser on its own.

The secret is in the **prompting**. We *pass tools to the LLM through the prompt*, essentially telling the model which tools exist and how to invoke them. Prompt engineering becomes the gateway that connects the LLM’s mind to its toolbelt.

Remember: LLMs *only take text input and produce text output*. They have no built-in button to call a function. So, what we do is **describe the available tools in the system prompt or context** and instruct the model on the format to use them.

```
system_message="""You are an AI assistant designed to help users efficiently and accurately. Your primary goal is to provide helpful, precise, and clear responses.

You have access to the following tools:
{tools_description}
"""
```

For example, we might include in the prompt: “*You have access to a Calculator tool. To use it, output `Calculator(input)` with your query.*” By providing a clear description syntax, the model can “decide” when to use the calculator by emitting the text `Calculator(2+2)` as part of its response. The agent’s runtime system will detect this and actually execute the tool, then return the result to the model.

In essence, we **teach the model** how to use tools via the prompt. The system prompt might list each tool’s name, function, and usage format. The model then knows: “If user asks something requiring calculation, I should use the Calculator tool by outputting a call.” The agent framework (the code around the LLM) reads that and does the calculation, then feeds the answer back for the LLM to continue. All these steps are orchestrated by carefully engineered prompts and a loop that alternates between the LLM’s thoughts and tool executions.

To illustrate, imagine the user asks: “*What was the score of yesterday’s game?*” The system prompt has given the agent a `WebSearch` tool. The LLM might output a *thought* like “The user is asking for yesterday’s game score; I should use `WebSearch`.” Then it outputs an *action*: `WebSearch("yesterday's game score")`. The agent intercepts this, runs the web search, gets an observation (say, “Team A 3 – Team B 1”), and feeds it back. The LLM sees the new info and finally outputs: “The score was 3–1 for Team A.” All the user sees is the final answer; behind the scenes the **prompt guided the LLM** to invoke the right tool and incorporate the result.

**Prompt engineering is foundational** for agent behavior.

We have to explicitly funnel knowledge of tools into the model through text. This often means giving very precise, structured prompts (sometimes using formats like JSON or function definitions) so the LLM understands how to call each tool.

*(By the way, frameworks like LangChain or the new OpenAI function-calling API automate much of this prompt engineering. They prepare a system message listing the tools, enforce the format of the LLM's response, and parse out tool calls. But under the hood it's still the same principle: the LLM is **conditioned via prompts** to use tools.)*

### (3) From Chain-of-Thought to ReAct: Reason + Action

Now we move to one of the most important frameworks discussed today: **ReAct**, which stands for “**Reasoning and Acting**.” This approach is all about combining the LLM's reasoning ability with tool use in a single, coherent loop. To understand why ReAct is needed, let's first recall **Chain-of-Thought (CoT)** prompting.

#### Standard Prompting

##### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

##### Model Output

A: The answer is 27. ❌

#### Chain-of-Thought Prompting

##### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

##### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅



Chain-of-Thought prompting is when we encourage the LLM to **think step-by-step** like showing its work for a math problem. If you prompt ChatGPT with “*Let’s think through step by step,*” it will start writing out intermediate reasoning steps before giving the final answer.

CoT is great for **making the model’s reasoning explicit and logical**. It can dramatically improve accuracy on complex tasks by breaking them down. However, **vanilla CoT has a limitation**: those thought steps are purely internal to the model. The model isn’t interacting with the outside world; it’s just reasoning on its own knowledge.

If a needed fact isn’t already in the model’s brain, it might just **make something up** – in other words, **CoT alone lacks the ability to take *actions*** – it’s reasoning in a vacuum.

This is where **ReAct** comes in. The ReAct framework (introduced by Yao et al. in 2022) augments chain-of-thought reasoning with the ability to **act on the environment**.

## ReAct: Synergizing Reasoning and Acting in Language Models

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao

While large language models (LLMs) have demonstrated impressive capabilities across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with external sources, such as knowledge bases or environments, to gather additional information. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate effectiveness over state-of-the-art baselines, as well as improved human interpretability and trustworthiness over methods without reasoning or acting components. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes issues of hallucination and error propagation prevalent in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generates human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. On two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples. Project site with code: [this https URL](https://github.com/yaozhu1/ReAct)

Instead of just thinking through a solution, the agent can **think, then act (use a tool), then observe the result, then think some more, and so on**.

An agent following ReAct will produce a series of *Thought* → *Action* → *Observation* (repeat), concluding with an answer. This means the agent can use tools mid-thought to fetch information, then use that info to refine its next thought, and so forth.

Let's contrast to make it clearer. Imagine a question: *"Who won the first Oscar that Russell Crowe received, and who directed that film?"* A pure CoT approach would have the model try to recall Russell Crowe's awards and films, step by step, from its training data. It might recall *"Gladiator"* but if it's unsure, it could hallucinate a director or mix things up. In fact, research noted that *"the reason-only baseline (CoT) suffers from misinformation... as it is not grounded to external knowledge"*.

Now consider a ReAct approach: The agent might **think** "I need to find Russell Crowe's first Oscar-winning film." Then **act** by calling a search tool for "Russell Crowe first Oscar." The **observation** might return: "Russell Crowe won his first Oscar for *Gladiator* (2000)." The agent then **thinks** "The question also asks who directed *Gladiator*; I should find that." **Action**: call the wiki tool on "Gladiator film director". **Observation**: "*Gladiator* was directed by Ridley Scott." Now the agent has the pieces and can **answer**: "Russell Crowe's first Oscar was for *Gladiator*, which was directed by Ridley Scott." In this process, the agent **alternated between reasoning and tool use** with each tool result informing the next reasoning step. It didn't have to know everything upfront – it **figured out what it needed and fetched it**.

The ReAct framework formalizes this pattern. It uses a prompt structure that encourages the model to produce a **verbal chain-of-thought and an action plan interwoven**. Typically, the prompt might look like:

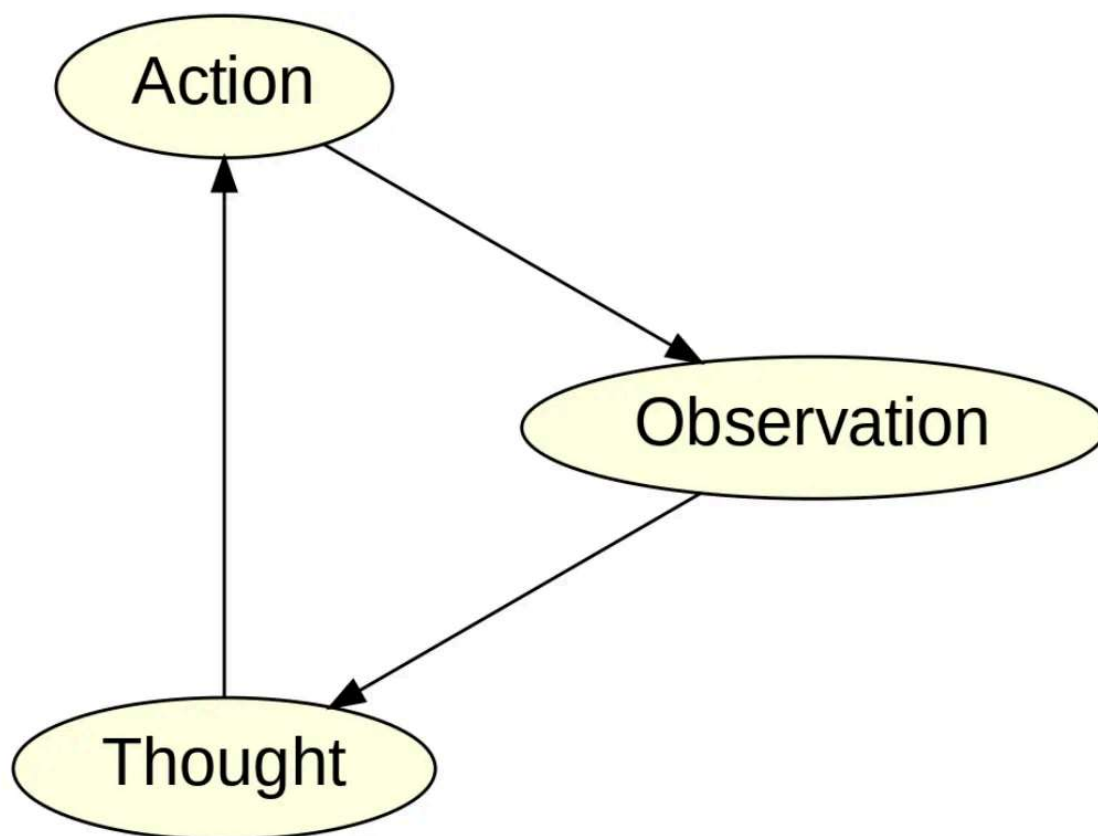
- **Thought**: "...reason about problem..."
- **Action**: "...choose a tool and input..."
- **Observation**: "...result of the action..."
- (Loop back to Thought, and repeat as needed)
- **Final Answer**: "..."

To sum up, **ReAct = CoT + Tool Use**. It was a major step in evolving AI agents from just being clever chatbots to being actual problem-solvers that can **think and do**. The industry quickly embraced this: many agent frameworks (like LangChain agents) are



essentially implementations of the ReAct pattern. Now let's zoom in on the core of ReAct – the TAO loop.

## (4) The TAO Loop: Thought → Action → Observation (Repeat)



## Thought-Action-Observation Loop

The **Thought-Action-Observation (TAO) loop** is the heartbeat of the ReAct agent. It's the cycle that lets an agent iteratively approach a solution. Here's the breakdown of each phase:

- **Thought:** The agent's LLM “thinks” about what to do next. This is a reasoning step, like planning or analyzing the problem state. For example: *“Hmm, to answer this question I might need data X; perhaps I should use tool Y to get it.”*

- **Action:** Based on that thought, the agent takes an action by invoking a tool. E.g. *Action:* call the search tool with query “data X”.
- **Observation:** The agent then gets the result of that action – new information from the tool. E.g., the search results come back with a relevant snippet. The agent *observes* this and incorporates it.

At that point, one **TAO cycle** is complete, but the task may not be finished. The crucial part is the loop can iterate: the agent goes back to **Thought** with the new information in mind, then possibly takes another **Action**, gets another **Observation**, and so on. This repeats until the agent’s thought process determines it has enough to conclude with a **Final Answer**.

This loop is powerful. It means the agent can **adjust on the fly**. If one tool’s observation brings up an unexpected clue or a new requirement, the next Thought can pivot accordingly. Just like how humans might try one approach, see the outcome, then re-evaluate, the TAO loop lets the AI do the same. It’s essentially a feedback loop for problem solving. As IBM’s guide describes, the agent “*iteratively repeats this interleaved thought-action-observation process*”, using each observation to inform the next step.

A concrete example: Let’s say our agent is troubleshooting why a website is down. It might start with Thought: “Maybe the server is unreachable. I should ping the server.” Action: use a Ping tool. Observation: ping failed, host not found. Next Thought: “The domain might be wrong or DNS failed. I should verify the URL.” Action: use DNS lookup tool... and so forth. The agent continues looping until it converges on a solution or an answer to report.

One thing to highlight is that the **TAO loop requires the model to remember what happened in previous steps** – which brings us to the importance of memory, next.

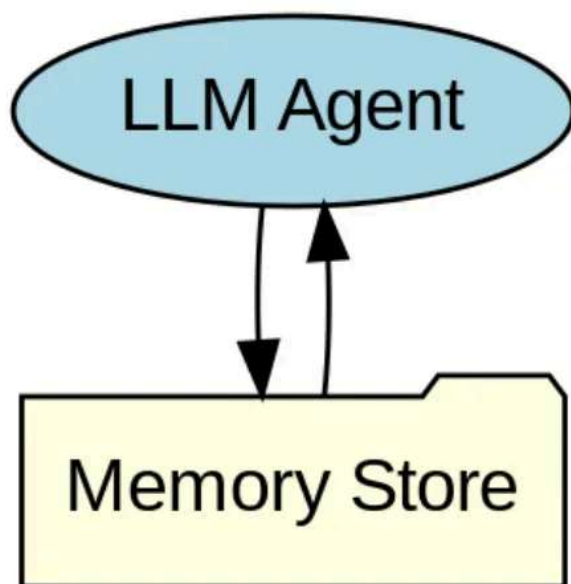
## **(5) Don’t Forget: Memory in Agent Workflows**

We've given our agent the ability to think and use tools – one more piece of the puzzle remains: **memory**. Human intelligence relies heavily on memory, and so do AI agents. Today we learned that equipping agents with memory, both short-term and long-term, is crucial for truly autonomous behavior.

Why does memory matter? Imagine an agent that *forgets* everything after each Thought-Action cycle – it would be like the movie *Memento*, where the protagonist has no short-term memory.

The agent would not learn from tool outputs or remember what it was doing a minute ago! In fact, without memory, the model would have to start fresh at each step, unaware of what it has already done or learned.

## Agent with Memory (stores past information)



*An agent with a memory store. The agent can write information to its memory (down arrow) and retrieve information from memory later (up arrow). This allows it to accumulate knowledge over time.*

## (6) Wrapping Up Day 3

---

## How AI Agents really work: The ReAct Framework



A huge thank you to all the attendees of Day 3! 🎉 We covered a lot of ground, and all asked fantastic questions. By now, you should have a solid mental model of **how LLM agent actually works under the hood**: it thinks, it uses tools, it observes results, it remembers, and it repeats this cycle until it achieves its goal. This combination of reasoning + acting is precisely what turns static AI models into dynamic problem-solving agents.

Coming up next: Lecture 4 will be all about building agents using popular frameworks.

### Sources and Further Reading:

- Yao et al., “*ReAct: Synergizing Reasoning and Acting in Language Models*”, 2023
  - IBM AI Agents Guide on ReAct – (excellent explanation of the TAO loop)
  - Hugging Face *Agents* Course – *What are Tools?*
-

## Subscribe to Vizudara's Substack

Launched 7 months ago

My personal Substack

akshayredekar4441@gmail

Subscribe

By subscribing, I agree to Substack's [Terms of Use](#), and acknowledge its [Information Collection Notice](#) and [Privacy Policy](#).



5 Likes

### Discussion about this post

Comments

Restacks



Write a comment...