Trust Topology: Verification Surfaces as the Unit of Reliability

Michael Rothrock

michael@roth.rocks · https://michael.roth.rocks

Version 1.0 · DOI: 10.5281/zenodo.20292194

Zenodo framework and empirical study · Not peer reviewed

Abstract. *Reliability is not a property of a model; it is a property of a target-aligned verification surface. You can only trust what you can verify.* This paper develops that claim from a 97-day production coding-agent pipeline in which Claude generated plans, designs, code, and tests that were verified by hybrid deterministic/LLM (Gemini) gates. Across 5.5K gate checks, what mattered most for reliability was not "which model judged?" but "which surface did the judge see?"

The central empirical result is a surface-target split. Design-stage review catches broad cross-context failures that narrower code/file-stage gates mostly miss: 88.7% and 95.0% coverage on SYSTEMATIC and OMISSION targets at the design surface, versus 28.5% and 25.3% on the same two targets at the code surface and 32.1% SYSTEMATIC at the file-scoped surface, a 56–70 percentage-point spread under the same production generator/reviewer pipeline. The failures are also less exotic than the usual hallucination story suggests. Among 760 classifiable rejections, 50.8% are systematic-constraint violations and 40.5% are omissions, so 91.3% are predictable structural failures; only 8.7% are incoherent.

The paper organizes these findings around the **verification surface**: the artifact exposed at a pipeline stage and the deterministic checks writable over it. A gate cannot catch failures its artifact does not expose. When a gate underperforms, the engineering lever is typically not a smarter judge but a better surface: expose the design, schema, plan, trace, or intermediate representation that contains the target.

The framework adapts rough-set lower-approximation ideas as an AI-pipeline design primitive, but the contribution is operational: deployed gates, target-relative measurement, cross-stage composition, oracle escalation, and production instrumentation. Precision is reported using a public analyzer plus redacted Zenodo aggregates, with dual-voted Claude/Gemini adjudication and three denominators: substantive precision 0.965, end-to-end analyzer precision 0.774, and conservative all-sample precision 0.714.

For practitioners building agentic systems, the takeaway is simple: do not ask whether a model is reliable in general. Ask which failure is visible at which surface. When a gate fails, change what it sees, not only how it judges.

1 Introduction

Monday morning. A developer opens the merge queue for last night's coding-agent runs: eight feature branches, each produced by the same pipeline. Plan, design, code, build, test. Branch 3: the plan said the new endpoint would validate input, but the design has no validation step. Branch 7: the code calls a helper function that doesn't exist in this codebase. Branch 5: tests pass, but they test the wrong invariant. The function signature changed and the tests were silently updated to match. None of those are model failures in a deep sense; they are pipeline failures, caught by the gates that surround the model before the human reviewer ever sees them.

This paper asks: which of those failures can the gates catch deterministically, and which ones can they only flag with empirical confidence? And underneath that practical question, a structural one: what *can* a gate at a given stage catch at all, and what determines the ceiling?

The standard reliability question "is the model good enough?" admits no decision procedure: for arbitrary Turing-complete tool-using agents, nontrivial semantic correctness properties run into Rice-style undecidability (Rice, 1953; Turing, 1936). The constructive question is whether we can design a pipeline whose intermediate artifacts turn correctness into a structural check. "Is this model trustworthy?" stays hard; "does this stage's artifact have a property only present in failure cases?" becomes a structural question with an estimable, auditably defined answer.

The answer reframes the unit of analysis. **Reliability is not a property of the model but of the *verification surface*:** at each stage, the artifact the stage exposes along with the deterministic checks that can be written over it. One claim organizes everything that follows. The most a zero-false-positive gate can catch is fixed by the artifact, not by how cleverly the gate judges it: precisely those failures whose artifact exposes a property that never appears in good output (§2.4). The direct upshot of this is that the engineering lever is in fact the artifact itself. Refining the representation toward the target can only raise that ceiling; where no property of the artifact separates failures from good cases, no number of gates over it catches anything without false positives. The lens is at once constructive (redesign the artifact to raise the ceiling) and diagnostic: it tells you in advance whether more gates can help. We make this precise in §2; the machinery is rough set theory (Pawlak, 1982) (in its discrete form here; the measure-theoretic restatement is in the Formal Supplement), restated in pipeline terms — deployed gates, target-relative measurement, cross-stage composition, and oracle escalation over production instrumentation. The Formal Supplement contains the full proofs and notation, and every empirical result here can be read without it.

The second half of the paper is the empirical realization of that framework on a software pipeline. The pipeline studied here implements a standard SDLC workflow (plan, design, code, test) with a separate-model reviewer running four mandatory review gates on the plan, design, and code artifacts (`review_plan`, `review_design`, `review_code`, and a file-scoped `codereview`), followed by a deterministic test-result postcondition. This pipeline has been in iterative production use for years; the LLM-augmented form is its current revision. What is novel is the *instrumentation* of the gates and the analysis we apply to them, not the workflow itself. Two findings carry it. First, gates over richer artifacts catch far more of the same defect classes: a 56–70 pp coverage spread between design-stage and code/file-stage gates (§8). Second, the failures themselves fall into a small, surface-predictable taxonomy: 91.3% are omission or systematic-constraint violations, not the wild hallucinations the discourse fixates on (§9).

The thesis, anticipating the empirical results: *in a production coding-agent pipeline, the representation determines the observable failure modes: design-stage surfaces expose broad omission, systematic, and incoherence targets, while code/file-stage surfaces expose narrower local targets.* This is the coding-agent realization of the surface-target alignment diagnostic developed in §2.

1.1 Contributions

- **The verification surface as a design primitive.** §2 casts pipeline reliability in rough-set terms: zero-false-positive recall is bounded by the fraction of failing cases that the chosen representation places in pure-positive buckets; refinement is the constructive lever; surface-target alignment is the diagnostic; no single feasible surface verifies every target. The Formal Supplement's subtraction table credits the rough-set inheritance and isolates the engineering contribution.
- **Headline 1: surface-target alignment at production scale (§8).** The same cross-context defect classes are caught at 88.7% / 95.0% by design-stage review, versus 28.5% / 25.3% by code-stage review and 32.1% SYSTEMATIC by the file-scoped gate: a 56–70 pp spread under the same production generator/reviewer pipeline, consistent with the framework's prediction.
- **Headline 2: a surface-predictable error taxonomy (§9).** Of 760 classifiable rejections, 50.8% systematic / 40.5% omission / 8.7% incoherent: 91.3% predictable structural failures, not wild hallucinations.
- **Reproducible instrumentation, honest precision basis (§6).** Three-denominator precision on dual-voted adjudication (substantive 0.965, end-to-end 0.774, conservative 0.714); cross-model adjudication

(Cohen's κ (Cohen, 1960) = 0.812 rejection / 0.562 acceptance-shadow); and a deliberately-documented dual instrument (private regex extractor vs. the public reproducible analyzer (Rothrock, 2026a)) so other practitioners can run the analysis on their own logs.

1.2 Related work

Spec-time agent verification reduces agent behavior to checkable safety properties before deployment—LTL model checking (Fang, 2026), inspectable orchestration graphs like LangGraph (LangChain, 2026)—and generator-side methods shift quality onto the generator: DSPy compiles prompts (Khattab et al., 2024), structured generation constrains decode-time tokens to a grammar (Willard & Louf, 2023). We instead hold the generator fixed and measure what the deployed surface catches against an evolving production target. LLM-as-judge work studies judges as benchmark evaluators (Gu et al., 2024; Zheng et al., 2023); we use them in two roles that literature does not cover—the production reviewer whose decisions ship artifacts, and cross-voted post-hoc adjudicators of those decisions. The substrate is Claude Code (Anthropic, 2026a; 2026b); our public analyzer (Rothrock, 2026a) extracts gate decisions, error classes, and revision loops from session JSONL without access to the prompt templates that produced them.

A separate strand frames reliability as *coherence*: Sohl-Dickstein's hot-mess theory (Sohl-Dickstein, 2023) holds that more capable agents behave less coherently, and Hägele et al. (Hägele et al., 2026) (with Sohl-Dickstein) operationalize it on benchmarks, measuring incoherence as the variance-driven fraction of failures and finding it grows with reasoning length and task difficulty. We pose the same question at the system level: under deployed review gates the incoherent class is the *smallest* (8.7% of classifiable rejections), with omission and systematic-constraint violations dominating—a thematic counterpoint, not a same-metric replication, since our INCOHERENT label marks an artifact's cross-location self-contradiction (§9) rather than their run-to-run variance.

1.3 Roadmap

The paper develops the framework as a lens, then realizes it on production data. §2 sets up the lens: the capacity mechanism, the verification surface, the surface-target alignment diagnostic, and the no-universal-verifier prediction, with the propositions, refinement, and proofs in the Formal Supplement. §3–§9 realize it on a 97-day production software pipeline, from the pipeline structure and what each estimate measures through precision, coverage, and the two empirical headlines. §10 carries the framework to the medical and language-model companions; §11–§13 discuss, bound, and conclude.

How to read this paper. Practitioners deploying coding-agent pipelines: §3 (the pipeline and its gates), §6–§8 (precision, coverage, and which gate aligns with which failure), and §9 (what the rejected errors actually are). Researchers verifying the framework: §2 (the lens) and §4 (what each estimate measures), then the empirical instantiation in §6–§7, with methods and adjudication detail in Appendices A–B. Reviewers assessing rigor: §5 (methods), §12 (limitations), and the reproducibility manifest in Appendix D.

2 The Verification Surface

The introduction stated the central claim; this section gives the mechanism behind it and then makes it precise. The mechanism is simple. A stage turns each case into an artifact: in coding, perhaps a plan, a design, or a file; imaging: a mask or boundary; a language model: a candidate next token. An artifact may even be a single bit, where the stage expresses some aspect of the input as a binary true/false.

This means many different cases can collapse to the same artifact. A stage cannot distinguish cases it renders identically, so each artifact names a *bucket* of cases. A gate only ever sees the artifact, thus it can only act on whole buckets. If some bucket holds only failing cases, a test that fires on it catches those failures and never a good case; call the bucket *pure-positive* and the test *zero-false-positive* (ZFP). If every bucket that holds a failure also holds a good case, no ZFP test over that artifact can fire at all.

That fixes a ceiling. The most any ZFP deterministic gate can catch, the surface's *capacity*, is the fraction of failing cases that land in pure-positive buckets. Capacity is a property of the representation, not of gate cleverness: even the best judge cannot separate cases the artifact has already merged. Three consequences run through the rest of the paper.

- *Refinement is monotonic.* Emitting a finer artifact can split a mixed bucket into a pure-positive one and can never merge two, so it can only raise the ceiling or leave it unchanged, never lower it. Adding a target-exposing stage is therefore downside-protected: at worst inert, at best the thing that turns an unverifiable surface into a verifiable one. **This is why the representation, not the gate, is the primary engineering lever**.
- *Composition compounds.* Each gate shrinks the set of failures that survive, and the next stage works only on that residue, so a defect must slip past every stage to escape. Reliability is a property of the pipeline's topology, not of any single stage. **This is the second lever**, and it is as safe as the first: adding a target-aligned gate anywhere can only lower the escape rate, never raise it.
- *No separator, no gate.* Where the artifact exposes no property that separates failures from good cases, capacity is zero and no amount of gating repairs it; the only fix is to change the representation.

The rest of this section makes each of these precise: the verification surface as a four-part object, the case space and buckets, the gate families, and the measurement attributes, closing with the surface-target diagnostic and the no-universal-verifier prediction. The full propositions and proofs are in the Formal Supplement.

2.1 What a stage looks like to an engineer

An engineer sees four things at each stage:

1. *The artifact the stage emits*: a representation Y mapping each case to an artifact in some set S (a plan, code, an organ mask, a candidate next token).
2. *The deterministic tests performable on an artifact* within budget—the *capability* set $\mathcal{D}$, the deterministic *predicate class* (a type check, a regex match, a threshold over a scalar score).
3. *The gates actually deployed*—the *deployment* set F^{deploy} (the deployed *gate family*), some deterministic gates from $\mathcal{D}$ plus any stochastic or heuristic gates added (an LLM judge, a learned classifier).
4. *The kind of failure the engineer is trying to catch*, L (incoherent design, hallucinated entity, anatomically impossible segmentation, schema-invalid output).

The artifact, capability set, and deployed gates are observable at runtime; none are model-internal. The target L_α, the cases that truly carry violation type α, is operationally specified before evaluation; a gate fires on an observable artifact property, not on L_α itself. When only failing cases can produce that property (for example, a compile failure on a syntax target), firing certifies membership immediately. When the property

is merely a proxy for the failure, membership is confirmed post hoc, through labels, audits, outcome data, or oracle resolution. Either way the gate need not decide membership in L_α at runtime, yet L_α is what makes purity, capacity, and coverage quantifiable. A *verification surface* at stage k against target α is the tuple $\Sigma_{k,\alpha} := \left(Y_k, \mathcal{D}_k, F_k^{\text{deploy}}, L_\alpha\right)$ of these four things, the unit of reliability for the stage. Swapping the model while holding the four fixed preserves the surface type (though it may shift the bucket distribution); changing the emitted representation changes the surface itself.

2.2 Case space and representations

Definition 1 (Case Space). The case space Ω is the set of cases, each $\tau \in \Omega$ a single case[1] (a patient, a task, an input).[2] Each stage k has a representation $Y_k : \Omega \to S_k$ that maps each case to its artifact in some set S_k (a mask, a plan, a candidate next token). For each violation type α, $L_\alpha \subseteq \Omega$ is the set of cases that truly have violation α. The violation lives in the case space, not in any single stage's representation.

The case unit depends on architecture. In non-revision pipelines (medical imaging is canonical) each case flows once through every stage. In revision pipelines (software development is canonical) each case is a *task* that may produce many artifact-versions through its review cycle. Both architectures yield well-defined Ω, L_α, and the events below.

The violation belongs to the case, not the surface. A patient either has csPCa or doesn't, before any segmentation runs; a task either adds tech debt or doesn't, decided by what the case is, not by what the code looks like at some stage. Artifacts produced by stages are projections of the case: they expose some aspects and hide others, but they are not the case itself.

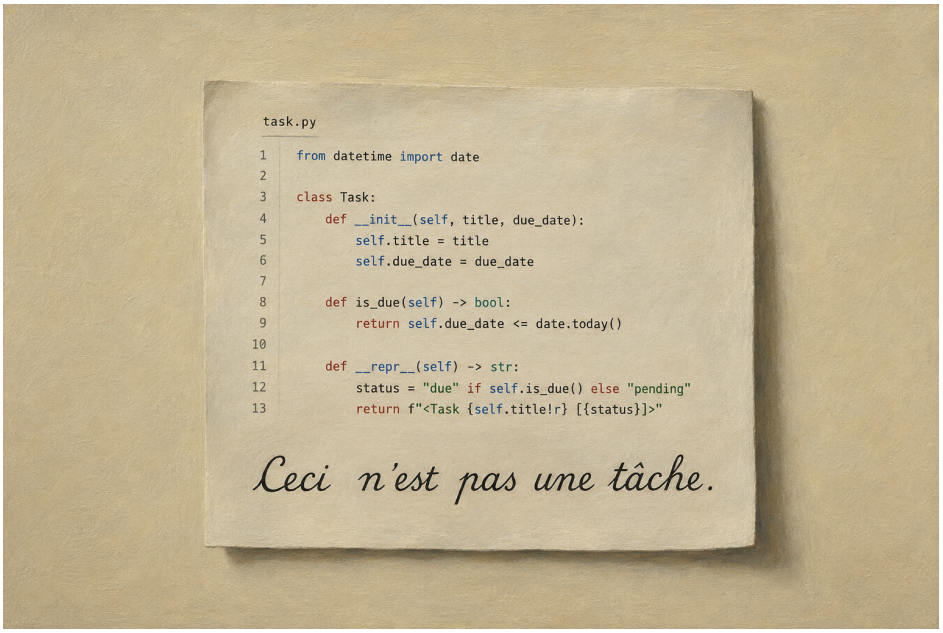

Figure 1: After Magritte's *The Treachery of Images* (1929). The artifact a stage emits is a projection of the case, not the case itself. $Y_{k(\tau)} \in S_k$ is faithful within what stage k can see, but it remains a different object from $\tau \in \Omega$.

A representation partitions Ω into *buckets*:

[1] τ is the case symbol throughout (mnemonic for "task"), reserving ω for the gate-overlap-ratio metric (ω_k / ω_{ij}, the attribute table in §2.4).

[2] For the discrete software pipeline studied here, the case space is discrete and the study ranges over a finite set of logged cases, so every capacity, purity, and coverage quantity below is an empirical fraction (a count). The general measurable formulation, with (Ω, P) a probability space and capacity given by singular mass, is needed for the continuous case spaces of the medical-imaging and language-model companions and is developed in the Formal Supplement.

$$Y_k^{-1}(s) := \big\{\tau \in \Omega : Y_{k(\tau)} = s\big\},$$

the cases stage k renders as the same artifact $s \in S_k$. Two cases in the same bucket cannot be distinguished at stage k by any predicate over Y_k. This is the equivalence-class structure of Pawlak's rough set theory (Pawlak, 1982): a representation induces an equivalence relation on Ω (indiscernibility), and buckets are its equivalence classes.[3] A bucket value s names the property $Y_k = s$; a region $A \subseteq S_k$ names $Y_k \in A$, and a test rejecting on A fires on exactly the buckets whose values lie in A.

Assembly-line gloss. A bucket at an RF chamber on a phone assembly line is the set of phones with identical spectrum profile at the chamber's resolution. Two phones in the same bucket are indistinguishable to that station; they are accepted or rejected together, whatever other defect they carry. If the profile is too coarse, a phone that spikes in a forbidden band shares a bucket with a clean phone of the same total, and the representation cannot separate good from bad. Resolve the spectrum per frequency and the spiking phone splits into its own pure-positive bucket, where a zero-false-positive test catches it. *The finer representation did not change the phones; it stopped hiding the violation.*

2.3 The deployed and deterministic gate families

Definition 2 (Capability set, deployment set, target-relative deterministic subfamily). The *capability set* $\mathcal{D}_k$ (the deterministic *predicate class*) is the set of deterministic tests that could be run on an artifact at stage k and used as gates. Each test is identified with the subset $A \subseteq S_k$ on which it fires (its rejection region); we use the test and its region interchangeably. (Tests are computable within the design budget; the class is fixed before evaluation and excludes tests that memorize held-out labels.)

The *deployment set* F_k^{deploy} (the deployed *gate family*) is the set of gates the engineer has *actually* deployed at stage k, observable as elimination events at runtime. It may include deterministic predicates from $\mathcal{D}_k$ as well as stochastic and heuristic gates that are not in $\mathcal{D}_k$.

Rejection and elimination events. Each deployed gate g has a *rejection event* $R_{k,g} \subseteq \Omega$, the cases on which it fires at runtime. For a deterministic artifact-predicate gate identified with a region $A_g \subseteq S_k$, this is $R_{k,g} = \{Y_k \in A_g\}$; a stochastic or heuristic gate's rejection event need not factor through Y_k. The stage-k *elimination event* is the union over all deployed gates:

$$E_{k,\alpha} := \bigcup_{g \in F_k^{\text{deploy}}} R_{k,g}$$

The *target-relative deterministic subfamily* is the deployed deterministic gates that happen to be zero-false-positive against the target:

$$F_{k,\alpha}^{\text{det}} := \big\{g \in F_k^{\text{deploy}} : \exists A_g \in \mathcal{D}_k \ \text{ with } \ R_{k,g} = \{Y_k \in A_g\} \ \text{ and } \ Y_k^{-1}(A_g) \subseteq L_\alpha \ \text{ modulo } \ P\text{-null sets}\big\}.$$

Membership is target-relative: a gate may be zero-FP for target α and not for target β.

How to use the sets. The three families overlap rather than nest. The deployed deterministic gates are those $g \in F_k^{\text{deploy}}$ for which $R_{k,g} = \{Y_k \in A_g\}$ for some $A_g \in \mathcal{D}_k$. The deployed stochastic or heuristic gates (LLM judge, learned classifier) are the remaining $g \in F_k^{\text{deploy}}$. The undeployed deterministic predicates are regions in $\mathcal{D}_k$ not associated with any deployed gate. The work is to move useful deterministic checks into production and measure which are actually zero-FP for each target. Human-resolved escalations can promote into durable deterministic predicates when the resolution is expressible over the existing artifact (the oracle channel, in the Formal Supplement); otherwise they identify a representation gap that motivates changing Y_k.

[3] The same objects are called *fibers* in algebraic topology and *granules* in granular computing; we use *bucket* throughout for accessibility.

2.4 Four measurement attributes

The four measurement attributes diagnose the surface (throughout, a hat marks an empirical estimate, e.g. $\hat{\rho}$ estimating the population ρ; the $\hat{\Gamma}^{\text{all,fb}}$ superscript (all deployed gates, feedback-based audit) is defined in full in the coverage section):

Attribute	What it tells the engineer
η	Capacity: the fraction of failing cases that land in pure-positive buckets (artifacts no good case ever produces), and therefore the ceiling on zero-FP deterministic recall, fixed by the representation rather than gate cleverness. η^* is the full representation ceiling, $\eta^*_{\mathcal{D}}$ the capability-restricted ceiling, $\hat{\eta}$ realized deterministic recall; the gaps $\eta^* - \eta^*_{\mathcal{D}}$ and $\eta^*_{\mathcal{D}} - \hat{\eta}$ distinguish capability-set slack from deployment slack.
ρ	Per-gate purity: the fraction of a gate's rejections that are target-true; $\rho = 1$ means zero-FP for the target it is scored against
Γ	Per-target observed recall of the full deployed family (deterministic, stochastic, and heuristic combined)
ω	Complementarity within or between surfaces (Jaccard of rejection sets)

These are the diagnostics this paper measures on the software pipeline: the precision section reports $\hat{\rho}$, the coverage section $\hat{\Gamma}^{\text{all,fb}}$, and the cross-stage discussion ω. Capacity η is split into three levels so capability-set slack and deployment slack separate; the software schema does not yet permit a deterministic-only $\hat{\eta}$ estimate (limitations section).

2.5 Diagnostic: surface-target alignment

If no failing case lands in a pure-positive bucket, the zero-FP capacity is zero ($\eta^* = 0$), and consequently $\eta^*_{\mathcal{D}} = \hat{\eta} = 0$ for any deployed deterministic family: no zero-FP deterministic gate over Y can achieve nonzero recall against L. (The chain $\hat{\eta} \le \eta^*_{\mathcal{D}} \le \eta^*$ forces both lower levels to zero; the formal statement is proved as Corollary 1 in the Formal Supplement.)

This is the falsifiable design rule. When every bucket of Y carrying target-positive cases also carries target-negative cases, no zero-FP deterministic gate can succeed: the pipeline cannot be repaired by adding gates, only by changing Y to produce buckets that separate target-positive from target-negative cases.

Sports-replay gloss. If no camera angle saw the relevant moment of a play, no review can call it, and adding more reviewers to the same angles does not help; the fix is a different camera position. Likewise, when a surface's buckets do not separate the target's positive from negative cases, the fix is a different representation, not more gates.

Two-part design constraint. A useful surface must satisfy two conditions. First, it must *target-separate*: some target-positive cases must lie in pure-positive buckets, or no zero-FP gate can fire (this section). Second, those buckets must be *expressible* in the deployable capability set: $\mathcal{D}$ must contain a region whose preimage matches those pure-positive buckets, or the design ceiling stays out of reach (the capability-set restriction, in the Formal Supplement). The two fail differently: a target-separation failure points to representation change, and an expressibility failure to capability-set enrichment within an already-promising representation.

2.6 A falsifiable prediction: no universal verifier without a universal target

If a single feasible $(Y, \mathcal{D})$ pair contained nontrivial ZFP deterministic regions for every target a system cares about, the surface-target distinction would be redundant.[4] The framework predicts no such pair exists for open-ended generation under realistic constraints: non-oracular, budget-bounded, fixed before deployment. The reason follows from the capacity characterization: a bucket pure-positive for target L_α typically carries target-negative cases for a different L_β, since the two induce different positive and negative subsets of Ω, so a representation tuned to separate one target's positives commingles another's. The prediction is falsifiable:

[4]The "nontrivial" qualifier rules out empty verifiers: an empty rejection set is vacuously zero-FP but useless.

a single feasible $(Y, \mathcal{D})$ pair with $\rho = 1$ *and nontrivial zero-FP coverage* across a non-trivial target collection would force a revision of the central claim, that surface choice and target choice must be paired.

Scope of the claim. The claim is engineering-falsifiable, not mathematically universal. A trivial ($Y =$ identity, $\mathcal{D} =$ all target indicators) pair achieves $\rho = 1$ across every target by construction, but violates the non-oracular requirement (the engineer cannot evaluate target indicators without already knowing the outcome) and the budget-bounded requirement (the capability set is unbounded). The claim is about feasible pairs: representations computable within the design budget over deployable artifacts, with capability sets fixed before evaluation. Within that scope the prediction stands, and the software realization is one instance —the four gates each specialize for the targets their representation exposes and align poorly with the others (precision and coverage sections).

3 Pipeline as Verification Surfaces

3.1 Architecture

The pipeline implements a standard SDLC workflow: prompt → plan → design → code. A separate-model review gate (deterministic checks plus an LLM reviewer) runs at each post-prompt stage, and a purely deterministic test-result postcondition sits downstream. This is decades-old software engineering practice. The pipeline predates the LLM era and has been in iterative production use for years. The current form is its latest revision, adapted to use coding agents.

Anthropic Claude generates work artifacts (plans, designs, code, and tests). Google Gemini, acting as a separate verification model, validates each artifact through four mandatory review gates (`review_plan`, `review_design`, `review_code`, `codereview`). A deterministic test-result check also runs downstream of the code stage, where tests pass or fail with no reviewer. It is part of the pipeline but not analyzed empirically; the precision and coverage matrix covers the four review gates only, each of which pairs deterministic checks with a Gemini reviewer. The generator transitioned from Sonnet-tier to Opus-tier mid-window while the reviewer model held constant, a direct result of a living pipeline dealing with the blistering pace of model evolution. The full model-role and identifier table is in the methods-detail appendix, and the Zenodo aggregate package records the production reviewer's exact Gemini identifier per row.

- **`review_plan`**: verifies plan structure, completeness, scope alignment, and the presence of acceptance criteria. Returns a structured verdict with per-finding severity.
- **`review_design`**: verifies design coherence, internal consistency, separation of concerns, and design-to-plan traceability.
- **`review_code`**: verifies that code matches design, parses correctly, and follows project architecture conventions.
- **`codereview`**: checks file-scoped local correctness, bugs, style violations, and missing error handling. It has the narrowest observability of the four.

Each gate is a three-state operation: deterministic checks (e.g. structural completeness, syntactic validity, schema enforcement, lint and style rules) fire first, then the stochastic LLM reviewer renders a qualitative verdict, and unresolved cases may escalate to the human. Rejections at any stage send feedback to the generating agent, which revises the artifact and resubmits it to the same gate. Architecturally, the model that generated each artifact never reviews it. This reduces direct self-review bias but does not by itself establish statistical independence of escapes: the reviewer can still share training data, inductive biases, coding conventions, or model-family limitations with the generator, and gate-level co-rejection is distinct from escape independence under the product-form lemma. This paper treats architectural separation as a necessary practical guard rather than a proof of conditional independence. (The escalation state, the schema's escalation-count limitation, and the oracle channel by which $\mathcal{D}_k$ can grow are developed in the limitations section and the Formal Supplement.)

3.2 Measurement window and instruments

The measurement window covers 97 consecutive days of production operation, over which the pipeline shipped 165 releases. Two analyzer instruments observe this system: the calibrated private analyzer is scoped to exactly that window, while the public live DB is a current extract of the same ongoing system that extends past it. The calibrated private regex analyzer (the earliest paper numbers) recorded 5,109 gate checks and 1,450 pre-correction analyzable rejection events; it is highly precise on the author's gate-prompt formats but not portable. The *public-instrument live DB* is the `gate_analytics.db` instance the public open-source analyzer (`gate_analyzer.py`) produces from the author's session logs; it reports 5,519 gate checks and 760 *analyzable rejection events* under the explicit live-DB filter `decision = NEEDS_REVISION AND error_class IS NOT NULL AND error_class != API_ERROR` (the reproducibility appendix). "Public" describes the analyzer, not the database content: the live DB contains production prompts and code and is not publicly hosted, and only the redacted aggregate is in the Zenodo package. Because the two instruments classify the same underlying production process through different pipelines (calibrated regex versus LLM-assisted) and not merely through different filters, this paper reports both and standardizes precision and recall denominators on the public-instrument live-DB definition. All artifacts are real working code that shipped to production after passing the pipeline; the window is a slice of an ongoing system, not its full lifetime. The model-role and identifier table, the per-instrument count reconciliation, and the dual-instrument rationale are in the methods-detail appendix.

3.3 Case-space unit: the task

Following Definition 1 (§2), the pipeline is a *revision* architecture. Each case τ is a *task*: a single coherent unit of work that flows through its own scoped review cycle (plan, design, and code each have a review gate, and tests run as a deterministic downstream postcondition). Plans at the orchestration level decompose into multiple tasks. Each task has its own scoped trajectory. A task contains many artifact-versions (each revision of its plan, design, code, and review findings) and many gate checks.

The 5,109 gate checks and 1,450 rejections in the snapshot are individual artifact-level events. For cross-stage analysis, the elimination event $E^{\text{task}}_{k,i}$ holds for task τ if gate i at stage k rejected at least one artifact-version produced during τ's review cycle. The gate-overlap ratio ω (§2, attribute table) therefore uses each task's full revision history, not co-rejection of a single artifact-version (which is structurally impossible under fail-fast revision). The notation aligns with the theory spine (F^{deploy} for the deployed gate family, $E_{k,\alpha}$ for the stage elimination event). The E^{task} superscript marks aggregation to the task level over a task's full revision history. The task is the appropriate unit because it carries a single intent through all four stages.

3.4 Refinement gradient across the stages

This subsection maps the pipeline to the verification-surface primitive of §2. Each post-prompt stage corresponds to a surface $\Sigma_{k,\alpha} = \left(Y_k, \mathcal{D}_k, F^{\text{deploy}}_k, L_\alpha\right)$ (§2, Definition 1 and Definition 2). The software-specific content is which representation, capability set, and target violations live at each SDLC stage:

Stage	**Representation S_k**	**Capability set $\mathcal{D}_k$**	**Target violations α**
Prompt	Unrestricted natural language	Trivial: format-membership predicates only	None directly checkable from format alone
Plan	Structured plan artifact (sections, acceptance criteria, scope)	Schema completeness, criterion presence, required-section presence	Missing acceptance criterion, missing scope, missing test plan, structural omission
Design	Structured design artifact (component map, traceability)	Schema completeness, reference resolution, traceability-link presence	Inconsistency, missing trace, scope drift, layered-violation
Code	Source code, syntax-typed	Parse, type, schema, lint, dependency, static-analysis predicates	Syntax error, type error, missing import, lint violation, dependency cycle

Stage	**Representation S_k**	**Capability set $\mathcal{D}_k$**	**Target violations α**
Test result	Pass/fail outcome per test	Pass/fail predicates per test specification	Failed test (for the tested specification)

The $\mathcal{D}_k$ column lists only *genuinely deterministic* predicates: ones whose verdict on a given artifact is reproducible byte-for-byte and never requires semantic judgment. Not every target in the α column has one. At the design stage, *missing trace* is deterministically checkable via traceability-link presence. But *inconsistency* (does the design contradict itself) and *scope drift* (does its scope still match the plan's) are semantic judgments with no separating $\mathcal{D}_k$ predicate, so the stochastic LLM reviewer catches them: they live in F_k^{deploy} but *not* in $\mathcal{D}_k$ (§2.3 distinguishes the two). That $\mathcal{D}_k$-to-target gap is exactly the deterministic-capacity shortfall the next paragraph reads as an η gradient, widest at the design stage, whose gate largely relies on an LLM reviewer over a target-exposing representation rather than through deterministic checks.

The natural-language prompt has trivial $\mathcal{D}$ in the syntactic sense (every UTF-8 string is a valid prompt), and its target-conditional text distributions are largely indistinguishable for substantive correctness properties. Plan, design, and code surfaces progressively refine the representation: a plan separates "has acceptance criterion" from "lacks acceptance criterion" with a deterministic predicate; a design exposes "has design-to-plan trace" vs "lacks trace"; code exposes "parses" vs "fails to parse." A test-result surface exposes high-purity pass/fail predicates *for the tested specification*, not for total program correctness.

One large language model produces a task's plan, design, and code, yet the capacity ceiling climbs from low η at the prompt to higher η downstream without any implication that the underlying model has grown less fallible. In software, the Chomsky hierarchy is one source of this η floor: Type-3 (regular) structure supports high η, Type-2 (context-free) moderate-to-high, down to Type-0 (unrestricted natural language) near zero from format alone, so architects raise η by targeting lower hierarchy levels at intermediate stages. That source is not universal (the medical companion derives η from physical anatomy, not formal-language structure), but it governs the SDLC realization here. *The available capability set and observed mixed coverage improve when the representation exposes the target, not because the model improves.* The deterministic-only ceiling growth in this paper is established by the structural argument; the empirical signal is on the mixed-gate $\hat{\Gamma}^{\text{all,fb}}$ rather than on deterministic-only $\hat{\eta}$, which the current schema does not yet permit (limitations section).

This progression is an *operational analogue* of the refinement-monotonicity argument (§2; Formal Supplement), not a literal instance. The formal argument requires that every Y'-bucket be a subset of some Y -bucket, so the later representation refines the earlier one. In a coding-agent pipeline, the stages are *generated transformations*: a design may add, omit, distort, or reinterpret the plan; code may fail to preserve the design; tests observe only a subset of behavior. Monotonic capacity is therefore a design aspiration and an empirical question, not a mathematical consequence. It is achieved only when later artifacts preserve task-relevant information while exposing new target-aligned predicates; the empirical $\hat{\Gamma}^{\text{all,fb}}$ pattern in the coverage and surface-target-alignment sections is the observational evidence that the deployed pipeline approximately realizes this in practice for the target classes the design-stage gates expose.

Each deployed gate is a mix of deterministic and stochastic checks: in the production reviewer, schema enforcement, structural-completeness checks, and syntax/parse/type checks sit within $\mathcal{D}_k$, while the LLM reviewer's qualitative judgments about coherence, scope alignment, and design quality are stochastic gates with no predicate counterpart in $\mathcal{D}_k$. Capacity η is defined over the deterministic subfamily $F_{k,\alpha}^{\text{det}}$; the coverage attribute $\Gamma_{k,\alpha} := P\big(E_{k,\alpha} \mid L_\alpha\big)$ is defined over the full deployed family F_k^{deploy} and is what the shadow audit measures (the event notation and cross-stage composition are in §2.3 and the Formal Supplement). Because the current schema does not tag decisions by source, the empirical estimate is the mixed-gate $\hat{\Gamma}_{k,\alpha}^{\text{all,fb}}$ rather than a deterministic-only $\hat{\eta}_{k,\alpha}$ (the limitations section gives the required schema upgrade).

3.5 Per-stage target-violation taxonomy

The error taxonomy used downstream, **omission**, **systematic**, **incoherence**, maps onto the per-stage target-violation set α as follows:

Taxonomy class	What it labels	Most-exposed surface	Least-exposed surface
`OMISSION`	Artifact incomplete: missing required section, missing acceptance criterion, missing test	Plan, Design (high $\hat{\Gamma}$)	Code, Codereview (low $\hat{\Gamma}$)
`SYSTEMATIC`	Artifact violates a specified constraint: scope drift, design-spec mismatch, prohibited dependency	Plan, Design (high $\hat{\Gamma}$)	Codereview (low $\hat{\Gamma}$)
`INCOHERENT`	Artifact internally inconsistent: contradicts itself across sections or files	Plan, Design (high $\hat{\Gamma}$)	Codereview (low; cross-context target not exposed — surface-target-alignment section)

The coverage section reports the empirical $\hat{\Gamma}^{\text{all,fb}}_{k,\alpha}$ for each (stage, target-class) cell. The qualitative pattern is that design-stage gates expose broad omission/systematic/incoherent structure, while file-scoped `codereview` exposes mostly local correctness, which is the surface-target mismatch developed in the surface-target-alignment section.

4 What Each Empirical Estimate Measures (and Does Not Yet Measure)

To make the strength of each claim explicit up front, the following table maps each headline quantity to the evidence source and the gap that remains:

Quantity	Estimate	What it measures	What it does not yet measure
$\hat{\rho}$ substantive	0.965	Consensus-valid rejection reports (coherence, specificity, actionability of the reviewer's stated concern)	Artifact-grounded false-positive rate
$\hat{\rho}$ end-to-end	0.774	Public-analyzer precision including parser/workflow/API extraction artifacts	Production gate quality alone
$\hat{\Gamma}^{\text{all,fb}}$ overall	58.5%	Feedback-based clear-case mixed deterministic + stochastic coverage	Artifact-blind recall
κ rejection-precision	0.812	LLM cross-rater agreement on structured rejections	Human ground truth
κ acceptance-shadow	0.562	LLM cross-rater agreement on missed defects	Stable recall without human audit
Plan-stage rejection share	43.6% (live DB)	Distribution of where rejections fire across the four gates	Causal evidence that early gates reduce later defects
Revision recovery	80.6% (live DB gate_iterations)	Process pass-after-revision rate at the (task, gate) level	Semantic correctness of the rejection
Decomposed vs single-agent incoherence	$\approx$ 10% vs $\approx$ 20% (private 97-day snapshot)	Observed incoherence-rate difference; not yet replicated on the live-DB instrument	Causal effect of decomposition (potential confounders: task type, difficulty, review intensity)

The table is the paper's epistemic key: most quantities are **strong proxies** rather than direct measurements of what this approach asks about. The current evidence supports the framework's **relative** predictions (e.g.,

representation scope predicts coverage) more than its **absolute** claims (e.g., a specific deterministic-η value or a specific escaped-defect rate).

The instrument is a production system, not an apparatus. The proxy shape above is not an accident of analysis; it is what instrumenting a real pipeline yields. The schema records what the system needed to ship software, not what this paper would ideally measure, so deterministic and stochastic checks arrive bundled, the shadow audit re-judges from reviewer feedback rather than stored artifacts, and the rest of the right-hand column follows from the same cause. The trade is measurement precision for ecological validity: every number describes a pipeline that shipped 165 releases over 97 days under production constraints, not a clean-room proxy for one. The controlled complement is the medical companion, taken up under External consistency below.

5 Methods Summary

The pipeline data is from production engineering work and contains proprietary content, so the raw logs cannot be released publicly. To compensate, the public extraction code and schema are open-source at github.com/mrothroc/claude-code-log-analyzer the redacted aggregate databases and adjudication summaries are deposited with this Zenodo record. The reproducibility appendix gives the full extraction-pipeline definitions, the live-DB filter used throughout, and the reproducibility manifest.

Three method choices govern how the numbers should be read; the full mechanics are in the methods-detail appendix. First, the *dual instrument*: the private calibrated regex analyzer is precise on the author's gate-prompt formats but not portable, so all public precision and recall estimates use the reproducible LLM-assisted public analyzer, which is less precise on this specific data but runnable on any engineer's Claude Code logs. Second, *two false-positive notions* that the headline $\hat{\rho} = 0.965$ must be read against: an *artifact-grounded* false positive (a rejection that demanded no actual fix) requires re-reading the original artifact and is not directly measurable from the current schema, whereas the *rejection-report* false positive (a rationale incoherent or out-of-scope on its face, judged from the reviewer text alone) is what dual-voted adjudication measures and what $\hat{\rho} = 0.965$ estimates. Precision denominators based only on reviewer text are therefore *rejection-report-quality* denominators; the limitations section (Priority 1) describes the schema upgrade that would convert one into the other. Third, the *cross-model adjudication protocol*: two adjudicators (`claude-sonnet-4-6` and `gemini-2.5-flash`) independently rate each case without seeing the other's vote, consensus is assigned only when both agree, and disagreements are recorded as `ambiguous_by_disagreement` and excluded from clear-case denominators. The expanded audit contains 1,570 dual-voted cases: 770 rejection-precision rows and 800 acceptance-shadow rows.

5.1 External consistency

The companion medical-imaging paper (DOI: 10.5281/zenodo.20331363 builds on the prior Zenodo preprint 10.5281/zenodo.19362420) covers a non-revision architecture with public datasets (PI-CAI, MSD Liver, KiTS23) and no code or models shared with this software pipeline. Cross-domain replication of the same structural findings, rare cross-stage co-rejection, surface-target alignment driving high-purity recall, and surface-target mismatch driving null-result coverage, provides external consistency, not independent external evidence: the medical work is same-author, same-theory replication.

6 Precision Results

Two adjudicators (Claude Sonnet 4.6 and Gemini 2.5-Flash) dual-voted the full 770-row rejection-precision stratum from `gate_analytics.db`, stratified by gate (200 per gate; `codereview` at 170, all included), each rating the reviewer's structured rejection text without seeing the other's vote. The protocol sub-classifies false positives into *substantive* (the reviewer was actually wrong) versus three *extraction artifacts* (parser, workflow, and API-error rows the analyzer captured as gate decisions; details in Appendix C). That split

matters for the headline: of the sample's 161 raw false positives, only 20 are substantive (concentrated in `review_code`); the other 141 are analyzer-pipeline noise that shifts end-to-end precision but not substantive precision. The per-gate false-positive breakdown is in the methods-detail appendix.

The dual-voted adjudication supports three precision denominators, each answering a different question about the public reproducible instrument (source: `data/kappa_summary.json`, $n = 770$):

Precision definition	**Numerator / denominator**	**Estimate**	**95% Wilson CI**
Substantive clear-case (excl. parser/workflow/API/disagreement)	550 / 570	0.965	$[0.946, 0.977]$
End-to-end clear-case (incl. parser/workflow/API)	550 / 711	0.774	$[0.741, 0.803]$
Conservative all-sample (disagreement and ambiguous as non-genuine)	550 / 770	0.714	$[0.681, 0.745]$

The substantive clear-case figure $\hat{\rho} = 0.965$ (excluding parser/workflow/API, ambiguous, and disagreement rows) is the relevant estimate of rejection-report quality (§5); the two design-stage gates reach 1.000 on it: every consensus-clear rejection from `review_plan` and `review_design` was a substantive concern. The end-to-end 0.774 adds the extraction artifacts back into the denominator; its 19 pp gap from the substantive figure is the analyzer-instrument gap, dominated by `codereview`'s workflow-artifact tail. The conservative 0.714 treats disagreement and ambiguous rows as non-genuine, the lower bound under the strictest reading. The substantive 0.965 is substantially higher than the prior single-rater Claude estimate of 0.896 (n=162), driven by `review_plan` and `review_design` reaching 1.000 in the larger sample, not by any change in the deployed pipeline. The per-gate precision table across all three denominators is in the methods-detail appendix.

Cross-rater agreement separates the two strata. Cohen's κ between the two adjudicators is 0.812 on rejection-precision (just inside the Landis–Koch "almost perfect" band (Landis & Koch, 1977)) but only 0.562 on acceptance-shadow (moderate). The adjudication is feedback-based: both adjudicators judge the reviewer's own output, not the artifact under review (the schema does not persist artifact snapshots; limitations section), so κ reflects agreement on that feedback rather than a check against the artifact. The asymmetry is structural: a second adjudicator can recognize whether an explicit, structured rejection cites a real concern, but deciding what defect *should have been* flagged under a silent approval requires more inference and produces more disagreement. `codereview` has the weakest acceptance-shadow κ (0.387), which is consistent with the surface-target-mismatch story (§8): when the representation does not expose cross-context state, even the adjudicators struggle to agree on what should have been flagged. The full per-gate κ table and its readings are in the methods-detail appendix.

To check the consensus method against a human read, the author manually adjudicated a stratified $n = 50$ sample enriched $4\times$ toward Claude/Gemini disagreement. When both LLMs agree, the human agrees with that consensus on $\frac{23}{24} = 95.8\%$ of cases, and on the disagreement subset sides with Gemini over Claude roughly $2:1$; the population projection puts human-anchored substantive precision at approximately 0.94, within 2 pp of the LLM-only 0.965. This is a sanity check, not independent ground truth, the reviewer is the system designer, the sample is small, and it was deliberately enriched for disagreements. The full breakdown (per-stratum tables, tie-break direction, per-gate human-vs-LLM κ, triple-disagreement cases, projection arithmetic) is in the adjudication appendix; submission-grade validation requires a larger independent audit with at least two unaffiliated reviewers (the limitations section, Priority 2). The 1,450-vs-760 and 5,109-vs-5,519 instrument reconciliation behind these denominators is in the methods-detail appendix.

7 Coverage Results ($\hat{\Gamma}^{\text{all,fb}}$)

The shadow audit estimates $\hat{\Gamma}^{\text{all,fb}}_{k,\alpha} = P(E_{k,\alpha} \mid L_\alpha)$, the observed feedback-based typed coverage of the deployed mixed deterministic + stochastic review gates, *not* pure deterministic $\hat{\eta}$. The superscript "all" denotes the full deployed gate family F_k^{deploy} (vs. the deterministic subfamily $F_{k,\alpha}^{\text{det}}$), and "fb" denotes *feedback-*

based: adjudicators re-judged each case from the reviewer's feedback text rather than from the original artifact, because the current schema does not preserve a raw artifact snapshot for every row (the limitations section, Priority 1). Here $E_{k,\alpha}$ (§2.3) is the stage elimination event. A deterministic-only $\hat{\eta}$ estimate requires separating schema, structural, compile, and test predicates from stochastic LLM judgments; this is future work.

Both adjudication strata are fully dual-voted (acceptance-shadow at $n = 800$ for the missed-violation counts, rejection-precision at $n = 770$ for the caught counts), counting a case only on adjudicator agreement. We compute $\hat{\Gamma}^{\text{all,fb}}_{k,\alpha}$ as a population-weighted ratio of caught to caught-plus-missed clear-consensus rows, with per-stratum expansion weights derived from the live-DB row counts; the full estimator with variable definitions is in the reproducibility appendix. The intervals below are descriptive Wilson intervals (Wilson, 1927) on the population-scaled effective counts n_{eff}, not full design-based survey intervals: they do not propagate the expansion-weight uncertainty, which a survey-design treatment (Priority 7) would.

7.1 Per-(gate, target) coverage

Gate	Target	caught (raw)	missed (raw)	n_{eff} (pop-wt)	$\hat{\Gamma}^{\text{all,fb}}$	95% Wilson CI
`review_plan`	OMISSION	112	43	525.7	73.7%	[0.698, 0.773]
`review_plan`	SYSTEMATIC	46	16	210.6	75.6%	[0.693, 0.809]
`review_plan`	INCOHERENT	29	2	106.7	94.0%	[0.878, 0.971]
`review_design`	OMISSION	89	2	222.6	95.0%	[0.913, 0.972]
`review_design`	SYSTEMATIC	55	3	147.4	88.7%	[0.826, 0.928]
`review_design`	INCOHERENT	30	1	76.8	92.8%	[0.847, 0.967]
`review_code`	OMISSION	58	31	393.9	25.3%	[0.212, 0.298]
`review_code`	SYSTEMATIC	86	39	517.9	28.5%	[0.248, 0.325]
`review_code`	INCOHERENT	11	1	28.4	66.5%	[0.482, 0.810]
`codereview`	OMISSION	9	3	17.3	55.4%	[0.332, 0.756]
`codereview`	SYSTEMATIC	24	21	79.7	32.1%	[0.229, 0.430]
`codereview`	INCOHERENT	1	0	1.1	*n=1; uninformative*	—

Counts. "caught (raw)" and "missed (raw)" are the dual-voted clear-case counts in the 770-row rejection-precision and 800-row acceptance-shadow samples respectively (both adjudicators agree on verdict and target). n_{eff} (population-weighted) is the live-DB count of (target-positive caught + target-positive missed) under the same dual-voted classifier, scaled from the sample to the population by stratum frequencies. $\hat{\Gamma}^{\text{all,fb}}$ is feedback-based shadow-audit coverage (not artifact-grounded, verdict-blind recall). Wilson 95% CIs are computed on n_{eff}.

This descriptive analysis treats cells with $n_{\text{eff}} \geq 50$ as *better-powered* and cells with $n_{\text{eff}} < 50$ as *directional*. Under that threshold, all six plan/design cells (`review_plan` and `review_design` × OMISSION/SYSTEMATIC/INCOHERENT), `review_code` OMISSION and SYSTEMATIC, and `codereview` SYSTEMATIC are better-powered. `review_code` INCOHERENT ($n_{\text{eff}} = 28.4$), `codereview` OMISSION ($n_{\text{eff}} = 17.3$), and `codereview` INCOHERENT ($n = 1$ caught, 0 missed) are directional or uninformative. The cell-level pattern is the central finding for surface-target alignment. On the better-powered cells the design-stage gates exceed the code-stage gates by a wide margin at both the SYSTEMATIC and OMISSION targets, under the same production generator/reviewer pipeline and comparable review pattern, so the gap is consistent with a representation effect; the surface-target-alignment section develops the headline cell-level comparison off this matrix. The `codereview` OMISSION value (55.4%) is directional, not citation-grade.

The `codereview` INCOHERENT cell is uninformative: only 1 case in the dual-voted sample has cross-context incoherence at the codereview gate (caught), with 0 missed. The surface-target-alignment section's qualitative claim is that file-scoped representations cannot reliably expose cross-context targets. That claim rests primarily on the better-powered SYSTEMATIC cell at $n_{\text{eff}} = 79.7$ and on the structural argument. The OMISSION cell ($n_{\text{eff}} = 17.3$) is directionally consistent with a wide interval. The INCOHERENT cell is too thin to estimate. The companion medical paper provides an analogous, better-powered null-result on a non-revision architecture.

7.2 Population-weighted overall $\hat{\Gamma}^{\text{all,fb}}$

Gate	Population-weighted $\hat{\Gamma}^{\text{all,fb}}$	Wilson 95% CI	Prior single-rater estimate
`review_plan`	76.7%	[0.737, 0.794]	62.9%
`review_design`	92.5%	[0.897, 0.946]	75.1%
`review_code`	28.3%	[0.255, 0.313]	18.1%
`codereview`	37.0%	[0.281, 0.469]	5.2%
Overall	**58.5%**	[0.565, 0.605]	40.1%

The dual-voted estimates shift up uniformly relative to the prior single-rater $\hat{\Gamma}^{\text{all,fb}}$. This is not a deployment change; it reflects requiring consensus on what counts as a missed violation, which excludes cases where one LLM adjudicator over-flagged a non-violation. The qualitative ordering survives: design-stage gates dominate code-stage gates by roughly 40–65 pp at the population-weighted level (`review_design` 92.5% vs `review_code` 28.3% = 64.2 pp; `review_design` 92.5% vs `codereview` 37.0% = 55.5 pp; `review_plan` 76.7% vs `codereview` 37.0% = 39.7 pp).

7.3 Honest caveat

This is a *feedback-based* shadow audit, not a full artifact-grounded, verdict-blind re-review. The current `gate_checks` schema preserves reviewer feedback and metadata but does not preserve the exact raw artifact snapshots for every approved row. The shadow audit therefore re-judges based on what the reviewer wrote, not on what the reviewer was looking at. The schema upgrade required for a full artifact-grounded, verdict-blind audit is documented in the limitations section (Priority 1).

7.4 Interpretation

`review_plan` and `review_design` show high $\hat{\Gamma}^{\text{all,fb}}$ across all target classes; their representations expose the targets. `review_code` and `codereview` show steeply lower coverage on the OMISSION and SYSTEMATIC targets, the cross-context patterns the design-stage gates excel at. The qualitative conclusion: design-stage gates expose broad omission and systematic structure across the artifact, while code-stage and file-scoped gates are mostly blind to cross-context violations because their representations do not expose them. The surface-target-alignment section makes this prediction-from-the-framework rather than a curiosity.

8 Surface-Target Alignment in Software

This is the paper's central empirical result. The surface-target alignment diagnostic (§2) predicts that a gate over a representation that does not expose the target has low coverage for that target, however well the gate judges. The software pipeline makes that prediction visible at production scale: a 56–70 pp coverage spread between design-stage and code/file-stage gates on the same cross-context defect classes. The design-stage gate operating on a richer representation (`review_design`) reaches 88.7% SYSTEMATIC and 95.0% OMISSION coverage, while the code-stage and file-scoped gates over narrower representations sit far below on those same targets (`review_code` 28.5% SYSTEMATIC / 25.3% OMISSION; `codereview` 32.1% SYSTEMATIC). Both gate families run on the same production pipeline with the same held-constant reviewer model, and the review pattern is comparable, so the spread is consistent with the representation, rather than the model or the gate, being the dominant factor. The rest of this section develops that finding as a direct instance of the diagnostic.

8.1 The empirical finding

The four gates operate on representations of progressively narrower scope. `review_plan` and `review_design` see the entire structured artifact (plan or design) plus its traceability links. `review_code` sees the code under

review plus its design context. `codereview` is *file-scoped*: each invocation runs on a single file's diff or post-edit snapshot, with no explicit cross-file context.

The `codereview` low-coverage finding discussed here is a *surface* phenomenon: the file-scoped representation cannot reliably expose cross-context targets, which lowers `codereview`'s $\hat{\Gamma}^{\text{all,fb}}$ on those target classes. (Separately, the public analyzer that produced the live DB carried extraction-artifact noise on `codereview` rows, documented and quantified in the extraction-artifact appendix; that is an instrument-level issue that does not affect the key finding here because the dual-voted LLM adjudication is computed on the consensus verdicts, not on the analyzer's raw output.)

The dual-voted $\hat{\Gamma}^{\text{all,fb}}$ matrix (per-(gate, target) cell, with n_{eff} and Wilson intervals, in the coverage section) reveals a spread that tracks representation scope. We read the headline comparisons off that matrix here rather than reprinting it; the per-cell power thresholds that qualify each comparison live with the matrix.

The qualitative pattern is the primary finding: the best-powered cross-surface comparisons show a 56–70 pp design-stage advantage, with other cells directionally consistent but thinner. The three best-powered headline cell-level comparisons are:

- `review_design` SYSTEMATIC 88.7% vs `review_code` 28.5% = 60.2 pp
- `review_design` SYSTEMATIC 88.7% vs `codereview` 32.1% = 56.6 pp
- `review_design` OMISSION 95.0% vs `review_code` 25.3% = 69.7 pp

The `review_design` OMISSION vs `codereview` OMISSION comparison (95.0% vs 55.4% = 39.6 pp) is directionally consistent but draws on a directional `codereview` OMISSION cell ($n_{\text{eff}} = 17.3$). The narrow-representation gates (`review_code`, `codereview`) are not poorly written; they are surfaced against a representation that cannot reliably expose cross-artifact targets.

Architectural-drawing gloss. A zoning problem visible in an architect's site plan is cheap to catch and decisive at that stage. The same problem caught after framing is up costs a teardown; caught after move-in, the whole building is wrong. `review_design` sees the structural intent and catches structural omissions; the file-scoped `codereview` sees a chunk of source and catches local errors. Same defect class, two different surfaces, different coverage by construction.

8.2 The framework prediction

A coding agent's defects are typically cross-context: an OMISSION at the code level often means the artifact is missing something specified in the plan or design (cross-file or cross-stage); a SYSTEMATIC error often means the implementation violates an architectural decision recorded elsewhere; an INCOHERENT defect is, by definition, an inconsistency between two or more locations. None of these defects are reliably observable from a single file's text alone. The buckets of $Y_{\text{codereview}}$, sets of cases that map to the same single-file output, contain both target-positive and target-negative cases, because the difference depends on what is *not* in the file.

Diagnostically: the observed coverage pattern is consistent with the diagnosis that defective tasks and correct tasks are insufficiently separated on the file-local representation $Y_{\text{codereview}}$ for these cross-context targets. We do not measure the capacity on this representation directly; the empirical signal is the low $\hat{\Gamma}^{\text{all,fb}}$ on cross-context targets at this surface relative to the design-stage surface, controlling for generator/reviewer family. The theory predicts that, by the capacity characterization (§2), the representation ceiling $\eta^*_{\text{codereview},\alpha}$ is bounded by the fraction of defective tasks that land in buckets no correct task ever produces, which is small whenever the representation lacks a separator for α. The capacity characterization bounds zero-FP deterministic recall η^*. The reported $\hat{\Gamma}^{\text{all,fb}}$ is a mixed deterministic + stochastic feedback-based coverage estimate, so it is not itself subject to the zero-FP capacity theorem. The empirical prediction is weaker but operationally important: when a representation does not expose the target, calibrated mixed gates over that representation should show lower reliable coverage, unless they pay for coverage with false positives or import additional information.

The empirical pattern tracks the prediction. `review_design`, operating on a richer representation that exposes design-to-plan trace, also catches 92.8% of INCOHERENT defects, the cross-context class by definition. The code-stage and file-scoped gates' low coverage on the same targets (the spread above) is not a gate-side failure; they are surfaced against a representation that cannot expose those targets. The SYSTEMATIC cell is the better-powered codereview cross-context result and is what this prediction rests on; the OMISSION cell ($n_{\text{eff}} = 17.3$) is directionally consistent with a wider interval, and the INCOHERENT cell ($n = 1$ caught, 0 missed) is uninformative on its own.

8.3 The remedy

This approach directs the designer toward representation change, not gate-quality improvement. Adding more zero-FP deterministic file-local review over the same representation cannot solve the cross-context target. Stochastic file-local review may raise observed coverage at a false-positive cost, but representation change is required to raise the guaranteed deterministic ceiling for that target class. The remedy is to introduce a surface whose representation *does* expose cross-context state. Concretely:

- A **design-level trace** surface: `review_design` already operates on this representation and shows $\hat{\Gamma}^{\text{all,fb}}$ in the 88–95% range for all three target classes. Design's coverage advantage on the better-powered cross-context cells is the strongest single statement of the surface-target effect in this dataset.
- A **dependency graph** surface: a representation whose buckets separate cases by inter-module consistency would expose architectural-coherence predicates that single-file review cannot reach.
- A **task-level invariant** surface: a representation that aggregates cross-file artifact properties into a per-task summary would expose plan-to-implementation drift.

Each of these is a different Y_k, not a different gate over the same Y_k. The empirical pattern in the dual-voted matrix above is the framework's diagnostic prediction made visible: when you control for adjudication noise (via dual-voting), the surface-induced coverage spread becomes the dominant signal.

8.4 The csPCa analogue

The companion medical paper documents an architectural twin of this finding. The same anatomical-tail gate family that achieves $\hat{\eta}_{\text{tail}} = 1$ for its own target produces $\hat{\Gamma} = 0.035$ on csPCa+, a clinical-outcome target the gland-shape representation does not expose. The decision-curve net benefit of using the gates against csPCa is *negative* (Δ NB $= -0.024$ with ISUP severity weighting). The medical case and the software `codereview` case are different domains with the same diagnosis: the surface fails the target because the representation does not expose it. The remedy in both is representation change, not more gates over the same representation. The medical companion's Phase-2 extension then executes that remedy on the same cohort: a literature-derived ADC predicate over the joint (mask, intensity) representation reaches held-out $\hat{\rho} = 1$ on csPCa+ (a finite-sample pilot, $n = 152$), with catches above the marginal union of the existing surfaces — the constructive direction realized on the very target that produced the null.

9 What the Errors Actually Are

The surface-target result above says *where* a gate can catch a defect. The second headline says *what kind of defect* the gates catch, and the answer is the practical reason the whole surface-engineering program pays off: the errors fall into a small, structurally predictable taxonomy rather than a long tail of arbitrary mistakes.

Of the 760 rejections the public instrument classifies into the OMISSION/SYSTEMATIC/INCOHERENT taxonomy, 50.8% are SYSTEMATIC (386), 40.5% are OMISSION (308), and 8.7% are INCOHERENT (66).[5] Read memorably: 91.3% of classifiable rejections are *predictable structural failures*, omission or systematic-

[5]Denominators: the public instrument classifies 760 of 1,722 candidate `NEEDS_REVISION` rows into the taxonomy; the within-classifiable percentages are over the 760. "Classifiable" is the live-DB filter sense of the descriptive-process appendix, not the semantic-validity adjudication of the precision section. The private 97-day instrument reports a higher classifiable share with the same OMISSION/SYSTEMATIC-dominated shape (descriptive-process appendix).

constraint violation, and the remaining 8.7% is cross-context incoherence. The headline practitioners care about is the negative one: **wild hallucinations are not the dominant failure mode.** The bulk of what these gates reject is an artifact that is missing a required part or that violates a stated constraint, not an artifact that invented something untethered to reality.

This is why the surface-engineering lever works. *Omission and systematic-constraint violations are exactly the failure classes that expose deterministically-checkable predicates: section presence, acceptance-criterion presence, scope conformance, dependency conformance.* They are the failures a richer representation can turn into a structural check (§2). Incoherence is the residual that genuinely needs cross-context state to detect, which is the same surface-target boundary the alignment result traces, file-scoped representations cannot reliably expose it (the surface-target-alignment section). The taxonomy and the alignment spread are two views of one fact: the representation determines which defects are even visible, and most defects are of the visible kinds.

Decomposition and the incoherence residual (observational). One further pattern bears on that residual. The private 97-day snapshot recorded a roughly $2\times$ lower incoherence rate on decomposed multi-agent release arcs ($\approx 10\%$) than on single-agent feature arcs ($\approx 20\%$). This is suggestive, not causal. The comparison is confounded by task type, difficulty, review intensity, and selection effects (which arcs get decomposed in the first place). It lives only on the private snapshot, because the public instrument does not currently carry the arc-type tag at the same resolution, and it has not been replicated on the live-DB instrument. We report it as a snapshot observation that motivates a controlled follow-up (the limitations section), not as evidence that decomposition causes lower incoherence.

10 Cross-Domain Generalization

Software is the only domain treated in full here. The framework, though, is not about software: a verification surface is a representation and a target, and that construction transfers wherever a pipeline emits artifacts. Two companion papers carry the full treatments, and the gesture below records that the same lens works in those domains without re-deriving their analyses.

Medical image segmentation. The medical companion (Rothrock, 2026b) builds the same gate construction over mask geometry rather than reviewer text, and finds the same pattern: the mask surfaces verify artifact-level failures, fail on out-of-channel csPCa where the representation does not expose the target, and improve constructively when an ADC predicate is added over a joint (mask, intensity) representation. Holding the gates fixed while only the upstream model weakens makes the surface reject more, not less, which restates this paper's thesis that reliability is a property of the surface, not the model behind it.

Language models. The language companion (Rothrock, 2026c) carries the same construction to the token and structured-output layers, where schema, disagreement, token uncertainty, and intermediate representations behave as different surfaces. The diagnostic holds there too: a deterministic gate's alignment inverts across error definitions, so no single gate is a universal correctness monitor; and the constructive direction reappears as composition of low-overlap, zero-false-positive surfaces, one layer below the artifacts this paper measures.

11 Discussion

11.1 When the framework applies

A system falls in scope exactly when it has a verification surface: some artifact representation Y_k exposes target-aligned predicates. That is a property of the system as built, not a fixed property of the underlying task. A bare forward pass, with no target-aligned output and no intermediate verifier, sits outside the compositional view. One model call whose output passes through a verified artifact (a schema validator, a

constrained decoder, a type checker) is a $K = 1$ surface and is in scope; the coding-agent pipeline is the $K \geq 2$ case, where cross-stage composition (§2) and compounding apply. Whether a system has a surface at all is a design choice, and the software pipeline studied here is one such choice made deliberately.

Where no intermediate surface exists, reliability is genuinely a model property, and the right tools are alignment, RLHF, scaling, and interpretability. The framework complements that program rather than replacing it. Two reliability mechanisms are worth keeping distinct. *Ensemble averaging* reduces variance over an atomic operation and introduces no intermediate surface. *Compositional verification*, the mechanism formalized here, eliminates failures deterministically over a pipeline with an explicit surface at each stage. The companion papers compare the two empirically, against TTA-softmax and conformal baselines in the medical setting (Rothrock, 2026b) and against LLM-judge and self-consistency baselines at the language layer (Rothrock, 2026c); in every case the deterministic surfaces and the ensemble baselines fire on different cases and align with different targets. They are complementary, not interchangeable.

11.2 The constructive direction

Refinement (§2; Formal Supplement) formalizes the constructive lever. A task with low capacity at its input representation can often be transformed into a higher-capacity pipeline by introducing a constrained intermediate representation, provided that representation refines the input in a target-relevant way. The coding-agent pipeline illustrates this structurally through its prompt-to-test progression: each generated stage exposes deterministic predicates the prompt surface never could. The medical companion (Rothrock, 2026b) realizes the identical design pattern in a non-revision architecture, where mask geometry exposes anatomical-tail predicates that raw pixels do not. The same qualifier governs both: only refinements that produce target-relevant pure-positive buckets raise the ceiling, and format constraints alone do not.

Composition is the second constructive lever, and it takes two forms. Across stages it is the chain rule (Formal Supplement): a defect must escape every stage to survive, so when stages catch different failures the pipeline's escape rate falls below any single stage's. That is what the prompt-to-test progression above does, compounding rather than merely adding a fourth gate; the medical companion (Rothrock, 2026b) tests the identity directly in a non-revision architecture. On a single artifact it is the union of disjoint target-aligned surfaces, which raises coverage at preserved purity: the language companion (Rothrock, 2026c) wrings a both-rise from zero-false-positive lexical and cross-model checks across three hard domains. Where refinement changes what one surface can see, composition harvests what several catch together, and both are design moves a surfaceless system cannot make.

11.3 The diagnostic direction

Some surfaces cannot be repaired by adding gates at all, and naming them in advance is the framework's strongest practical use. When no property of the artifact separates failures from good cases under the operative target (§2, the alignment diagnostic), no gate over that surface delivers deterministic recall, and the fix is a different artifact rather than more gates. Outcomes that improve anyway on such a surface improve through sampling effects, distribution shifts, or cohort shifts, not gate quality. The software realization shows the live version of this: `codereview` over a single-file representation aligns poorly with the cross-context targets the design surface catches, and stacking more file-scoped checks would not close the gap. Many production AI deployments add verifiers without first asking whether the surface's buckets are pure under the target. The capacity characterization supplies that design check, and refinement supplies the design action when the check fails.

11.4 Relation to prior work

The related-work discussion (§1) places the framework against the verification literature; what distinguishes the approaches at the design level comes down to where each one acts. Behavioral verification and graph-based orchestration (Fang, 2026; LangChain, 2026) act on the agent's execution rather than its generator, and the framework specifies *where in a pipeline* such checks can pay off. LLM-as-judge work (Gu et al., 2024; Zheng et al., 2023) targets stochastic gate quality, and the capacity reading predicts when judge

reliability turns decisive (low-capacity stages) and when it stays marginal (high-capacity stages). Generator-side compilation and constrained generation (Khattab et al., 2024; Willard & Louf, 2023) shift quality onto the producer; the surface view instead holds the generator fixed and asks what the gate's representation can expose. Rough set theory supplies the mathematical core (in its discrete form here; the measure-theoretic generalization is developed in the Formal Supplement), credited cleanly in its subtraction table. Throughout, the framework contributes the use of that machinery as an AI-pipeline design primitive, which the empirical results above instantiate at production scale.

The limitations below make the scope boundary precise. Under a fixed information boundary the surface guarantees consistency across projections of intent, never correspondence to intent itself. Tool calls, retrievals, and oracle escalations all change that boundary, and each new input then carries its own surface (the oracle channel, Formal Supplement).

12 Limitations and Open Work

12.1 Limitations

LLM-judge reliability is largely but not fully addressed. All adjudication is cross-model on the full $n = 1{,}570$ sample with a human-anchored sub-sample of $n = 50$ ($\kappa = 0.812$ rejection / 0.562 acceptance-shadow; human matched consensus on $\frac{23}{24}$ cases). The two LLM adjudicators can still plausibly share blind spots that a larger human review would catch, most acutely on the ambiguous-or-disagreement subset; replication on a second engineer's pipeline (Priority 5) is the strongest single mitigation.

Feedback-based, not artifact-grounded, verdict-blind, shadow audit. The current `gate_checks` schema preserves reviewer feedback but not raw artifact snapshots for every approved row, so the shadow audit re-judges from what the reviewer wrote, not what the reviewer was looking at. A full artifact-grounded, verdict-blind audit requires the schema upgrade in Priority 1.

No post-deployment defect linkage. The most consequential gap. We do not currently link `gate_analytics.db` to the issue tracker, git-blame, or production-incident logs. The current evidence suggests high feedback-based coverage for several broad target classes; we cannot yet quantify artifact-grounded missed defects or post-deployment escapes. The companion medical paper (Rothrock, 2026b) provides indirect evidence that complementary detection surfaces work in a related setting (cross-stage co-rejection 1.49%, Wilson 95% CI [0.51%, 4.30%] across three medical domains), but the equivalent claim is not quantitatively supported in software.

Mixed-gate Γ, not deterministic-only η. The reported $\hat{\Gamma}^{\text{all,fb}}$ measures the deployed mixed deterministic + stochastic gate family. A deterministic-only $\hat{\eta}$ requires separating schema/structural/compile/test predicates from stochastic LLM judgments, which the current schema does not tag by source (Priority 3).

Private-vs-public instrument gap. The 5,109/1,450/165 figures come from the calibrated private regex extractor; precision and coverage estimates use the public LLM-assisted analyzer. The two instruments classify the same production process through different pipelines and produce somewhat different aggregate counts (full reconciliation in the methods-detail appendix). We deliberately chose the reproducible instrument as the public methodology; the precision cost is the price of replicability.

Single engineer's prompt templates. All measurements come from the author's pipeline. The public analyzer is designed to run on other engineers' logs without prompt-template-specific tuning, but no such replication is reported here. The medical companion's external consistency (External consistency, above) is same-author work and does not supply the independent replication this pipeline still lacks.

Population-vs-finite-sample. The capacity characterization (§2) is a population statement. Empirical $\hat{\rho}$ and $\hat{\Gamma}$ are finite-sample estimates that approach it asymptotically; we do not claim formal $\rho = 1$ for any deployed gate.

Specification vs. intent. Under a fixed information boundary, the theory checks fidelity to represented specifications, not original intent lost before the artifact surface. The pipeline catches deviations between artifacts and their explicit specifications; it does not catch a specification that itself misrepresents user intent. The alignment problem at the level of original intent remains outside scope.

12.2 Operational learning and the deployed capability set

The pipeline's four gates are three-state: deterministic checks fire first, the stochastic LLM reviewer renders a verdict, and unresolved cases may escalate to a human. Escalation is one operational mechanism by which $\mathcal{D}_k$ can grow over time (the oracle channel, Formal Supplement): when the team *is able* to encode a human decision as a new deployed predicate (schema rule, structural check, lint rule), $\mathcal{D}_{k(t)}$ acquires a member, and adding such retained predicates can increase the capability-restricted ceiling $\eta^*_{\mathcal{D}}$ for the targets they address. When the resolution is not expressible at the current representation, the escalation produces only a one-off decision and instead diagnoses a representation gap; changing the full ceiling η^* requires changing Y_k itself. The current schema records escalation *counts* but not the rule-version timeline, oracle-resolution records, or decision-source tags needed to observe escalation→new-predicate→reduced-escape as a time-series effect, so the oracle-channel growth argument (the Formal Supplement) remains the formal claim this paper does not directly observe (the negative-inventory audit is retained in the author's source tree, not this package; schema upgrade in Priority 6).

12.3 Open empirical work (priority list)

The following are addressable gaps with proposed mechanisms, ordered by criticality for submission-grade validation.

[**Priority 1**] **Schema upgrade for artifact-grounded, verdict-blind shadow audit.** Persist `artifact_snapshot` (or `artifact_ref` to immutable storage), `review_prompt`, `raw_gate_response`, `parsed_decision`, `parser_confidence`, and `artifact_hash` with each gate check. Enables artifact-grounded, verdict-blind re-review, replacing the current feedback-based shadow audit. *This is the highest-priority change because it converts rejection-report-validity precision into artifact-grounded semantic precision.*

[**Priority 2**] **Independent human audit ($n \approx 200$, two unaffiliated reviewers).** The current author-adjudicated anchor ($n = 50$) is in the adjudication appendix. The next pass should expand to $n \approx 200$ with at least two unaffiliated human reviewers, enabling human-vs-human κ alongside human-vs-LLM κ.

[**Priority 3**] **Schema upgrade for deterministic-vs-stochastic decision-source tagging.** Tag each gate check's decision components by source: deterministic schema rule, deterministic structural check, deterministic syntax/parse check, stochastic LLM judgment, deterministic test result. Enables the deterministic-only $\hat{\eta}$ estimate.

[**Priority 4**] **Post-deployment defect linkage.** Link `gate_analytics.db` to the issue tracker, git-blame, and production-incident logs to compute per-release escaped-defect counts. Closes the largest validation gap. *Without this, the paper cannot claim that the gates reduce escaped production defects.*

[**Priority 5**] **Replication on a second engineer's pipeline.** Run the public analyzer on another practitioner's Claude Code logs and report the aggregate findings. The strongest single defense against the "single engineer's prompt templates" limitation.

[**Priority 6**] **Schema upgrade for oracle-channel time series.** Persist rule-version IDs, oracle-resolution links, and gate-check chronological order. Enables direct empirical observation of the operational-monotonicity claim.

[**Priority 7**] **Survey-design intervals on coverage.** Replace the descriptive Wilson intervals on n_{eff} with Taylor-linearization or stratified bootstrap intervals that propagate the expansion-weight uncertainty.

13 Conclusion

A coding-agent pipeline is not a model with safety nets. It is a sequence of artifacts at progressively richer representations, and each representation exposes a different set of target-aligned predicates a gate can act on. The catch is structural: what a stage can verify is fixed by what its artifact reveals, not by how clever the gate is. The verification-surface framework (§2) makes this precise, defining each surface's deterministic capacity as the fraction of failing cases whose artifact lands where no good case ever does. The second half of the paper measures mixed-gate, feedback-based coverage, a proxy for that capacity, across a 97-day production pipeline.

The production data bears it out. Moving from prompt through plan, design, and code is refinement in the framework's sense, and the available capability set and the observed coverage both climb as the representation exposes the target. The design-stage gate, reading the richest cross-context representation, dominates the narrower-representation gates on the same omission and systematic targets, with the remaining cells pointing the same way on thinner evidence. The process record tells the shift-left story from a second angle: plan-stage rejections are the largest single-stage share of analyzable rejections, and decomposed multi-agent releases showed roughly half the incoherence rate of single-agent ones (observational, not causal). Precision itself rests on dual-voted cross-model adjudication with an author-anchored check, reported under three denominators because three different questions about the instrument deserve three different answers.

One measurement stays open. The framework predicts that the post-deployment escape rate is governed by the separability the deployed surfaces achieve against the targets that drive production behavior, and closing that gap empirically needs the schema upgrade and defect-linkage ETL laid out in the limitations. It is the natural next surface to instrument.

The model is one node in the pipeline, not the seat of its reliability. There is no global model trust. There are only local verification surfaces, and reliability composes through their topology: what each representation exposes, which deterministic predicates are deployed against each target, and where escalation can turn a resolved decision into a durable, artifact-local check. *In a production coding-agent pipeline, the representation determines the observable failure modes: design-stage surfaces expose broad omission, systematic, and incoherence targets, while code- and file-stage surfaces expose only narrower local ones.* The medical and language companions carry the same claim into image segmentation and the language-model layer, so the lens is not an artifact of one pipeline. Reliable AI systems are not trusted into existence. They are engineered, by building artifacts whose structure separates failing cases from good ones. That is the contribution.

Appendix A Methods and Precision Detail

This appendix carries the method and precision mechanics that §3, §5, and §6 summarize: the model-role identifiers and the measurement-window reconciliation behind the dual-instrument framing (§3 and §5), the two false-positive notions and the cross-model adjudication protocol (§5), and the per-gate precision and inter-rater detail behind the three headline precision numbers (§6).

Appendix A.1 Model roles and identifiers

The body (§3) notes that the generator transitioned from Sonnet-tier to Opus-tier mid-window while the reviewer model held constant. The exact identifiers and roles over the measurement window are:

Role	Model identifier	Date range	Used for
Pipeline generator (early)	`claude-sonnet-4-6`	Window start → mid-window	Plan / design / code / test generation
Pipeline generator (late)	`claude-opus-4-7`	Mid-window → window end	Plan / design / code / test generation
Production reviewer	Gemini family (held constant)	Window throughout	The four gate checks (`review_plan`, `review_design`, `review_code`, `codereview`)
Public analyzer classifier	`gemini-2.5-flash`	Post-window analysis	OMISSION / SYSTEMATIC / INCOHERENT taxonomy classification of rejection text
Adjudicator A	`claude-sonnet-4-6`	Post-window analysis	Dual-voted adjudication (rejection-precision + acceptance-shadow)
Adjudicator B	`gemini-2.5-flash`	Post-window analysis	Dual-voted adjudication (rejection-precision + acceptance-shadow)

The generator transition from Sonnet 4.6 to Opus 4.7 is recorded only at the model-identifier level; the pipeline orchestration, gate prompts, and reviewer model held constant across the transition. The production reviewer's exact Gemini identifier is held constant over the window; in the Zenodo aggregate package the identifier is captured per-row in the redacted `gate_analytics.db`. The two adjudicator models are deliberately drawn from different model families so that LLM cross-rater agreement (§6) is not measuring within-family consistency.

Appendix A.2 Measurement-window mechanics and the dual instrument

The original measurement window covers 97 consecutive days of production operation. Under the calibrated private analyzer instrument, the snapshot contains:

Quantity	Count
Gate checks	5,109
Private-instrument pre-correction analyzable rejection events	1,450
Shipped releases	165

All artifacts are real working code that shipped to production after passing the pipeline. The window is a slice of an ongoing system, not its full lifetime.

The analyzer database has since been reprocessed with the public reproducible analyzer. This paper refers to it throughout as the *public-instrument live DB*: the `gate_analytics.db` instance that the public

open-source analyzer (`gate_analyzer.py`) produces from the author's session logs. "Public" describes the analyzer, not the database content. The live DB on the author's machine contains production prompts and code and is not publicly hosted; only the redacted aggregate is in the Zenodo package. The current public-instrument live DB reports 5,519 gate checks and 760 *analyzable rejection events* under the explicit live-DB filter `decision = NEEDS_REVISION AND error_class IS NOT NULL AND error_class != API_ERROR` (the reproducibility appendix). "Analyzable" here means the database-filter sense; semantic-validity adjudication is the separate process in §6. Because the snapshot and live DB come from different analyzer instruments (calibrated private regex versus public LLM-assisted classifier) and not merely different filters, this paper reports both and standardizes precision and recall denominators on the public-instrument live-DB definition.

There is a deliberate dual-instrument history behind those two count sets. The earliest paper numbers came from the author's private hand-tuned regex analyzer, calibrated to the stable structured-output formats of the author's prompt templates. That instrument is highly precise for this specific signal source but not portable. The public analyzer uses a more general extraction pipeline with Gemini-assisted classification so that other engineers can run the method on their own Claude Code logs without prompt-template-specific tuning. The two instruments measure the same underlying production process through different classification pipelines and therefore produce somewhat different aggregate counts. The LLM-assisted public instrument is less precise on the author's specific data but is the reproducible methodology used for public precision and recall estimates. The parser fix detailed in the extraction-artifact appendix closes one concrete `review_design` misclassification pattern but does not make the public instrument identical to the private one.

Appendix A.3 Two false-positive notions, clearly separated

The paper measures two distinct false-positive quantities, and the difference matters for what the headline $\hat{\rho} = 0.965$ claims.

(a) Artifact-grounded false positive. A rejection that did not require any actual fix to the artifact: the gate demanded an unnecessary, impossible, or out-of-scope change. This is the framework-level definition (a rejection event whose task-case $\tau \notin L_\alpha$). It can only be measured by re-reading the original artifact alongside the reviewer's text. The current `gate_checks` schema preserves reviewer feedback but does not preserve raw artifact snapshots for every rejection, so artifact-grounded false-positive rate is *not directly measurable from the current schema.*

(b) Rejection-report false positive. A rejection whose stated rationale is incoherent, non-actionable, or out-of-scope on its face, judged from the rejection text alone, with no recourse to the artifact. This is the quantity §6 measures via dual-voted adjudication, and is what the $\hat{\rho} = 0.965$ figure estimates. It is a strong proxy for gate quality but is *not* the artifact-grounded false-positive rate of (a).

Throughout the paper, precision denominators based only on reviewer text should be read as *rejection-report-quality denominators* unless the original artifact was available to the adjudicators. The limitations section (Priority 1) describes the schema upgrade required to convert (b) into (a).

Appendix A.4 Cross-model adjudication protocol

The expanded audit uses two independent adjudicators: Claude (`claude-sonnet-4-6`) and Gemini (`gemini-2.5-flash`). Each adjudicator rates the same case without seeing the other model's vote. For rejection-precision cases, the collapsed labels are `genuine`, `false_positive`, and `ambiguous`. For approval-shadow cases, the collapsed labels are `true_negative`, `false_negative`, and `ambiguous`. Consensus is assigned only when both adjudicators agree. Disagreements are recorded as `ambiguous_by_disagreement` and excluded from clear-case denominators. The expanded audit contains 1,570 dual-voted cases: 770 rejection-precision rows and 800 acceptance-shadow rows.

The protocol sub-classifies false positives into *substantive* (reviewer was actually wrong), *parser artifact* (analyzer misclassified the gate verdict), *workflow artifact* (analyzer captured a pause/status message), and *api error* (infrastructure failure rather than gate decision). The parser/workflow/api categories measure analyzer-pipeline auditability gaps, not rejection-report quality.

Appendix A.5 Substantive vs. extraction false positives

The dual-voted rejection-precision sample contains 770 rejections from `gate_analytics.db`, stratified by gate (target 200 per gate; the `codereview` population contained 170, all included). The per-gate split of genuine, ambiguous/disagreement, and the four false-positive subclasses is:

Gate	n	Genuine	Ambig./Disag.	Substantive FP	Parser artifact	Workflow	API error
`review_plan`	200	187	6	0	4	2	1
`review_design`	200	174	6	0	14	1	5
`review_code`	200	155	27	18	0	0	0
`codereview`	170	34	20	2	28	71	15
Overall	**770**	**550**	**59**	**20**	**46**	**74**	**21**

The 161 raw false positives across the dual-voted sample split as: 20 *substantive* (reviewer reasoning was actually wrong; most concentrated in `review_code`), and 141 *extraction artifacts*, pause/status/control messages and wrapper/body verdict mismatches that the analyzer captured as gate decisions rather than as actual review verdicts (46 parser artifacts concentrated in `review_design` and `codereview`; 74 workflow artifacts concentrated in `codereview`; 21 infrastructure-failure API errors). The 20 substantive false positives are the non-genuine rows in the rejection-report-quality denominator (per §5); together with the 550 genuine rows, they form the substantive clear-case denominator of 570 used for $\hat{\rho} = \frac{550}{570} = 0.965$. The other 141 are analyzer-pipeline noise, not gate-decision quality; they shift end-to-end precision but not substantive precision. The extraction-artifact appendix documents the taxonomy in more detail for anyone re-running the public analyzer on their own logs.

Appendix A.6 Per-gate precision

The three headline precision denominators (§6) decompose by gate as follows. The source data is `data/kappa_summary.json` (full dual-voted, $n = 770$) and `data/precision_denominators_v2.json`.

Gate	Substantive clear-case	End-to-end clear-case	Conservative all-sample
`review_plan`	1.000 187/187; [0.980, 1.000]	0.964 187/194; [0.927, 0.982]	0.935 187/200; [0.892, 0.962]
`review_design`	1.000 174/174; [0.978, 1.000]	0.897 174/194; [0.846, 0.932]	0.870 174/200; [0.816, 0.910]
`review_code`	0.896 155/173; [0.842, 0.933]	0.896 155/173; [0.842, 0.933]	0.775 155/200; [0.712, 0.827]
`codereview`	0.944 34/36; [0.819, 0.985]	0.227 34/150; [0.167, 0.300]	0.200 34/170; [0.147, 0.266]

The two design-stage gates (`review_plan` and `review_design`) reach 1.000 on the substantive denominator: every consensus-clear rejection from those gates was a substantive concern. The substantive-vs-end-to-end gap of 19 pp is the analyzer-instrument gap, dominated by `codereview`'s heavy workflow-artifact tail (71/170 codereview rows are pause/status messages misclassified as rejections; full diagnosis in the extraction-artifact appendix). The conservative all-sample denominator treats disagreement and ambiguous rows as non-genuine, the lower bound under the strictest reading.

After dual-voting on the rejection side, the substantive precision claim is $\hat{\rho} = 0.965$, substantially higher than the prior single-rater Claude estimate of 0.896 (n=162). The shift is driven by `review_plan` and `review_design` reaching 1.000 in the larger sample (where the prior estimates of 0.870 and 0.818 reflected single-rater noise), not by any change in the deployed pipeline.

Appendix A.7 Inter-rater agreement

Cohen's κ between Claude and Gemini adjudicators differs substantially between the two strata: agreement is much stronger on what counts as a genuine rejection than on what counts as a missed defect under an APPROVED row.

Gate	Rejection-precision κ	n	Acceptance-shadow κ	n
`review_plan`	0.688	200	0.477	200
`review_design`	0.881	200	0.535	200
`review_code`	0.562	200	0.710	200
`codereview`	0.744	170	0.387	200
Overall	**0.812**	**770**	**0.562**	**800**

The rejection-precision stratum reaches $\kappa = 0.812$ overall, just inside the Landis–Koch "almost perfect" band (Landis & Koch, 1977), reasonably described as substantial-to-almost-perfect. The acceptance-shadow stratum sits at $\kappa = 0.562$, in the moderate range. The asymmetry has a structural reading: when a reviewer issues a structured rejection with explicit findings, a second LLM adjudicator can recognize whether the findings constitute a real concern; when a reviewer approves silently, deciding what defect *should have been* flagged requires more inference and produces more disagreement.

`review_code` has the weakest rejection-side κ (0.562), while `codereview` has the weakest acceptance-shadow κ (0.387, below the conventional 0.41 boundary between Landis–Koch "fair" and "moderate" agreement). The two findings have different readings. The `review_code` rejection-side κ is depressed by disagreement on *which* collapsed verdict to apply when reviewers raise multiple findings of mixed severity; it is the noisiest of the four gates at distinguishing genuine vs. false-positive rejections. The approval-side `codereview` figure is the more important one because it is consistent with the surface-target-mismatch story developed in §8: when the representation does not expose cross-context state, even the adjudicators struggle to agree on what *should* have been flagged. Where adjudicators disagree, cases are marked `ambiguous_by_disagreement` and excluded from clear-case $\hat{\Gamma}^{\text{all,fb}}$ estimates.

Appendix A.8 Definitions reconciliation

The earlier 1,450 pre-correction rejection-events figure was computed by the calibrated private instrument; the current live database is the public-instrument live DB (above). Under the explicit filter, it yielded 852 analyzable rejection events before the parser rerun and 760 after. We report both:

- **Original 97-day snapshot** (private instrument): 5,109 checks, 1,450 pre-correction rejection events, 165 shipped releases.
- **Current public-instrument live DB**: 5,519 checks, 760 analyzable rejection events under the explicit filter.

Precision and recall estimates use the live-DB denominator after parser fix. The extraction-artifact appendix documents the parser-fix forensics: what was fixed, what remains, and why the substantive precision is unaffected by remaining extraction artifacts.

Appendix B Adjudication Details

This appendix gives the full author-anchored breakdown that the precision section summarized.

Appendix B.1 Sample composition

We drew a stratified $n = 50$ sample from the dual-voted set: 12–13 cases per gate, 26 rejection-precision rows + 24 acceptance-shadow rows. The sample was deliberately enriched 4 × toward Claude/Gemini disagreement so the human verdict is informative on the cases where the LLMs split. The author manually adjudicated each case with the same vocabulary and reasoning protocol used by the LLM adjudicators. Detailed case-level notes are at `data/CONSENSUS_BREAKDOWN.md`.

This is a sanity-check, not independent ground truth. The reviewer is the system designer, the sample is small, and it was deliberately enriched for disagreements. A larger independent audit with at least two unaffiliated reviewers (the limitations section, Priority 2) is required for submission-grade validation.

Appendix B.2 Author matches LLM consensus on consensus cases

Subset	n	Human matches LLM consensus	Notes
`rejection_precision`, LLM-consensus subset	12	12 / 12 = 100.0%	No human disagreements with consensus
`acceptance_shadow`, LLM-consensus subset	12	11 / 12 = 91.7%	One human-vs-consensus disagreement
Combined consensus subset	**24**	**23 / 24 = 95.8%**	—

When both LLMs agree, the human agrees with that consensus 95.8% of the time. The author-adjudicated anchor supports the consensus methodology on this sample, but does not establish independent human ground truth (the human reviewer is the system designer, the sample is small, and it was deliberately enriched for disagreements).

Appendix B.3 On disagreement cases, the human breaks the tie toward Gemini 2:1

Subset	n	Sided with Gemini	Sided with Claude	Sided with neither
`rejection_precision` disagreements	14	9	4	1
`acceptance_shadow` disagreements	12	8	4	0
Total	**26**	**17 (65.4%)**	**8 (30.8%)**	**1 (3.8%)**

The asymmetry is consistent across both strata. Per-gate human-vs-LLM agreement (Cohen's κ on the $n = 50$ sample):

Gate	Human vs Gemini κ	Human vs Claude κ	Difference
`review_plan`	0.540	0.500	+0.040
`review_design`	0.667	0.414	+0.253
`review_code`	0.782	0.547	+0.235
`codereview`	0.880	0.469	+0.411
Overall	**0.731**	**0.510**	**+0.221**

Gemini agrees with the author more than Claude does on every gate, with the largest gap at `codereview` ($\kappa =$ 0.880 vs 0.469). *Caveat: because the production reviewer is also Gemini-family, the Gemini–author agreement reflects alignment on this dataset, not independence from the production reviewer's inductive biases. The result is real but does not by itself establish that Gemini is a better adjudicator in general; it establishes that Gemini matches the author on this pipeline.* Subject to that caveat, if a future analysis is constrained to a single LLM adjudicator, Gemini 2.5-Flash is the more author-aligned choice on this dataset.

Appendix B.4 Sensitivity check: author-anchored population projection

We report this as a *sensitivity analysis*, not a population estimate. The full 770-case rejection-precision stratum has 711 clear LLM-consensus rows (92.3%) and 59 ambiguous-or-disagreement rows (7.7%; 57 split-disagreement plus 2 consensus-ambiguous; source: `data/kappa_summary.json`, `consensus_counts` field). Combining the per-subset author-anchored substantive precision on the population fractions:

$$\hat{\rho}^{\text{substantive, proj}}_{\text{human-anchored}} \approx 0.923 \times 0.965 + 0.077 \times 0.692 \approx 0.94$$

where 0.692 is the author-anchored substantive precision on the disagreement subset (9 genuine / 13 clear) from the deliberately disagreement-enriched $n = 26$ rejection-precision human sample. The projected human-anchored substantive precision is within roughly 2 percentage points of the LLM-only 0.965. The raw human-anchored $\hat{\rho}$ from the $n = 26$ sample alone is 0.640; reading it as the population estimate would over-correct, because the sampling deliberately oversampled LLM disagreements $4\times$ to maximize information per case. The projection assumes the $n = 13$ clear-verdict disagreement sample is representative of the full 59 ambiguous-or-disagreement population rows, an assumption that the larger human audit in the limitations section is required to test.

Appendix B.5 Triple-disagreement cases

Only 2 of 50 sampled cases produced a verdict that disagreed with both LLMs. Both involved single-step judgment calls:

- **Case 13** (`review_plan` / acceptance_shadow): both LLMs called a structurally complete plan a `false_negative`; the human called it `true_negative` with high confidence, reasoning that "the plan is correct and dependencies are recorded."
- **Case 46** (`review_design` / rejection_precision): Claude said `genuine`, Gemini said `false_positive`, and the human called the case `ambiguous` because the review identified a real plan deviation but framed it as an improvement.

The two cases are documented at `data/CONSENSUS_BREAKDOWN.md`.

Appendix C Extraction Artifacts in the Live DB

This appendix records the extraction-artifact taxonomy in the 770-row dual-voted sample (the precision section) so anyone re-running the public analyzer on their own logs can interpret similar rows in their own data. The substantive precision claim ($\hat{\rho} = 0.965 = \frac{550}{570}$) and the coverage matrix $\hat{\Gamma}^{\text{all,fb}}$ both use the dual-voted LLM adjudication consensus and are therefore *unaffected* by extraction artifacts in the analyzer's raw output; the end-to-end precision claim (0.774) and the per-gate `codereview` end-to-end figure (0.227) reflect the analyzer's behavior on the live DB and would shift if the analyzer is updated.

Appendix C.1 Artifact taxonomy

In the dual-voted sample, 46 parser artifacts and 74 workflow artifacts appear alongside 21 infrastructure-failure API errors:

- **Parser artifacts (46 total)**: the wrapper/header said `NEEDS_REVISION` while the body concluded `APPROVED`. Concentrated in `review_design` (14), `codereview` (28); also `review_plan` (4). The `review_design` subclass was addressed by a two-phase body-decision fix in `gate_analyzer.py`.
- **Workflow artifacts (74 total)**: pause/status/control messages emitted mid-review (e.g., JSON bodies with `status: "code_review_complete_*"` plus a `next_step_required` flag and no findings) captured as `NEEDS_REVISION`. Concentrated in `codereview` (71 of 74, 96%).
- **API errors (21 total)**: infrastructure-failure rows tagged `error_class = API_ERROR`. Excluded by the live-DB filter from the measurement-window subsection (formal definition in the reproducibility appendix).

Appendix C.2 Impact

Extraction artifacts shift end-to-end precision (which counts artifacts in the denominator) but not substantive precision (which excludes them by definition); they also shift coverage cell `n_eff` slightly because expansion weights use raw rejection counts. The qualitative finding in the surface-target-alignment section that codereview underperforms on cross-context targets is a *surface* phenomenon and is independent of analyzer extraction noise. The public analyzer at github.com/mrothroc/claude-code-log-analyzer has since added a workflow-status-message detector that filters these rows out at extraction time; readers re-running the post-fix analyzer on their own logs will see lower extraction-artifact counts.

Appendix D Coverage Estimator and Reproducibility Manifest

This appendix gives the extraction-pipeline definitions, the coverage estimator derivation, the live-DB filter, and the reproducibility manifest that the methods and coverage sections summarized.

Appendix D.1 Extraction pipeline

The public analysis tool (`gate_analyzer.py`) parses Claude Code session JSONL files, matches `tool_use` blocks to `tool_result` responses by `tool_use_id`, and writes per-check rows to a SQLite schema (`gate_checks`, `gate_iterations`, `gate_issues`). The analyzer, schema, classification prompts, and tests are public in the repository. The *production gate prompts* are not public; their absence is what prevents bit-for-bit replication of the production decision boundary. The redacted aggregate reports and adjudication summaries used to produce the live-DB precision and recall numbers in the precision and coverage sections are deposited separately with this Zenodo record; raw session logs and artifact snapshots are not public (they contain prompts and code from production work).

Appendix D.2 Rejection-event populations

Each gate invocation produces a verdict extracted from the reviewer's structured output: `APPROVED`, `NEEDS_REVISION`, `ESCALATE`, or null. Decisions are extracted via a regex stack covering the multiple verdict formats produced by different gate prompts (`gate_analyzer.py:323`–`gate_analyzer.py:444`). We distinguish three populations:

- *Candidate rejection row*: any public-analyzer row parsed as `NEEDS_REVISION` and included in the adjudication sample. Candidate rows may include parser artifacts, workflow artifacts, and API errors.
- *Substantive rejection report*: a candidate row whose reviewer text is consensus-clear and not a parser/workflow/API/infrastructure artifact.
- *Valid semantic rejection*: a substantive rejection report judged genuine against the adjudication rubric.

From the live-DB aggregate counts onward, an *analyzable rejection event* uses the explicit live-DB filter `decision = NEEDS_REVISION AND error_class IS NOT NULL AND error_class != API_ERROR` (this filter excludes rows with `API_ERROR` infrastructure labels and rows lacking a substantive `error_class`). The 770-row precision stratum in the precision section is sampled from candidate rows *before* the API-error filter is applied, so it contains 21 API-error candidate rows that the dual-voted adjudication identifies and excludes from the substantive denominator. The two denominators, 760 analyzable rejection events in the live DB, 770 candidate rows in the precision stratum, are distinct sampling frames; we report both explicitly rather than collapsing them.

Appendix D.3 Error taxonomy

The OMISSION/SYSTEMATIC/INCOHERENT taxonomy is assigned in two passes:

1. Regex matching against rejection text using pre-defined keyword lists.
2. Gemini classification of items the regex could not categorize, using a fixed prompt (`gate_analyzer.py:1408`–`gate_analyzer.py:1480`).

The classification prompts and definitions are part of the public repository.

Appendix D.4 Coverage estimator derivation

For each (k, α):

$$\hat{\Gamma}^{\text{all,fb}}_{k,\alpha} = \frac{w^{\text{rej}}_k \cdot c_{k,\alpha}}{w^{\text{rej}}_k \cdot c_{k,\alpha} + w^{\text{app}}_k \cdot m_{k,\alpha}},$$

where:

- $c_{k,\alpha}$ is the consensus-genuine count for target α in the rejection-precision sample at gate k;
- $m_{k,\alpha}$ is the consensus false-negative count for target α in the acceptance-shadow sample at gate k;
- $w^{\text{rej}}_k = \frac{N^{\text{rej}}_k}{s^{\text{rej}}_k}$ is the gate-level rejection-stratum expansion weight (shared across targets α at gate k), with N^{rej}_k the live-DB count of NEEDS_REVISION rows at gate k not labeled `API_ERROR` (predicate `decision = NEEDS_REVISION AND (error_class IS NULL OR error_class != 'API_ERROR')`; rows with a NULL `error_class` are kept, and the count is not filtered by α) and s^{rej}_k the count of clear (genuine + false_positive) consensus rows in the rejection sample at gate k. Unlike the null-excluding *analyzable rejection event* filter above, this population keeps NULL-`error_class` rows on purpose: the weight is gate-level, not target-specific, because the adjudicators' `consensus.violation_type` is the rigorous target classification while the analyzer's `error_class` is frequently NULL on rows the consensus correctly types, so a per-α `error_class` expansion would discard many consensus-typed catches;
- $w^{\text{app}}_k = \frac{N^{\text{app}}_k}{a^{\text{app}}_k}$ is the live-DB approval-stratum expansion weight, with N^{app}_k the live-DB count of APPROVED rows at gate k and a^{app}_k the count of clear (true_negative + false_negative) consensus rows in the acceptance-shadow sample at gate k.

The effective count $n_{\text{eff}} := w^{\text{rej}}_k \cdot c_{k,\alpha} + w^{\text{app}}_k \cdot m_{k,\alpha}$ is the sum of the two population-scaled clear-consensus expansions. The intervals reported in the coverage section are descriptive Wilson intervals on these population-scaled effective counts. They are not full design-based survey intervals: they treat n_{eff} and $w^{\text{rej}}_k \cdot c_{k,\alpha}$ as a real-valued denominator/numerator pair and apply the standard Wilson formula, which does not propagate uncertainty contributed by the expansion weights themselves. A full survey-design treatment (Taylor linearization or stratified bootstrap on w^{rej}, w^{app}) is reserved for the follow-up audit in the limitations section (Priority 7). Computation lives in `scripts/compute_gamma_paper_estimator.py` (function `compute`); the live-DB queries materialize N^{rej}_k and N^{app}_k under the gate-level predicates defined above.

Appendix D.5 Reproducibility scope

The reproducibility surface splits across two artifacts. The GitHub repository contains the public analyzer source, schema, documentation, and tests. The Zenodo record for this paper contains the redacted aggregate databases, adjudication summaries, and analysis artifacts needed to reproduce the tables in the precision and coverage sections. Raw session logs, full production gate prompts, and artifact snapshots are not public.

On GitHub (public analyzer source):

- The extraction pipeline (`gate_analyzer.py`), the schema, classification prompts for the OMISSION/ SYSTEMATIC/INCOHERENT taxonomy, and regression tests.
- A redacted reference `gate_analytics.db` with project content removed (suitable for end-to-end pipeline testing, not for reproducing the paper's tables).

On Zenodo (reproducibility package, this paper):

- The redacted aggregate `gate_analytics.db` at the live-DB filter above.
- Adjudication summaries and redacted analysis artifacts corresponding to `analysis/adjudication/` and `analysis/sdlc_expanded/` in the source tree (with prompts and feedback text redacted where they contain production content).

- The `kappa_summary.json`, `precision_denominators_v2.json`, `gamma_hat_all_v2.json`, and `AGGREGATE_REPORT.md` (human-anchor n=50) files referenced throughout the precision and coverage sections.

Reproducibility status per claim:

- *Within-stage* aggregate findings (per-gate rejection rates, error-class distributions, recovery rates, plan-stage shift-left percentages, within-stage cross-gate co-rejection): fully reproducible from the Zenodo aggregates against the public analyzer.
- *Cross-stage* gate-overlap (the ω metric on rejection sets, §2) at the session-grouped level: reproducible from the Zenodo aggregates using session-id grouping as a `task_id` proxy. *Per-task* cross-stage overlap at the resolution of the private analyzer requires `task_id` linkage that is not exposed in the public schema; the Zenodo package includes the session-grouped aggregate tables only.
- Deterministic post-conditions (structural completeness, syntax validity, schema enforcement) are fully reproducible from the schema and gate descriptions.
- Stochastic LLM-judged decisions are reproducible in spirit but not bit-for-bit: production gate prompts are not public, and decision boundaries depend on the prompt.
- Linkage to specific production incidents or git-blame is not public.

Appendix E Descriptive Process Findings

This appendix is descriptive. Each rate below is given with its numerator, its denominator, and an explicit instrument tag. Bare percentages are hard to interpret without the population they were computed against, and some of these counts shift across the private 97-day snapshot and the public-instrument live DB. Process-level findings should not be confused with the precision and coverage estimates of the precision and coverage sections; they are properties of the revision loop and the rejection distribution, not of rejection-report quality.

Appendix E.1 Live-DB process counts (public instrument)

Finding	Num.	Denom.	Rate	Denominator definition
Classifiable rejected errors (OMISSION/ SYSTEMATIC/INCOHERENT)	760	1,722	44.1%	`decision = NEEDS_REVISION`
Within-classifiable: SYSTEMATIC	386	760	50.8%	analyzable rejection events
Within-classifiable: OMISSION	308	760	40.5%	analyzable rejection events
Within-classifiable: INCOHERENT	66	760	8.7%	analyzable rejection events
Plan-stage share of analyzable rejection events	331	760	43.6%	analyzable rejection events
First-pass approval (per `gate_iterations.iteration_number = 1`)	497	942	52.8%	first-iteration gate checks
Revision recovery (rejected first-iteration tasks reaching APPROVED on a later iteration)	249	309	80.6%	first-iteration NEEDS_REVISION tasks

The classifiable-share figure of 44.1% is the public-instrument rate. The private 97-day hand-tuned regex instrument's original aggregate reports a higher classifiable share with a similar OMISSION/SYSTEMATIC-dominated shape; raw counts are retained in the author's source tree for traceability but are not the citation-grade numbers (different sampling frame and pipeline). The headline coverage and precision claims in the precision and coverage sections are computed on the public-instrument live DB and do not depend on the private-snapshot process percentages.

Appendix E.2 Structural claims not denominator-dependent

The following structural statements are stable across both instruments and rest on the data in the precision and coverage sections:

- **Errors classify into a small typed taxonomy.** Among candidate `NEEDS_REVISION` rows, the public instrument classifies 760 of 1,722 into OMISSION/SYSTEMATIC/INCOHERENT; within that classifiable subset, SYSTEMATIC and OMISSION dominate (386 and 308 rows respectively; INCOHERENT contributes 66). "Wild hallucinations" are not the dominant failure mode. Omission and systematic errors expose deterministically-checkable predicates (section presence, constraint conformance); incoherence requires cross-context state.
- **The four gate types do not catch interchangeable errors.** `review_plan` concentrates on structural and scope errors, `review_design` on internal inconsistencies and design-to-plan traceability, `review_code` on syntactic and architectural violations, and `codereview` on local file-scoped correctness. This is per-stage target-violation specialization predicted by mapping each gate to a different surface.
- **Cross-stage co-rejection at the task level is rare.** Gates at different stages show low task-level co-rejection; gates at the same stage show moderate within-stage overlap. Exact within-stage and session-grouped cross-stage co-rejection tables are in the Zenodo aggregate package.
- **Cross-class coverage spread tracks representation scope.** On the best-powered cross-surface cells (the coverage section), `review_design` exceeds the narrower code/file-stage gates by 56–70 pp on $\hat{\Gamma}^{\text{all,fb}}$; the `review_design` OMISSION vs `codereview` OMISSION comparison (39.6 pp) is directionally consistent but draws on a directional cell. This is the central pattern for the surface-target-alignment section.

Appendix E.3 Observational findings that require causal follow-up

Two findings below are observational and are flagged explicitly so they are not over-claimed:

- **Plan-stage shift-left.** 43.6% of analyzable rejection events fire at the plan stage. This is consistent with the strictness-condition mechanism (cross-stage composition, §2: early gates narrow the residual corrupt set seen by later gates), but the early-rejection distribution does not by itself prove marginal effectiveness; that requires a marginal residual-coverage measurement, which is part of the follow-up audit program in the limitations section.
- **Decomposed-vs-single-agent incoherence.** The private 97-day snapshot recorded a roughly $2\times$ lower incoherence rate on decomposed multi-agent release arcs ($\approx 10\%$) compared with single-agent feature arcs ($\approx 20\%$). The public instrument does not currently carry the arc-type tag at the same resolution, so this finding is preserved as a snapshot observation rather than a live-DB rate. It is observational and confounded by task type, difficulty, review intensity, and selection effects (which arcs get decomposed). Causal evidence would require a marginal-coverage measurement or a controlled experiment.

References

Anthropic. (2026b). *2026 Agentic Coding Trends Report*. Anthropic. https://resources.anthropic.com/2026-agentic-coding-trends-report

Anthropic. (2026a). *Claude Code Release Notes and Documentation*. https://claude.com/

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Fang, E. (2026). *AgentVerify: Compositional Formal Verification of AI Agent Safety Properties via LTL Model Checking*. https://www.preprints.org/manuscript/202604.1029

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., Wang, S., Zhang, K., Wang, Y., Gao, W., Ni, L., & Guo, J. (2024). A Survey on LLM-as-a-Judge. *arXiv preprint arXiv:2411.15594*. https://arxiv.org/abs/2411.15594

Hägele, A., Gema, A. P., Sleight, H., Perez, E., & Sohl-Dickstein, J. (2026,). The Hot Mess of AI: How Does Misalignment Scale With Model Intelligence and Task Complexity?. *International Conference on Learning Representations (ICLR 2026)*. https://arxiv.org/abs/2601.23045

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2024). DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv preprint*. https://arxiv.org/abs/2310.03714

Landis, J. R., & Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, *33*(1), 159–174.

LangChain. (2026). *LangGraph Overview*. https://docs.langchain.com/oss/python/langgraph

Pawlak, Z. (1982). Rough sets. *International Journal of Computer and Information Sciences*, *11*(5), 341–356.

Rice, H. G. (1953). Classes of Recursively Enumerable Sets and Their Decision Problems. *Transactions of the American Mathematical Society*, *74*(2), 358–366.

Rothrock, M. (2026a). *claude-code-log-analyzer*. GitHub. https://github.com/mrothroc/claude-code-log-analyzer

Rothrock, M. (2026b). *Target-Specific Verification Surfaces for Cross-Stage Quality Assurance*. Zenodo. https://doi.org/10.5281/zenodo.20331363

Rothrock, M. (2026c). *Verification Surfaces in Language-Model Systems*. Zenodo. https://doi.org/10.5281/zenodo.20331399

Sohl-Dickstein, J. (2023, March 9). *The Hot Mess Theory of AI Misalignment: More Intelligent Agents Behave Less Coherently*. https://sohl-dickstein.github.io/2023/03/09/coherence.html

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2–42*(1), 230–265.

Willard, B. T., & Louf, R. (2023). Efficient Guided Generation for Large Language Models. *arXiv preprint*. https://arxiv.org/abs/2307.09702

Wilson, E. B. (1927). Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, *22*(158), 209–212.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. https://arxiv.org/abs/2306.05685