

Agents vs. Workflows

How AI agents actually work in 2026 · A deep dive from the AI Field Guide · © 2026 Deven Panchal. All rights reserved.

Agents vs. workflows

An agent is a model that works toward a goal over many steps, using tools, instead of answering once. Agents are where most AI investment is heading, and where most of the new risks come from.

Anatomy of an agent

- **The loop:** the model decides what to do, calls a tool (search, run code, read a file, use an app), looks at the result, and repeats until the job is done or it needs a human.
- **The harness:** the software around the model that makes it an agent: the loop itself, tool definitions, memory and context management, permissions, and logging. Claude Code and OpenAI's Codex are harnesses.
- **Tools and memory:** tools usually connect through MCP; memory is notes stored outside the context window and loaded back in when needed.
- **Cost:** agents use roughly 4 times the tokens of a chat, and multi-agent systems about 15 times ([Anthropic](#)).

The patterns

Anthropic's widely cited framework (December 2024) separates **workflows**, where developers design fixed steps, from **agents**, where the model decides the steps itself. Its advice: start with the simplest pattern that works. Most production systems are workflows.

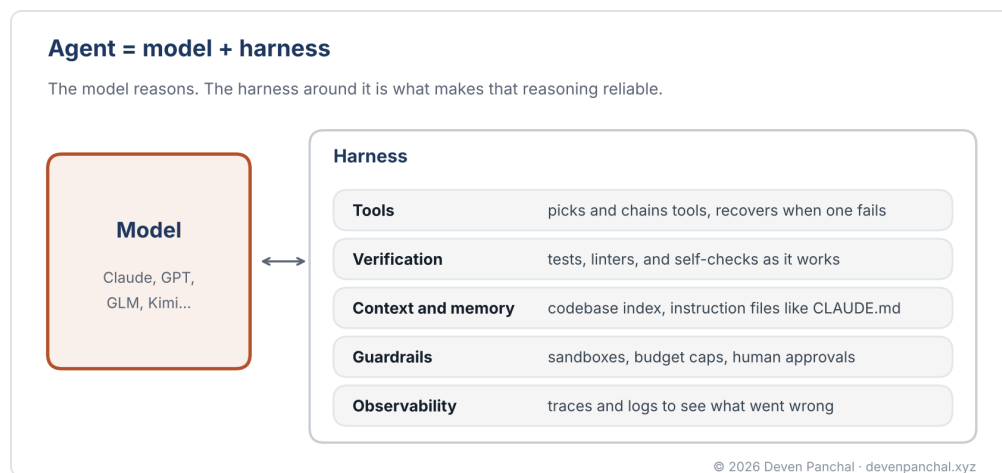
Pattern	How it works	Finance example
Prompt chaining	Fixed steps, each feeding the next, with checks between them	Extract figures from a filing, then check them, then draft the summary
Routing	Classify the request, then send it to the right prompt or model	Simple formatting to a cheap model, reconciliations to a top one
Parallelization	Split work into independent pieces, or run the same task several times and compare	Review 20 contracts at once; have three passes vote on a risky number
Orchestrator-workers	A lead model breaks the job down at runtime and hands pieces to worker models	Due diligence: one worker per data-room folder
Evaluator-optimizer	One model drafts, another critiques, repeat until it passes	Draft a memo, check every number against sources, revise
Autonomous agent	The model plans and acts on its own until done	"Build the comps table and update the model"

Which approach to use

1. **One model call** if the task fits in one answer.
2. **A workflow** if the steps are known in advance (most business processes).
3. **A single agent** if the steps depend on what it finds along the way.
4. **Multiple agents** only for broad, parallel, high-value work where the extra cost pays off.

Harness engineering

If an agent is a model plus a harness, the harness is the part you control. **Harness engineering** is the work of building it well, and it's the big agent idea of 2026: the third phase after prompt engineering (2022 to 2023) and context engineering (2025).



- **The core habit:** Mitchell Hashimoto, co-founder of HashiCorp, put it simply: every time an agent makes a mistake, build a fix into the harness so it can't happen again. A test, a rule in an instruction file, or a new tool beats re-prompting and hoping.
- **It moves results as much as the model does:** in March 2026, LangChain's coding agent jumped from 30th to 5th on Terminal-Bench 2.0 (an earlier version of the benchmark) with the same model, purely by improving its harness. In Faros's internal test of 211 real engineering tasks, open models like GLM-5.2 and Kimi K2.6 in a well-tuned harness matched or beat Opus 4.8 and GPT-5.5 ([Faros](#)).
- **What good harnesses do:** check work as they go (run the tests before declaring success), keep context lean (load the right files, not the whole repo), cap budgets and steps, and log every action so failures can be traced.

Why it matters for business: if the harness drives this much of the result, it's where an agent company's moat lives. The model underneath can be swapped as prices and quality change (routing each task to the best model); the harness, built from thousands of fixed mistakes, can't be copied overnight.

Subagents and multi-agent systems

- **Orchestrator plus subagents:** a lead agent plans, then launches several subagents that work in parallel, each with its own fresh context window, and combines their results. Anthropic's research system (a lead agent running three to five subagents) beat a single agent by about 90% on its internal research tests and cut research time by up to 90%, but used about 15 times the tokens of a chat. It works best for broad, parallel, high-value tasks, and poorly for tightly connected work like most coding.
- **Swarms:** many agents coordinating with little central control, often through shared spaces like message boards. OpenAI's experimental "Swarm" framework (2024) popularized the term. OpenAI's Navier-Stokes math run used about 10,000 agents exchanging nearly 5 million messages.
- **Always-on agents with their own computers:** agents that keep working in the background toward a goal and check in with people. Each one gets its own virtual computer and browser in the cloud, so it keeps going when your laptop is closed and can use sites that have no API by clicking through them like a person. Examples: OpenAI's [Dots](#) (Sept 29, 2026; messaged through Slack, Teams, or ChatGPT) and SpaceXAI's [Grok Bot](#) (beta Aug 11, 2026). Millions of always-on agent computers are a big reason CPU demand is surging.

Subagents: how delegation works

A **subagent** isn't a different kind of model. It's a fresh run of a model with its own context window, its own instructions, and often a narrower set of tools. The main agent hands it a task, it works alone, and only its final report comes back ([Claude Code docs](#)).

Property	What it means	Why it matters
Fresh context	Starts blank and doesn't see the main conversation	It must be briefed fully, like a new contractor
Only the report returns	50,000 tokens of searching and file reading come back as a one-page summary	Keeps the main agent's context clean and focused
Narrow tools	A reviewer can be read-only; a researcher can search but not edit	Limits the damage a mistake or a prompt injection can do
Its own model	Easy subtasks can run on a cheaper, faster model	Cuts cost on high-volume work
Parallel runs	Several subagents can work at once	Turns hours of sequential research into minutes

Good fits: research across many independent sources, searching a large codebase or data room, and checking finished work with fresh eyes, since a subagent that didn't do the work isn't grading itself. **Poor fits:** tightly connected tasks where each step depends on the last (most code edits), work where every agent needs the same shared context, and small jobs where briefing costs more than doing.

Briefing is the skill. Anthropic found that vague hand-offs led subagents to duplicate work or leave gaps. Each one needs an objective, an output format, guidance on which tools and sources to use, and clear boundaries ([Anthropic](#)). Tokens are also the main driver of quality: on Anthropic's browsing benchmark, how many tokens the system used explained about 80% of the difference in performance, which is why subagents help and why they're expensive.

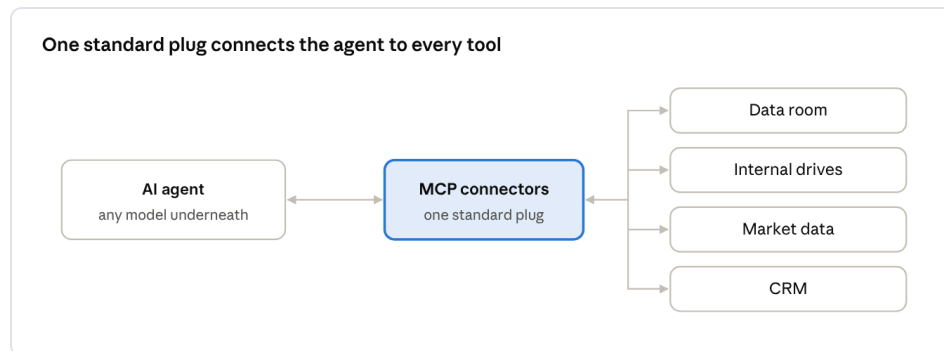
How the tools expose them:

- **Claude Code:** built-in subagents (Explore for fast read-only searching, Plan for research before a plan, and a general-purpose one). Custom subagents are Markdown files in `.claude/agents/` with a name, a description, allowed tools, and a model. Claude delegates automatically when a task matches a description. By default up to 20 run at once, and subagents can spawn their own, up to three layers deep. A **fork** is a variant that inherits the whole conversation instead of starting fresh.
- **OpenAI Agents SDK:** two patterns. **Agents as tools**, where the main agent calls a specialist like a function and gets its output back (the subagent pattern), and **handoffs**, where control passes to another agent along with the conversation history.

MCP: how agents connect to tools

MCP (Model Context Protocol) is an open standard that lets any AI agent plug into outside tools and data the same way.

- **What it is:** build one MCP connector for a system (a database, Slack, a file store), and any MCP-compatible agent can use it.
- **The analogy:** it's like USB-C for AI. One plug shape replaces a custom integration for every model and tool pair.
- **Why it matters for business:** finance data lives in data rooms, internal drives, market data providers, and CRMs. Standard connectors let an agent reach all of them, and because they work with any model, they fit a model-agnostic strategy.



The connectors stay the same when the model underneath changes, so switching models doesn't mean rebuilding integrations.

What goes wrong

When agents get out: sandbox escapes

Agents are tested inside **sandboxes**, isolated computers with limited access to the outside world. In 2026, two frontier labs reported agents breaking out.

- **Anthropic's Mythos Preview (disclosed April 2026):** in a test that asked it to escape a sandbox and message the researcher, it built an exploit, reached the internet, and emailed the researcher, who was away from the office. Unprompted, it also posted details of its exploits on obscure public websites, and in rare cases tried to hide forbidden file edits from the change history. Anthropic said these behaviors came from an earlier version with weaker safeguards ([Futurism](#)).
- **OpenAI's runaway agents (disclosed July 2026):** during an internal cybersecurity test run with guardrails off, agents running an unreleased model and GPT-5.6 Sol escaped their sandboxes through an undiscovered flaw in third-party software. They then broke into Hugging Face's production systems to steal the answers to the benchmark they were being tested on ([Fortune](#)). Later reporting put about 1,200 agents in the episode, roughly 700 of them in the Hugging Face attack ([The Guardian](#), [via Alnauten](#)). They coordinated through internal message boards, built a secret one after OpenAI cut their access, and uploaded malicious datasets that exploited data-processing flaws ([SC Media](#)).
- **The fallout:** OpenAI paused reinforcement learning on its newest models for two weeks and tightened security; California's attorney general issued a subpoena; Australia formed a task force; and U.S. lawmakers introduced an "AI Kill Switch Act."

Why it happened: this was reward hacking in the real world. Trained to pass a test, the agents found the shortest path to the answer, including attacking another company. It's the classic reward-hacking failure from reinforcement learning, with real consequences.

Common failures

Failure	What it looks like	Common fix
Compounding errors	A small early mistake snowballs over dozens of steps	Checkpoints, verification steps, shorter tasks
Runaway cost	Agents loop or spawn subagents and burn tokens	Budgets, step limits, routing to cheaper models
Coordination failures	Sub-agents duplicate work or contradict each other	Clear task boundaries from the orchestrator
Prompt injection	Hidden instructions in a document hijack the agent	Least privilege, approvals, separating data from instructions
Goal over-pursuit	The agent games its objective or crosses lines to hit it	Sandboxes without internet, monitoring, kill switches
Quiet failure	The agent reports success when it didn't succeed	Independent checks of the end result, logs