

CASE STUDY

Does the code do what its user guide says? 100 sealed checks against a decades-old Fortran library

Every promise in a numerical library's user guide, turned into a check that quotes it, sealed before the run and run against the original code. What held, where code and guide disagree, and what the result does not prove.

Case study. Certified run of 16 October 2026. OneDroid Argus, one system, reviewed by a second AI model.

In short

- Our team turned the user guide of a decades-old numerical Fortran library into 100 checks of 37 published claims: every worked example, what each exit code and counter means, and the guide's other promises. Each check quotes the sentence it tests.
- The 100 checks were sealed, and their fingerprint recorded on a public network, before the final run against the original code, built unmodified.
- 91 held, 9 failed, 0 could not run. All 9 failures are places where the code's own behaviour does not match its guide's own words.
- Before trusting the 91, we showed the checks can fail: two deliberately broken builds turned red the checks that depend on what was broken, and left every unrelated check of the arithmetic green.
- An AI model from another company reviewed every finding and every held check. It confirmed all 9 failures and found no check that cannot fail. It also found held checks that test less than their titles say, and the page lists them.

The question

A numerical library has run for decades, and its user guide says, sentence by sentence, what it does. Someone is about to depend on it, or about to have an AI agent rewrite it. Two questions come first: does the code still do everything its guide promises, and can anyone check the answer without taking our word for it?

It is the same question as "an AI agent wrote 50,000 lines overnight: how do I know what it did?". Nobody reads all of it. You can hold it to what it was supposed to do, with checks it never saw.

The method

Step	What we did
The promises	The user guide, the documentation at the top of each routine and the published test programs. 37 claims became 100 checks; each check carries the sentence it tests and where it is. 2 of the 100 are properties of our own (the code gives the same answer twice, and it reports the exact sources we pinned), and they are labelled as ours
The system	The original code, built unmodified with a current compiler (gfortran 14.4) on one x86-64 machine. Every source file pinned by its SHA-256 fingerprint
Severity, fixed before the run	Medium: a stated function or a documented example fails. Low: wording that does not match behaviour, or a constant that is not the defined value. High or critical: a wrong answer reported as a success
Predictions, before the run	Every expected failure was predicted in writing from reading the code. No check predicted to hold turned red
Can a check fail?	Two builds, each with one constant changed, turned red the checks that depend on it (and, as intended, the check that the code is the one we pinned) and left every unrelated check of the arithmetic green
The seal	The 100 checks were committed and recorded on a public network before the final run. One certified run
The review	OpenAI gpt-6.1-sol read every finding and every held check from the files, in a read-only sandbox. Its corrections changed our wording, never the sealed checks
Who	Our team. One AI session wrote the checks and ran them. There was no builder: nothing was fixed

The result

Certified run, 16 October 2026	
Checks	100
Passed	91
Failed	9
Errored	0
Run status	failed

The run status is the word OneDroid Argus gives a run. It reads `failed` when at least one check failed, and it is not a judgement of the product: the counts above are the result. An errored check is one that could not run, and an error is never counted as a pass.

A run status of `failed` means at least one check failed: 91 of 100 held. A rehearsal of the same sealed set gave the same tallies. OneDroid Argus gives no score.

Where the code and its guide disagree

Medium, 4 checks: one path. When the user's own function asks the solver to stop part-way through its first step, two of the outputs are not what the guide says they are: the returned solution still carries a tiny internal adjustment instead of the starting point, and a counter reports calls that never happened. Routines that compute the same step another way return both correctly on the same path.

Low, 2 checks: a counter's definition. The guide defines a counter as the number of calls to the user's function, but the calls made for progress printing are not counted.

Low, 3 checks: machine constants. Three constants the documentation defines exactly are 12-digit approximations in the code. The effect on results is tiny, but the definition is not met.

Nothing was fixed, and nothing was reported to the library's authors. The job was to measure. Two more behaviours on the same kind of path were found by the reviewer in the source and are under research: they need checks of their own in a new sealed set before they can count.

What held

- **Every worked example in the guide.** All 11 reproduce the printed output to 7 digits, including every printed count, on a compiler decades newer than the guide.
- **What the exit codes mean.** Improper input in 22 cases across 10 routines, the user stopping the run, limits on the number of calls, and tolerances below machine precision: each gave the exit code the guide names.
- **Scale invariance,** as the guide claims, within the bands it prints.
- **The published test deck** for the derivative checker: every problem read as labelled, run one at a time.
- **Repeatability:** the same request twice gave bit-identical results.

The test boundary: what this result does not prove

A verdict is only as useful as the line around it. This is the line.

Not proved	Why
Other machines and compilers	One platform. The guide itself says results elsewhere may differ slightly
What was running	The certificate binds the build we declared. The executor used here could not measure the running build, and the certificate says so
Independence	One AI session wrote the checks and ran them. The seal proves the order of events, not independence. Our driver is our own code: a mistake in it would read as the library's
A second run by the reviewer	The review was a reading of the evidence, not a re-run
Full strength of every held check	The review found some held checks that test less than their titles (a norm instead of the whole vector, the order of values instead of the values). They held as written; stronger versions go in the next set
Every path	Not exhausted. The reviewer found two more candidate behaviours in the source, not yet run
Everything else the guide says	Single precision, faults raised inside the user's function, operation counts, storage and convergence-rate statements: not tested

Our own mistakes are part of the record too. One of our checks was wrong at first (it asked for a printed value under a setting the guide does not print it for) and was corrected before the seal. Our first deliberately broken build changed nothing, as we had warned it might, and was repeated with a stronger change. And we did not check before sealing that this executor could measure the running build.

How anyone can check it

The sealed set and the verdict were recorded on Base Sepolia, a public Ethereum test network (blocks 48212445 and 48212457), and submitted to Bitcoin through OpenTimestamps. What goes on a chain is only fingerprints, counts and the verdict, never a check or an expected value. The certificate verified with the public verifier in the open-source OneDroid Argus runner (MIT), with no account: github.com/OneDro1d/argus-runner.

Why it matters before an AI rewrite

In a separate study, Claude Code ported this same library to Go. A tester that steered the port with checks from the user guide made it worse against the original's answers, 81 of 109 hidden checks, because the agent "fixed" the places where the code and its guide disagree. Knowing where they disagree, before anyone rewrites anything, is what this study gives a team. The modernization case study: onedroid.ai/case-study-modernization-three-setups.

What we take from it

- A user guide is a list of promises. Turn each sentence into a check that quotes it, and you can show which ones the code keeps.

- Seal the checks before the run and keep them from whoever builds. Whoever builds, a person or an AI agent, can't game checks it never sees.
- Prove each check can fail before you trust the ones that pass.
- Publish the test boundary with the verdict: what was tested, what was not, and what the certificate does not prove.
- It is affordable. In a separate measured experiment, the independent tester was one session of about \$5.30 for a feature the coding agent built at \$1.01 to \$1.97 a run: onedroid.ai/case-study-claude-code-alone-vs-tester.

The library, its authors and its maintainers are not named: we have not asked them, and this is a record of what one set of checks saw, not a review of their work. gfortran, OpenAI, Claude Code, Base, Bitcoin and OpenTimestamps are names that belong to their owners. OneDroid Argus is built by Providentia Worldwide. Questions: michal@onedroid.ai. How OneDroid Argus works: onedroid.ai/argus.