

The Verification Gap

Generation is solved. Verification is not — and it is the exact bottleneck the leading AI coding tools leave open. This document names the gap, then shows the methodology that closes it.

THE PROBLEM IS ALREADY NAMED

“Verification, code review, and security became the bottlenecks once generation was cheap.”

Anthropic, on running an AI-native engineering organisation (2025)

Every tool is the same loop

Plan → edit → run → verify. Devin, Copilot, Claude Code, Cursor, Amp, Factory, Jules — all of them are a harness around a model looping over tools.

The model is not the difference

They all rent frontier models. The real difference between them is whether the **verify** step is something the system enforces — or something the model may skip.

The winner in this market will be decided by who can make “done” mean “verified”.

It's us — already built, already running.

Ten of twelve tools leave verification optional

TOOL	MANDATORY VERIFICATION GATE BEFORE "DONE"?
Factory (Missions)	Yes — a validation phase runs build, test, lint, typecheck, browser
Amp (Ship)	Yes — runs the full test suite before push
Devin	Partial — video QA, self-described "sanity check, not a replacement for tests"
Copilot Coding Agent	Partial — tests/linters, gated by repo hooks
v0 (Vercel)	Partial — the Vercel build/CI gate only
Claude Code	No — hooks only; the model may skip them
Cursor	No — hooks only; "user is responsible for verifying"
Jules	No — user setup script only
Replit / Lovable	No — opt-in / on-demand
Bolt / Windsurf	No — none documented / lint auto-fix only

A green check that never ran is not a pass. In ten of twelve tools, "done" does not mean "verified" — it means the model stopped.

Five principles that make verification structural

- 1 The gate is structural, not a choice.**
Required checks are matched *by name*; a skipped or neutral check is a fail — never a silent pass.
- 2 Prove the check can fail.**
Every guard is mutation-tested: break the guard, confirm it goes red. A check that has never failed proves nothing.
- 3 The reviewer must not grade its own work.**
Review runs on a different model than the coder, against its own evidence — not the coder's whole context.
- 4 Failures become memory.**
Every self-inflicted breakage is filed with a detection rule, and fed back to the reviewer on every turn.
- 5 Evidence carries provenance.**
A result must name who produced it, that it ran, and what would falsify it.

THE NUMBERS

Built and running — not a proposal

130

verify-* guard scripts across one codebase, each asserting a real failure mode

26

documented self-inflicted breakages, each with a detection rule

502

mutation tests — every text-falsifiable guard proven to fail (backlog 44 → 5)

–78%

reviewer context tokens with **zero** regression in what it catches

98.7%

of spend is the pipeline — so the lever is input tokens, measured, not guessed

0

gates that read “green” when nothing actually ran

WHAT THIS GIVES YOU

A drop-in trust layer for any agent toolchain

For a coding agent

A **named-gate contract** so a green PR means the gates ran — plus a reviewer that can never grade its own work.

For model routing

Measured role assignment: planner / coder / reviewer on separate, A/B-tested models — not vendor defaults.

For evidence

A **trust ladder** so every result names its producer, its falsifier, and its spread.

This is the layer your product is currently missing — and it is the layer that decides whether customers **trust the code you generate.**

Built, running, and ahead of the field — on the one dimension that decides trust.

Available to **license**, **integrate**, or **acquire**. The system, the receipts, and the full 12-tool survey are ready to share under NDA.