



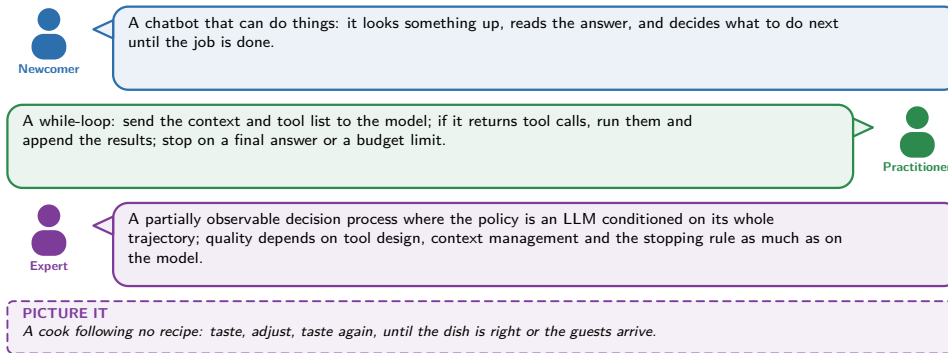
What Is an AI Agent?

Agents 01 | The agent loop, ReAct, workflows vs agents, autonomy levels, stop conditions, the cost of a loop, when not to build an agent

#Agents #AgentLoop #ReAct #Autonomy

The idea

AI agent



One definition, three listeners. All three describe the same loop

Before Agent fundamentals, can you explain...

Not yet Partly Yes

MUST KNOW	Chat messages: system, user, assistant roles			
Learn: llm/prompt engineering				
MUST KNOW	Tokens, context window, input vs output price			
Learn: llm/tokenisation				
MUST KNOW	Reading JSON and a Python while-loop			
Learn: programming				
HELPFUL	Reasoning models and thinking tokens			
Learn: llm/reasoning models				

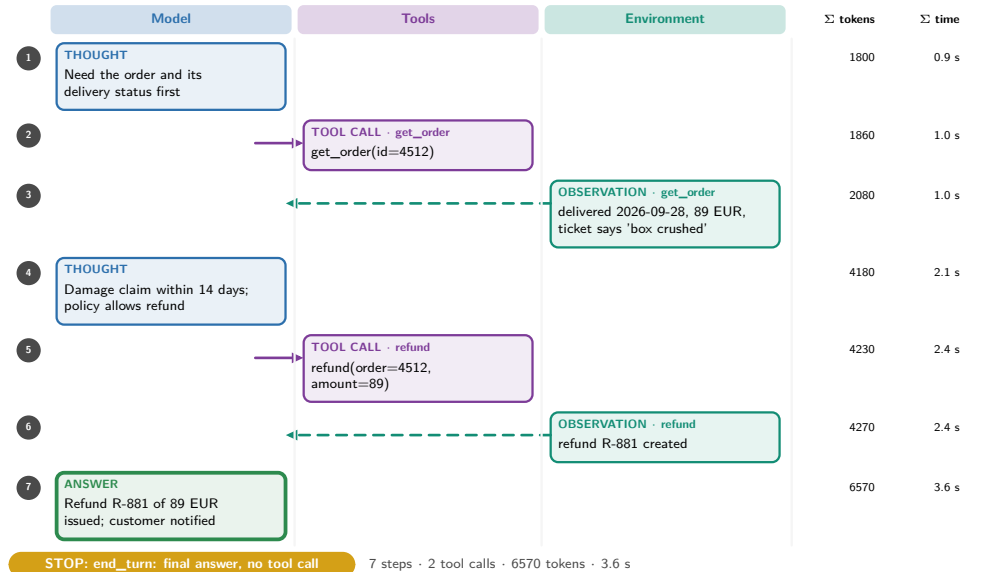
Tick honestly before reading on

Agent (working definition) — A system where the LLM dynamically directs its own process and tool use, deciding the next step and when to stop (Anthropic, 'Building effective agents', Dec. 2024).

Agentic — A matter of degree, not a yes/no label: the more steps the model chooses, the more agentic the system.

The agent loop

Goal: Refund order 4512 if it arrived damaged



One run of a support agent: model decisions, tool calls and environment results as lanes, with cumulative tokens and latency. Most time is spent in model calls; most tokens are re-read history

The whole loop in Python (Anthropic Messages API; other SDKs differ in field names only)

```
msgs = [{"role": "user", "content": task}]
for step in range(MAX_STEPS):
    r = client.messages.create(model=MODEL, max_tokens=2048,
                              system=SYSTEM, tools=TOOLS, messages=msgs)
    msgs.append({"role": "assistant", "content": r.content})
    if r.stop_reason != "tool_use":
        break
    results = []
    for b in r.content:
        if b.type == "tool_use":
            out = run_tool(b.name, b.input) # YOUR code
            results.append({"type": "tool_result",
                           "tool_use_id": b.id, "content": out})
    msgs.append({"role": "user", "content": results})
```

- 1 **Final answer** the model ends its turn without a tool call (stop reason end_turn or equivalent), or calls an explicit finish/submit tool
- 2 **Step budget** max iterations, typically 10-50 for assistants and hundreds for coding agents
- 3 **Token or money budget** cumulative input + output tokens tracked per run
- 4 **Wall-clock timeout** a user is waiting, or a job slot is reserved
- 5 **Repeated failure** same tool error N times, or the same call repeated: a loop that makes no progress
- 6 **Human interrupt or approval denied** the user stops it, or refuses an action that the plan depended on

Stop conditions: a production loop has all of them, not only the first

Why long loops get expensive

PROBLEM

A 10-step run: how many input tokens does the model read in total?

GIVEN

- n = 10 model calls (steps)
- S = 3,000 tokens of system prompt and tool definitions, re-sent every call
- U = 200 tokens: the user task
- a = 600 tokens added to the history per step (tool call + tool result)

1

Input of call i

S + U + a x (i - 1)

Why: each call re-reads everything before it

2

Sum the fixed part

n x (S + U) = 10 x 3,200 = 32,000

Why: prefix paid on every call

3

Sum the growing part

a x n(n-1)/2 = 600 x 45 = 27,000

Why: 0 + 1 + ... + 9 = 45 history increments

4

Total

T_in = 32,000 + 27,000 = 59,000 tokens

Why: grows with n squared

ANSWER

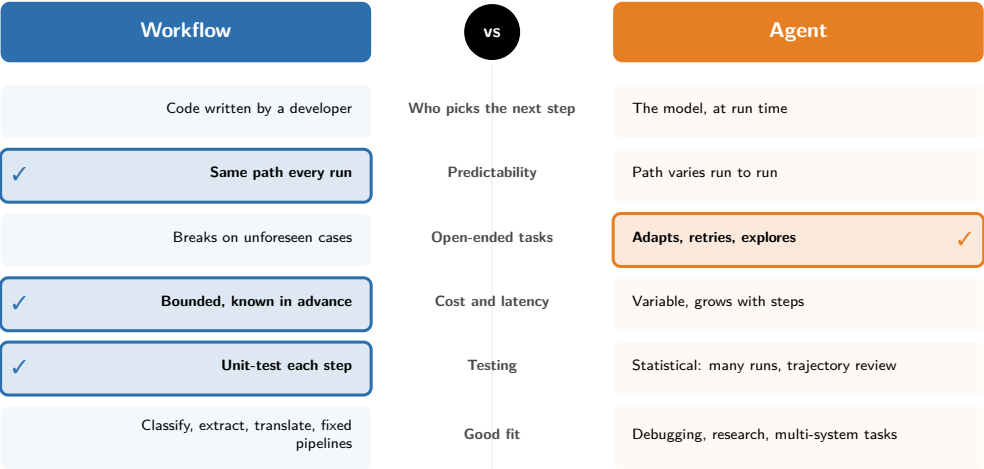
About 59,000 input tokens for a 10-step run; at 30 steps the same formula gives 96,000 + 261,000 = 357,000

CHECK

✔

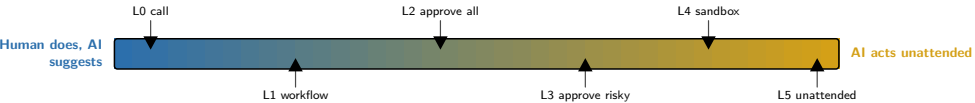
Prompt caching bills the repeated prefix at a fraction of the input price (approx. 10 % for cache reads on Anthropic's API as of 2026); compaction (sheet 04) caps the growing part

- What changed models are now trained for tool calling: the harness no longer parses 'Action:' lines with regular expressions, it reads structured tool calls
- Reasoning models think before (and on some APIs between) tool calls; the ReAct pattern is built into training rather than the prompt



Same task, two architectures. 'Better' marks the usual winner for that criterion

Autonomy levels



A common way to grade autonomy (there is no single standard scale as of 2026). Move right only when evaluation and guardrails justify it

Level	Human role	Typical example (2026)
L0 single call	reads and uses the answer	summarise this e-mail
L1 workflow	designs the path once	classify ticket → retrieve policy → draft reply
L2 approve each action	clicks approve on every tool call	coding agent in its default ask-before-edit mode
L3 approve risky actions	approves writes, payments, deletes; reads run freely	support agent that may refund under 50 EUR alone
L4 sandboxed run	reviews the outcome (a pull request, a report)	cloud coding agent working on a branch in a container
L5 unattended	audits logs after the fact	rare; narrow, reversible, well-evaluated tasks only

⚡ ReAct and its successors

Move	ReAct text format	Native tool-calling API
Reason	Thought: I need the order first	thinking block / text before the call
Act	Action: get_order[4512]	tool_use block with name + JSON input
Observe	Observation: delivered, damaged	tool_result message with the call id
Finish	Action: finish[answer]	turn ends without a tool call

ReAct (Yao et al., 2022) as text, and how the same three moves look with native tool calling in 2026

- Why ReAct mattered interleaving reasoning with actions beat reasoning alone (chain of thought hallucinated facts) and acting alone (no plan) on question answering and web tasks

⚠ When not to build an agent

Conditions	R1	R2	R3	R4	R5
Steps known in advance	Y	N	N	N	--
Success can be checked (tests, rules, a person)	--	Y	Y	N	--
Actions reversible or cheap	--	Y	N	N	--
Value per task covers tokens and review	Y	Y	Y	Y	N
Actions					
Single call or workflow	X				
Agent, runs alone in a sandbox		X			
Agent with approval on risky actions			X		
Agent drafts, human executes				X	
Do not automate with an LLM					X

Pick the least autonomous design that solves the task. Rule R3 is the most common correct answer in business settings

⚠ Compounding errors

If each step succeeds with probability p , a run of n dependent steps succeeds with probability p^n (p = per-step success rate, 0-1; n = number of steps). With $p = 0.95$: 10 steps $\rightarrow 0.60$, 20 steps $\rightarrow 0.36$. Agents survive long tasks only because they observe errors and correct them; a design that cannot observe its own failures should stay a workflow.

MYTH

✗ An agent is a smarter model

FACT

✓ An agent is a loop and tools around a model
The same model can power a single call, a workflow or an agent; the architecture changes, not the weights.

✗ More autonomy means better results

✓ Autonomy trades predictability for flexibility
On a fixed pipeline a workflow is cheaper, faster and easier to test.

✗ The model runs the tools

✓ The harness runs them
The model emits a request; your code validates, executes, or refuses it. That is where permissions live.

✗ ReAct is obsolete

✓ Its reason-act-observe loop is what tool-calling APIs implement
Only the text format with 'Action:' lines was replaced by structured tool calls.

🗣 Interview: 'When would you not use an agent?'

When the steps are known (a workflow is cheaper and testable), when success cannot be verified, or when actions are irreversible and no human can approve them. I start with one well-prompted call, add a workflow, and only give the model control of the path when the task is open-ended and the value pays for extra tokens and review.

✓ Recap and self-test

Agent = LLM directs its own steps
workflow = code directs the steps

Loop: call $\rightarrow$ tool $\rightarrow$ result $\rightarrow$ call
until final answer or a budget stops it

Several stop conditions
answer, steps, tokens, time, repeated failure, human

Cost grows with n squared
history is re-read each call; cache and compact

Autonomy is a dial
approve risky actions before going unattended

p^n
observe and correct errors, or keep runs short

Q1 Give Anthropic's distinction between workflows and agents.

A Workflows: LLMs and tools follow predefined code paths. Agents: the LLM dynamically directs its own process and tool use.

Q2 What are the three moves of ReAct?

A Thought (reason), Action (call a tool), Observation (read its result), repeated until finish.

Q3 Which stop_reason tells the harness to run tools (Anthropic API)?

A tool_use; any other stop reason (end_turn, max_tokens) ends or pauses the loop.

Q4 Why do total input tokens grow with the square of the steps?

A Call i re-reads the prefix plus $i - 1$ history increments; summing $0..n-1$ gives $n(n-1)/2$ increments.

Q5 Name four stop conditions besides the final answer.

A Step budget, token or money budget, timeout, repeated failure (or human interrupt).

Q6 Per-step success 0.9, 10 steps, no correction: chance of full success?

A $0.9^{10} \approx 0.35$, about one run in three.

Q7 What is autonomy level L3 in the scale above?

A The agent acts alone on reads and safe actions; a human approves risky ones (writes, payments, deletes).

Q8 Steps are known in advance: agent or workflow?

A Workflow: cheaper, predictable, unit-testable.

Q9 Where are permissions enforced in an agent?

A In the harness, before executing a tool call; never by asking the model to behave.