

The residue contract: the audit layer AI code review is missing

Platforms already have one model review another model's work, across families. What no platform records is what that review left open.

Nicolás Rocchia · Pelatech · August 2026 · pelatech.com.ar/blog/residue-contract

THE GAP

The round is a commodity now. The record does not exist.

GitHub added a cross-family second opinion to Copilot CLI, generally available since June 2026. The premise is right. What it produces dies with the session.

When one model writes and another critiques, the most valuable part of that critique is precisely what did **not** close: the unverified assumption, the tradeoff someone accepted, the escalation waiting for a human decision. That is the review's residue.

Today that residue has no record, no owner, and no enforcement, on any platform. The critique feeds the agent's reasoning and disappears: who reviewed whom, with what independence, what was dismissed without evidence. In Copilot CLI's own repository there are user reports of hand-extracting evidence from process logs to prove which model reviewed which, because the independence guarantee can fail silently and lands nowhere.

This case documents how I went after that gap: method with data first, then the tool, then the public claims. Everything dated and verifiable, because that is the whole point.

A note on this page: Pelatech is a Spanish-language site based in Argentina, so the navigation stays in Spanish. This article is the author's translation of the Spanish original, which is the version of record.

THE BUILD

Method, tool, proof and claim. In that order.

The order matters: each piece exists because the previous one backs it, and the public dates show it.

First, the method

I ran 52 days of disciplined adversarial review across five professional projects: one assistant generates, another from a different model family attacks, every finding driven to a terminal state. I published the method with the data before writing a single line of the tool: 91 documented review events, false positives and misses included. It is **the experience report** that already lives on this blog, with a Zenodo DOI.

Then, the tool

disensor (MIT, on PyPI and the GitHub Marketplace) runs the round and leaves it declared. It builds the package the reviewer receives, runs whichever assistant is on the machine, measures the working tree before and after, and anchors the result to the commits it actually reviewed. Out of that comes the JSON declaration: who generated, who attacked (model family plus the hash of the adversarial brief), every finding with its terminal state, and the declared residue. A deterministic validator applies the artifact rules and a CI action applies the gate checks on every PR. There is no language model anywhere in the validation path and no API keys: the gate only reads artifacts that live in the repository.

The reviewer is not tied to any vendor: any assistant with a command line works, and nothing gets copied and pasted between one model and the other. When no second model family is available, the round still runs and is declared as the degraded mode it is, with the correlation it failed to cover recorded as residue. A declared degraded round is worth more than a round that never happened.

The proof, in its own repository

The disensor repository governs itself: every PR passes through the published version of disensor, and the declarations directory holds the real, timestamped record of its own development, including the rounds that rejected my own changes. Releases ship via Trusted Publishing with OIDC: zero tokens on any machine.

The public claim, dated

I filed **the first recorded request** for review critiques to leave a verifiable record, in Copilot CLI's own tracker, motivated by their reports of independence failing silently. If the platform builds that layer tomorrow, the public record shows who specified it first and where they read it.

THE NUMBERS

All verifiable, none round.

Every figure has a public source: the report with its DOI, the repository, the PyPI records.

91

Review events measured before the tool was released

52

Days of documented pilot across five projects

363

Tests, in a repository gated by its own tool

03

Model families orchestrated in real rounds

ENGINEERING

Decisions I can defend in any room.

A control that decides whether a merge is allowed cannot run on good intentions. These are the decisions that hold it up.

Fail-closed everywhere.

No PR range means no pass; a malformed policy rejects instead of applying defaults. A control that degrades gracefully under a condition the attacker controls is not a control.

Policy is read from the target branch.

A PR cannot relax the rules that judge it, because the rules in force are those of the branch it wants to enter.

Evidence is append-only.

Review history cannot be rewritten by the change it reviewed.

The pin is a commit, and a command resolves it.

A tag can be moved, so it is not a root of trust. After a real incident (a hand-typed SHA my own test suite stopped), resolution stopped being manual work: the tool does it, dodging the annotated-tag trap that returns the tag object instead of the commit.

The pilot's most instructive defect family

: the test that verifies the substitute instead of the behavior. I documented it nine times, each caught by the other family's reviewer and recorded in a public declaration. A mock that accepted any argument, a test that read the source instead of executing it, a capture fixture that hid a real encoding crash.

PUBLIC RECORD

The scrutiny the method invites.

A method about review honesty is judged by how it answers when it gets reviewed itself.

The audit I did not ask for

When I presented the tool in a spec-driven development community, an independent reviewer read the source before commenting, verified that the declared epistemic boundary ships in the code, and pressed exactly where one should press: vacuous evidence, role separation, the risk of the control becoming ceremony. Their opening: that this closes a real gap in every SDD harness they had examined. Their closing, after the full exchange: a meaningful improvement over the alternatives. That public thread is the kind of scrutiny the tool exists to invite.

The rounds, lived

Development itself runs on the method: Anthropic models generate, OpenAI and Google models attack. One recorded event carries two reviewers because the first ran out of quota mid-round; the declaration says so, because declarations record what happened, not what should have happened.

The limits, in writing

The gate detects the empty field and the generic marker, not a well-written lie. Human sampling stays in the loop by design, and the open problems are public issues, not fine print: representing external evidence, and correlating declared residue with the defects that later escape.

THE ARCHIVE

This case, to go.

The case as a PDF to attach or print, and the primary sources to verify every claim.

ARCHIVE		PDF + SOURCES
EN	Download the case residue-contract.pdf · 6 pages · 297 kB · author's translation	↓
ES	Descargar el original en español contrato-del-residuo.pdf · 6 páginas · 301 kB · versión de registro	↓
GIT	The disensor repository MIT · the code, the real declarations of its own development, the open issues	↗
WEB	disensor.dev The product page, with the adoption guide	↗

CONTACT

Does your team review AI-generated code and cannot answer what the review left open?

That is the problem I have been solving, with the method and the tool in the open. If you are living it, or want to argue the design, write me.

EMAIL info@pelatech.com.ar

LINKEDIN nicolas-rocchia ↗

TAMBIÉN EN PELATECH		03
01	Controlled disagreement The experience report behind this case, with its DOI	→
02	Minimal adoption recipe The method on one page, ready to apply	→

Productos propios y trabajo a medida para clientes