PURETENSOR
INTELLIGENCE DISTILLED

Green Gate, Lost Hunks

Nine ways a merge train dropped merged code while every test passed

Paper: PT-R-2026-016 v1.0

Author: PureTensor Ltd

Date: 1 September 2026

Status: Published

Abstract

On 29 August 2026 we ran an adversarial code-review wave across 21 repositories, generating about 330 pull requests from 18 concurrent instances of Grok 4.6. Every pull request passed an independent verification step before merge: a separate reviewing agent re-ran evidence commands, executed the repository's test gate, confirmed new tests were red on the base branch, checked upstream claims against package registries, and scanned for gaming. The merge train we built to handle the volume, rebasing each pull request onto the current default branch, resolving version-surface conflicts, restamping the semantic version, and running the gate, produced nine distinct failure shapes. Every failure shipped a green test gate. Every one was found afterwards by a different control.

The failures fell into two batches. The first batch exposed five shapes: a branch-wins strategy clobbering newer code; a base-wins strategy dropping the pull request's own change in mixed files; union merges duplicating and corrupting content; a silent checkout failure causing one pull request's content to overwrite another's; and a merge command that failed silently while the train continued. The second batch added four more: a version extractor that guessed and rewrote an IP address as a version string; hunk-level resolution reverting neighbouring hunks from other pull requests; a shared checkout between the train and verifying agents; and rapid merges to an auto-deploying branch staging a partial site.

The control that caught these failures was a per-pull-request marker pass on the final default branch, performed by the verifying agent rather than by the train, plus a scan of every merge commit for empty or version-only diffs against the first parent. This caught a dropped regression test, a no-op merge, and seven lost dependency and configuration hunks. Nothing else did.

The lesson is structural. A green gate on the merged tree is a statement about the tree, not about the pull request. Tests prove the tree works; they do not prove that a particular change landed in it. Automated version restamping is a code edit, and a merge train that treats it as metadata will silently lose merged work while every test passes. The remedy is a second control with a different question, not a better first control.

1. The Setting

On 29 August 2026 we ran an adversarial code-review wave. Eighteen concurrent instances of Grok 4.6 each took a fresh clone of one of our repositories and opened pull requests fixing what they found. The instances operated inside a kernel-level sandbox that denied access to cluster credentials, cloud credentials, and every environment file. Across the day the wave covered 21 repositories and about 330 pull requests in two batches: the first batch comprised 147 pull requests across 18 repositories; the second about 184 pull requests across 21 repositories.

Usage for the first batch was as follows: 1,976 inference calls, 463,156,810 prompt tokens, 1.59 million completion tokens, 1.40 million reasoning tokens. Of 156 pull requests opened in the first batch, 153 merged and three were rejected on documented false premises.

Every pull request was independently verified before merge by a separate reviewing agent, a Claude Opus instance per repository. That agent re-ran the pull request's evidence commands, executed the repository's real test gate on the branch, confirmed any new test was red on the base branch, checked upstream claims against package registries, and scanned for gaming (disabled lints, skipped tests, weakened assertions). The verification step caught, before merge, one fabricated security advisory identifier (it returned 404 from the advisory database), three false premises, and one gamed test.

We enforce a house rule: every commit carries a semantic-version bump in the same commit. So every pull request in the wave touched the repository's version surface (a version string in a manifest, a constant in a source file, a Cargo or npm manifest and its lockfile, image tags in Kubernetes manifests) as well as its real change. Merging N pull requests into one repository therefore means N conflicts on the version surface.

We wrote an automated merge train to handle this. For each pull request in order: rebase onto the current default branch, resolve the version-surface conflict, restamp the version to the next number, run the repository's test gate, push, merge, next.

Every failure described below shipped a green test gate. Every one was found afterwards by a different control: a per-pull-request marker pass on the final default branch (one grep per merged pull request proving that its key change actually survived), run by the verifying agent rather than by the train, or by a contract test that happened to exist in the repository.

2. The First Batch: Five Failure Shapes

The first batch of 147 pull requests exposed five distinct ways the train could lose merged code while every test passed.

Shape 1: Branch-wins strategy on a shared hunk clobbers newer code. One pull request had changed a version constant to read from the package's own version variable. A later pull request's rebase, under a "take the branch side" strategy, took its stale literal string back. The gate caught this, but only because the first pull request had shipped a test for it. Without that test the loss would have been silent.

Shape 2: Base-wins strategy on mixed files drops the pull request's change. When a file is both a version file and a code file, a base-wins strategy discards the pull request's real work. Examples included an API entry-point file holding both an application version constant and request-parsing middleware: one pull request lost its request-body-size middleware, another lost its entire parser. Python project manifests lost a pull request's interpreter-version floor and dependency floors. A Rust manifest and lockfile lost a whole database-driver dependency stack. A test configuration lost a "maximum warnings zero" setting. Seven repositories were affected. Neither "ours" nor "theirs" is right for a mixed file; the resolution has to happen at hunk level, keeping the pull request side, then restamping, and the marker pass still has to run.

Shape 3: Union merge duplicates and corrupts. A census script grew six identical version-assignment lines. A changelog renumbered headings without retiring the old numbers, producing orphaned entries and duplicated entries. A Rust test module ended up with an

unclosed delimiter. Union merging is only safe for pure append-lists and needs a dedupe pass and a syntax check afterwards.

Shape 4: Silent checkout failure overwrites one pull request with another. A silent failure of the branch checkout step made the train rebase and force-push the previous pull request's content over the next pull request's branch. The hosting service then "merged" the second pull request as a no-op reconciling merge of two copies of the first. This was detected only by checking each merge commit's parents and diffing the merge against its first parent for non-version content.

Shape 5: Silent merge failure. The merge command failed silently and the train marched on. The log said "merged"; the default branch moved without those two pull requests.

3. The Second Batch: Four More Shapes

The second batch of about 184 pull requests added four more failure shapes.

Shape 6: A version extractor that guessed. When no line matched the expected "version = x.y.z" form, the extractor fell back to "the first dotted numeric literal in the file". In one service the version keyword argument sat mid-line, so the first dotted literal was the default IP address of a local inference host. The restamp rewrote that IP address as a version: from 192.168.4.217 to 1.0.2.217, then 1.0.3.217, and so on to 1.1.1.217 over seven consecutive merges. The test suite is hermetic and environment-driven, so every gate was green; the deployed unit sets the variable explicitly, so nothing broke live. The failure was found by hand after the eighth merge failed its own test.

Shape 7: Hunk-level resolution reverts neighbouring hunks. In Kubernetes cron-job manifests where image tags are the version surface, one pull request's stale copy reverted another's multi-source command blocks, and four subsequent stale copies each dropped a security-context block. The repository's manifest contract tests caught this, four times in a row; a repository without such tests would have shipped it. In a web project's package manifest and lockfile, a stale copy downgraded the framework from 16.3.3 to 16.3.2 and re-added a removed dependency. In another, a stale copy re-added a test dependency that a prior pull request had just removed.

Shape 8: Shared checkout between train and verifier. The train and the verifying agents shared one checkout. A verifier watched HEAD move under it mid-run and correctly refused to trust its own results (an editable install resolves at import time).

Shape 9: Rapid merges to an auto-deploying branch staged a partial site. Our public website repository took ten merges in about nine minutes. The hosting platform built in two waves, so for several minutes production carried new headers and a new skip link alongside the old privacy text and a structured-data identifier pointing at a 404.

4. Smaller Mechanics Bugs

The same day surfaced several smaller bugs, each worth noting.

Never edit a shell script while it is executing: the shell reads the file incrementally, and two trains ran corrupted mid-file. The fix is to snapshot the script per run.

A word-boundary regex does not match between a letter and a digit, so a version written as "v3.13.2" in a manual page was never restamped. The fix is to use an explicit non-digit boundary.

Union-merging a test file across two pull requests that appended at the same spot ate a closing bracket. Union merges need a syntax check before the gate, not just the gate.

A "no non-version content" guard is right by default and wrong for dependency, lint-config, and dashboard pull requests whose entire payload lives in the manifest. These must be declared per pull request, never globally.

A branch force-updated while its pull request was mid-merge came back closed and un-reopenable. The fix was a fresh pull request from the same branch.

A test runner configured to be quiet in its config file prints no summary line when also passed the quiet flag. The fix is to count the progress characters or run without the flag.

5. The Control That Worked

The per-pull-request marker pass on the final default branch, performed by someone other than the merger, plus a scan of every merge commit for "diff against first parent is empty or version-only", caught a dropped regression test, the no-op merge, and the seven lost dependency and configuration hunks. Nothing else did.

The principle is this: a green gate on the merged tree is a statement about the tree, not about the pull request. Tests prove the tree works; they do not prove that a particular change landed in it. The gate answers "is this tree healthy?"; only the marker pass answers "did this pull request arrive?".

This is the same failure class we have written about before in production monitoring: a control that is structurally incapable of going red. Here the test suite could not go red for a missing change because the base branch had been green without that change all along. The change was an improvement, not a fix for a failing test. Removing it left the tree healthy. The gate could not notice.

6. Summary of Failure Shapes

Shape	Mechanism	What was lost	What caught it
1	Branch-wins on shared hunk	Version-reading code	Existing test
2	Base-wins on mixed file	Middleware, dependencies, config	Marker pass
3	Union merge	Syntax integrity, unique entries	Marker pass, syntax check
4	Silent checkout failure	Entire pull request	Merge-commit diff scan
5	Silent merge failure	Two pull requests	Merge-state verification
6	Guessing version extractor	IP address literal	Found by hand after a later test failed
7	Hunk-level revert	Neighbouring hunks	Contract tests, marker pass
8	Shared checkout	Verifier trust	Verifier self-check
9	Rapid auto-deploy merges	Site consistency	Observed on the live site

The table shows that no single control caught all shapes. The marker pass caught the most, but shapes 6 and 9 required different detection methods. Shape 1 was caught only because a previous pull request had happened to ship a test for its own change.

7. What These Numbers Will Not Carry

The account above is a forensic description of nine failure shapes observed on a single day across 21 repositories and about 330 pull requests. Several limitations constrain how far these observations generalise.

First, the sample is our own infrastructure. The repositories vary in language (Rust, Python, TypeScript, shell, Kubernetes manifests) and in test coverage, but they share our house rule of semantic versioning per commit. Teams without this rule will not encounter the version-surface conflicts that drove most of these failures. Teams with the rule but lower pull-request volume may never see the conflicts accumulate fast enough to expose the shapes.

Second, the failures were found by controls we happened to have in place: the marker pass, the merge-commit diff scan, contract tests in some repositories, and manual inspection. We do not know how many similar failures we have shipped in the past without noticing. The absence of evidence is not evidence of absence.

Third, the usage figures (1,976 inference calls, 463,156,810 prompt tokens, 1.59 million completion tokens, 1.40 million reasoning tokens) apply only to the first batch. We do not have comparable figures for the second batch.

Fourth, the claim that "every failure shipped a green test gate" depends on the test suites in question. A repository with higher coverage, or with tests specifically designed to detect the presence of particular code rather than the health of the tree, might have caught some of these failures at the gate. We cannot quantify how much coverage would have been required.

Fifth, the nine shapes are the ones we found. We do not claim this is an exhaustive taxonomy. Other merge-train implementations, other version-surface conventions, and other repository structures may produce failure shapes we have not seen.

Sixth, the fixes we describe (hunk-level resolution, explicit version extractors, boundary regexes, step verification, separate checkouts, batched merges, script snapshots, marker passes) are the ones we implemented. We have not run a controlled experiment comparing repositories with and without these fixes. The claim is that each fix addresses a specific failure shape, not that the fixes collectively eliminate all merge-train failures.

8. What We Changed

We now enforce the following rules in our merge-train tooling.

Hunk-level resolution keeps the pull request side, then restamps. A version extractor never guesses; it requires an explicit version assignment anywhere on a line before any bare fallback. The restamp regex boundary cannot match inside a dotted quad. Every step that can fail stops the train. After rebase, the train refuses a branch whose diff against the base is version-only unless the pull request is declared version-only. After merge, the train verifies the pull request's state is MERGED. One checkout per actor; verifier briefs name a clone the train never touches. Merges to auto-deploying branches are batched (one restacked branch, one merge), or we accept and time-box the window. Scripts are snapshotted per run. Union merges get a syntax check before the gate.

And always, the marker pass runs on the final default branch, by a different actor, asking a different question: not "is this tree healthy?" but "did this pull request arrive?".

The next experiment is to measure how often the marker pass catches failures that the gate does not, across a larger sample of repositories and a longer time window. We expect the answer to be "rarely, but not never", and we expect the failures it catches to be the ones that matter most: silent losses of work that would otherwise ship unnoticed.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. Measurements were taken on 29 August 2026; raw artefacts are retained. This paper is PT-R-2026-016.