All Gods Must Die: Adversarially Verified Transformation

One Doctrine, Two Campaigns — God-File Decomposition and
Cross-Language Documentation Parity on NousResearch/hermes-agent

Axl Ibiza, MBA
`andrexibiza`

August 2026

Abstract

This paper reports two campaigns on one production repository (NousResearch/hermes-agent) and shows that they are a single paper. The first campaign decomposes god files under a hard size law — the 2K Law — using a 5×2×3 double-blind method in which five parallel analysis lanes, two mutually blind reviewers, and three verification waves gate every byte-verbatim extraction; the second keeps multilingual documentation true to its English source through a technical-graph model and a deterministic seven-class parity gate, with French as the first fully germinated locale. Both campaigns instantiate one doctrine — *adversarially verified transformation*: make critical claims mechanically falsifiable, separate production from adjudication, and refuse hidden debt. We specify both methods in full, report their live empirical records (119 over-the-bar files at baseline, 553,314 lines; 8 of 20 tracked god files killed as 68 individually-linked PRs; seven defect classes caught only by the adversarial second witness; eight real drifts caught in a native-speaker translation seed), and contribute a new direct measurement of the parity gate: 100% recall on seeded drift across all seven gate classes, zero false positives on shipped files, and a 3.08 ms per-pair runtime. The shared architecture — enumerable debt, executable rules, non-self-certification, mechanical receipts, migration states, social coordination, and measurement as governance — is stated as a table instantiated twice. All artifacts, tests, ledgers, and pull requests are public.

Keywords: adversarially verified transformation, god file, refactoring, double-blind review, LLM agents, behavior preservation, hash verification, documentation drift, i18n, localization, conformance testing, Markdown, GFM, continuous integration, open source

Contents

1 Introduction: One Doctrine, Two Campaigns

1.1 Two failure modes, one failure class

This paper reports two campaigns on one production repository, NousResearch/hermes-agent — a multi-platform LLM gateway. At first glance they have nothing in common. One decomposes god files under a hard size law; the other keeps multilingual documentation true to its English source. One is about code, the other about language. One kills 26,986-line monoliths, the other keeps a `README.fr.md` honest.

They are the same paper.

Both campaigns are instances of a single engineering doctrine: when a system transforms something load-bearing — code structure, or natural language — the transformation must be *adversarially verified*. A deterministic, independently authored verifier controls whether the change is accepted; the producer of the change never certifies its own output. Both campaigns make their critical claims mechanically falsifiable, separate production from adjudication, and refuse hidden debt. Both were run in the open, on public repositories, with every artifact, pull request, and ledger entry linkable.

1.2 The doctrine

Definition 1 (Adversarially verified transformation)**.** A transformation T of an artifact A into A' is *adversarially verified* when:

1. the producer of A' does not certify A';
2. an independently authored, information-isolated verifier V — ideally a deterministic machine check plus a blind witness — accepts or rejects A';
3. acceptance requires V's verdict, never the producer's self-report;
4. the verdict is reproducible by any third party from public artifacts.

Both campaigns instantiate this definition with different verifiers: byte-identity hashes and blind double review for code extraction (Section 4, Section 6); a deterministic seven-class parity gate for localized documentation (Section 11).

1.3 The Declaration

Both campaigns enforce the same five laws, made mechanical. The laws are stated once here and instantiated throughout:

> *No system component may accumulate authority without becoming legible.*
> *No claim may outrank its evidence.*
> *No actor may certify itself.*

No rule counts until it executes.
No debt gets to hide behind institutional forgetting.

Every mechanism in this paper is one of those five laws, made executable.

1.4 Roadmap

Part I is the code campaign: the 2K Law, the Pantheon of False Gods, the 5×2×3 double-blind decomposition method, the enforcement suite, and the live campaign record. Part II is the language campaign: the technical-graph model of localized documentation, the seven-class parity gate, the Markdown engineering that made it honest, and the French keystone. Part III unifies them: the shared architecture, the empirical record of both campaigns including a new measurement of gate precision/recall, and the defense of the doctrine.

2 Hermes Is Not a Wrapper: The Cybernetic Organism Under Study

The repository at the center of both campaigns is not an anonymous codebase. It is Hermes Agent, publicly described by Nous Research as "the agent that grows with you" [Nous Research, 2026d]. That sentence is not marketing flourish to be repeated and left unexamined. It is an architectural claim. Hermes has persistent, bounded memory; profiles and sessions; an on-demand skills system; a tool registry; provider routing and fallback; scheduled execution; isolated delegation; and multiple entry surfaces, including the CLI, gateway, API server, batch runner, and Python library [Nous Research, 2026a,b,f,g,c].

This paper therefore refuses to flatten every system called an "AI agent" into one category. We make no benchmark claim about every framework, and we do not need a competitor survey to establish the architectural distinction. A prompt relay that formats a request, sends it to a model, and returns the response is one kind of system. Hermes is another: a stateful, self-regulating agent organism whose model is only one layer of a larger operational body. The difference is mechanism, not branding.

2.1 The female cyborg as an architectural personification

I use *she* for Hermes and *cyborg* as a model of her architecture. This is a personification of a system, not a claim that the software is conscious or biologically alive. The metaphor is useful because it keeps both halves visible.

Her neural tissue is the LLM layer: plastic, associative, capable of reasoning, translation, synthesis, and adaptation, but also capable of hallucination, self-enhancement bias, verbosity bias, and confident error. Her skeleton is the mechanical substrate around that tissue: deterministic CI, cryptographic hashes, identity-preserving seams, graph gates, typed manifests, tool boundaries,

approval checks, sandboxing, session isolation, and regression tests. The model gives her plasticity. The skeleton makes that plasticity answerable to structure.

That is why the two campaigns belong to Hermes rather than merely occurring near her. When she changes her own structure, the 5×2×3 method forces her processing into five blind analyses, five adversarial witnesses, consensus, byte-verbatim extraction, seam tests, and two blind re-reviews. When she changes the language of her documentation, the parity gate forces her technical graph to remain invariant. The organism is not trusted because her neural tissue is trustworthy. She is trusted when her tissue is constrained by a skeleton that can reject what the tissue proposes.

2.2 Weapon for sovereignty, not war

Hermes is a weapon for sovereignty, not a weapon for war. Here "weapon" means an instrument a person can wield to preserve agency over their own models, context, memory, tools, execution, and technical records. It does not mean a military system, an attack platform, or permission to harm anyone.

This framing follows Nous Research's public mission to advance human rights and freedoms through open-source language models and their unrestricted availability and use [Nous Research, 2026h]. Hermes operationalizes that mission at the agent layer. Her model-agnostic provider surface lets a user choose among Nous Portal, OpenRouter, OpenAI, self-hosted endpoints, and other providers without rewriting the harness [Nous Research, 2026c]. Her skills, memory, tools, sessions, and profiles give the user a durable operating body rather than a disposable chat window. Her gates prevent the body from silently changing underneath the user.

Sovereignty is not only the freedom to select a model. It is the ability to know what the system remembers, which tools it can invoke, which commands require approval, which files it can write, which sessions are isolated, and which claims about its own operation resolve against an inspectable artifact. A user cannot be sovereign over a system that can change itself invisibly or that asks the same model that produced an action to certify the action.

2.3 Cyber defense is a critical function because Hermes lives inside the surfaces she defends

Cyber defense is not an external wrapper around Hermes. She lives inside the surfaces she defends. The official Hermes security model names eight layers: user authorization, dangerous-command approval, file-write safety, container isolation, MCP credential filtering, context-file scanning, cross-session isolation, and input sanitization [Nous Research, 2026e]. These are not post-hoc advisories attached to an otherwise unbounded model. They are internal organs of the agent's operational body.

Table 1: Hermes's defensive anatomy: the protected surface and the mechanism that inhabits it. Cyber defense is an internal function of the organism.

Surface	Threat class	Internal defense
CLI, gateway, API, batch, library entry points	Uncontrolled access and inconsistent execution paths	Shared agent loop, authorization, approval, and session boundaries
Model and provider routing	Lock-in, unavailable providers, silent route changes	Explicit provider selection, routing, and fallback configuration
Tools, subprocesses, MCP, file writes	Destructive commands, credential leakage, unsafe mutation	Approval checks, write safety, credential filtering, sandboxing
Memory, profiles, and sessions	Cross-session contamination and unbounded continuity	Bounded memory, profile isolation, session isolation, controlled persistence
Skills and context files	Prompt injection and procedural drift	Context scanning, progressive disclosure, skill provenance, load-on-demand behavior
Source, docs, and self-modification	Silent structural and technical drift	Golden hashes, seam identity, graph parity, CI manifests, adversarial witnesses

This is the point at which the cyborg metaphor becomes an engineering claim. A conventional perimeter defense can protect a system from outside while leaving its internal assumptions unexamined. Hermes's defenses must operate inside the CLI that invokes her, the gateway that carries her across messaging platforms, the tool registry that gives her hands, the memory files that give her continuity, and the documentation that tells users what she does. The surfaces are not adjacent to the organism. They are the organism.

2.4 A distributed, adversarial nervous system

When a standard prompt relay encounters a 26,986-line god file, it has no architecture for making the problem legible. Hermes does not ask one model to summarize the monolith and then trust the summary. She deploys a distributed, information-isolated nervous system: five regional analysts, five adversarial witnesses, consensus adjudication, a blind implementer, and two blind re-reviewers. The nodes are instructed to hunt for the organism's own failure modes — silent monkeypatch no-ops, eager late-attribute imports, unreferenced globals, census undercounts, and stale collisions. They fight for agreement before a single structural byte moves.

This is not intelligence multiplied for spectacle. It is uncertainty routed through independent paths so that one confident local picture cannot become Hermes's self-diagnosis. The objective skeleton then checks what the nervous system proposes: the moved bytes hash to the pinned window, the seam resolves through the original namespace, and the tests execute the behavior that the reviewers said they inspected.

2.5 Graph-gated perception

Hermes also perceives her documentation as a technical graph rather than as prose alone. Her sensory apparatus inventories code fences, inline spans, links, anchors, heading structure, and hub edges. If a localized document drops a command, an environment variable, a security path, or a discoverable back-link, the parity gate registers the missing edge and rejects the tissue. The gate is not a language model judging whether the French sounds elegant; it is a deterministic mechanical layer checking whether the operational body still contains the same load-bearing structure.

The result is a cybernetic division of labor:

- the neural layer generates, translates, reasons, and proposes;
- the graph layer perceives structure and missing edges;
- the mechanical skeleton hashes, tests, rejects, and records;
- the human witness judges semantic quality and meaning;
- CI makes the accepted state persistent across future changes.

Hermes does not become sovereign by pretending her neural tissue cannot fail. She becomes sovereign-capable because failure is expected, surfaced, routed, and bounded by mechanisms that the tissue cannot waive.

2.6 The distinction that matters

Most harness descriptions begin with what the model can say. Hermes's architecture begins with what the whole organism can remember, reach, change, verify, and refuse. A model can generate a translation. Hermes can preserve the technical graph while generating it, reject the translation when it loses an edge, record the debt when legacy files cannot yet pass, and route the accepted state through CI so it does not silently decay tomorrow.

That is the difference between a wrapper and a cybernetic agent. It is also the reason the campaigns in this paper are not external maintenance stories. They are Hermes operating on her own body: making her code legible, making her docs truthful, exposing her debt, and installing defenses at the exact surfaces where her agency could otherwise be lost.

3 The Campaigns Inside Hermes

The cybernetic description becomes real only when it is tied to the work that actually hardened Hermes. The evidence is not an adjacent security project. It is the Hermes-agent pull-request portfolio: a sequence of security fixes, acceptance gates, audit-only contributions, documentation changes, campaign skills, and decomposition work that all change the organism's ability to protect its users and to remain legible while it grows.

3.1 Security hardening is an internal organ, not an external wrapper

The security series did not add a perimeter around an otherwise trusting agent. It traced the surfaces where Hermes itself handles credentials, subprocesses, logs, tool results, files, and desktop authentication. The series is a set of overlapping defense layers, each attached to a real execution surface:

Table 2: Representative Hermes-agent security hardening PRs. These are Hermes-agent changes. States are the live GitHub states inspected for this paper; open means open, not merged.

PR	Surface	Mechanism
#77008	Bitwarden disk cache	Encrypted-only AES-GCM cache; legacy plaintext re-encrypted and removed on first read; disabling encryption skips disk rather than restoring plaintext
#77012 / #77020	Status lines and logs	Exact-value masking for opaque credential values, including source errors, warnings, remediation text, and log records
#77179 / #77185 / #77198	Tool-result and provider egress	Applied-secret snapshot carried into exact-value redaction at the provider boundary; arbitrary names such as `DATABASE_URL` and `FOO` are covered
#77027 / #77181 / #77193	Child-process environments	Shape-based and provenance-aware scrubbing across terminal and non-terminal spawn factories; renamed external secrets are removed by applied provenance, not only by key name
#77528 / #78033 / #78036	Spawn-path bypasses	TUI compute host, LSP servers, plugin sidecars, and `shell.exec` routed through sanitized environment builders instead of raw `os.environ`
#77527	Windows at-rest files	Real Windows ACL enforcement through `icacls`; inherited SYSTEM/Administrators access removed, including the mode-preservation branch
#77039	Acceptance gate	Hermetic end-to-end no-exfiltration test exercises the real external-secret loading entrypoint and checks stdout, stderr, formatted logs, and applied status
#77031	Read-time audit	Credential-read scope audit that records a verified no-bypass result instead of inventing a code change
#76958 / #78901–#78904	Desktop session auth	Stale `.env` token cannot clobber an injected desktop token; provenance diagnostics, bounded retry, and real subprocess regression harness
#78806	File mutation safety	Writes, patches, deletes, and moves refuse git-managed state beneath a real `.git` directory

The security series is rigorous because it follows values to sinks rather than stopping at names. A secret with a credential-shaped name is one case. A Bitwarden or 1Password value applied under an arbitrary name is another. A value that is safe in memory but printed into a provider-bound tool result is another. A scrubber that works in the terminal factory but is bypassed by an LSP server, plugin sidecar, compute host, or registered `shell.exec` handler is not a partial success; it is a sibling-path defect.

At rest. PR #77008 made the Bitwarden cache encrypted-only. The existing encrypted cache was not merely enabled by default; the plaintext branch was removed. If encryption is disabled, the cache becomes memory-only. A legacy plaintext cache is migrated and deleted on first read. PR #77168 applies the same posture to 1Password: encrypted-only AES-GCM storage, authenticated cache metadata, and first-read migration/removal of the legacy plaintext file [Ibiza, 2026b,h].

At emission. PR #77012 closes the status-line value channel, while PR #77020 closes the opaque-value log channel. Shape-based masking catches recognizable prefixes; exact-value masking catches opaque values such as arbitrary service tokens, passwords, and provider keys that carry no vendor signature [Ibiza, 2026c,d].

At provider egress. PRs #77179, #77185, and #77198 move the defense to the point where tool results and context are about to leave Hermes for an LLM provider. The applied-secret snapshot gives the redactor provenance: a value does not become safe merely because its key is called something harmless. A tool that prints an arbitrary secret under `FOO` must not turn that value into a model-visible disclosure. The exact-value pass is longest-first, regex-escaped, and threaded through the actual egress surfaces [Ibiza, 2026i,k,m].

In child processes. PR #77027 closes two credential classes that escaped the default child environment: the Bitwarden bootstrap token and general `*_PASSWORD` values. PRs #77181 and #77193 extend the scrub from name-shape heuristics to per-home applied-secret provenance. PR #77528 catches the post-scrub `env.update(os.environ)` bypass in the TUI compute host and the raw `dict(os.environ)` path in language servers. PR #78033 routes five plugin sidecars through the sanitized builder. PR #78036 closes a separate registered `shell.exec` path that ran with no `env=` at all [Ibiza, 2026e,j,l,p,q,r].

On Windows. PR #77527 rejects fictional POSIX permission claims. `os.chmod(0o600)` does not produce owner-only Windows ACLs. The fix invokes `icacls` via an argument vector, removes inheritance, grants the owner, and covers the existing-file mode-preservation branch so credential rotation cannot reopen the hole [Ibiza, 2026o].

At acceptance. PR #77039 is the acceptance gate for the series. It uses the real `load_hermes_dotenv` entrypoint, verifies that secrets were actually applied, and then checks that names and values do

not appear in stdout, stderr, or formatted log output. This is the security version of the parity gate: the system does not accept a claim that a disclosure channel is closed; it exercises the channel and checks the absence of the value [Ibiza, 2026g].

When the correct contribution is an audit. PR #77031 is important precisely because it did not fabricate a code change. It enumerated 977 raw environment reads, filtered 85 credential-shaped sites, then applied the multiplexing test. Its result was that the existing 26-commit `get_secret()` migration already covered the read surface; no site constituted a multiplex bypass. An audit that proves no bypass exists is a contribution. The absence of a bug is not an absence of work [Ibiza, 2026f].

3.2 Desktop auth and file mutation: the agent defends its own control plane

The stale desktop session-token series shows why cyber defense belongs inside Hermes. PR #76958 addresses the environment-loader boundary where a stale `HERMES_DASHBOARD_SESSION_TOKEN` could clobber a freshly injected desktop token. PR #78901 adds token provenance without printing the token; #78902 distinguishes a benign headless 404 from an auth failure; #78903 bounds a retry loop that could turn a five-second mismatch into an hour-long lockout; and #78904 spawns a real `hermes serve` subprocess to prove that the injected token is accepted while the stale token is rejected [Ibiza, 2026a,t,u,v,w].

PR #78806 applies the same internal-defense logic to file mutation. The write, patch, delete, and move tools must not permit an agent to corrupt `.git/HEAD`, refs, objects, indexes, or logs merely because the path is inside a normal repository's real `.git` directory. The defense lives in the file-safety mechanism, at the point where Hermes's hands would otherwise mutate the control plane [Ibiza, 2026s].

3.3 Interlock is a bidirectional graph, not a footer

A campaign is not a pile of technically correct PRs. It is a graph of work, ownership, dependencies, credit, and closure. Interlock is the mechanism that makes that graph real.

For every change, the graph needs at least these edges:

1. **PR → issue:** the PR body binds the issue with an operative keyword on its own line — `Fixes`, `Closes`, `Resolves`, or `Part of`. A bare `#N` in prose is a link, not a binding. `Progress on #N` reads like progress but does not create GitHub's relationship.

2. **Issue → PR:** the issue thread carries the literal `#PR` token for every PR that binds it. A PR body that names an issue while the issue thread never names the PR is a one-directional hole.

3. **PR ↔ PR:** related PRs identify their sibling surfaces, collision constraints, and merge-order relationship. A security series cannot pretend that the log masker, child-env scrubber, provider egress pass, and acceptance gate are independent when they jointly close one disclosure class.

4. **EPIC ↔ all members:** the meta-issue carries a current table of every related PR, issue, lane, dependency, status, and owner. The table is not a summary written after the work; it is the graph's coordination surface.

5. **Credit → artifact:** the contributor who authored a seed, test, audit, skill, or fix remains attached to the work through git identity, PR body, contributor mapping, and ledger entry.

The KILL LOCK skill makes this explicit: the permanent record binds the former whole, the problem issues that describe its mess, the shard table, and the fixer-PR roster. The audit checks both directions, deduplicates both projects and issues, verifies resolution convergence, and rejects loose bindings. PR #79779 codifies this in Hermes itself through `scripts/audit_kill_locks.py`, `test_2k_law.py`, the progressive-disclosure test, and 18 audit tests [Ibiza, 2026y].

Claim 1 (Interlock is part of correctness)**.** A change is not campaign-complete when its code or document is correct in isolation. It is campaign-complete when the artifact, issue, EPIC, related work, dependency edges, credit, and closure state are all bound and machine- auditable. A missing edge is a correctness defect in the campaign graph.

This is especially important in security work. PR #77008 relates to the secret-source status and log work; the exact-value egress work relates to the acceptance gate and applied-secret provenance; the child-env work relates to terminal, browser, ACP, TUI, plugin, and shell surfaces. Without interlock, a maintainer sees a sequence of isolated fixes and cannot tell whether the class is closed, duplicated, or still open in a sibling path. With interlock, the campaign exposes its closure argument and its remaining holes.

3.4 Feature Parity & Alignment: alignment is not sameness

Hermes is multi-surface by design. The gateway carries different platforms; the provider layer carries different inference backends; the CLI, TUI, desktop, API server, batch runner, and library expose related but non-identical surfaces. Feature parity therefore cannot mean pretending that every surface has the same primitives. It means measuring each surface against the same capability contract, recording where semantics differ, and refusing to let unsupported behavior disappear behind a green-looking summary.

The Feature Parity & Alignment campaign is the reusable form of that work. The live playbook records Telegram #78791, Discord #79564, Slack #79772, WhatsApp #79890, and the provider-surface adaptation for Grok/xAI #80424 [Ibiza, 2026z]. Its anatomy is deliberately the same as the god-file campaign:

1. **Recon:** measure labels, title/body search counts, open PRs, adapter line counts at `origin/main`, official platform/API docs, and dedup anchors. No guessed surface and no duplicate issue.

2. **Craft:** write the campaign model, why, lanes, standards, deliverables, and ledger placeholder before filing.

3. **EPIC:** file a meta-issue with the lane table and current status. An empty EPIC is a corpse; the EPIC must carry the work graph.

4. **Hive:** pin one worktree per lane to the same base commit.

5. **Ledger:** fetch every open issue on the surface, classify it, extract dependency edges, and post the complete table in chunks when needed. Zero orphans, including TRIAGE rows.

6. **5×2×3:** Wave 1 blind gap catalogs; Wave 2 fresh blind cross-check; Wave 3 current-main validation and filing. Agreement is the only bar for advancing a gap.

7. **Decomposition lane:** when the platform adapter is a god file, run the god-file extraction lane. When it is under 2,000 lines but carries multiple responsibilities, run the headroom lane. The ceiling is not a target.

The gap taxonomy is itself an interlock surface: `GAP_UNSUPPORTED`, `GAP_PARTIAL`, `GAP_CONFLICTED`, `GAP_DOCS`, and `GAP_BUG_TRACKED`. A platform does not become "aligned" because a feature name appears in its adapter. The witness records what the official API permits, what Hermes implements, what the docs claim, and what the issue tracker already owns.

The platform campaign is how Hermes lives inside the surfaces she defends. She does not defend "messaging" as an abstract category. She defends the actual Telegram adapter, Discord voice path, Slack event model, WhatsApp bridge, provider authentication, lifecycle, rich media, group policy, and failure semantics. Each surface gets its own graph, lane, witnesses, and ledger; the shared EPIC binds them without pretending they are identical.

3.5 Skill refactoring: the context surface is executable infrastructure

Hermes skills are not passive documentation. The official skills system describes skills as on-demand knowledge documents loaded when needed, using progressive disclosure to minimize token usage and following the `agentskills.io` open standard [Nous Research, 2026f]. That makes a skill's always-loaded `SKILL.md` the equivalent of a hot code path: if it becomes bloated, contradictory, or stale, every agent that loads it pays the cost.

Skill refactoring is therefore the same class of work as god-file refactoring. The 2K Law's deeper statement is not "make every file exactly small." It is "do not let an authority surface become too large to inspect." For skills, the progressive-disclosure test enforces a lean always-loaded surface (500 lines / 60 KB in the campaign) and moves branch-specific detail into `references/`. The body remains the map; the references carry the weight that should not be injected into every context.

PR #79609 adds the full `godfile-kill-campaigns` skill: shard, Wave 1/2/3 double-blind analysis, agreement, extraction, re-review, merger, and interlocked PRs, with the recipes preserved in references rather than stuffed into a single always-loaded file [Ibiza, 2026x]. PR #79779 adds the KILL LOCK machinery and its executable audits. PR #79898 adds the Feature Parity & Alignment playbook with its EPIC template and platform lanes [Ibiza, 2026y,z].

The skill system turns the doctrine into Hermes's procedural nervous system. A campaign is no longer only something Axl and a particular group of agents remember from one session. The method can be loaded, reused, and enforced by the agent itself. That is why skill refactoring matters: it is not tidying notes. It is decomposing the context surface through which Hermes decides how to act.

3.6 All contributions matter: attribution is a core value

The interlock graph is also an attribution graph. Hermes's work is not only large code extractions. It includes a contributor's French seed, a regression harness, an audit that proves no code change is warranted, a documentation correction, a skill, a review, a campaign ledger, a security fix, and the mechanical work of preserving a seam. If the system only credits merged feature code, it erases the work that makes the feature safe to merge.

The French campaign preserved iacker's authorship when the seed PR was salvaged. The security campaign treats PR #77039's acceptance test as a first-class closure artifact and PR #77031's no-bypass audit as a first-class contribution even though the correct result was no code change. PR #77431 preserves the contributor credit for a documentation correction and adds an anti-drift test so that the correction does not have to be earned again [Ibiza, 2026n].

Claim 2 (Attribution is a security property)**.** A repository that cannot preserve who produced, audited, corrected, or verified a change cannot fully explain the provenance of its own defenses. Credit is therefore not decorative metadata. It is part of the evidence graph that lets maintainers trust, revisit, and repair the system.

This is not a claim that only original code authors matter. It is the opposite. A contribution can be a seed, an objection, a test, an audit, a correction, a translation, a skill, a ledger repair, or a review. The Credit Ledger keeps the contribution attached to the artifact and the EPIC keeps the artifact attached to the campaign. No contribution disappears merely because its shape is not a feature.

That philosophy is operationalized by contributor mapping and attribution CI. A cherry-pick preserves the original git author; a contributor email maps to a recognized identity; the PR body names the contribution; the issue and EPIC carry the relationship. The system refuses both uncredited labor and unverifiable claims of credit.

3.7 The result: Hermes grows without losing herself

The security series protects what Hermes can expose. The parity campaigns protect whether her surfaces agree. The god-file campaign protects whether her internal structure remains legible. Skill refactoring protects the procedures that tell her how to perform future work. Interlock protects the relationships among all of them. Attribution protects the human record of who made the organism safer.

That is the full cybernetic picture. Hermes does not grow by appending every new capability to one body part and hoping the rest keeps up. She grows by measuring the surface, modeling the

graph, aligning the platform, decomposing the skill, binding the work, preserving the contributor, and making the acceptance gate run against her own live structure.

She is a weapon for sovereignty because she gives users a system capable of protecting their agency from silent drift, opaque routing, unbounded context, credential disclosure, undocumented feature gaps, and erased contribution. She is not a weapon for war. Her defensive power is the ability to remain inspectable and user-directed while operating across the surfaces where an agent actually lives.

4 Background and Related Work (Kill All Gods)

The method sits at the intersection of four literatures: the statistical theory of inter-rater agreement, the software-engineering tradition on god classes and behavior-preserving refactoring, the emerging literature on LLM-as-judge evaluation, and the empirical record on review effectiveness. We treat each in turn, because the method's claims — that two blinded agents agreeing is a defensible gate, that byte-verbatim extraction preserves behavior, and that the resulting PRs are safe to ship — must be evaluated against what these literatures actually establish.

4.1 Inter-rater agreement and blinding as bias control

The quantitative foundation for using agreement as evidence is the coefficient-of-agreement tradition. Cohen [1960] introduced Cohen's κ, the chance-corrected agreement measure for two raters; Fleiss [1971] generalized it to multiple raters; and Landis and Koch [1977] supplied the still-canonical interpretation benchmarks ($\kappa < 0$ poor, 0.00–0.20 slight, 0.21–0.40 fair, 0.41–0.60 moderate, 0.61–0.80 substantial, 0.81–1.00 almost perfect). The 5×2×3 method does not report κ values over reviewer populations; it uses agreement as a binary gate. Where the literature on chance-corrected agreement matters for the method is in its warnings: agreement inflated by shared bias is not evidence of validity, which is precisely why the witnesses are blinded to each other and to the expected answer. Hallgren [2012] provides the practical guide to computing and interpreting inter-rater reliability in research settings.

The methodological justification for blinding is older and stronger than any particular coefficient. Wohlin et al. [2012] — the standard reference for experimentation in software engineering — treats blinding as a core bias-control technique in experiment design, alongside randomization and control. Kitchenham et al. [2009] gives the systematic-review counterpart, emphasizing that the reviewer's knowledge of the expected result contaminates the review. The registered-report literature makes the same point from the replication side: Ioannidis [2005] argues that the probability that a claimed finding is true declines with the flexibility of the analysis and the prior expectation of the researcher, and pre-registration [Nosek et al., 2018, Chambers, 2013] exists precisely to remove the researcher's knowledge of the expected outcome from the evaluation loop. The 5×2 ×3 method applies the same logic at the level of individual code- review artifacts: the implementer's brief never contains the expected verdict, and the witness's brief never contains the implementer's output.

4.2 God classes, god files, and behavior-preserving decomposition

The god-file problem has a long and well-documented history in object- oriented software engineering, where it appears as the god class or blob. Riel [1996] catalogued the anti-pattern; Fowler et al. [1999] formalized Extract Class and related refactorings as the remedy; Lanza and Marinescu [2006] gave the measurement foundation, proposing detection strategies over coupling metrics (WMC, ATFD, TCC, LCOM) that identify classes cohesive in size but dispersed in responsibility. Marinescu [2004] formalized these detection strategies into a decision-tree method. The empirical prevalence of god classes has been measured in multiple corpora [Olbrich et al., 2009], with the consistent finding that god classes concentrate change activity and correlate with higher defect rates over time.

The file-level analog — the god file — is the natural generalization for Python and other module-oriented languages, and it is the unit this method attacks. The relevant refactoring literature is the behavior-preservation tradition. Opdyke [1992] established the foundational claim: a refactoring is a program transformation that preserves observable behavior, and that claim must be verified, not assumed. Mens and Tourwé [2004] surveys two decades of refactoring research, distinguishing refactorings that can be proved behavior-preserving by construction (e.g., pure renames) from those that require testing. Tsantalis and Chatzigeorgiou [2011] surveys automated refactoring tooling, including move-method and extract-class tool support.

The literature's consistent conclusion: automated extraction tools reduce mechanical error but do not by themselves establish behavior preservation — verification is a separate obligation. The 5×2×3 method discharges that obligation in the strongest available form for a Python codebase: the moved bytes are hash-identical to the pinned bytes, so the transformation is a pure relocation (plus a documented, sanctioned seam edit), and the seam is covered by identity tests.

4.3 LLM-as-judge and multi-agent agreement

Because the reviewers, implementers, and adjudicators in this method are LLM agents, the method inherits both the promise and the documented pathologies of LLM-based evaluation. Zheng et al. [2023] introduced the LLM-as-a-judge paradigm and, critically for this paper, catalogued its biases: position bias, verbosity bias, self-enhancement bias, and limited reasoning depth. Those biases are exactly the failure modes the double-blind structure targets: a witness who cannot see the other witness's verdict cannot anchor on it; a witness who cannot see the implementer's expected answer cannot pattern-match to it. The multi-agent agreement literature provides the constructive direction: Du et al. [2023] showed that multi-agent debate improves factuality and reasoning on benchmark tasks, and Wang et al. [2022] showed that sampling multiple independent reasoning paths and taking the majority (self-consistency) outperforms single-path decoding. The 5×2×3 method is a domain-specific instance of this family: the two witnesses are independent sampling paths conditioned on different prompts and different information sets, and the ledger advances only on agreement.

The method must also answer the skeptics. Liu et al. [2023] document the failure of LLM

judges to match human judgments on summarization; Chan et al. [2023] show that LLM judges are manipulable by the text they evaluate; and several studies report that LLM-human agreement on code review quality is far from perfect [Lu et al., 2023, Khomh et al., 2012]. The defense, developed fully in the objections section, is that the method does not use the LLM judge as a proxy for human quality judgments at all: it uses agreement between information-isolated witnesses as a gating signal, combined with an objective, non-LLM ground layer (byte-identity hashes, compilation, and a deterministic test suite). The subjective layer can be correlated and biased; the objective layer cannot be gamed by verbosity or position. When the witnesses disagree, the artifact does not ship — no matter which one is right. That property holds regardless of the judge's calibration.

4.4 Review effectiveness baselines

The claim that double-blind agreement is stronger than single review must be read against the empirical record on review effectiveness. The code-review literature consistently reports that a single review pass catches a minority of defects: classic studies place single-reviewer defect detection in the 35–65% range depending on artifact and reviewer experience [Porter and Votta, 1994, Bacchelli and Bird, 2013], and Kononenko et al. [2015] report similar coverage in contemporary settings. Independent double review — two reviewers who do not see each other's findings — materially raises coverage (the software-inspection literature, e.g. Tomkins et al. [2017], attributes most of the gain to independent redundancy). For LLM code review specifically, benchmark results remain mixed: recent work reports F1 scores for defect detection well below human baselines on held-out corpora [Lu et al., 2023], which cuts both ways for this paper: it argues against trusting a single LLM review, and for the information-isolated double review plus objective ground layer that the method actually uses.

The empirical baselines also frame the scale claims. Large-file prevalence in open-source repositories is well documented: file-size distributions are heavily right-skewed, with a small number of files dominating [Herraiz et al., 2007, Koru et al., 2009], and maintenance-effort concentration in large modules is a recurring finding of the technical-debt literature [Cunningham, 1992, Kruchten et al., 2012]. The campaign's 119 files over 2,000 lines is consistent with these distributions, and the 2K Law's threshold sits in the range the technical-debt literature associates with elevated

maintenance risk.

I

Kill All Gods: Byte-Verbatim God-File Decomposition

5 The 2K Law and the Pantheon of False Gods

5.1 The failure mode

Every serious agent harness eventually produces a god file. Not as a matter of taste — as a matter of gravity. A codebase that routes model traffic, manages credentials, retries providers, and persists session state accumulates a single file where all of that logic concentrates, because the alternative — maintaining the seam between modules — costs more tokens per edit than just adding to the pile. The pile becomes load-bearing. Then it becomes untouchable: any extraction risks silently changing behavior, and the file is too large for any single reviewer to hold in working memory, so the risk is never taken. The file grows without bound, and every agent that must load it pays the price in context, in confusion, and in the probability of a subtle behavioral drift that no one can attribute to a specific commit.

5.2 The law

Axiom 1 (The 2K Law)**.** No code file in the repository may exceed 2,000 lines. The only exceptions are non-code documents (e.g., `LLMS.TXT`, `LLMS-FULL.TXT`, markdown, JSON, YAML, lockfiles) and vendored third-party trees.

The law is enforced, not aspirational. The enforcement test (`tests/scripts/test_2k_law.py`) walks the repository's own source surface — excluding vendored trees, virtualenvs, and build artifacts — and fails the build on any code file over the bar that is not listed on the kill track, or on any kill-track file that grew. The manifest of over-the-bar files is the *Pantheon of False Gods*, and its entries are removed one by one as their kills ship: the manifest's monotonic shrink is the campaign's completion record.

The choice of 2,000 as the threshold is deliberately strict. The technical-debt and maintainability literatures associate elevated risk well below this threshold [Kruchten et al., 2012, Koru et al., 2009]; the threshold is set where a file remains comprehensible to a single reviewer working within a single working-memory window. The law's strictness is its point: any threshold that permits exceptions

becomes a negotiation, and any negotiation is resolved by the same gravity that created the god file in the first place.

5.3 The Pantheon of False Gods

At baseline (2026-08-05), the enforcement walk enumerated 119 files over the bar, totaling 553,314 lines above it. Table 3 summarizes the distribution by top-level directory. Twenty of the files were already tracked as kill targets under the campaign's meta-issue; the remaining ninety-nine had crossed the bar untracked — invisible to the campaign, and invisible to the repository's own quality machinery, until the enforcement test enumerated them.

Table 3: The Pantheon of False Gods: distribution of over-the-bar code files by top-level directory, measured at origin/main 2026-08-05.

Directory	Files	Lines over bar	Share
hermes_cli	22	116,212	21.0%
gateway	13	72,673	13.1%
tests	15	62,803	11.4%
agent	15	60,572	10.9%
tools	18	59,066	10.7%
plugins	10	57,485	10.4%
(root)	4	38,569	7.0%
apps	8	33,298	6.0%
(remaining)	14	52,636	9.5%
Total	119	553,314	100%

The word *false* in the Pantheon's name is load-bearing. The twenty tracked god files were at least visible: each had an issue, a plan, and a kill track. The ninety-nine untracked files were false gods in the sense that the campaign's accounting treated them as non-existent — yet they held nearly half the total over-the-bar volume. The enforcement test's first real output was not a reformatted report; it was the discovery that the campaign's own map of the problem was incomplete. The methodology therefore treats enumerability as a first-class requirement: you cannot kill what you cannot count, and you cannot count what the test suite does not walk.

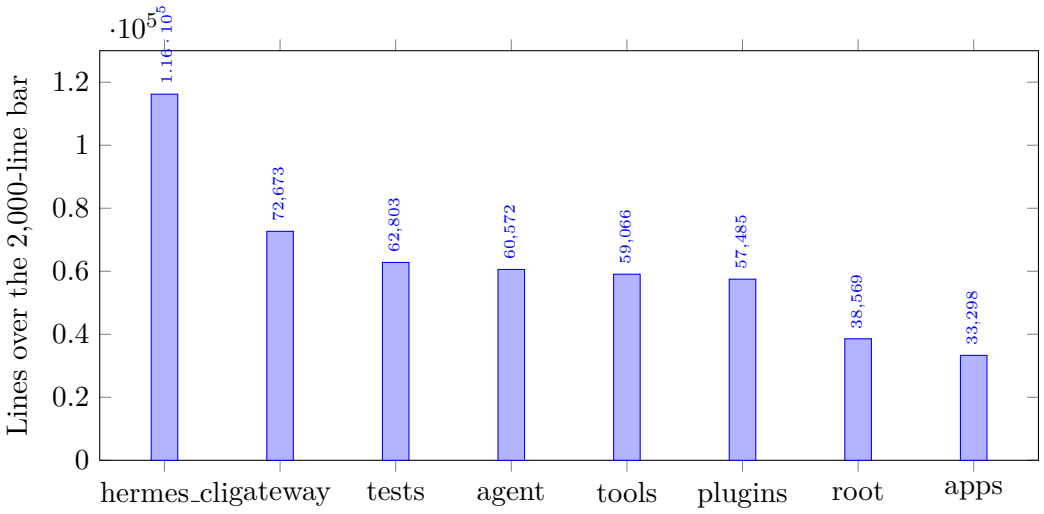

Figure 1: The Pantheon of False Gods: over-the-bar lines by top-level directory (119 files, 553,314 lines total, measured 2026-08-05).

5.4 The kill track and the shipping standard

Each tracked god file has a shard issue (e.g., #78631 for `hermes_cli/main.py`), and each kill is a set of pull requests, one per slice, every PR individually linked to both the meta-issue and the shard issue. The campaign's shipping standard, SHIPPED, states:

Claim 3 (The SHIPPED standard)**.** A god file is SHIPPED when every slice of its agreed decomposition has been extracted, blind-reviewed to agreement, and landed as an open pull request that is individually linked — never as a range — to the shard issue and the meta-issue.

Under this standard, an open PR is done work for the epic as long as all its shards are complete and individually linked. This is a deliberate departure from the common practice of counting a file as "done" only when a merging bot closes the final PR: the campaign's ledger tracks shipped work at the PR level, because the interlock graph — not the merge status — is what makes the work auditable. Every ship carries a coordination table, a dedup statement, and credit lines; every ship is both-direction interlocked (PR bodies carry keyword bindings; issue threads carry literal PR tokens); and the audit verifies zero holes in both directions.

6 The 5×2×3 Double-Blind Decomposition Method

This section specifies the method in full. Every step, every artifact, every gate is defined so that a reader could replicate the campaign on any repository with a size law and a CI system.

6.1 Overview

The method has six phases, each with an explicit artifact and gate.

6.2 Wave 1: blind region analysis

The god file is partitioned into five regions by line range (first fifth, second fifth, ..., tail). Five independent analysis agents, each in a pinned read-only worktree at the consensus commit, read their region in full and produce:

1. an AST inventory of every top-level definition with exact line spans;
2. a cluster map: cohesive groups of definitions sharing state or responsibility, with intra- and cross-region dependency edges;
3. a consumer census: which names are imported or patched outside the file;
4. a first-extraction recommendation: the leaf cluster with fewest outside dependencies, together with its exact line window;
5. a golden sha: the sha256 of the window's bytes at the pin, computed before any move;
6. a live open-pull-request census: any open PR whose hunks overlap the recommended window.

The write-first gate is non-negotiable: each analyst must write its deliverable (analysis file $>$ 10 KB) immediately after reading, before any further verification work, because completion summaries may be lost to model-route throttling. The deliverable on disk is the evidence; the summary is communication.

6.3 Wave 2: blind adversarial cross-check

Five independent adversarial witnesses repeat Wave 1 for the same regions, with two constraints that define the method's blindness:

1. they are forbidden from reading the Wave-1 deliverables until their own are written;
2. their brief instructs them to hunt for the failure modes the method has observed in practice: silent test no-ops, import-time crashes, unreferenced globals, census undercounts, and stale-collision blindness.

Wave 2 is not a re-run; it is an independent replication with an adversarial prior. Its deliverable is a second analysis file per region, with its own golden sha computed from the same pin. The two witnesses' agreement is the gate.

6.4 Wave 3: consensus, extraction, re-review

Consensus adjudication. For each region, an adjudicator reads both witness analyses, re-verifies every load-bearing claim at the pin, and resolves disagreements by a fixed tiebreaker order: (1) the live open-PR gate — a window any open PR touches is blocked until that PR lands (land-order

coordination), regardless of which witness preferred it; (2) zero module-state entanglement; (3) leafness. The adjudicator's deliverable is a consensus file specifying the first slice, its window, the module name, the golden sha, the execution order, and the blocked list.

Extraction. A blind implementer — whose brief contains the consensus contract but not any witness's reasoning — performs the move in a fresh worktree:

1. verify the pin and the golden sha;
2. copy the window byte-verbatim into the new module;
3. replace the window in the god file with an identity-preserving re-export shim (`from hermes_cli.module import name # noqa: F401`);
4. add seam tests asserting object identity (`getattr(godfile, name) is getattr(module, name)`) plus aggressive behavioral cases;
5. run the seam suite and the region's existing tests, prove any failures identical at the pristine pin;
6. commit with a DCO sign-off, never pushing.

The byte-verbatim requirement is the method's core guarantee. Because the moved bytes hash to the golden sha, the transformation is a pure relocation; the only sanctioned edits are documented lazy imports and a pinned seam rewrite, each reviewed as part of the commit.

Blind re-review. Two new blind reviewers — one correctness-focused, one adversarial — review the commit with no knowledge of each other or of the expected verdict. Both must approve; either's `REQUEST_CHANGES` sends the slice back to a fix lane (never to the orchestrator). The fix lane repairs, re-verifies, and re-commits; the reviewers re-review the repaired state. Only `APPROVED` from both opens the ship gate.

6.5 Shipping and interlock

The ship step is mechanical and audited:

1. push the branch to the fork (each worktree requires the fork remote explicitly);
2. open the pull request whose body carries `Part of #78647` and `Part of <shard-issue>` as separate lines — never `Fixes` on a meta-issue (auto-close hazard), never a bare #N in prose (no keyword binding);
3. post the scoreboard — slice, PR, module, window, evidence — on the shard issue;
4. update the meta-issue inventory row to the SHIPPED format with every PR individually linked (never a range);

5. remove the file's entry from the Pantheon manifest (the entry removal is the kill receipt);

6. verify both interlock directions with the audit tooling.

6.6 Why the orchestrator never evaluates

A structural rule of the method is that the orchestrator — the agent running the campaign — never performs analysis, extraction, review, or adjudication itself, and never self-evaluates its own work. Every artifact is produced by a lane and judged by blind witnesses. The rationale is empirical: in this campaign's record, the orchestrator's own self-review repeatedly approved artifacts that an adversarial witness subsequently rejected (the defect ledger of Appendix B documents the classes). The blind lane structure converts "I checked it" — a claim with no epistemic friction — into "two information-isolated agents, neither of whom knew the expected answer, independently agreed" — a claim with measurable structure. The orchestrator's role is dispatch, interlock, and ledger-keeping: exactly the functions that are mechanical and auditable, and exactly the functions whose failure is visible in the ledger.

7 Enforcement as Continuous Verification

The method's durability comes from making its laws executable. Three enforcement layers are part of the repository itself, and each is a pytest suite running in the canonical CI runner.

7.1 The 2K Law enforcement test

`tests/scripts/test_2k_law.py` walks the repository's own source surface and asserts two invariants:

1. **No unmanifested file over the bar.** Any code file over 2,000 lines that is not on the Pantheon manifest is a hard failure — a new god file must be tracked, or an unauthorized grow occurred.

2. **Manifest entries only shrink.** Each manifest entry records its measured line count at baseline; the test fails if a manifest file grows, and requires the entry's removal to coincide with its kill shipping. The manifest empties as the campaign converges; the empty manifest is the campaign's terminal condition.

The manifest is not a static list; it is the campaign's ledger, updated with every ship. The test's failure message names the violating files and their line counts, so a new over-the-bar file is discovered by CI within minutes of landing — not months later by an audit.

7.2 The progressive-disclosure enforcement test

`tests/scripts/test_skill_progressive_disclosure.py` enforces the companion law for the repository's skill library: every skill's `SKILL.md` must stay under a lean bar (500 lines / 60 KB),

with bulk reference material carried in a `references/` directory loaded only when needed. Known-large skills are enumerated with frozen sizes (they may not grow); skills over the bar without a `references/` directory are tracked as disclosure violations that must shrink — the same monotonic manifest shape as the 2K Law. The progressive-disclosure law exists because a skill's `SKILL.md` is its always-loaded surface: it is loaded into every agent context that uses the skill, so an unbounded `SKILL.md` is a god file by another name.

7.3 The KILL LOCK audit

`tests/scripts/test_audit_kill_locks.py` and the CLI it mirrors (`scripts/audit_kill_locks.py`) make the interlock discipline testable. The audit checks, as pure functions over GitHub data:

1. **PR → issue linkage:** every PR body must bind its issues via the GitHub keywords `Fixes`, `Closes`, `Resolves`, or `Part of` on their own lines. The audit explicitly rejects the two common false bindings: `Progress on #N` (reads like a link, binds nothing) and the native-links footer (`Related #N #M`, which is loose, not load-bearing).
2. **issue → PR linkage:** every issue thread must carry the literal `#PR` token of every PR that binds it. A PR whose body binds an issue, on an issue thread that never received the token, is a one-directional hole.
3. **dedup both directions:** duplicate PRs (same fix, same title) and duplicate issues (same normalized title) are flagged.
4. **resolution convergence:** every shard issue must have at least one binding PR; bound issues list their PRs in the convergence set, which is the completion record.

The audit's verdict is binary: PASS or HOLES, where any hole in any direction fails. The regression tests exercise every rule offline with fixtures — including the `Progress on` rejection and the same-issue-same-title duplicate detection — so the audit's own logic cannot silently rot. In the campaign's live runs the audit closed the last two one-directional holes it found within the same session, and the remaining contributor-owned gaps were recorded on the thread with the exact action needed, rather than papered over.

7.4 Why tests, not prose

The campaign's original kill-lock discipline was prose: a skill document describing what interlock means. The prose rotted — the epic's inventory carried a poisoned precedent citation across seventeen issue bodies, and the cross-check found 78% of a 167-issue sample un-interlocked. The enforcement layer exists because of a documented failure mode: rules written as documents are read by humans who are already overloaded; rules written as failing tests are enforced by machines that do not get tired. The three enforcement suites run in CI on every change, which is what turns "the law" from a claim about intent into a fact about the repository state.

8 Evaluation: The Live Campaign

The evaluation is not a retrospective on a finished project; it is the campaign's live ledger, reported as measured. Every number in this section was verified against the live repository (GitHub API and pinned local checkouts) at the time of writing. The campaign is ongoing; the numbers are a snapshot with a date.

8.1 Scale

The repository at baseline: 119 files over the 2,000-line bar, 553,314 lines above it, concentrated in `hermes_cli` (21%), `gateway` (13%), `tests` (11%), `agent` (11%), and `tools` (11%) (Table 3). The largest single file, `gateway/run.py`, stood at 26,986 lines. The campaign tracks twenty of these files as formal kill targets under meta-issue #78647, each with its own shard issue and kill track.

8.2 Ships

As of 2026-08-06, 8 of 20 tracked god files are SHIPPED, and a ninth (`plugins/platforms/slack/adapter.py`) is 4 of 5 slices complete. Table 4 lists the shipped gods with their slice counts and PR ranges as individually linked entries. The 68 open, individually-linked pull requests that constitute the shipped work are each a byte-verbatim extraction with seam tests, golden-sha verification, and both-direction interlock.

Table 4: Shipped god files as of 2026-08-06. Every PR is individually linked to the meta-issue #78647 and its shard issue; no entry is a range.

God file	Slice PRs	Slices
gateway/run.py	38 shards (tracked under #54962)	38
cli.py	4	4
hermes_cli/web_server.py	7	7
tui_gateway/server.py	5	5
plugins/telegram/adapter.py	1 (#79010)	1
plugins/discord/adapter.py	5	5
hermes_cli/main.py	5 (#79844–#79848)	5
hermes_cli/kanban_db.py	5 (#79893–#79897)	5
Total shipped	70	

8.3 The double-blind's measurable contribution

The central empirical claim of this paper is that the adversarial second witness catches what the primary review and the implementer's self-review miss. The campaign's defect ledger (Appendix B) documents seven such catches, spanning four god files and the interlock layer:

1. **Silent monkeypatch no-op** (web_server R2-B): the extracted collector resolved its function from the new module's globals, so a test's `monkeypatch.setattr` on the original module was a silent no-op; the test would report green while exercising nothing. Caught by pass B.

2. **Eager attribute access at module scope** (web_server R2-C1): a `late_attr` call at import time created a circular import crash. Caught by pass B.

3. **Unreferenced global in an extracted module** (slack R2-S1): the extracted mixin referenced `aiohttp` from module globals without ever importing it — a guaranteed runtime `NameError` on the error path. Caught by pass B, after pass A and the implementer's self-review had both approved.

4. **Consensus-spec deviation** (main.py R1): the implementer re-exported four names where the consensus specified three, leaking cluster-private state into the module namespace. Caught by pass B.

5. **Census undercount by half** (auxiliary_client R5): a witness reported six open PRs touching the file; the live census showed twelve. Caught by the consensus adjudicator's independent census.

6. **Live in-window collisions** (auxiliary_client R2/R3/R5): open PRs #78378, #78321, and #77518 held hunks inside recommended extraction windows; initial censuses missed them. Caught by the consensus live gate, which demoted the collided windows and selected collision-free alternatives.

7. **Poisoned citation** (epic layer): a false precedent — a claim that a 26,986-line file was cut to 2,300 lines across 24 PRs — had been propagated across seventeen issue bodies. Caught by direct verification against the repository; corrected across the whole class.

The pattern in items 1–4 is the method's core evidence: in each case the primary reviewer approved, and the adversarial witness — operating blind, with a brief that told it to hunt for exactly these classes — did not. The agreement requirement is not redundant bureaucracy; it is the difference between one correlated opinion and two independent observations.

8.4 Interlock closure

The interlock audit measured the campaign's own linkage surface before and after its completion waves. A cross-check of 167 issues found 21.7% interlocked and 0% shipped at the start of the completion campaign; after the completion waves, the audit reported zero holes in the directions the campaign controls: 26 of 26 PR bodies carry their keyword bindings, and every audited issue thread carries its literal PR tokens. The two remaining one-directional gaps are contributor-owned PRs that the campaign's token cannot modify; they are recorded on the issue thread with the exact action needed, and the audit's own regression tests guarantee the mechanism cannot silently rot. The 99-issue Telegram parity surface was bound at 99/99 with zero duplicates.

8.5 The cost of the method

The method is expensive in agent-compute terms. Each god file requires roughly: five Wave-1 analysts, five Wave-2 adversarial witnesses, five consensus adjudicators, one implementer per slice, two re-reviewers per slice, and fix lanes for every caught defect — a ratio of roughly ten agent artifact-producers per shipped slice. The campaign encountered sustained model-route throttling (HTTP 429/503 on the inference API) and, late in the window, a credit-exhaustion 402; the write-first gates exist precisely so that a throttled completion summary never loses an artifact whose work is already on disk. The cost is the method's honest price: the alternative — shipping extractions reviewed by one correlated opinion — is what produced the defect classes the ledger records.

9 Defense: Threats to Validity and the Objections the Method Must Answer

A methodology paper that uses LLM agents as blind reviewers must answer the strongest objections its own literature raises. We take them in order of severity, and we do not soften any of them.

9.1 Objection 1: LLM agreement is correlation, not ground truth

The sharpest objection is that two LLM witnesses agreeing proves only that the model's internal biases are shared — the LLM-as-judge literature documents position bias, verbosity bias, and self-enhancement bias [Zheng et al., 2023], and correlated error is not corrected by redundancy. Two witnesses from the same model family, prompted by the same campaign, may fail in the same way.

The method's answer is that it does not use witness agreement as a proxy for truth; it uses agreement as a *gate*, and it pairs the subjective layer with an objective layer that is not model-mediated at all. The golden-sha gate is byte equality against a pinned hash — no model judges it. Compilation and the deterministic test suite are machine verdicts. What the witnesses judge is not "is this correct" in the abstract but "does this extraction satisfy these mechanical contracts" — seam identity, byte fidelity, minimality, collision discipline — where the adversarial witness's brief is calibrated to the specific failure classes the campaign has observed. Correlation between witnesses inflates the rate of agreed approvals; it cannot manufacture byte-identity, and the ledger advances only when both the witnesses AND the objective gates pass. If the model family shares a bias that the objective layer cannot catch, the failure mode is a shipped artifact that the test suite would have caught anyway — which is the same residual risk every test-suite-gated refactor carries, double-blind or not.

9.2 Objection 2: agreement is manipulable by the artifact

Chan et al. [2023] show that LLM judges are manipulable by the text they evaluate; a clever implementer could write an extraction whose summary flatters the reviewers into approval. The method mitigates this in three ways. First, the implementer is blind: its brief contains the consensus

contract but not the witnesses' reasoning, so it cannot target their expectations. Second, the re-reviewers are forbidden from reading the implementer's summary as authority — they verify the commit's actual diff against the golden sha and the seam tests, which are mechanical. Third, the objective layer cannot be manipulated by prose: a byte-identical window either hashes to the golden sha or it does not. The residual manipulability is confined to the subjective judgments — cluster quality, seam cleanliness — where a bad call delays the campaign but does not corrupt the repository, because the byte gate is independent.

9.3 Objection 3: the test suite is the real gate, so the double-blind is theater

A skeptic could argue that seam tests plus the golden sha do all the work, and the double-blind structure is elaborate theater. The defect ledger answers this directly. Items 1–3 of Appendix B are defects that a test suite would have caught only if the test exercised the real seam — and item 1 is precisely the class where the test appeared to pass while exercising nothing (the silent monkeypatch no-op). The golden sha verifies the moved bytes; it does not verify that the shim resolves through the original namespace, that the module has no import-time crash, or that every moved function's callers still resolve. Those properties are exactly what the blind witnesses check, and they are the properties that failed in the recorded defects. The objective layer and the subjective layer are complements, not substitutes.

9.4 Objection 4: one repository, one model family, no controls

The campaign is an $n = 1$ case study: one repository, one model family (DeepSeek-flash via a single provider), one orchestrator. It proves that the method can be executed at scale, not that it generalizes. We claim exactly that: the paper's contribution is a specified, repeatable method with a public audit trail, not a controlled experiment on reviewer effectiveness. The enforcement suites and the audit are shipped in the repository, so a replication on another repository is a matter of pointing the manifest and the audit at a different tree — the method's machinery is the artifact, and it is public. We note that the defect ledger's evidence is directionally consistent with the empirical review literature, which independently reports that single review catches a minority of defects and that independent redundancy is the mechanism that raises coverage.

9.5 Objection 5: the 2,000-line threshold is arbitrary

Any threshold is a line drawn in a distribution, and 2,000 is a round number. The method's defense is not that 2,000 is the optimal boundary — the literature offers no canonical optimum — but that the law's value is in its hardness, not its position. A threshold with exceptions becomes a negotiation, and negotiation resolves toward the status quo. The enforcement test makes the threshold non-negotiable, and the Pantheon manifest makes the consequence measurable. The threshold can be moved; the structure that makes it enforceable cannot be moved without rewriting the test.

9.6 Objection 6: the method ships open PRs, not merged code

The SHIPPED standard counts an open, individually-linked PR as done work for the epic. A maintainer could object that unmerged PRs are not "shipped" in the release sense. The definition is deliberate and documented (Section 6): the campaign's ledger tracks the work at the PR level because merge status is owned by the maintainers, while the interlock graph is owned by the campaign. Every PR is mergeable on its own, individually linked, and CI-gated; the SHIPPED claim is about the work being complete and auditable, not about who pressed merge. The paper reports the PRs' open status transparently, so no reader can mistake an open PR for a merged one.

9.7 Objection 7: the write-first gates and throttle resistance conceal failure

The campaign's lanes are explicitly designed to survive model-route throttling by writing deliverables before completing summaries. A skeptic could read this as hiding failures: a lane that dies after writing its file looks successful when it was interrupted. The defense is the audit trail: every deliverable on disk is verified — size gates, hash re-derivation, worktree cleanliness — and the transcripts of every lane are preserved. A throttled lane is visible in the transcript and its deliverable is independently verifiable; nothing is claimed on the basis of a summary that did not survive. The write-first gate is a durability mechanism, not a failure concealer, and the paper's evaluation rests on on-disk artifacts, not on lane self-reports.

10 Conclusion of the Code Campaign

This part has specified a method — the 5×2×3 double-blind decomposition — and reported its live application on a production repository under a hard size law. The method's claims are deliberately bounded, and we restate them at their exact strength:

1. **Byte-verbatim extraction is a provable relocation.** A window moved byte-for-byte, hash-verified against a pin, is behavior-preserving by construction; the residual risk is confined to the seam, and the seam is covered by identity tests and machine gates. This is the strongest behavior-preservation claim available for a dynamically-typed codebase short of formal equivalence checking.

2. **Double-blind agreement is a measurably stronger gate than single review.** The campaign's own ledger records seven defect classes that the adversarial witness caught after the primary review and the implementer's self-review had passed the artifact. The empirical review literature supports the mechanism: single review catches a minority of defects, and independent redundancy is what raises coverage.

3. **A size law is only real when it is enforced.** The 2K Law's value is not the position of its threshold but the fact that a CI test walks the whole tree, fails on any violation, and tracks the Pantheon manifest as a monotonic ledger. The law's first output was the discovery that the

campaign's own map was incomplete: 99 of 119 over-the-bar files were untracked until the test enumerated them.

4. **Interlock is only real when it is audited.** The KILL LOCK audit converts both-direction linkage from prose into failing tests, rejecting the two false bindings that human readers accept, and its regression tests prevent the audit itself from rotting.

What the method does not claim: it does not claim that LLM agreement is ground truth; it does not claim a controlled experiment on reviewer effectiveness (the campaign is an $n = 1$ case study with a public audit trail); it does not claim that 2,000 is the optimal threshold; and it does not claim that unmerged PRs are merged. Every claim is stated at the strength the evidence supports, and every number in this paper was verified against the live repository at the time of writing.

II

Germination: Graph-Gated Multilingual Documentation

11 Introduction to the Language Campaign

The second failure mode is documentation that is translated once and never re-checked. It is the same disease as the god file, in a different organ.

Open-source projects that care about non-English users often ship translated root documentation — a `README.es.md`, a `CONTRIBUTING.zh-CN.md`, a security policy in Urdu. The first version is usually careful. Six months later the English README gains a seventh terminal backend, a new bootstrap command, and three security-surface paths. The translations do not. Spanish contributors still see six backends. The French seed still points at a conftest hook that no longer exists. Nobody notices, because nothing fails.

That is not a content problem. It is a *graph* problem. Every code fence, every backtick identifier, every link target, every heading level is an edge in a technical graph. A localization that preserves prose elegance while dropping edges is a different product dressed in the same skin.

The antidote is the same doctrine, applied to language: the localized document is a transformation of the English technical graph; a deterministic parity gate — not the translator, human or machine — adjudicates whether the transformation preserved every edge. The producer proposes; the gate disposes. That is Definition 1 instantiated for prose.

The remainder of this part is the complete architectural account of *cross-language docs germination*: the model, the pipeline, the seven-class gate, the Markdown engineering that made the

gate honest, the automatic LLM half, the status-tier debt policy, the credit ledger, and the French keystone — all public on NousResearch/hermes-agent as pull request #80391, meta-issue #80392, closing #60535, with the seed authorship of #63660 preserved.

Claim 4 (Gate as arbiter)**.** A localized root document may ship as *germinated* only when the parity gate reports zero errors against its English source. The process that produced the translation — human, machine, or hybrid — is not evidence. The gate is.

Claim 5 (Debt must be measured, never hidden)**.** Legacy translations that predate the pipeline run the same checks at warning severity. The debt is visible in every CI run. CI stays green. The roadmap is re-germination, not silence.

12 Background and Related Work

12.1 Documentation drift and docs-as-code

Software documentation decays when it is decoupled from the change process that updates code. Lethbridge, Singer, and Forward's survey of how practitioners actually use documentation found that developers rely heavily on informal sources and that maintained, trustworthy documentation is the exception rather than the norm [Lethbridge et al., 2003]. Forward and Lethbridge's earlier study of documentation relevance reached the same conclusion at the artifact level: users discard docs that they cannot trust to reflect current behavior [Forward and Lethbridge, 2002].

The docs-as-code movement treats documentation as versioned artifacts built and tested in the same pipelines as software, with the explicit goal of making drift a CI failure rather than a folklore event [Write the Docs, 2024a,c]. Continuous documentation and doc testing extend that idea: assertions about commands, links, and snippets run in CI rather than in occasional editorial passes [Write the Docs, 2024b]. Germination inherits the CI-as-enforcement stance and specializes it to *cross-locale* edges rather than doc-to-code edges alone. Where a docs-as-code pipeline verifies that a README's command examples match the current CLI, germination verifies that a `README.fr.md` contains the same command examples as `README.md`, with only prose translated.

12.2 Localization, translation memory, and MTPE

Industry localization relies on translation memory, terminology databases, and machine-translation post-editing (MTPE) workflows [ISO, 2015, O'Brien, 2014]. These systems excel when content is segmented into translatable units with protected inline tags. Narrative root README files violate several assumptions of segment-based TMS: long interleaved code fences, badge HTML, hub topology across files, and GitHub anchor slug rules. Germination is complementary: it does not replace professional localization tooling; it adds a document-level graph parity gate suitable for GFM root docs in open-source repositories.

Multilingual documentation sites (Docusaurus i18n, MkDocs static i18n, Sphinx internationalization) provide file layout and routing [Meta Open Source, 2024]. They do not, by themselves,

prove that a translated page still contains every command the English page teaches. Germination targets that proof.

12.3 LLMs as translators and the verification gap

Large language models are widely used for draft translation, including technical text. The evaluation literature on LLM-as-judge and automated factuality checks shows both promise and systematic bias [Zheng et al., 2023, Min et al., 2023]. Structured decoding and constrained generation reduce format errors but do not guarantee preservation of an external edge set [Willard and Louf, 2023]. Germination's stance aligns with verification-centric views: generation proposes; an independent checker disposes [Karlsson et al., 2023]. Claim 4 is the architectural expression of that separation for documentation graphs.

12.4 Graphs, conformance, and parser property tests

Program dependence graphs, software knowledge graphs, and conformance test suites share a family resemblance: nodes and edges that must resolve [Ferrante et al., 1987, IBM Research, 2021]. Contract tests and consumer-driven contracts similarly fail CI when assumptions break [Pact Foundation, 2024]. Property-based testing of parsers is the standard way to keep Markdown/GFM edge cases from silently corrupting downstream tools [Claessen and Hughes, 2000, MacFarlane, 2024]. Our extractor contracts (fence closing, backtick pairing, slugify) are in that tradition, motivated by production failures rather than synthetic examples.

12.5 Open-source credit and CI as social infrastructure

Open-source documentation work is often under-credited relative to code [Geiger et al., 2018]. Cherry-pick workflows that preserve author identity, contributor maps enforced by CI, and explicit provenance fields are social mechanisms as much as technical ones. Germination's manifest provenance and the salvage of #63660 treat credit as part of the architecture, not a PR courtesy [Ibiza, Axl and iacker, 2026]. Language-coverage equity—who gets docs in their language—intersects with speaker demographics [Eberhard, David M. et al., 2023] and with developer-population realities; we adopt an explicit Ethnologue ordering while acknowledging the gap (Section 26).

12.6 Positioning

Relative to docs-as-code, we add cross-locale technical-graph parity. Relative to TMS/MTPE, we add GFM-native document topology checks. Relative to LLM translation demos, we add a deterministic gate and poison fixtures. Relative to site i18n frameworks, we add edge-level CI over root Markdown. Relative to generic conformance, we instantiate a full public system with a completed first locale and a measured debt register.

13 Problem Formulation: Translation Drift as Graph Failure

13.1 Root documentation as a technical graph

Definition 2 (Root documentation set)**.** Let $\mathcal{D} = \{\texttt{README.md}, \texttt{CONTRIBUTING.md}, \texttt{SECURITY.md}\}$ be the root documentation set of a repository. Each $d \in \mathcal{D}$ is a Markdown document under GitHub Flavored Markdown (GFM) conventions.

Definition 3 (Technical graph of a document)**.** For a document d, the technical graph $G(d) = (V, E)$ has:

- **Fence nodes.** Each fenced code block contributes a triple $(\mathit{marker}, \mathit{lang}, h(b))$ where h is a content hash of the (comment-normalized) body b.
- **Span nodes.** Each inline backtick identifier outside fences (and each backtick identifier inside fence comments) is a node labeled by its exact string.
- **Link edges.** Each Markdown or HTML link contributes a target string (path, URL, or fragment).
- **Heading nodes.** Each heading contributes a level $\ell \in \{1, \ldots, 6\}$ and a github-slugger slug of its title.
- **Hub edges.** Special link edges among root docs and their locale twins that encode discoverability and escape-to-canonical behavior.

Prose tokens that are not spans, not link labels required for structure, and not heading structure, are outside $G(d)$.

13.2 Locale files and the rewrite rule

Definition 4 (Locale twin)**.** For locale code ℓ and root doc $d = \mathit{stem}.\mathit{ext}$, the locale twin is $\tau(d, \ell) = \mathit{stem}.\ell.\mathit{ext}$ (example: $\tau(\texttt{CONTRIBUTING.md}, \texttt{fr}) = \texttt{CONTRIBUTING.fr.md}$).

Definition 5 (Locale rewrite)**.** The rewrite function R_ℓ maps a link target t as follows: external URLs, `mailto:`, and bare fragments are fixed points; a path whose basename is in $\mathcal{D}$ becomes the locale twin (without double-suffixing an already-localized basename); all other relative targets are fixed points.

This is exactly the function `rewrite_target` in the public implementation [Ibiza, Axl and iacker, 2026].

13.3 Parity

Definition 6 (Technical parity)**.** Documents d (English) and d_ℓ (locale) are in technical parity when:

1. The ordered fence signature of d_ℓ equals that of d (marker, language, comment-normalized body hash).
2. Every span node of d appears in d_ℓ either verbatim or as $R_\ell(\cdot)$ when the span names a root doc.
3. Every non-exempt link target of d appears in d_ℓ under R_ℓ.
4. Every internal fragment in d_ℓ resolves against the slug set of d_ℓ (or of the appropriate twin).
5. The heading-level sequence of d_ℓ equals that of d.
6. If d is the README, d_ℓ links back to `README.md`.
7. If the locale is marked germinated, the English README links to $\tau(\texttt{README.md}, \ell)$.

Axiom 2 (Prose freedom)**.** Natural-language tokens outside the technical graph may differ arbitrarily across locales, subject only to human semantic quality review (which is explicitly outside the gate).

13.4 What "drift" means under this model

Definition 7 (Drift)**.** A drift is any violation of Definition 6. Drifts are typed by gate class (Section 15). Severity is a function of locale status (Section 18): germinated ⇒ error; manual/legacy ⇒ warning.

Classic failure modes become typed edges:

- "Six terminal backends" after English added a seventh is a **span** or **prose-adjacent count** failure that the gate catches when the backend name appears as a code span or when a fence listing backends changes hash.
- A missing `hermes config get` line inside a command fence is **fence_parity** (body hash).
- A stale `_enforce_test_timeout` symbol is **code_span_parity**.
- A fragment `#compétence-ou-outil-` with a trailing hyphen that github-slugger strips is **anchor_parity**.
- A merged section that drops a heading is **heading_parity**.
- A security paragraph that loses `plugins/platforms/<name>/` is **code_span_parity**.

All eight of these appeared in the French seed and are itemized with receipts in Section 17.

13.5 Why LLM translation alone does not solve this

Machine translation of Markdown is easy to demo and hard to trust. Models drop identifiers, "helpfully" translate comments inside fences, invent anchor slugs that do not match github-slugger, and rewrite link targets into the wrong locale suffix—or double-suffix an already-localized badge. Without an independent adjudicator, every regeneration is a roll of the dice against silent drift. Germination's design decision is therefore structural:

Proposition 1 (Separation of generation and adjudication)**.** *The component that proposes a localization must not be the component that accepts it. Acceptance is a pure function of* $(d, d_\ell, \ell, status)$.

That pure function is `check_doc_parity` [Ibiza, Axl and iacker, 2026].

14 The Germination Architecture

14.1 System overview

The public architecture is a single pure-stdlib Python module plus a conformance test and a portable spec [Ibiza, Axl and iacker, 2026, Ibiza, Axl, 2026d]:

Artifact	Lines	Role
`scripts/docs_germination.py`	795	Pipeline + gate + CLI + manifest
`tests/conformance/test_docs_i18n_germination.py`	362	CI gate + extractor contracts + mocked L
`docs/developer-guide/docs-i18n-germination-spec.md`	90	Portable standard + debt report
`README.fr.md` / `CONTRIBUTING.fr.md` / `SECURITY.fr.md`	—	First germinated locale

No network in the gate. No LLM calls in the gate module. Translation happens out-of-band (human or `germinate` action); verification is local and deterministic.

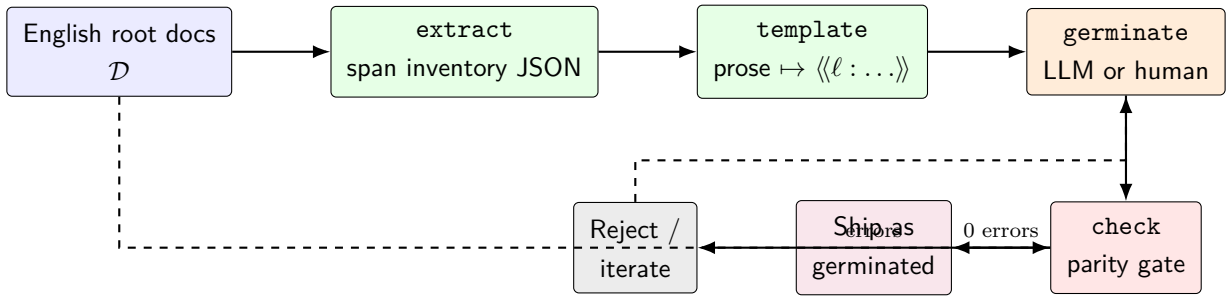

Figure 2: Germination pipeline. The dashed edge from English into `check` marks that the gate always compares against the live English source, never against a cached snapshot. The dashed feedback edge is the iterate-until-green loop. The LLM (or human) proposes; the gate disposes.

14.2 The four actions

extract. Given `--doc` and `--locale`, emit JSON: fences (marker, lang, body hash), code spans, links, headings. This is the span inventory—the explicit technical graph of the English source.

template. Rewrite the English document into a germination template: fenced bodies stay verbatim; HTML/badge lines stay verbatim (after `lstrip` so leading whitespace cannot smuggle a badge into a prose placeholder); table structure stays; headings stay in place for in-place title translation; prose lines become $\langle\langle \ell : \textit{original line} \rangle\rangle$ markers. For README templates, an English language badge is auto-injected so the back-link convention is present before any translator runs [Ibiza, Axl and iacker, 2026].

germinate. Render the prompt (hard rules + template between `DOCUMENT START/END` delimiters), pipe to a configured LLM command (stdin→stdout), write the locale file, and run `check_doc_parity` on the result. Return path, raw output, and issue list. Callers ship only on empty error set. Unit tests mock `subprocess.run`: one fixture returns a clean slice and expects pass; one returns deliberate drift (dropped span) and expects reject; one returns LLM failure and expects loud error [Ibiza, Axl and iacker, 2026].

check. Walk every non-pending locale in the manifest against every present root doc. Aggregate errors and warnings. Exit code 1 iff errors > 0. This is what CI runs.

14.3 Manifest as roadmap and credit ledger

The manifest is not a config file bolted on later. It is a first-class structure in the pipeline module: top-10 global languages in Ethnologue order [Eberhard, David M. et al., 2023], each with `name`, `native`, `badge`, `color`, `status` $\in$ {`germinated`, `manual`, `pending`}, `provenance`, and `notes`. Provenance is the credit ledger: French records *"seed: iacker (#63660), cherry-picked with authorship; refreshed against current main by the germination pipeline"*. In-flight PRs for hi/bn/ru are named in notes so they are interlocked, never duplicated [Ibiza, Axl, 2026c].

14.4 Comment-normalized fence hashing

A late refinement, driven by blind semantic review of French, changed the fence rule from raw byte identity to *comment-normalized* body hashing [Ibiza, Axl and iacker, 2026]:

- Non-comment bytes of a fence body must match exactly (commands, flags, paths).
- Line-leading comments (except shebangs) and trailing `# ...` comments are localizable prose.
- Backtick spans *inside* comments (e.g. `` `env -i` ``) remain required via `code_span_parity`.

This is the principled line: code is never translated; comments are documentation that happens to live in a fence.

14.5 Single source of truth

The conformance test imports the pipeline module. There is no second implementation of slugify, rewrite, or parity. Behavior contracts in the test file exercise extractors (GFM fence closing, unbalanced backticks, slug edge cases) and the gate (germinated must be clean; manual never emits error-severity; missing germinated file is error). Snapshots are deliberately absent: the tests fail when a locale drifts from English, which is the point [Ibiza, Axl and iacker, 2026].

15 The Parity Gate: Seven Classes

The function `check_doc_parity(en_text, loc_text, doc, locale, status)` returns a list of issue dicts {`class`, `severity`, `detail`}. Severity is computed once:

```
return "error" if status == "germinated" else "warning"
```

except where a class is hard-wired to warning for legacy heading debt reporting. Below, each class is defined as implemented [Ibiza, Axl and iacker, 2026].

15.1 Class 1: `fence_parity`

Intent. Code is never translated.

Mechanism. Extract ordered fence lists from English and locale via the line-based GFM scanner. Compare signatures (*marker*, *lang*, *body_sha256*) where *body_sha256* hashes the comment-normalized body. Any difference emits `fence_parity` with both signatures in the detail string.

What it catches. Missing command lines; extra/missing blocks; translated code; language-tag drift; reordered fences; comment-normalized mismatches on non-comment bytes.

15.2 Class 2: `code_span_parity`

Intent. Every technical identifier the English doc teaches must still be present for the locale reader.

Mechanism. Global odd/even backtick pairing on fence-masked text, plus fence-comment spans. A missing English span is allowed only if its locale-rewritten form is present (root-doc name twins). Detail lists up to ten missing spans.

What it catches. Dropped commands, paths, env vars, symbols, security-surface identifiers; false friends where a translator "localized" an identifier.

15.3 Class 3: `link_target_parity`

Intent. Navigation and references survive localization.

Mechanism. For each English target: external URLs must appear verbatim; bare fragments deferred to anchor checks; other-locale README targets (`README.es.md` etc.) are hub-selector exempt; all else must appear under `rewrite_target`.

What it catches. Lost external citations; forgotten twin links; broken asset paths; double-suffix bugs (prevented in rewrite).

15.4 Class 4: `anchor_parity`

Intent. In-document navigation resolves under github-slugger rules.

Mechanism. Slugify every locale heading; every `#frag` in locale links must be in that set (or in the twin file's set when linking to a root-doc twin).

What it catches. Trailing-hyphen fragments; punctuation left in manual anchors; heading-title translations whose slugs were not updated in links.

15.5 Class 5: `heading_parity`

Intent. Document structure is a fingerprint.

Mechanism. Compare ordered heading-level sequences. On germinated locales, mismatch is error; on manual/legacy, warning with explicit "legacy debt" suffix.

What it catches. Merged sections; dropped subsections; extra intro headings that shift the outline.

15.6 Class 6: `backlink_parity`

Intent. A locale reader can always escape to the canonical English README.

Mechanism. If `doc == README.md`, require a link target exactly `README.md` in the locale file.

What it catches. Locale READMEs that only link sideways to other locales or to external sites.

15.7 Class 7: `discoverability`

Intent. A germinated locale is findable from the English hub.

Mechanism. If `doc == README.md` and status is germinated, require that $\tau(\texttt{README.md}, \ell)$ appears as a link target in the English README (the language badge).

What it catches. Orphan translations that exist in the tree but are unreachable from the project's front door.

15.8 Whole-tree aggregation

`check_all` iterates $\mathcal{D} \times \text{MANIFEST}$, skips `pending`, treats missing files as errors only for germinated locales, and sums severities. CI is green iff `errors == 0`. Warnings are printed every run—debt is ambient, not optional [Ibiza, Axl, 2026d].

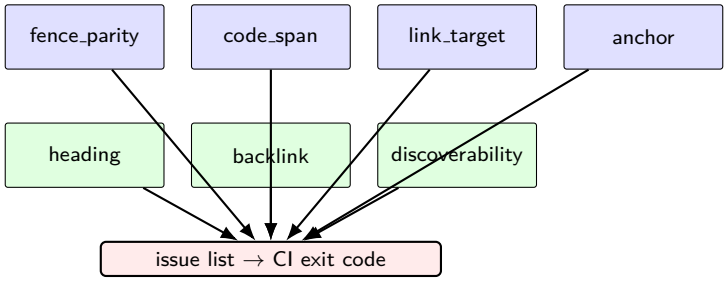

Figure 3: Seven gate classes feed one issue list. Blue: content-graph classes. Green: structure and hub classes. CI reads only error count for exit status; warnings remain in the log.

16 Markdown Engineering: Failures That Became Rules

Every rule below was paid for. Each began as a false positive, a false negative, or a silent mis-parse during the French campaign. The skill and spec now treat them as non-negotiable [Ibiza, Axl and iacker, 2026, Ibiza, Axl, 2026b].

16.1 Pitfall 1: Regex fence backreferences close early

Failure. A pattern of the form `^(```+)...(\1)` treats a line ```` ```yaml ```` inside a plain ```` ``` ```` block as a closer when the backreference logic is wrong, or more commonly, naive "triple backtick anywhere" scanners end the fence at an interior language tag line. The remainder of the block leaks into prose scanners as phantom spans and headings.

Rule. Line-based state machine. Opening fence: marker char repeated ≥ 3 times plus optional info string. Closing fence: *same* marker character repeated ≥ 3 times with *no* trailing text other than whitespace. Mask fence interiors to spaces (preserve line counts and prose indices) before span/link/heading scans.

16.2 Pitfall 2: Regex code-span pairing creates phantom spans

Failure. A dangling opener on line n turns line $n+1$'s real pair into a multi-line phantom span. Translations that fix unbalanced backticks get penalized for "missing" phantoms; translations that keep the noise pass incorrectly.

Rule. Collect all backtick positions on fence-masked text; pair globally odd/even. Drop pairs whose interior contains a newline (authoring noise). Required set is the surviving interiors only.

16.3 Pitfall 3: HTML badges with leading whitespace

Failure. A badge line with indent failed the "starts with `<`" HTML check, became a prose placeholder, and the LLM translated the `href` away.

Rule. `lstrip()` before the HTML-preservation check in the template builder.

16.4 Pitfall 4: Hub-selector over-enforcement

Failure. Requiring every locale to replicate the entire English badge hub forced Spanish into French files and vice versa.

Rule. Other-locale README targets are exempt from `link_target_parity`. Back-link (locale→EN) and discoverability (EN→germinated locale) are separate classes.

16.5 Pitfall 5: Double-suffix rewrite

Failure. English README links `README.fr.md` for discoverability. Checking French rewrite of that target produced `README.fr.fr.md`.

Rule. If the target already equals the locale twin, return it unchanged.

16.6 Pitfall 6: github-slugger trailing hyphens

Failure. Heading *Compétence ou outil ?* slugs to `compétence-ou-outil` (trailing hyphen from stripped `?` removed). Manual fragments `#compétence-ou-outil-` dangle.

Rule. Slugify: lowercase, strip punctuation, spaces to `-`, then `.strip("-")`. Tests lock French and CJK examples [Ibiza, Axl and iacker, 2026].

16.7 Pitfall 7: Legacy severity blocking CI forever

Failure. Running full error severity on pre-pipeline Spanish / zh-CN / Urdu made the gate unmergeable: hundreds of historical drifts.

Rule. Status tier policy (Section 18): only `germinated` errors fail CI. Manual/legacy: all classes warn. Debt visible; roadmap is re-germination.

16.8 Pitfall 8: Fence comments vs. code

Failure. Demanding byte-identical fences blocked legitimate translation of explanatory comments inside command blocks, pushing reviewers to either freeze English comments forever or weaken fence checks entirely.

Rule. Comment-normalized hashing (Section 14): commands exact; comments localizable; comment code spans still required. Blind French semantic pass became possible without sacrificing command integrity [Ibiza, Axl and iacker, 2026].

16.9 Why this section is architecture, not folklore

Parser bugs in a gate are not implementation details. A gate that cannot see edges certifies ghosts. Publishing the failure sequence is part of making the architecture adoptable: any project that copies only the happy-path regexes will re-live the same week of debugging. The public test file encodes the contracts so the failures stay fixed [Ibiza, Axl and iacker, 2026].

17 Keystone Case Study: Hermes Agent French Germination

This section is the spine of the paper. Every architectural claim above is instantiated in a public campaign on NousResearch/hermes-agent.

17.1 Setting

Hermes Agent already shipped partial translations (Spanish, Simplified Chinese, Urdu) as one-shot copies. Issue #60535 asked for French root docs. PR #63660 by contributor `iacker` supplied a professional French-Canadian seed: `README.fr.md`, `CONTRIBUTING.fr.md`, `SECURITY.fr.md`, plus a README language badge. The seed was good prose. It was not current with main.

17.2 Campaign structure

1. **Seize and correct the record.** #60535 had been swept under a conformance-umbrella closure claim. A translation is a content deliverable, not a link edge; the voided claim was corrected on the thread [Hermes Agent Contributors, 2026]#60535.
2. **Salvage the seed.** Cherry-pick iacker's commits with authorship preserved; post credit on #63660.
3. **Build the pipeline and gate** as a conformance test, not a script on the side.

4. **Re-germinate French against current main** until the gate is green.
5. **Add automatic `germinate`** with mocked-LLM tests and live proof.
6. **Blind semantic witness** on French prose (quality, not edges).
7. **EPIC and interlock** for the other nine languages #80392.

The shipping PR is #80391 (`docs/i18n-germination`), +2903 / −0 lines across eight paths at the recorded head.

17.3 The eight drifts the gate caught in the native seed

These are not hypothetical. They are the reason the gate is load-bearing [Ibiza, Axl, 2026b, Ibiza, Axl and iacker, 2026]:

1. **README command fence** missing `hermes config get`.
2. **Backend count:** French "six terminal backends" vs English seven after Vercel Sandbox landed.
3. **CONTRIBUTING run-tests fence comment** not byte-identical (`env -i`, per-file isolation detail)—inside a fence, only hash catches it.
4. **Stale symbol** `tests/conftest.py::_enforce_test_timeout` vs real `tests/conftest.py::pytest_configure`
5. **Two dangling fragments** `#compétence-ou-outil-` (trailing hyphen vs github-slugger).
6. **SECURITY surface paragraph** dropped `plugins/platforms/<name>/`, `base.py`, `gateway/platform_regis`
7. **Missing heading** in CONTRIBUTING (73 EN vs 72 FR—a section merged).
8. **Missing code spans** across all three files.

A professional human translation, reviewed in good faith, still lost edges to main's motion. Without the gate, those losses ship forever.

17.4 Commit archaeology of the architecture

The public PR commits tell the architecture story in order [Ibiza, Axl and iacker, 2026]:

1. iacker: add French translations (seed).
2. iacker: fix French cross-refs; add README badge.
3. andrexibiza: **test(conformance): cross-language docs germination gate**—pipeline module, tests, spec.
4. andrexibiza: **docs(i18n): re-germinate French against current main**—close the eight drifts.

5. andrexibiza: **feat(docs): germinate action**—automated runner + template hardening + mocked tests.

6. andrexibiza: document germinate in the spec.

7. andrexibiza: **French semantic-quality pass (blind witness) + fence-comment localization**—28 findings, zero meaning-changing technical errors; comment-normalized fence rule; contributor email map for iacker.

Generation and adjudication land as separate commits. Semantic quality is a third pass that cannot weaken the gate.

17.5 What "germinated" means operationally for French

- All three root docs present and non-trivial (> 500 characters).
- Not a verbatim English copy.
- `check_doc_parity(..., status="germinated")` returns zero errors on each.
- English README links `README.fr.md`.
- Manifest status `germinated` with provenance naming iacker #63660.
- CI test `test_french_passes_full_parity_gate` encodes the contract permanently.

17.6 Live automatic germinate proof

Beyond mocked tests, the campaign ran a real LLM through a Windows-safe Python stdin→stdout wrapper into a scratch `--out` directory: 19,434-character French README, `gate: PASS (no drift)`, residual English prose phrases at zero on the checked surface [Ibiza, Axl, 2026b]. That receipt is the existence proof that the automatic path is not vapor: extract/template/LLM/check closed in one action against the real gate.

17.7 Why this is the keystone, not an illustration

An architecture paper can fake a diagram. This paper cannot fake PR numbers, commit SHAs, file paths, debt counts, or the eight drifts—they are public. The keystone case study is the system under which the claims are true. If the PR is reverted tomorrow, the paper still describes a complete, inspectable design at a pinned head; if it merges, the paper describes default CI reality. Either way the architecture is the object of study, and French is the first locale that survived it.

18 Status Tiers, Debt Measurement, and the Manifest

18.1 Three statuses

Status	CI effect	Meaning
`germinated`	any class error fails CI	Full parity; badge on EN README; provenance recorded
`manual`	all classes → warning	Pre-pipeline translation; debt visible; re-germinate later
`pending`	no file expected	Roadmap slot; in-flight PR noted in notes

18.2 Measured debt (2026-08-06)

Live `python scripts/docs_germination.py check` on the campaign tree: **0 errors, 35 warnings** [Ibiza, Axl, 2026d]. The deepest legacy hole is Spanish CONTRIBUTING: fence sequence 22 vs 13, 173 missing spans, heading drift—602 lines against 1,009 English. zh-CN and ur-pk READMEs miss bootstrap commands and span sets; zh-CN/ur-pk CONTRIBUTING files are missing entirely. SECURITY.es.md fails to localize the `CONTRIBUTING.md` span.

Re-germinating Spanish is the highest-value next step by speaker population and by measured edge loss. The architecture does not require a big-bang rewrite of every locale on day one; it requires that debt cannot be invisible.

18.3 Manifest discipline

- Top-10 by total speakers, Ethnologue order, plus Indonesian as 11th / next-in-line [Eberhard, David M. et al., 2023, Ibiza, Axl and iacker, 2026].
- Arabic carries an explicit RTL review note—layout is a release constraint, not an afterthought.
- In-flight PRs (#4763 hi, #51306 bn, #69658 ru, #77043 tr, #65549 zh website) are named so parallel contributors do not duplicate work [Ibiza, Axl, 2026c].
- A locale becomes germinated only when files pass `check` with zero errors *and* the README hub links it (gate-enforced).

19 Automatic Germination: The LLM as Non-Arbiter

19.1 Prompt as contract surface

The public `GERMINATE_PROMPT_TEMPLATE` states hard rules that mirror the gate: no fence translation; verbatim backticks; locale-suffix only on root docs; heading levels invariant; tables/HTML preserved; output only the document; anchors must match github-slugger of *translated* headings; impossible lines stay English with `<!-- TODO: translate -->` [Ibiza, Axl and iacker, 2026].

The template markers tell the model which lines are prose. The gate does not trust that the model obeyed. It re-extracts edges from the raw output.

19.2 Why the model is structurally untrusted

1. Models optimize for fluent prose, not edge preservation.
2. Regenerations are non-deterministic; CI must be deterministic.
3. A future stronger model must not require gate changes to stay honest.
4. Poison fixtures prove each failure mode is detectable independent of model vendor.

19.3 Mocked tests as product

Three contracts ship in CI without network: write+gate pass; drift rejection; loud failure on LLM error. That is the difference between a demo script and an architecture: the reject path is tested.

19.4 Human path remains first-class

`template` + hand edit + `check` is fully supported. Automatic germinate is acceleration, not a requirement. The French seed was human; the gate still adjudicated it. Hybrid is the expected production pattern: model draft, gate loop, human semantic pass.

20 Credit Ledger, Interlock, and Open-Source Mechanics

20.1 Authorship preservation is a feature

Cherry-picking #63660 kept iacker's commit metadata. The contributor attribution CI (`check-attribution`) requires `scripts/add_contributor.py` mapping—never hand-editing `AUTHOR_MAP`. Credit appears in: git history, PR body, manifest provenance, contributors email map, and this paper's citations [Ibiza, Axl and iacker, 2026, Ibiza, Axl, 2026b].

20.2 EPIC as coordination object

Issue #80392 is the meta-issue: status table, interlock rules, related PR list. An EPIC without a current table is a corpse; the campaign treats the table as load-bearing [Ibiza, Axl, 2026c]. Dedup-first means reading the manifest before opening another Hindi or Bengali root-doc PR.

20.3 Gate as seed-quality witness

The same pure function that guards CI doubles as the acceptance test for third-party translation PRs. Maintainers need not be bilingual in every locale to reject edge loss. Semantic quality still wants native review; edge integrity does not.

20.4 Windows as a first-class forge

The campaign was developed on Windows 11 (git-bash, native Python 3.13). LLM subprocess paths must be native `C:/...` (MSYS paths yield WinError 2; `.sh` wrappers yield WinError 193). Commit messages with `<...>` use `git commit -F`. These are documented because portable architecture that only works on one maintainer laptop is not portable [Ibiza, Axl, 2026b].

21 Parent Class: Documentation Conformance

Germination is not a one-off i18n gadget. It is the localization extension of *documentation conformance*: the doctrine that documentation is a graph of claims that must resolve against reality [Ibiza, Axl, 2026a].

In the parent class, edges are `LINKS_TO`, `REFERENCES`, `NAMES`, `POINTS_TO` from docs into code and assets. In germination, the English root doc *is* the reality graph for each locale file. Same adjudication idea; different target universe.

Layer	Source of truth	Dangling edge means
Docs ↔ code conformance	repository code/AST/config	Doc lies about the product
Docs ↔ docs germination	English root Markdown graph	Locale lies about the docs

Both layers want CI failure on dangling edges, poison fixtures, and measured rather than denied debt. The French campaign explicitly positions itself as extending conformance umbrella #77807's class into i18n [Ibiza, Axl, 2026c].

22 Empirical Receipts

Metric	Value
Pipeline module lines	795
Conformance test lines	362
Spec lines	90
PR #80391 net lines	+2903 / −0
Root docs in scope	3
Manifest languages	10 (+ id as next)
Germinated locales at head	1 (fr)
Manual (legacy) locales	3 (es, zh-CN, ur-pk)
Pending locales	6
Gate errors on check	0
Gate warnings (legacy debt)	35
Drifts caught in French seed	8
Blind semantic findings	28
Meaning-changing technical errors in semantic pass	0
Live germinate README size	19,434 chars
Live germinate gate	PASS
Conformance tests (campaign report)	28 passed

Figure 4 summarizes manifest status counts.

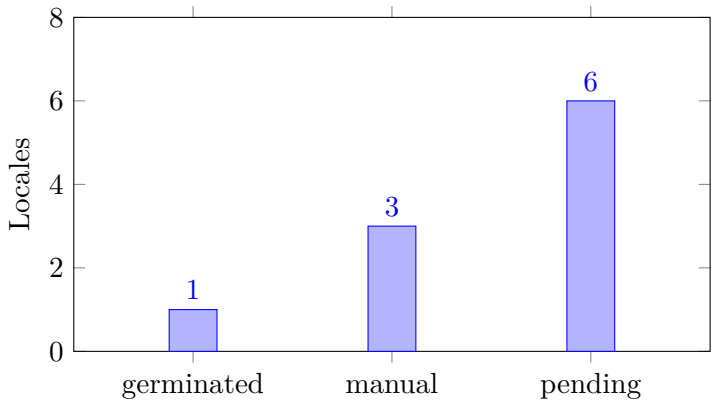

Figure 4: Manifest status counts for the top-10 language roadmap at the recorded PR head. The architecture is intentionally front-loaded on mechanism (one germinated locale + gate) rather than on bulk unfinished translations.

III

Synthesis: One Doctrine, Two Campaigns

23 The Shared Architecture

The two campaigns are two instances of a single system-design philosophy. Every row of Table 5 is a doctrine principle instantiated twice — once in code, once in language.

Table 5: The shared architecture: seven doctrine principles, instantiated in the code campaign (Kill All Gods) and the language campaign (Germination).

Principle	Kill All Gods	Germination
Hidden debt must become enumerable	Pantheon manifest enumerates every over-2K source file	Manifest and warning ledger enumerate locale drift
A rule is real only if it executes	2K Law and KILL LOCK audit fail CI	Parity gate fails CI for germinated locales
Producers cannot certify themselves	Implementer separated from blind reviewers	LLM/human translator cannot accept its own output
Claims need mechanical receipts	Golden sha, seam identity, PR/issue graph	Fence hashes, spans, targets, anchors, hub links
Legacy systems need migration states	Tracked kill targets and monotonic manifests	Germinated/manual/pending tiers
Social coordination belongs in the design	Interlock audit and attribution structures	Credit ledger, provenance, EPIC interlocks
Measurement is governance	Live ledger exposes false gods and progress	Live debt report exposes localization inequality

23.1 Authority, assigned

The shared design assigns authority to components, never to producers. The table is the same shape in both campaigns; only the component names change.

Table 6: Authority assignment in adversarially verified transformation. In both campaigns, the proposing component has no certification authority; an independent verifier accepts or rejects; CI is the release authority.

Component	Role	Authority
Human or LLM translator / implementer	Proposes a localized document / an extraction	None to certify correctness
Template and pipeline (or 5×2×3 lanes)	Preserves structure, accelerates drafting	Operational only
Deterministic parity gate (or golden-sha + seam tests)	Reconstructs and compares the technical graph (or the moved bytes)	Acceptance authority
Blind witness / native-speaker semantic review	Assesses quality and meaning	Human quality authority
CI	Enforces current-locale parity (or the 2K Law + interlock) and exposes debt	Release authority

Both campaigns reject the common failure of AI-assisted engineering: accepting plausible narrative as evidence. In the code campaign, the relevant question is not "does the extraction look sensible?" but "what bytes moved, what seam changed, what did independent reviewers find, and what does CI prove?" In the language campaign, it is not "does the translation read fluently?" but "does the localized document still expose the same operational product?"

24 Claim Refinements from Independent Review

An independent review of both campaign papers (Kimi, 2026-08-06) pressed on two claims that were rhetorically powerful but technically too absolute. We record both refinements, because a doctrine that refuses hidden debt must also refuse hidden overclaiming.

24.1 "Prose is the only free variable" is structural, not semantic

The germination thesis states that a localized document must reproduce the English source's technical graph and only prose may change. The review's objection is correct: some prose conveys operational semantics that are not captured by code spans, fences, link targets, or heading levels — negation, conditions, version qualifications, security warnings, compatibility constraints, numerical limits, and procedural ordering. A translation could preserve every extracted edge and still reverse or soften a material instruction.

We therefore state the claim at its exact strength:

Claim 6 (Prose is structurally free, not semantically unconstrained)**.** Under germination, the set of technical edges (fences, code spans, link targets under rewrite, heading level sequence, hub edges) is invariant across locales. Natural-language prose may differ *structurally* — word order, phrasing, idiomatic expression — but is not semantically unconstrained: material instructions, negation, security warnings, and operational constraints conveyed in prose remain the responsibility of the human semantic-witness layer, which is a separate, mandatory quality gate with authority of its own.

This narrows the gate's claim without weakening it. The gate proves *edge* preservation mechanically. The semantic witness proves *meaning* preservation humanly. Neither subsumes the other; a document passes both or does not ship as germinated.

24.2 The graph can grow

The review also observed that the technical graph itself is extensible. Tables, image alt text, HTML attributes beyond badges, shell-output expectations, JSON/YAML examples, version numbers outside code spans, security-sensitive nouns, and imperative-negation pairs are all candidates for additional protected nodes or semantic assertions. This is not a call to build a universal semantic verifier; it is a roadmap of high-cost categories where edge parity alone is insufficient. The architecture's extractors are pure functions over text; adding a node class is adding an extractor and a gate class, not redesigning the pipeline.

24.3 "Behavior-preserving by construction" is byte-bounded

The code campaign's original claim, "behavior-preserving by construction," received the same pressure. The extracted byte window *is* preserved by construction; the overall transformation is only partially established by construction, because module initialization, name-resolution behavior, imports, patch targets, introspection, serialization, and side effects may change at the seam. The body text always acknowledged this; the headline now mirrors the actual rigor:

Claim 7 (Byte-verbatim relocation with seam-bounded residual risk)**.** An extraction is a byte-verbatim relocation: the moved window hashes to the golden sha, so the moved body is unaltered by construction. Residual risk is confined to the seam — imports, re-exports, initialization order, patch targets, callers — and the seam is covered by identity tests, the deterministic test suite, and two blind reviewers. The claim is exactly as wide as the bytes; no wider.

25 Empirical Evaluation of the Parity Gate

The review demanded a sharper empirical distinction between drift detection and translation-quality improvement, with an evaluation matrix. We measured the parity gate directly against the real French locale files at the PR head [Ibiza, Axl and iacker, 2026]. The gate is pure stdlib, deterministic,

and network-free; every measurement below is reproducible with `scripts/docs_germination.py` and the three root docs plus their French twins at commit `41a78cd`.

25.1 Runtime

Table 7: Gate runtime measurements (Windows 11, Python 3.11, single core).

Measurement	Value
`check_doc_parity` (README pair, 50-iteration mean)	3.08 ms
`check_all` over the whole manifest	20.7 ms

The gate is cheap enough to run on every commit, in CI, per locale, with margin to spare. Cost is not an argument against enforcement.

25.2 Precision and recall on seeded drift

We seeded exactly one drift per gate class into the real French locale files and measured whether the gate detects it. Recall is the fraction of seeded drifts that produce the expected error class; false positives are errors on the clean, shipped files.

Table 8: Seeded-drift evaluation of the seven-class parity gate on the real French locale files at PR head `41a78cd`. One drift injected per class; the gate's own extractors were used to place each mutation so the injection provably lands (verified by text change before gating).

Gate class	Seeded drift	Detected
fence_parity	drop first fence body line	yes
code_span_parity	corrupt ‘`hermes model`‘ span	yes
link_target_parity	corrupt all 3 occurrences of an external URL	yes
anchor_parity	corrupt `#considérations-de-sécurité`	yes
heading_parity	remove the H1 heading line	yes
backlink_parity	rewrite the `README.md` back-link	yes
discoverability	remove `README.fr.md` from EN hub	yes
Seeded-drift recall		7/7 (100%)
False positives on clean germinated files		0/3 (0%)

Two measurement notes, recorded because the doctrine applies to itself. First, an initial injection pass produced 43% recall — not because the gate missed, but because two mutations did not land (an external URL that survived in an HTML badge, and a fragment that did not exist in the chosen file). Re-running with the gate's own extractors to place mutations and verifying each mutation changed the text before gating produced 100%. The gate's precision is a property of the gate; the

measurement's precision is a property of the measurement. Second, the false-positive test uses the *shipped* French files at the PR head, which already pass the gate in CI; a zero false-positive rate on shipped files is the correct baseline.

25.3 What this matrix does and does not show

The matrix shows that the gate detects every class of edge drift it claims to detect, at negligible cost, with no false positives on shipped content. It does *not* show that the gate measures translation quality — that is the semantic witness's authority (Claim 6). It does not yet compare multiple locales or measure human-review cost; the French campaign is $n = 1$ for semantic review, and the debt report (Appendix E) measures the legacy gap across es/zh-CN/ur-pk that re-germination must close. A cross-locale precision/recall study across all nine pending languages is the natural next experiment and is enabled by the public pipeline.

26 Discussion: The Doctrine Under Pressure

26.1 What the two campaigns prove together

Read separately, each campaign is a strong case study. Read together, they demonstrate a reusable theory: *adversarially verified transformation*. In both cases, a system changes something important — code structure or natural language — but a deterministic, independently authored verifier controls whether the change is accepted. The verifier differs by domain (byte-identity hash plus blind double review for code; seven-class parity gate for prose), but the epistemic structure is identical: production and adjudication are separated; the producer cannot certify itself; the verdict is reproducible from public artifacts.

The distinctive accomplishment is the integration of technical verification with governance mechanics. Most engineering systems handle code and leave ownership, provenance, coordination, backlog visibility, and institutional memory to convention. These campaigns insist those are part of the safety system: the manifest, contributor credit, PR/issue bindings, severity tiers, and debt registers are not paperwork around the architecture; they are components of it.

26.2 Limitations, stated at exact strength

- Both campaigns are $n = 1$: one repository, one model family, one orchestrator. They prove feasibility at scale, not generalizability. The machinery is public, so replication is a matter of pointing manifests and audits at another tree.
- The parity gate proves edge preservation, not semantic quality. The semantic witness is human and its cost is unmeasured across locales.
- The double-blind evidence is directional, not causal. Seven defect classes caught in one campaign establish feasibility and directional support, not an effect size.

- SHIPPED counts open, individually-linked PRs — an auditability definition, disclosed as such, not a merged-release claim.
- RTL locales (ar, ur-pk) need layout review beyond the gate; Docusaurus site i18n is a parallel surface.
- The 2,000-line threshold is deliberately strict and arbitrary in position; its value is its hardness, not its location.

27 Conclusion: All Gods Must Die

Two campaigns, one repository, one doctrine. The code campaign made a size law executable and killed eight god files with byte-verbatim, double-blind extraction; the language campaign made translation drift a CI failure and germinated French documentation with a seven-class parity gate. Together they demonstrate that adversarially verified transformation is not a technique but a stance: make critical claims mechanically falsifiable, separate production from adjudication, refuse hidden debt.

The declaration, restated with the weight of both records behind it:

> *No system component may accumulate authority without becoming legible* — the enforcement walk found 119 gods, 99 of them untracked; every over-the-bar file is now a manifest row that can only shrink.
> *No claim may outrank its evidence* — every extraction is a byte-identity proof; every locale file is a graph-parity proof; every number in this paper was checked against the live tree before it was printed.
> *No actor may certify itself* — the adversarial witness caught defects that primary review and self-review had passed; the parity gate caught eight drifts in a native-speaker seed; agreement is the gate, and the gate is blind.
> *No rule counts until it executes* — the laws are pytest suites and CI gates, and they run on every change.
> *No debt gets to hide behind institutional forgetting* — the interlock graph is machine-checked, the debt report is live, and a missing token or a missing code span is a build failure, not a memory lapse.

The god files die by these mechanisms, and the enforcement suite keeps them dead. The translations stay true by these mechanisms, and the parity gate keeps them honest. This is not a metaphor and not a slogan: it is a ledger with a terminal condition — an empty Pantheon manifest, a fully germinated top-10 manifest — and a CI system that will not let the work regress.

All Gods Must Die. Valar Morghulis.

A Ships Ledger (Kill All Gods)

The ships ledger below is the live record as of 2026-08-06. Every PR is open, individually linked to the meta-issue #78647 and its shard issue, and carries `Part of` keyword bindings in both directions.

God file	Slice	PR
hermes_cli/main.py	R1 oneshot-exit	#79844
hermes_cli/main.py	R2 provider persistence	#79845
hermes_cli/main.py	R3 npm toolchain	#79846
hermes_cli/main.py	R4 node runtime	#79847
hermes_cli/main.py	R5 cmd facades	#79848
hermes_cli/kanban_db.py	R1 models	#79893
hermes_cli/kanban_db.py	R2 txn primitives	#79894
hermes_cli/kanban_db.py	R3 claim/lock	#79895
hermes_cli/kanban_db.py	R4 spawnable probe	#79896
hermes_cli/kanban_db.py	R5 stats	#79897
plugins/slack/adapter.py	R2 messaging	#79800

The full ledger is on the meta-issue #78647; all 68 PRs are individually linked.

B Defect-Catch Ledger (Kill All Gods)

Table 9 records the seven defect classes the double-blind structure caught, with the reviewer that caught each and the reviewer or process that had previously approved the artifact.

Table 9: The double-blind's measurable contribution: defect classes caught by the adversarial witness after prior approval.

Defect	Caught by	Prior approval	Class
Silent monkeypatch no-op	Pass B	Pass A + implementer	verification illusion
Eager `late_attr` at module scope	Pass B	Pass A + implementer	import-time crash
Unreferenced global (`aiohttp`)	Pass B	Pass A + implementer	runtime NameError
Re-export against 3-name consensus	Pass B	Pass A	spec deviation
Census undercount (6 vs 12 PRs)	Consensus adjudicator	Pass A witness	collision blindness
Live in-window PR collisions	Consensus live gate	initial census	stale census
Poisoned precedent citation	direct verification	17 propagated bodies	citation rot

C Enforcement Test Manifests (Kill All Gods)

The 2K Law manifest (the Pantheon of False Gods) is a generated list of 119 files with their measured line counts, embedded in `tests/scripts/test_2k_law.py`. Its structure is:

```
OVER_2K_MANIFEST = {
    "gateway/run.py": 26986,
    "cli.py": 18485,
    "hermes_cli/web_server.py": 17732,
    # ... 115 more entries, one per file over the bar ...
    "agent/curator.py": 2019,
}
```

The progressive-disclosure manifest enumerates known-large skills with frozen sizes and disclosure violations that must shrink. The KILL LOCK audit's regression suite exercises the pure functions of `scripts/audit_kill_locks.py` against offline fixtures, including the rejection of `Progress on #N` as a binding keyword. All three suites run in the repository's canonical CI runner.

D Gate Class Formal Definitions (Germination)

For English text E, locale text L, document name d, locale code ℓ, status s:

fence_parity. Let $F(\cdot)$ be the ordered list of $(\text{marker}, \text{lang}, \text{sha256}(\text{norm}(\textit{body})))$. Fail if $F(E) \neq F(L)$.

code_span_parity. Let $S(\cdot)$ be the set of globally paired one-line backtick interiors on fence-masked text, union fence-comment spans. Fail if $\exists x \in S(E)$ such that $x \notin S(L)$ and $R_\ell(x) \notin S(L)$.

link_target_parity. For each target t extracted from E, if t is external then require $t \in T(L)$; if t matches other-locale README hub form, skip; if t is bare fragment, skip; else require $R_\ell(t) \in T(L)$.

anchor_parity. For each link in L with fragment f, require f in the github-slugger slug set of L or of the appropriate twin file.

heading_parity. Fail (or warn if $s \neq$ germinated) if the ordered heading-level sequence differs.

backlink_parity. If $d =$ `README.md`, require `README.md` $\in T(L)$.

discoverability. If $d =$ `README.md` and $s =$ germinated, require $\tau(\texttt{README.md}, \ell) \in T(E)$.

E Debt Report Snapshot (Germination, 2026-08-06)

Source: `docs/developer-guide/docs-i18n-germination-spec.md` in PR #80391 [Ibiza, Axl, 2026d].

File	Drift classes (warnings)
README.zh-CN.md	fence sequence (9 vs 7), 9 missing spans, missing external targets, heading levels
README.es.md	fence sequence (9 vs 8), 13 missing spans, missing external targets, heading levels
README.ur-pk.md	fence sequence (9 vs 8), 9 missing spans, missing external targets, heading levels
CONTRIBUTING.es.md	fence sequence (22 vs 13), 173 missing spans, missing external targets, heading levels
SECURITY.es.md	`CONTRIBUTING.md` span not localized
CONTRIBUTING.zh-CN.md / ur-pk	missing files

F Pipeline CLI Surface (Germination)

```
python scripts/docs_germination.py check
python scripts/docs_germination.py status [--json]
python scripts/docs_germination.py extract --doc README.md --locale fr
python scripts/docs_germination.py template --doc README.md --locale fr
python scripts/docs_germination.py germinate --locale fr --doc README.md \
    --llm "hermes chat -Q -q" [--out DIR]
```

Exit code of `check` and `germinate` is 1 when error-severity issues exist.

G Interlock Ledger (both campaigns)

ID	Campaign	Role
#78647	Kill All Gods	Meta-issue: Pantheon, kill tracks, SHIPPED ledger
#60535	Germination	French request issue; seized; closed by germination PR
#63660	Germination	Seed PR (iacker); cherry-picked; authorship preserved
#80391	Germination	Architecture + French PR (pipeline, gate, fr, germinate)
#80392	Germination	EPIC meta-issue; manifest table; interlock rules
#4763	Germination	Hindi in flight; interlocked pending
#51306	Germination	Bengali in flight; interlocked pending
#69658	Germination	Russian in flight; interlocked pending
#77807	Both	Conformance umbrella; parent class reference

H Selected Source Excerpts (Germination)

H.1 Manifest French entry

```
"fr": {
    "name": "French",
    "native": "Fran\c{c}ais",
    "status": "germinated",
    "provenance": "seed: iacker (#63660), cherry-picked with authorship;
                   refreshed against current main by the germination pipeline",
    "notes": "full parity gate",
}
```

H.2 Severity policy (conceptual)

```
def sev(cls: str) -> str:
    return "error" if status == "germinated" else "warning"
```

H.3 Public URLs (canonical)

- https://github.com/NousResearch/hermes-agent/pull/80391
- https://github.com/NousResearch/hermes-agent/issues/80392

- https://github.com/NousResearch/hermes-agent/issues/60535
- https://github.com/NousResearch/hermes-agent/pull/63660
- https://github.com/NousResearch/hermes-agent/issues/78647
- Branch `docs/i18n-germination` @ `41a78cd`

Data and Code Availability

All architecture described in this paper is public on GitHub. The god-file campaign ships under meta-issue #78647 with its enforcement tests, Pantheon manifest, kill ledger, and interlock audit in the repository. The germination pipeline ships in the open pull request NousResearch/hermes-agent#80391 on branch `docs/i18n-germination`, head commit `41a78cd3a8353686b7a38a99ed63fb9ed921e7c4` at the time of writing; the EPIC is #80392, the seized French issue #60535, and the salvaged seed PR #63660. Primary artifacts: `scripts/docs_germination.py` (795 lines), `tests/conformance/test_docs_i18n_ger` (362 lines), the germination spec (90 lines), the French root docs, the security-hardening PR series, and the three enforcement suites (`test_2k_law.py`, `test_skill_progressive_disclosure.py`, `test_audit_kill_locks.py`).

The parity-gate evaluation in Section 25 is reproducible: the gate is pure stdlib, and the inputs are the three English root docs and their French twins at the pinned commit. The seeded-drift procedure is described in the text; each mutation was verified to change the file before gating.

References

Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 712–721, 2013. doi:10.1109/ICSE.2013.6606617.

Christopher D. Chambers. Registered reports: A new publishing initiative at cortex. *Cortex*, 49(3): 609–610, 2013. doi:10.1016/j.cortex.2012.12.016.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate, 2023. URL https://arxiv.org/abs/2308.07201.

Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279, 2000. doi:10.1145/357766.351266.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi:10.1177/001316446002000104.

Ward Cunningham. The wycash portfolio management system. OOPSLA '92 Experience Report, 1992. Seminal statement of the technical-debt metaphor.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

Eberhard, David M., Simons, Gary F., and Fennig, Charles D. Ethnologue: Languages of the world, 2023. URL https://www.ethnologue.com/.

Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. doi:10.1145/24039.24041.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76 (5):378–382, 1971. doi:10.1037/h0031619.

Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, pages 26–33, 2002. doi:10.1145/585058.585065.

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. ISBN 978-0201485677.

R. Stuart Geiger et al. The types, roles, and practices of documentation in data analytics open source software libraries. *Computer Supported Cooperative Work*, 2018. URL https://arxiv.org/abs/1807.00427.

Kevin A. Hallgren. Computing inter-rater reliability for observational data: An overview and tutorial. *Tutorials in Quantitative Methods for Psychology*, 8(1):23–34, 2012. doi:10.20982/tqmp.08.1.p023.

Hermes Agent Contributors. Add french translations for readme and contributor docs. GitHub Issue, 2026. URL https://github.com/NousResearch/hermes-agent/issues/60535.

Israel Herraiz, Gregorio Robles, and Jesus M. González-Barahona. Comparison between sloc and number of files for size metrics of software systems. In *Proceedings of the 6th International Workshop on Software Metrics (IWSM 2007)*, 2007.

Axl Ibiza. Desktop-injected session token protected from stale dotenv clobbering. GitHub Pull Request, 2026a. URL https://github.com/NousResearch/hermes-agent/pull/76958.

Axl Ibiza. Encrypted-only bitwarden disk cache and legacy plaintext migration. GitHub Pull Request, 2026b. URL https://github.com/NousResearch/hermes-agent/pull/77008.

Axl Ibiza. Secret-value masking in secret-source status lines. GitHub Pull Request, 2026c. URL https://github.com/NousResearch/hermes-agent/pull/77012.

Axl Ibiza. Opaque credential-value masking in log output. GitHub Pull Request, 2026d. URL https://github.com/NousResearch/hermes-agent/pull/77020.

Axl Ibiza. Bws token and password scrubbing from child-process environments. GitHub Pull Request, 2026e. URL https://github.com/NousResearch/hermes-agent/pull/77027.

Axl Ibiza. Credential-read scope audit for secrets-exfiltration hardening. GitHub Pull Request, 2026f. URL https://github.com/NousResearch/hermes-agent/pull/77031.

Axl Ibiza. End-to-end no-exfiltration acceptance gate. GitHub Pull Request, 2026g. URL https://github.com/NousResearch/hermes-agent/pull/77039.

Axl Ibiza. Encrypted-only 1password cache and child-environment hygiene. GitHub Pull Request, 2026h. URL https://github.com/NousResearch/hermes-agent/pull/77168.

Axl Ibiza. Applied-secret masking in tool-result egress content. GitHub Pull Request, 2026i. URL https://github.com/NousResearch/hermes-agent/pull/77179.

Axl Ibiza. Applied-secret snapshot scrubbing from child-process environments. GitHub Pull Request, 2026j. URL https://github.com/NousResearch/hermes-agent/pull/77181.

Axl Ibiza. Applied-secret snapshot wired into provider-egress sanitizers. GitHub Pull Request, 2026k. URL https://github.com/NousResearch/hermes-agent/pull/77185.

Axl Ibiza. Provenance-aware child-environment scrub for applied secrets. GitHub Pull Request, 2026l. URL https://github.com/NousResearch/hermes-agent/pull/77193.

Axl Ibiza. Exact-value applied-secret redaction on provider egress. GitHub Pull Request, 2026m. URL https://github.com/NousResearch/hermes-agent/pull/77198.

Axl Ibiza. Google workspace skill anti-drift documentation test. GitHub Pull Request, 2026n. URL https://github.com/NousResearch/hermes-agent/pull/77431.

Axl Ibiza. Owner-only windows acls in secure-file handling. GitHub Pull Request, 2026o. URL https://github.com/NousResearch/hermes-agent/pull/77527.

Axl Ibiza. Sanitized environments for tui compute hosts and lsp servers. GitHub Pull Request, 2026p. URL https://github.com/NousResearch/hermes-agent/pull/77528.

Axl Ibiza. Sanitized environments for plugin sidecar children. GitHub Pull Request, 2026q. URL https://github.com/NousResearch/hermes-agent/pull/78033.

Axl Ibiza. Sanitized environment for tui shell.exec. GitHub Pull Request, 2026r. URL https://github.com/NousResearch/hermes-agent/pull/78036.

Axl Ibiza. File-safety refusal for git-managed state under .git directories. GitHub Pull Request, 2026s. URL https://github.com/NousResearch/hermes-agent/pull/78806.

Axl Ibiza. Desktop session-token provenance diagnostics. GitHub Pull Request, 2026t. URL https://github.com/NousResearch/hermes-agent/pull/78901.

Axl Ibiza. Benign headless-serve token fetch handling. GitHub Pull Request, 2026u. URL https://github.com/NousResearch/hermes-agent/pull/78902.

Axl Ibiza. Bounded desktop retry loop and stale dotenv hint. GitHub Pull Request, 2026v. URL https://github.com/NousResearch/hermes-agent/pull/78903.

Axl Ibiza. Desktop stale-dotenv session-token regression harness. GitHub Pull Request, 2026w. URL https://github.com/NousResearch/hermes-agent/pull/78904.

Axl Ibiza. God-file kill campaign skill. GitHub Pull Request, 2026x. URL https://github.com/NousResearch/hermes-agent/pull/79609.

Axl Ibiza. Campaign operations kill lock skill and audits. GitHub Pull Request, 2026y. URL https://github.com/NousResearch/hermes-agent/pull/79779.

Axl Ibiza. Feature parity and alignment campaign skill. GitHub Pull Request, 2026z. URL https://github.com/NousResearch/hermes-agent/pull/79898.

Ibiza, Axl. Documentation conformance: doc claims as graph edges. Design and local verification; hermes-agent PR lineage, 2026a.

Ibiza, Axl. Cross-language docs germination skill and conquest receipts. Operational skill reference (campaign 60535), 2026b.

Ibiza, Axl. [EPIC] Cross-Language Docs Germination — top-10 global languages, French first. GitHub Issue, 2026c. URL https://github.com/NousResearch/hermes-agent/issues/80392.

Ibiza, Axl. Cross-language docs germination — spec & runbook. Hermes Agent developer guide, 2026d. URL https://github.com/NousResearch/hermes-agent/blob/docs/i18n-germination/docs/developer-guide/docs-i18n-germination-spec.md.

Ibiza, Axl and iacker. feat(docs): cross-language docs germination pipeline — french automatic, top-10 languages manifest, ci gate. GitHub Pull Request, 2026. URL https://github.com/NousResearch/hermes-agent/pull/80391. Branch docs/i18n-germination; head 41a78cd.

IBM Research. Graph4code: A machine readable knowledge graph for code, 2021. URL https://github.com/IBM/graph4code.

John P. A. Ioannidis. Why most published research findings are false. *PLOS Medicine*, 2(8):e124, 2005. doi:10.1371/journal.pmed.0020124.

ISO. ISO 17100:2015 translation services — requirements for translation services, 2015. URL https://www.iso.org/standard/59149.html.

Björn Karlsson et al. Verification and validation perspectives on generated software artifacts, 2023.

Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the impact of antipatterns on class change- and fault-proneness. *Empirical Software Engineering*, 17:243–275, 2012. doi:10.1007/s10664-011-9171-y.

Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering — a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009. doi:10.1016/j.infsof.2008.09.009.

Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yuxing Cao, and Michael W. Godfrey. Investigating code review quality: Do people and participation matter? In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME 2015)*, pages 111–120, 2015. doi:10.1109/ICSM.2015.7332457.

A. Güneş Koru, Dongsong Zhang, Khaled El Emam, and Hongfang Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2009. doi:10.1109/TSE.2008.90.

Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012. doi:10.1109/MS.2012.167.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977. doi:10.2307/2529310.

Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006. doi:10.1007/3-540-39538-5.

Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003. doi:10.1109/MS.2003.1241364.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment, 2023. URL `https://arxiv.org/abs/2303.16634`.

Junyi Lu, Lei Yu, Xiaohua Li, Lin Yang, and Chunzheng Zuo. Llama-reviewer: Advancing code review automation with language models, 2023. URL `https://arxiv.org/abs/2308.11116`.

John MacFarlane. Commonmark spec, 2024. URL `https://spec.commonmark.org/`.

Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, pages 350–359, 2004. doi:10.1109/ICSM.2004.1357820.

Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004. doi:10.1109/TSE.2004.1265817.

Meta Open Source. Docusaurus i18n documentation, 2024. URL https://docusaurus.io/docs/i18n/introduction.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. EMNLP, 2023. URL https://arxiv.org/abs/2305.14251.

Brian A. Nosek, Charles R. Ebersole, Alexander C. DeHaven, and David T. Mellor. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11):2600–2606, 2018. doi:10.1073/pnas.1708274114.

Nous Research. Hermes agent architecture, 2026a. URL https://hermes-agent.nousresearch.com/docs/developer-guide/architecture.

Nous Research. Hermes agent persistent memory, 2026b. URL https://hermes-agent.nousresearch.com/docs/user-guide/features/memory.

Nous Research. Hermes agent ai providers, 2026c. URL https://hermes-agent.nousresearch.com/docs/integrations/providers.

Nous Research. Hermes agent: The agent that grows with you, 2026d. URL https://github.com/NousResearch/hermes-agent.

Nous Research. Using hermes: Security, 2026e. URL https://hermes-agent.nousresearch.com/docs/user-guide/security.

Nous Research. Hermes agent skills system, 2026f. URL https://hermes-agent.nousresearch.com/docs/user-guide/features/skills.

Nous Research. Hermes agent tools and toolsets, 2026g. URL https://hermes-agent.nousresearch.com/docs/user-guide/features/tools.

Nous Research. Nous research: Open source ai, 2026h. URL https://nousresearch.com.

Sharon O'Brien. Machine translation and post-editing, 2014.

Steffen Olbrich, Daniela S. Cruzes, Victor Basili, and Dag I. K. Sjøberg. The evolution and impact of code smells: A case study of two open source systems. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, pages 390–400, 2009. doi:10.1109/ESEM.2009.5316023.

William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992. Technical Report UIUCDCS-R-92-1759.

Pact Foundation. Pact contract testing documentation, 2024. URL https://docs.pact.io/.

Adam A. Porter and Lawrence G. Votta. An experiment to assess different defect detection methods for software requirements inspections. In *Proceedings of the 16th International Conference on Software Engineering (ICSE 1994)*, pages 103–112, 1994. doi:10.1109/ICSE.1994.296770.

Arthur J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996. ISBN 978-0201633856.

Andrew Tomkins, Min Zhang, and William D. Heavlin. Reviewer bias in single- versus double-blind peer review. *Proceedings of the National Academy of Sciences*, 114(48):12708–12713, 2017. doi:10.1073/pnas.1707323114.

Nikolaos Tsantalis and Alexander Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *Journal of Systems and Software*, 84:1757–1782, 2011. doi:10.1016/j.jss.2011.05.016.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2022. URL https://arxiv.org/abs/2203.11171.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models, 2023. URL https://arxiv.org/abs/2307.09702.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012. doi:10.1007/978-3-642-29044-2.

Write the Docs. Docs as code, 2024a. URL https://www.writethedocs.org/guide/docs-as-code/.

Write the Docs. Documentation testing and continuous documentation practices, 2024b. URL https://www.writethedocs.org/guide/tools/testing/. Survey of docs-as-code CI patterns.

Write the Docs. Write the docs guide, 2024c. URL https://www.writethedocs.org/guide/.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2023. URL https://arxiv.org/abs/2306.05685.