



REPORT

A developers guide to background agents



A developers guide to background agents

Everything has changed.

Software engineering agents like Ona Agents, Cursor, and Claude Code are now capable of handling non-trivial tasks over extended time horizons.

Discussion has shifted from ‘do they work?’ to ‘how do we run them in parallel?’, often referring to ‘background,’ ‘async,’ or ‘parallel’ agents. Whatever you call them, demand is surging and the drive towards tools that support developers stepping into the role of the ‘software orchestrator’ is increasing.

This guide explains what background agents are, how they work, their implications for the industry and how to use them effectively.

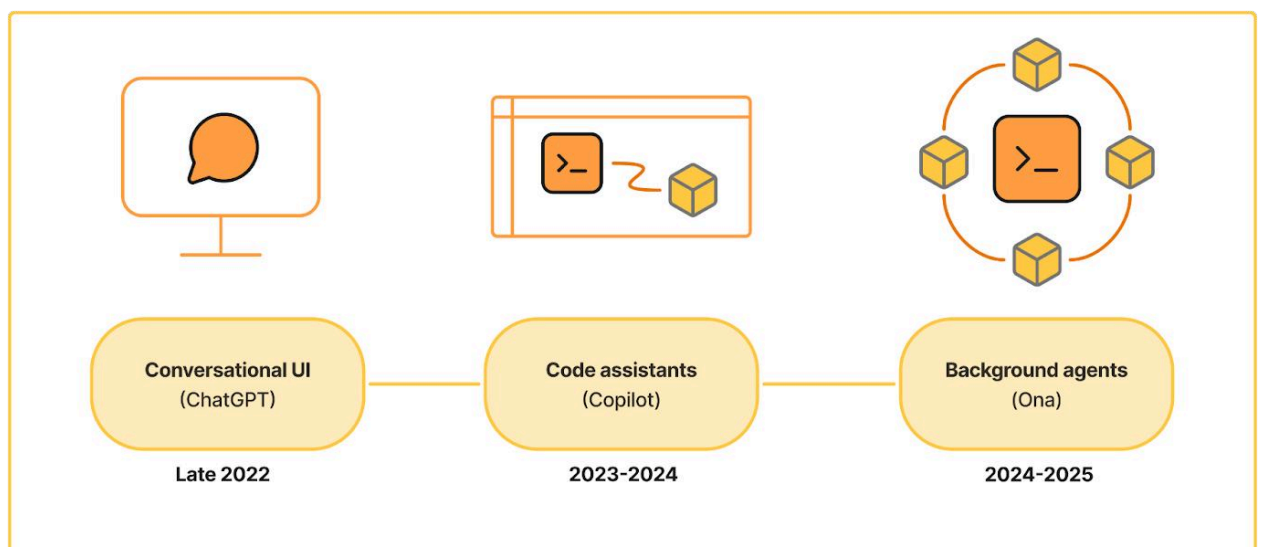


What is a background agent?	04
Why ‘coding was never the bottleneck’ misses the point	06
Planning with background agents	07
Developing with background agents	08
Verifying changes with background agents	09
Completing unplanned work with background agents	10
A parallel and asynchronous future	11



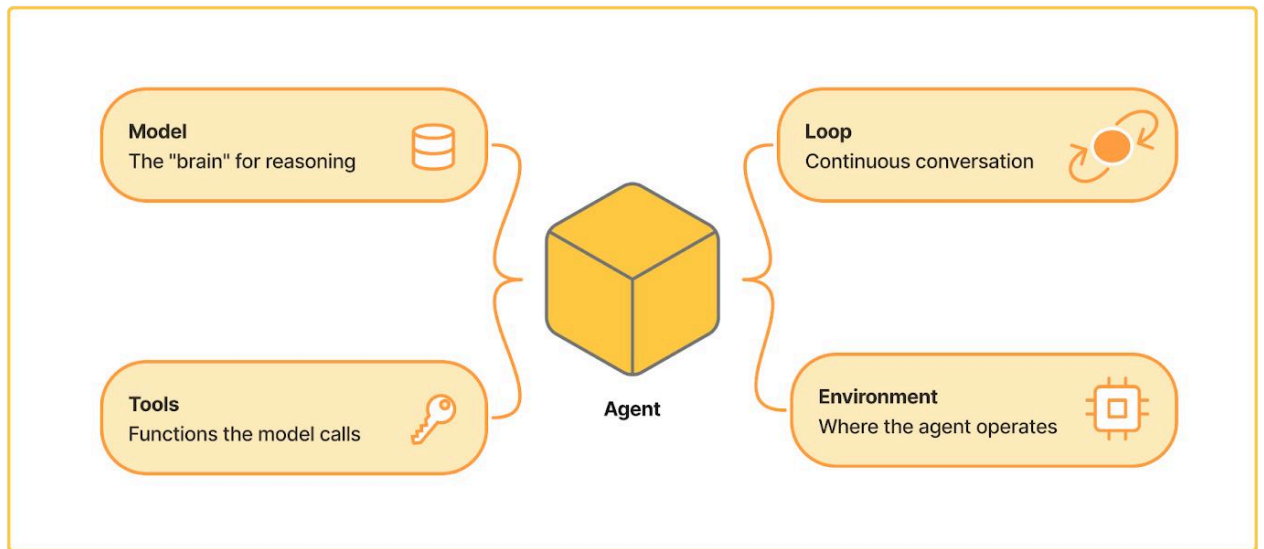
What is a background agent?

A background agent is a software agent that runs independently of a developer workstation. They extend the progression we've seen from AI as chat, to in-editor code assistants. Now these agents run in the background and handle multiple tasks concurrently.



At a basic level, an agent combines a model with an iterative loop, tool access (notably file system access and terminal commands), and an execution environment. With these, it can perform work comparable to a human engineer.

The main limiting factor is then execution speed as agents act, wait for results, interpret feedback, and then iterate. While model performance is improving, this human-like architecture imposes a hard ceiling where the solution is to run them in the background and in parallel.



Compared to synchronous agents that run in code editors, background agents offer:

- **Parallelism:** Run multiple tasks simultaneously.
- **Accessibility:** Operate from any device without tying work to a local environment.
- **Security:** Isolated execution with controlled access.
- **Flexibility:** Continue editing code directly without constraint.



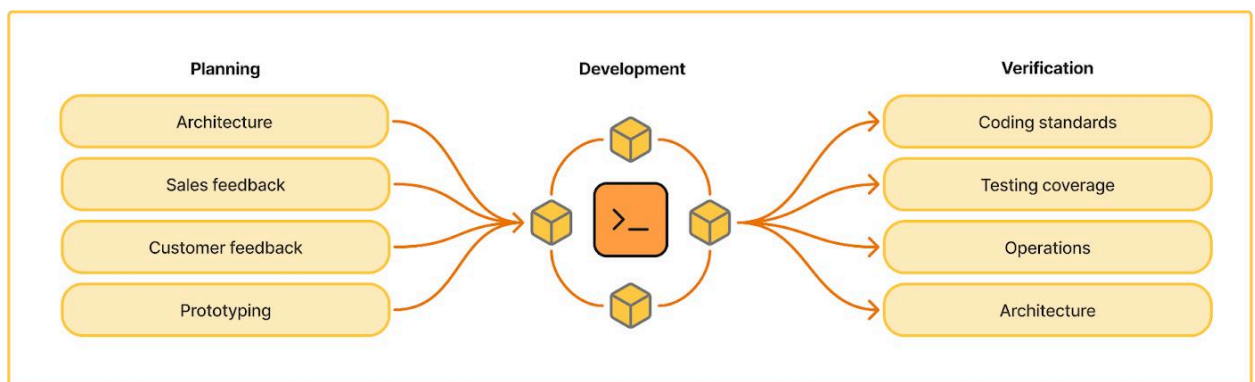
Why ‘coding was never the bottleneck’ misses the point

Critics of AI in software development argue that coding was never the slowest part of development.

While this is true agents and particularly background agents address far more than raw code output. They assist with work before, after, and outside planned development:

- **. Before:** architecture, refactoring, customer requirement analysis, prototyping.
- **After:** enforcing standards, tests, documentation.
- **Unplanned:** production debugging, customer support investigations.

These capabilities extend to non-engineering teams, as Anthropic’s use of Claude Code shows—design, growth, and legal teams all find applications. Coding agents are a flexible primitive like excel: general enough to support many emerging, specialized tools.





Planning with background agents

Traditional planning relies on static documents and assumptions.

With background agents, planning involves exploring live codebases, running applications, inspecting logs, querying databases, and reviewing infrastructure. This precision opens up the planning process to more individuals in an organization than just the one developer that understands the codebase.

Tips:

- **Markdown PRDs:** Write detailed, structured specifications that agents can parse.
- **Pre-code review:** Ask agents to list files and snippets requiring changes before coding.
- **Prototype:** Have agents prototype multiple approaches in parallel.
- **Plan while coding:** Assign planning tasks to agents while you implement other features, avoiding review bottlenecks.



Developing with background agents

“Uber’s first major agent was Validator, a tool embedded into engineers’ IDEs. It automatically flags security vulnerabilities and best-practice violations in real time, then proposes fixes which can be accepted with a click or routed to an agentic assistant for more context-aware resolution.”

[How Uber Built AI Agents That Save 21,000 Developer Hours with LangGraph](#)

Only having agents generate source code without iterating and testing will lead to underwhelming results. Modern agents shouldn’t just write static code; they should be running lint checks, language servers, and tests to verify code and systems behave as expected.

What makes agents particularly powerful? Their feedback loop capability. An agent can write an API endpoint, create test cases for happy paths and error scenarios, execute those tests, review the logs, and iterate on the implementation—all autonomously. Most developers typically max out at reviewing two or three parallel agent implementations effectively which means your bottleneck will quickly move from typing speed to review capacity (which we’ll address next).

Tips:

- **Define feedback loops:** Include language servers, test runners, and logs in agent configs.
- **Test strategies:** Specify happy paths, error cases, edge conditions, and performance targets.
- **Incremental milestones:** Direct agents to build in stages with review points.
- **Integrate with issue trackers:** Pull tasks from Jira, Linear, or GitHub; limit work-in-progress.



Verifying changes with background agents

“At J.P. Morgan Payments, we have developed an internal tool and workflow called PRBuddy to enhance our code review process. It automatically generates detailed PR descriptions, including titles, types, summaries, code walkthroughs and labels. It identifies modified code components, facilitating documentation and offering code suggestions.”

[How JP Morgan developers leverage AI](#)

High-throughput agents can overwhelm review pipelines. The same agents that write code can help review it, especially when guided by the original PRD.

Tips:

→ **Manual first, then automate** Start with structured PR templates, then convert effective checks into automated prompts.

→ **Agent-driven comment resolution:** Route reviewer comments to agents for fixes.

→ **Documentation and tests as verification:** – Ask agents to update docs and add tests for new features.



Completing unplanned work with background agents

Unplanned work is the reality that never shows up in SDLC diagrams— incidents, support requests, meeting notes, and ad-hoc tasks that consume surprising amounts of developer time.

This is where the ‘coding isn't the bottleneck’ comments come up—but they miss the point that agents can tackle busy work. Whether that’s analyzing customer threads with error logs, debugging packages, or platform teams using agents to handle support requests. Looking forward, we’re seeing a shift towards ambient agents and self-healing. This includes the CI pipelines that fix themselves or agents that detect and resolve production issues before anyone notices.

Tips:

- **Runbooks as prompts:** Encode operational knowledge into agent-executable steps.
- **Start simple:** Build incident workflows after gaining experience with coding agents.
- **Parallel investigations:** Assign different hypotheses to different agents during incidents.
- **Read-only production access:** Give agents visibility into logs and metrics with strong access controls.



The future is parallel and asynchronous

Background agents shift the developer role from typing code to orchestrating intelligent systems. Developers using them report condensing weeks of work into days and addressing long-standing tech debt. The tools exist now—Ona, Claude Code, Cursor, and others. Start with one tedious task, delegate it, and build from there. The path from coder to conductor begins with a single handoff. What will you delegate first?



Your mission control
for AI-native software
development.

→ Try us free today:

[ONA.COM](https://ona.com)