AI Coding Agent

Independent project · 2026 · Solo · Public repository · Beta

A coding agent that lives in the terminal: it reads and writes files, runs commands, and asks before doing anything it cannot take back, with whichever model you point it at.

Links: Repository (uhai) · Releases · Feature guide

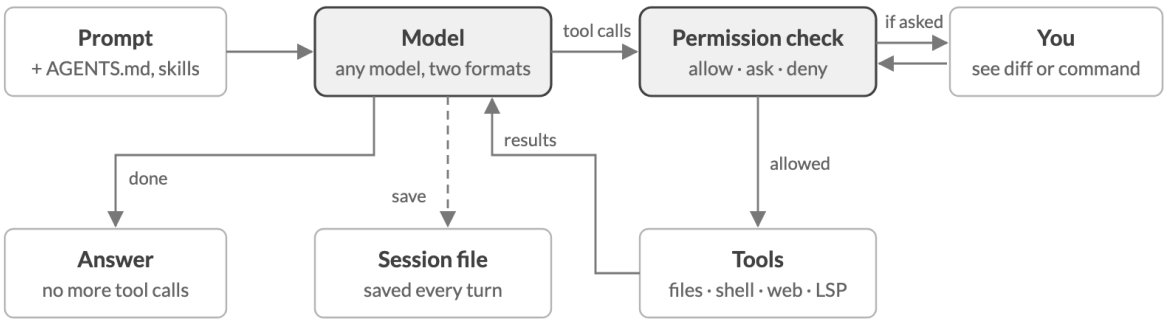
Why

Many coding agents are built around one vendor's models, with the agent loop inside a dependency. I wanted one that works with any model, including a company gateway or Ollama on my own machine, and is small enough to read end to end: about 15,000 lines of Go outside the tests.

What it does

- Works with Anthropic, OpenAI, or any endpoint that speaks either format; /model switches the model mid-conversation without losing history.
- Twelve tools: read, write and edit files, search, run shell commands, read the web, ask the language server about a symbol, keep a plan, and hand a job to a sub-agent.
- Asks before writing, running or reaching the network, and shows the diff or the exact command first.
- A background daemon holds the conversation, so work survives the terminal closing; every session is a file that can be resumed.
- Reads AGENTS.md, CLAUDE.md and skills in the layout Claude Code uses, so a project written for one needs nothing added.

Key flow



When the history fills the context window, it is compacted instead of failing the turn.

1. The prompt goes to the model with the project's AGENTS.md and the name and description of each skill.
2. The model answers with tool calls. Reading runs straight away; writing, running and network access check the permission rules and ask if needed.
3. Tool results go back to the model, and the loop repeats until it stops calling tools.
4. The session is saved after every turn. When the history fills the context window, it is compacted instead of failing the turn.

Key decisions

Decision	Why / trade-off
Providers split by wire format, not company	Two clients (Anthropic Messages, OpenAI Chat Completions) reach Gemini, Z.ai, OpenRouter and Ollama as custom endpoints. The cost: a custom endpoint has no known price, so none is shown rather than a wrong one.
A shell line is judged part by part	The line is split at &&, ||, ; and |, so allowing git does not quietly allow "git status && rm -rf /". A command that builds itself with $(...) always asks.
A daemon, with each background task in its own process	The conversation outlives the terminal, and a crashing task cannot take down the daemon or other projects' conversations. The cost is an HTTP API over a Unix socket between the UI and the daemon.
Skills load on demand	Only each skill's name and description go with every prompt; the body is read when the work needs it, so a project can keep many without paying for them each turn.
Docs held to the code by tests	A tool without a section in the guide, or a permission table that disagrees with the code, fails the build.
Tests fake the provider, not the agent	The real agent loop runs in tests against a fake model, so the loop itself is what gets tested.

Non-goals

- A Windows release: it compiles in CI but has never been run, so no binary is published.
- A Homebrew release before the first stable tag, so brew upgrade never moves anyone onto a beta.

Outcome

- Three beta releases, built by GoReleaser for macOS and Linux (amd64, arm64), plus go install.
- 355 tests, run in CI with gofmt, go vet and the race detector on every push.
- Lesson: a seam is proven by the next implementation, not the first. A third provider format landed without changing a single neutral type.

Code pointers

- internal/agent/agent.go: the agent loop and history compaction.
- internal/provider/provider.go: the neutral contract both wire formats implement.
- internal/config/permission.go: allow, ask and deny rules, and how a chained shell line is judged.

Stack

Go · Bubble Tea · Lip Gloss · LSP · Anthropic and OpenAI APIs · GoReleaser · GitHub Actions