What is a ReAct agent?

ReAct agent overview

A ReAct agent is an AI agent that uses the "reasoning and acting" (ReAct) framework to combine chain of thought (CoT) reasoning with external tool use. The ReAct framework enhances the ability of a large language model (LLM) to handle complex tasks and decision-making in agentic workflows.

First introduced by Yao and others in the 2023 paper, "ReACT: Synergizing Reasoning and Acting in Language Models," ReAct can be understood most generally as a machine learning (ML) paradigm to integrate the reasoning and action-taking capabilities of LLMs.

More specifically, ReAct is a conceptual framework for building AI agents that can interact with their environment in a structured but adaptable way, by using an LLM as the agent's "brain" to coordinate anything from simple retrieval augmented generation (RAG) to intricate multiagent workflows.

Unlike traditional artificial intelligence (AI) systems, ReAct agents don't separate decision-making from task execution. Therefore, the development of the ReAct paradigm was an important step in the evolution of generative AI (gen AI) beyond mere conversational chatbots and toward complex problem-solving.

ReAct agents and derivative approaches continue to power AI applications that can autonomously plan, execute and adapt to unforeseen circumstances.

How do ReAct agents work?

The ReAct framework is inspired by the way humans can intuitively use natural language—often through our own inner monologue—in the step-by-step planning and execution of complex tasks.

Rather than implementing rule-based or otherwise predefined workflows, ReAct agents rely on their LLM's reasoning capabilities to dynamically adjust their approach based on new information or the results of previous steps.

Imagine packing for a brief trip. You might start by identifying key considerations ("*What will the weather be like while I'm there?*"), then actively consult external sources ("*I'll check the local weather forecast*").

By using that new information ("*It's going to be cold*"), you determine your next consideration ("*What warm clothes do I have?*") and action ("*I'll check my closet*"). Upon taking that action, you might encounter an unexpected obstacle ("*All of my warm clothes are in storage*") and adjust your next step accordingly ("*What clothes can I layer together?*").

In a similar fashion, the ReAct framework uses prompt engineering to structure an AI agent's activity in a formal pattern of alternating thoughts, actions and observations:

- The verbalized CoT reasoning steps (*thoughts*) help the model decompose the larger task into more manageable subtasks.
- Predefined *actions* enable the model to use tools, make application programming interface (API) calls and gather more information from external sources (such as search engines) or knowledge bases (such as an internal docstore).
- After taking an action, the model then reevaluates its progress and uses that *observation* to either deliver a final answer or inform the next *thought*. The observation might ideally also consider prior information, whether from earlier in the model's standard context window or from an external memory component.

Because the performance of a ReAct agent depends heavily on the ability of its central LLM to "verbally" think its way through complex tasks, ReAct agents benefit greatly from highly capable models with advanced reasoning and instruction-following ability.

To minimize cost and latency, a multiagent ReAct framework might rely primarily on a larger, more performant model to serve as the central agent whose reasoning process or actions might involve delegating subtasks to more agents built using smaller, more efficient models.

ReAct agent loops

This framework inherently creates a feedback loop in which the model problem-solves by iteratively repeating this interleaved *thought-action-observation* process.

Each time this loop is completed—that is, each time the agent has taken an action and made an observation based on the results of that action—the agent must then decide whether to repeat or end the loop.

When and how to end the reasoning loop is an important consideration in the design of a ReAct agent. Establishing a maximum number of loop iterations is a simple way to limit latency, costs and token usage, and avoid the possibility of an endless loop.

Conversely, the loop can be set to end when some specific condition is met, such as when the model has identified a potential final answer that exceeds a certain confidence threshold.

To implement this kind of reasoning and acting loop, ReAct agents typically use some variant of *ReAct prompting*, whether in the system prompt provided to the LLM or in the context of the user query itself.

ReAct prompting

ReAct prompting is a specific prompting technique designed to guide an LLM to follow the ReAct paradigm of *thought*, *action* and *observation* loops. While the explicit use of conventional ReAct prompting methods is not strictly necessary to build a ReAct agent, most ReAct-based agents implement or at least take direct inspiration from it.

First outlined in the original ReAct paper, ReAct prompting's primary function is to instruct an LLM to follow the ReAct loop and establish which tools can be used—that is, which actions can be taken—when handling user queries.

Whether through explicit instructions or the inclusion of few-shot examples, ReAct prompting should:

- **Guide the model to use chain of thought reasoning:** Prompt the model to reason its way through tasks by thinking step by step, interleaving thoughts with actions.
- **Define actions:** Establish the specific actions available to the model. An action might entail the generation of a specific type of next thought or subprompt but usually involves using external tools or making APIs.
- **Instruct the model to make observations:** Prompt the model to reassess its context after each action step and use that updated context to inform the next reasoning step.
- **Loop:** Instruct the model to repeat the previous steps if necessary. You could provide specific conditions for ending that loop, such as a maximum number of loops, or instruct the agent to end its reasoning process whenever it feels it has arrived at the correct final output.
- **Output final answer:** Whenever those end conditions have been met, provide the user with the final output in response to their initial query. As with many uses of LLMs, as reasoning models employing chain of thought reasoning before determining a final output, ReAct agents are often prompted to conduct their reasoning process within a "scratchpad."

A classic demonstration of ReAct prompting is the system prompt for the prebuiltZERO_SHOT_REACT-DESCRIPTION ReAct agent module in Langchain's LangGraph. It's called "zero-shot" because, with this predefined system prompt, the LLM being used with the module does not need any further examples to behave as a ReAct agent.

```
Answer the following questions as best you can. You have access to the following tools:

Wikipedia: A wrapper around Wikipedia. Useful for when you need to answer general questions about people, places, companies, facts, histo
duckduckgo_search: A wrapper around DuckDuckGo Search. Useful for when you need to answer questions about current events. Input should be
Calculator: Useful for when you need to answer questions about math.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [Wikipedia, duckduckgo_search, Calculator]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}
```

Benefits of ReAct agents

The introduction of the ReAct framework was an important step in the advancement of LLM-driven agentic workflows. From grounding LLMs in real time, real-world external information through (RAG) to contributing to subsequent breakthroughs—such as Reflexion, which led to modern reasoning models—ReAct has helped catalyze the use of LLMs for tasks well beyond text generation.

The utility of ReAct agents is drawn largely from some of the inherent qualities of the ReAct framework:

- **Versatility:** ReAct agents can be configured to work with a wide variety of external tools and APIs. Though fine-tuning relevant ReAct prompts (using relevant tools) can improve performance, no prior configuration of the model is required to execute tool calls.
- **Adaptability:** This versatility, along with the dynamic and situational nature of how they determine the appropriate tool or API to call, means that ReAct agents can use their reasoning process to adapt to new challenges. Especially when operating within a lengthy context window or augmented with external memory, they can learn from past mistakes and successes to tackle unforeseen obstacles and situations. This makes ReAct agents flexible and resilient.

- **Explainability:** The verbalized reasoning process of a ReAct agent is simple to follow, which facilitates debugging and helps make them relatively user-friendly to build and optimize.

- **Accuracy:** As the original ReAct paper asserts, chain of thought (CoT) reasoning alone has many benefits for LLMs, but also runs an increased risk of hallucination. ReAct's combination of CoT with a connection external to information sources significantly reduces hallucinations, making ReAct agents more accurate and trustworthy.

ReAct agents vs. function calling

Another prominent paradigm for agentic AI is function calling, originally introduced by OpenAI in June 2023 to supplement the agentic abilities of its GPT models.

The function calling paradigm entails fine-tuning models to recognize when a particular situation should result in a tool call and output a structured JSON object containing the arguments necessary to call those functions.

Many proprietary and open source LLM families, including IBM® Granite®, Meta's Llama series, Anthropic's Claude and Google Gemini, now support function calling.

Whether ReAct or function calling is "better" will generally depend on the nature of your specific use case. In scenarios involving relatively straightforward (or at least predictable) tasks, function calling can execute faster, save tokens, and be simpler to implement than a ReAct agent.

In such circumstances, the number of tokens that would be spent on a ReAct agent's iterative loop of CoT reasoning might be seen as inefficient.

The inherent tradeoff is a relative lack of ability to customize how and when the model chooses which tool to use. Likewise, when an agent handles tasks that call for complex reasoning, or scenarios that are dynamic or unpredictable, the rigidity of function calling might limit the agent's adaptability. In such situations, it's often beneficial to be able to view the step-by-step reasoning that led to a specific tool call.

Getting started with ReAct agents

ReAct agents can be designed and implemented in multiple ways, whether coded from scratch in Python or developed with the help of open source frameworks such as BeeAI. The popularity and staying power of the ReAct paradigm have yielded extensive literature and tutorials for ReAct agents on GitHub and other developer communities.

As an alternative to developing custom ReAct agents, many agentic AI frameworks, including BeeAI, LlamaIndex and LangChain's LangGraph, offer preconfigured ReAct agent modules.