Agent Planning and Reasoning Patterns

Concrete patterns for an agent's "decide" step: ReAct, plan-and-execute, reflection, and when a loop should actually stop.

Intermediate 20 min

Last updated: August 22, 2026
Read online: https://www.learnforgex.com/en/topics/agent-planning-and-reasoning

---

Agent Planning and Reasoning Patterns

"What Is an AI Agent?" described the agent loop as observe, decide, act -- but left the "decide" step at "the model decides." That step is where most of the actual difference between agent designs lives. This lesson covers three concrete patterns for how a model turns "here's the goal and what's happened so far" into "here's the next action": **ReAct**, **plan-and-execute**, and **reflection** -- along with the question every one of them eventually has to answer: when does the loop stop?

The ReAct Pattern: Reasoning and Acting

ReAct (short for "Reason + Act") interleaves a visible reasoning step with each action: before choosing a tool call, the model first produces a short piece of text explaining its reasoning -- what it currently knows, what it still needs, and why the next action makes sense -- and only then emits the action itself. The next observation (the tool's real result) is fed back in, and the cycle repeats: reason, act, observe, reason, act, observe. The advantage is directness -- each decision is made with the freshest possible information, one step at a time, and the visible reasoning text makes it far easier to see *why* the agent chose a particular action instead of only seeing which action it chose. The cost is that the model is replanning, in a sense, at every single step, which can be less efficient than committing to a broader plan up front.

Plan-and-Execute: Planning Up Front

Plan-and-execute splits the loop into two distinct phases instead of interleaving them. First, given the goal, the model produces a multi-step plan up front -- "look up X, then compute Y using X, then combine both into an answer" -- before taking any action at all. Then, a (often much simpler) execution step works through that plan one step at a time, calling tools as needed. If a step's real result invalidates the rest of the plan, the plan can be

revised, but the default is to follow it through. This trades some of ReAct's step-by-step adaptiveness for a clearer, more inspectable picture of the whole intended sequence before anything actually runs -- useful when a goal decomposes cleanly into a known set of sub-tasks, and when seeing the full plan before execution matters (for a human review step, for instance -- see "Human-in-the-Loop: Approval Before Risky Actions" in "Controlling Agent Behavior").

Reflection: Checking and Revising Your Own Work

Reflection adds a distinct step after an action (or a whole attempt at the goal) completes: the model is asked to critique its own output before treating it as final -- does this actually answer the goal? does this result look right, given what was asked? -- and, if the critique finds a problem, to revise and try again rather than stopping. This directly targets the failure mode "Hallucination: Confident, Fluent, Wrong" (from "LLM Capabilities and Limitations") described at the level of a single response: a model producing a fluent, confident answer that happens to be wrong. Reflection doesn't eliminate that risk -- the same model doing the checking has the same limitations as the model that produced the answer -- but a dedicated critique step, treated as its own decision in the loop rather than skipped, catches some classes of mistake that would otherwise go straight through unchecked.

Choosing Among These Patterns

These patterns aren't mutually exclusive, and most real agents combine pieces of more than one. As a starting point: ReAct suits goals where the right next step genuinely depends on what the previous step returned, and the full sequence can't be known up front. Plan-and-execute suits goals that decompose cleanly into a known set of sub-tasks, especially when showing the plan before running it adds value. Reflection is less an alternative to the other two and more an addition to either of them -- a check applied before a result is treated as the final answer, regardless of how the steps leading up to it were decided.

Termination: Knowing When to Stop

Every one of these patterns eventually has to answer the same question: when is the loop actually done? Three conditions typically end it: the model itself decides the goal has been satisfied and produces a final answer (the normal case); an external limit is reached, such as a maximum number of steps, before the model reaches that conclusion on its own; or an unrecoverable error occurs (a tool consistently failing, for example) that no further looping can fix. The first case is the goal; the second is a safety net for when it doesn't happen -- a decision function that never concludes would otherwise loop forever, consuming time and cost with nothing to show for it. "Controlling Agent Behavior", next in this category, covers exactly how that safety net -- a step-limit guardrail -- gets built.

Best Practices

- Match the pattern to the goal's shape -- see "Choosing Among These Patterns" -- rather than defaulting to whichever pattern was used last.
- Keep a visible record of each decision (ReAct's reasoning text, or a plan-and-execute plan) rather than only the final answer -- it's what makes a wrong final answer debuggable after the fact.
- Add a reflection step specifically for outputs where "confident but wrong" is costly -- see "Reflection: Checking and Revising Your Own Work" -- rather than applying it uniformly regardless of stakes.

Common Mistakes

- **Treating ReAct, plan-and-execute, and reflection as mutually exclusive choices.** As "Choosing Among These Patterns" explained, most practical agents combine them -- reflection in particular is usually an addition, not an alternative.
- **Building a loop with no termination condition beyond "the model decides it's done."** "Termination: Knowing When to Stop" covered why a step-limit guardrail matters as a backstop, not as a substitute for the model's own stopping decision.
- **Assuming a reflection step removes the risk of a wrong answer entirely.** As "Reflection: Checking and Revising Your Own Work" noted, the same underlying model limitations from "Hallucination: Confident, Fluent, Wrong" still apply to the step doing the checking.

Summary, Cheat Sheet, and Glossary

Summary

- The agent loop's "decide" step from "What Is an AI Agent?" isn't one fixed mechanism -- ReAct, plan-and-execute, and reflection are three concrete patterns for how it can work.
- ReAct interleaves a visible reasoning step with each action, replanning at every step based on the freshest observation.
- Plan-and-execute separates planning (a full multi-step plan up front) from execution (working through that plan, revising only if needed).
- Reflection adds a self-critique step before treating an output as final, aimed at the same "confident but wrong" failure mode "LLM Capabilities and Limitations" described.
- These patterns combine in practice rather than being exclusive choices, and all of them eventually need a termination condition -- covered in depth next, in "Controlling Agent Behavior".

Cheat Sheet

- ReAct = reason -> act -> observe -> repeat, replanning every step.
- Plan-and-execute = plan fully up front -> execute steps -> revise only if a result invalidates the plan.
- Reflection = critique an output before finalizing it; catches some "confident but wrong" mistakes, doesn't eliminate the risk.
- Termination = model concludes goal met (normal case), OR a step limit is hit (safety net), OR an unrecoverable error occurs.

Glossary

- **ReAct:** an agent pattern interleaving a visible reasoning step with each action, deciding the next step fresh after every observation.
- **Plan-and-execute:** an agent pattern that produces a full multi-step plan before taking any action, then executes it step by step.
- **Reflection:** a self-critique step applied to an agent's output before treating it as final, aimed at catching confident-but-wrong mistakes.
- **Termination condition:** whatever causes an agent loop to stop -- the model concluding the goal is met, a step-limit guardrail, or an unrecoverable error.