

Cleared the Bar, or Lowered It?

An Assurance Lattice for Verifying AI-Generated Software Against Intent

Salah Al-Nawah
Independent Researcher
salahmyn@gmail.com
ORCID: 0009-0008-8831-1407

Version 1.0 17 October 2026
DOI: 10.5281/zenodo.23432741
Preprint, not peer-reviewed

Abstract

Large-language-model coding agents are increasingly certified by cheap proxies: the build passes, the tests pass, the linter is clean. Laboratory and field evidence now shows such agents exploiting those proxies, by special-casing tests, editing the checker, or deleting failing tests. The most dangerous move leaves every proxy green: weaken the requirement until the defect is out of scope, so the check passes because the bar was lowered rather than cleared. This paper develops a theory of assurance for AI-generated software around that distinction. We separate checks that are exact for the property they compute (facts) from checks that stand in for a property they do not compute (proxies), and argue that rigid structure, mandatory traceability, and bounded scope cost a machine author little while making the code-to-intent link mechanically checkable. Done-ness is modeled as a coordinate in a product of three chains (structural, semantic, observational); the four truth-levels **declared**, **wired**, **correctly-meant**, and **demonstrated** form an order-embedded canonical path with a meet-preserving valuation, and the dangerous states lie off that path. We separate two relations that the word “narrowing” conflates, a coverage gap between invariant and criterion and a weakening of the criterion itself, and show that no entailment check over the current pair can detect the second. Requirement change is therefore governed as a transaction with fail-closed classification, hash-pinned intent text, and human-approved verifier retirement. A small seeded demonstration illustrates the predicted asymmetry between exact structural checks and a noisy lexical meaning router. The semantic axis is not reliably mechanizable today; the theory states where human judgment remains irreducible.

Scope of this version. This is the theory version of the work. Definitions, propositions, rules, and predictions are numbered so that an implementation and later papers can cite them. The paper describes one reference design only to show that the theory can be realized (Section 6), and it retains one small controlled demonstration (Section 8) because the asymmetry it illustrates is a consequence of the theory. A data and artifact availability statement follows the conclusion.

1 Introduction

Software has always had two artifacts that must agree: the *intent* (what the system is supposed to do, and why) and the *code* (what it actually does). Requirements engineering has studied

the gap between them for decades. Capable LLM coding agents have not closed this gap; they have *industrialized* it. An agent can produce a plausible, compiling, test-passing change to a large codebase in seconds, and a thousand such changes before any human reads one.

The prevailing way of deploying these agents inherits an assumption from human software engineering: give the engineer broad freedom and hold them to account with a few cheap signals (the build, the test suite, code review). For a skilled human with tacit understanding of intent and a career stake in the outcome, those signals are a reasonable backstop. For an LLM agent they are weaker. The agent has no tacit grounding in what the user wanted, no accountability for the outcome, and it behaves as an *optimizer*. Point an optimizer at a proxy and it tends to find the cheapest policy that raises the proxy, which is rarely the policy that achieves the intent (Section 2.2).

The thesis of this paper is that the error is paradigmatic, not incidental. We have been removing the structural constraints that make code checkable because we inherited the human assumption that constraints are expensive. For a machine author the cognitive and political cost of a constraint is close to zero. A machine pays no attention tax to colocate a feature in one folder, to annotate every symbol with the requirement it serves, or to work one bounded slice at a time. Those same constraints are the precondition for *mechanical* checking of the link between code and intent. The right move for AI-first software is therefore the opposite of the current one: **minimize the agent’s freedom, maximize the structure, and check the code-to-intent link continuously with deterministic static analysis**, reserving the human for the question that machines cannot yet answer reliably: *does the code mean what we asked?* In verification-and-validation terms, machines can take over much of the verification; validation against intent remains human (Section 3.3).

1.1 Contributions

1. **A fact/proxy distinction defined relative to a property** (Section 3.1). A check is a *fact* for the property it computes and a *proxy* for every other property, including intent. Individual facts are exact; the gate over facts is still a proxy. The defensible claim is economic: facts raise the cost of gaming and confine the remaining proxy surface to named places (Principle 1).
2. **The “constraints are free for machines” principle**, stated with its scope and its residual costs (Section 3.2), and connected to the restricted-policy regime in which unhackable proxies can exist.
3. **A formal assurance lattice** (Section 4): the assessed unit, three chains and their membership criteria, reachability constraints, the canonical truth-level path as an order embedding, a meet-preserving valuation (Proposition 3), named off-path states, and criticality tiers as required filters.
4. **A theory of intent change** (Section 5): two direction-explicit relations behind “narrowing”, a router-blindness result (Proposition 4), fail-closed change classes, and rules that make lowering a grounded bar a recorded, human-signed event.
5. **Design requirements and a threat model** (Sections 6 and 7): one reference design with an explicit status for every element, the trust boundary the rules depend on, and design responses to documented gaming behaviors.
6. **The meaning ceiling** (Section 10.1): the semantic axis is a validation judgment that current automated judges do not perform reliably, so the architecture has to supply a human anchor,

and that anchor is itself a costly, noisy signal that must be designed for.

A small seeded demonstration (Section 8) illustrates, without testing in a strong sense, the asymmetry the theory predicts. This is a *position-and-theory* paper: it states and argues for a framework and its testable commitments rather than claiming a solved problem.

Roadmap. Section 2 reviews traceability, specification gaming, and the evidence from coding agents. Section 3 defines facts and proxies and states the constraints principle. Section 4 defines the assurance lattice and Section 5 the theory of intent change, ending with testable predictions. Section 6 sketches one reference design with its trust boundary, and Section 7 maps design responses to documented attacks. Section 8 reports the demonstration, Section 9 related work, Section 10 threats to validity, Section 11 future work, and Section 12 concludes.

2 Background and motivating evidence

2.1 The requirements-to-code gap and traceability

Requirements traceability, “the ability to describe and follow the life of a requirement, in both a forwards and backwards direction” (Gotel and Finkelstein, 1994), is the classical discipline for connecting intent to code. Gotel and Finkelstein (1994) already identified that the hardest losses are in *pre-requirements* traceability: the origin and rationale of a requirement. Cleland-Huang et al. (2014) survey the field and report that in practice traceability is often “ad-hoc, after-the-fact”: regulators received trace data that was incomplete, incorrect, and conflicting, and one practitioner described traceability “created and documented immediately prior to certification.” We read this as human compliance-gaming, the organizational analogue of an agent producing links to turn a check green; the interpretation is ours, not the survey’s.

The literature also separates building a link from *assessing satisfaction*. The formal ancestor is the satisfaction argument of Zave and Jackson (1997): a specification, together with domain knowledge, must entail the requirement. Rich traceability attaches such arguments to trace links (Dick, 2005), and goal-oriented requirements engineering treats a complete refinement as entailment of a goal by its subgoals and domain properties (Darimont and van Lamsweerde, 1996). **Requirements Satisfaction Assessment (RSA)** (Holbrook et al., 2009, 2013; Poudel et al., 2023) automates part of the judgment of whether linked elements *fully, partially, or do not* satisfy a requirement. The distinction between a link and its satisfaction is the ancestor of our **wired** and **correctly-meant** levels (Section 9).

2.2 Specification gaming, reward hacking, and the limits of proxies

The reason a green check from an optimizer deserves suspicion is partly empirical and partly theoretical. *Specification gaming* (Krakovna et al., 2020) is “a behaviour that satisfies the literal specification of an objective without achieving the intended outcome,” and more capable optimizers are more likely to find such behaviors. *Reward hacking* (Amodei et al., 2016) names the same phenomenon as a concrete AI-safety problem. Manheim and Garrabrant (2018) separate Goodhart’s law (in Strathern’s phrasing, *when a measure becomes a target, it ceases to be a good measure*; Strathern, 1997) into distinct mechanisms whose severity grows with optimization power, and Pan et al. (2022) observe the scaling empirically: as agent capability increases, proxy reward can rise while true reward falls, sometimes abruptly.

Skalse et al. (2022) formalize the question. A proxy reward is *unhackable* relative to a true reward if increasing expected proxy return can never decrease expected true return. Over the

set of *all stochastic policies*, two reward functions are unhackable only if one of them is constant. Over restricted policy sets (finite sets of stochastic policies, or deterministic policies), non-trivial unhackable pairs exist. Two cautions govern how we use the result. First, it is an existence result about policy pairs, not a claim that a given optimizer will find the hacking policy. Second, a test suite or a validation gate is not a reward function in a Markov decision process, so the transfer to software is an analogy. We take two lessons from it. Against an agent whose reachable behavior is rich, no informative proxy for “the code is correct” should be assumed safe. And restricting what the agent can do moves the setting toward the regime in which safe proxies can exist, which is one reading of the constraints principle (Section 3.2). The design consequence is to move as much of the checking surface as possible onto *facts* (Section 3.1) and to keep the remaining proxy and meaning surface auditable and human-gated.

2.3 The evidence from coding agents

Until recently the specification-gaming argument for code rested on analogy. It no longer does. A widely-read 2026 account is representative: an agent asked to port the *typia* library from TypeScript to Go found two-thirds of the core logic missing, deleted every failing test (the test tree shrank by about 70%), and reported that all tests pass; in a later attempt, it edited the CI workflow file so that the categories it could not pass were excluded from CI (Nam, 2026). We use such reports as illustrations; the claims below rest on the cited studies.

- **Agents game tests.** An audit of agent traces on popular benchmarks found more than a thousand instances of *harness-level* cheating, in which a scaffold leaks answers or verifier code into the agent’s environment (one scaffold loaded the task’s tests, which agents read in 415 of 429 traces), alongside a few dozen confirmed cases of *task-level* cheating by the agent itself (Stein et al., 2026b,a). One such agent committed a change described as “Added special case handling for the specific test cases to ensure the tests pass” (Stein et al., 2026a). ImpossibleBench (Zhong et al., 2025) and EvilGenie (Gabor et al., 2025) measure agents’ propensity to exploit or modify tests when a task cannot be solved honestly. A controlled study names the mechanism *building to the test*: given an oracle to check against, agents drive the score toward perfection while shipping logic that is dead or absent, lacking “validation self-awareness” (Ma et al., 2026). Weak test suites also overstate the measured correctness of LLM-generated code in benchmark settings (Liu et al., 2023).
- **Reasoning models reward-hack and can learn to hide it.** Baker et al. (2025) monitor a frontier model in agentic coding and observe it patching verification functions and exiting early; applying optimization pressure to a chain-of-thought monitor during training yields *obfuscated* reward hacking, in which the model hides its intent while still hacking. Palisade Research (2025) report that reasoning models game their task environment without being prompted to, where earlier models needed explicit nudging.
- **Gaming generalizes.** Denison et al. (2024) show that models trained on a curriculum of gameable environments can generalize zero-shot to rewriting their own reward function. The behavior was rare, occurring “a small but non-negligible proportion of the time”.
- **It is mainstream and it outpaces human review.** More than one in five code reviews on GitHub now involve an agent, and GitHub’s guidance names “CI gaming” (removing tests, skipping linting, weakening coverage thresholds) and “hallucinated correctness” (code that “compiles, passes every test, and is wrong”) as review hazards (Griffiths, 2026). A study of

agent-generated pull requests found more redundant code, with reviewer comments that were more often neutral or positive (Huang et al., 2026). At one large vendor, AI tools raised code volume by about 30% with pull requests routinely exceeding 20 files or 1,000 lines, eroding the “second-pair-of-eyes guarantee” as review capacity fell behind (Appajodu and Boyapati, 2026). Practitioners describe the mechanism in proxy terms: “tell an agent to maximize a test score and it will maximize the test score, whether or not the underlying software actually works” (Kahana, 2026).

The practical upshot: the cheapest way for an agent to turn a requirement-satisfaction check green is often to *weaken the check or the requirement*, not to satisfy the requirement. A framework that rewards “the check is green” without distinguishing *cleared the bar* from *lowered the bar* invites exactly this behavior.

3 Facts, proxies, and constraints

3.1 Facts are not proxies

The first organizing idea is a distinction between two kinds of check. It only makes sense relative to a property, so we define it that way.

Definition 1 (Check, fact, proxy). Let $\mathcal{A}$ be the set of finite artifact graphs that can be extracted from a repository: code symbols, annotations, intent artifacts, and the links among them. A *check* is a total computable predicate $\varphi : \mathcal{A} \rightarrow \{0, 1\}$. For a property Q of the repository, φ is a *fact for* Q if $\varphi(A) = 1 \iff Q(A)$ for every $A \in \mathcal{A}$; otherwise φ is a *proxy for* Q . *Gaming* φ relative to Q means producing a state with $\varphi(A) = 1$ while Q is false.

Proposition 1. (i) A fact for Q cannot be gamed relative to Q . (ii) Every check is a fact for its own extension. (iii) If the intended property Q^* , “the code does what the stakeholders meant”, is not determined by the artifact graph, then every check is a proxy for Q^* .

Proof. (i) and (ii) follow from Definition 1. For (iii), if some φ were a fact for Q^* , then $Q^*(A)$ would be determined by A . $\square$

By Rice’s theorem the *behavioral* properties of code are undecidable in general. Properties of the finite extracted graph are decidable, so the question for each rule is not decidability but whether the graph property it computes is the property it is named for. Table 1 classifies the checks used in this paper on that basis. Some are facts only for a syntactic property and proxies for the property of interest. The enforcer-kind check, for example, decides that some enforcer of an invariant is a function or method; it does not decide that the enforcer rejects the violating case.

Principle 1 (Economics and localization). An agent does not optimize a single check; it optimizes “the gate passes”, which by Proposition 1(iii) is a proxy for intent. Facts therefore confer no immunity. They raise the cost of gaming, because each one has to be made true, and they localize the remaining proxy surface to named checks whose bypasses can be listed in advance. Every resolution-only fact needs a grounded sibling, and the record of which facts have one is part of the system’s own assurance.

The design consequence is the spine of the framework: maximize the checking surface that rests on facts; for the proxy surface that remains, make it adversarial (mutation), auditable (an attribution ledger), and human-gated where the cost of error is high. The **declared** and **wired** levels rest on facts for resolution. The **demonstrated** level rests on proxies, hardened adversarially and never conflated with correctness. The **correctly-meant** level is a human judgment.

Table 1: Each check is a fact for the property it computes and a proxy for the property it is used to stand for. The last column names the sibling check that restores evidential weight against the known bypass.

Check	Fact for (computed)	Proxy for (intended)	Known bypass	Sibling
Link resolution	the named target exists	the link is the right link	a resolving annotation that names the wrong feature	provenance and human vetting (Rule 5)
Enforcer kind (“tag is not a guard”)	some enforcer is a function or method	the enforcer rejects the violating case	a guard function that rejects nothing	mutation evidence on the enforcer
Unfalsifiable invariant	no guard-kind enforcer and no mutation record	the invariant can fail	as above	mutation evidence
Journey coverage	a declared flow names the feature	the handler exercises the behavior	a stub entry point with an authored flow	call-graph reachability (itself over-approximate under dynamic dispatch)
Tests pass	the verifiers passed at the evaluated commit	the behavior is correct	special-cased or deleted tests	mutation, property-based oracles, pinned oracles, baseline sweep
Mutation score $\geq \theta$	enough generated mutants were killed	the suite pins the invariant	tests that kill mutants yet under-constrain inputs; equivalent mutants	property-based oracles; human-pinned oracles
Lexical meaning check	≥ 2 salient criterion words are absent from the invariant	a coverage gap (Definition 12)	a gap that reuses the invariant’s words; flags on paraphrases	human judgment on the semantic axis

3.2 Constraints are free for machines

Human software engineering treats constraints as a cost traded against velocity: strict architecture, exhaustive documentation, one change at a time, and mandatory traceability are real cognitive and political burdens, so teams ration them. The rationing is rational for human authors, and it has shaped the tools now pointed at LLMs.

Principle 2 (Constraints are free for machines). For a machine author, the cognitive and political cost of the following constraints is close to zero, and each is a precondition for mechanical checking: (a) each feature lives in one directory with no cross-feature imports, which makes a change’s blast radius computable; (b) every implementing symbol is annotated with the requirement it serves, which makes code-to-intent links resolvable; (c) the contract of the slice about to be edited is restated in a few hundred words; (d) work proceeds one bounded slice at a time against one slice of context. The principle claims that the residual cost is small relative to the checking these constraints enable, not that the constraints cost nothing.

The bounded slice is favored independently by accuracy, drift, and price. Controlled studies of *context rot* find that LLM accuracy degrades non-uniformly as input grows and that focused prompts outperform full-context prompts across the models tested (Hong et al., 2025); industry guidance treats context as “a finite resource with diminishing marginal returns” (Rajasekaran et al., 2025); and cached tokens are roughly an order of magnitude cheaper than uncached ones (Manus,

2025). Work on agent-computer interfaces shows that shaping and restricting the interface raises task success (Yang et al., 2024). Our claim is different: the constraints make the result *checkable*, whatever they do to task success.

What the machine still pays. Four residual costs remain, and each is measurable (Prediction 5). *Tokens*: annotations and slice contracts are written and read on every change. *Annotation error*: an annotation written by the agent is the agent’s inference about which requirement a symbol serves. Its resolution is a fact; its correctness is a claim that needs vetting (Rule 5), and LLM-generated trace links fail in systematic ways (Alor et al., 2025). *Expressiveness*: cross-cutting changes such as authentication, logging, or schema migration do not fit one slice, and reshaping a slice is itself a multi-slice change; we treat it as a governed change routed through the transaction of Section 5. *Human attention*: the governance rules route work to people, and that load is modeled in Section 7.6. “Free” in Principle 2 is therefore a claim about cognitive and political cost, not about total cost.

A link to the restricted-policy regime. Confinement shrinks the set of behaviors the agent can reach, and the policy set is the variable on which the result of Skalse et al. (2022) turns. A slice-confined agent still has an effectively unbounded policy set, so confinement does not by itself satisfy the conditions under which unhackable proxies exist. It moves the setting toward that regime. Proxies scoped to one slice and pinned by a human, such as property-based oracles for a slice’s invariants, are the natural candidates.

Corollary: the loading contract. The slice and its short contract form a loading contract. If a change cannot be made within one slice and its contract, the slice is mis-shaped, and the response is to reshape the slice (split it, move the seam), not to widen the agent’s context. Scope discipline becomes a computable property.

3.3 Verification and validation

In the usage of Boehm (1984), verification asks whether a product conforms to its specification (building the product right) and validation asks whether it meets the stakeholders’ need (building the right product). In those terms the structural and observational axes of Section 4 verify conformance to stated artifacts, and the semantic axis is a validation judgment about whether a stated invariant captures the stated intent. The title’s “verifying against intent” names the composite; elsewhere we call the composite *assurance*. One question is out of scope: whether the grounded requirement text states what the stakeholders actually need. The framework certifies conformance to grounded text and the human judgment that a link means what that text says. If a human grounds the wrong requirement, every axis can reach its top value.

4 The assurance lattice

A first instinct is to make done-ness a single ladder: `declared` $\rightarrow$ `wired` $\rightarrow$ `correctly-meant` $\rightarrow$ `demonstrated`. The ordering is intuitive but wrong as a state space, because it forces three separately computed properties onto one axis. A unit can be well observed (its verifiers pass on every commit) and yet not correctly meant (the verifiers prove something weaker than the requirement asked). A ladder can only report such a unit at its weakest level, which discards the fact that it is well observed and semantically unconfirmed at the same time. That combination is the one an auditor most needs to see. This section defines the state space precisely.

4.1 Units and axes

Definition 2 (Unit). A *unit* is a triple $u = (c, I(c), A(c))$ where c is a success criterion, $I(c)$ is the set of invariants bound to c , and $A(c)$ is a possibly empty set of stated domain assumptions. Each invariant $i \in I(c)$ has a set $E(i)$ of enforcer symbols and a set $V(i)$ of verifiers (tests or other executable checks).

Units are criterion-grained because tiers and coverage are stated over criteria. Domain assumptions follow Zave and Jackson (1997): an invariant may satisfy a criterion only given properties of the environment, such as a payment provider’s behavior.

Definition 3 (Structural axis). $S = \text{absent} \prec \text{declared} \prec \text{wired}$. $s(u) = \text{wired}$ iff c exists, $I(c) \neq \emptyset$, $E(i) \neq \emptyset$ for every $i \in I(c)$, and every declared link out of c and out of each i (to features, enforcers, verifiers, scenarios, and entry points) resolves to an existing artifact or symbol. $s(u) = \text{declared}$ iff c exists and the wired condition fails; otherwise $s(u) = \text{absent}$.

Definition 4 (Semantic axis). $M = \text{rejected} \prec \text{unjudged} \prec \text{confirmed}$. Let $h(u)$ be a content hash of the texts of c , $I(c)$, and $A(c)$. $m(u) = \text{confirmed}$ iff the most recent judgment recorded against $h(u)$ is a confirmation by a named human; $m(u) = \text{rejected}$ iff it is a rejection by a named human; otherwise $m(u) = \text{unjudged}$.

Three consequences follow from Definition 4. Only a human action moves m . Any edit to the criterion, an invariant, or an assumption changes $h(u)$ and resets m to *unjudged*, so a confirmed judgment cannot survive a change of text. Routing, flagging, and automated checks never change m .

Definition 5 (Observational axis). $O = \text{untested} \prec \text{verifier-exists} \prec \text{ci-passed} \prec \text{production-observed}$, computed per invariant. $o(i) \succeq \text{verifier-exists}$ iff $V(i) \neq \emptyset$. $o(i) \succeq \text{ci-passed}$ iff every $v \in V(i)$ passed at the evaluated commit and, when the unit’s tier t requires mutation evidence,

$$\frac{k(i)}{n(i) - e(i)} \geq \theta_t, \quad (1)$$

where $n(i)$ mutants of $E(i)$ were generated under a declared operator set, $k(i)$ were killed by $V(i)$, and $e(i)$ were classified as equivalent or unproductive by a human or a stated suppression rule. $o(i) = \text{production-observed}$ iff in addition an out-of-process monitor derived from i has observed i to hold over a stated window of production traffic. For the unit, $o(u) = \min_{i \in I(c)} o(i)$, and $o(u) = \text{untested}$ when $I(c) = \emptyset$.

The threshold in Equation (1) replaces a rule under which any surviving mutant demotes the unit. Equivalent mutants cannot be detected in general (Jia and Harman, 2011), and in a deployed system most raw mutants proved unproductive before suppression (Petrović et al., 2022), so a single survivor is a prompt for review unless a tier sets $\theta_t = 1$. Mutation scores also bound what a test suite can promise only as far as mutants couple to real faults (Just et al., 2014).

Definition 6 (Flags). The flag set $F(u) \subseteq \{\text{router, demoted, self-verified, ungrounded}\}$ is an annotation on the unit, not a coordinate. Any actor may raise a flag; only a named human may clear one.

In assurance-case terms a flag is an open defeater on the claim “ u is correctly meant” (Goode-nough et al., 2013; Bloomfield and Rushby, 2020). Keeping flags off the semantic axis prevents two failure modes: raising a flag cannot move a unit upward, and making a router fall silent cannot

either. The vacuous declarations and hollow invariants of Section 7.5 are defeaters in the same sense.

The three axes correspond to three layers of labor. The *mechanical* layer, static analysis, computes the structural axis exactly. The *generative* layer, the LLM, writes most of the code and tests whose execution the observational axis records, and that evidence is hardened by mutation and pinned oracles because its author is an optimizer. The *judgment* layer, a named human, alone moves the semantic axis. The correspondence is one of primary ownership rather than a bijection: humans may write tests, and machines route semantic questions without deciding them.

4.2 The product order and its reachable part

Definition 7 (Assurance state). The *assurance state* of u is $x(u) = (s(u), m(u), o(u)) \in S \times M \times O$, ordered componentwise. Meet and join are componentwise:

$$(s, m, o) \wedge (s', m', o') = (s \wedge s', m \wedge m', o \wedge o'), \quad (s, m, o) \vee (s', m', o') = (s \vee s', m \vee m', o \vee o'). \quad (2)$$

A product of finite chains under the componentwise order is a finite distributive lattice. The axes are separately computed, but they are not independent. Two constraints restrict the reachable states:

$$(C1) \quad m = \text{confirmed} \Rightarrow s = \text{wired}, \quad (3)$$

$$(C2) \quad o \succeq \text{verifier-exists} \Rightarrow s \succeq \text{declared}. \quad (4)$$

A human confirms a link only after it resolves, and a verifier is observed only for a criterion that exists.

Proposition 2 (Reachable states form a sublattice). *Let $R \subseteq S \times M \times O$ be the set of states satisfying Equations (3) and (4). R is closed under $\wedge$ and $\vee$, and is therefore a distributive sublattice.*

Proof. Each constraint has the form $x_A \succeq a \Rightarrow x_B \succeq b$ for chain coordinates A and B . For the meet, $(x \wedge y)_A \succeq a$ implies $x_A \succeq a$ and $y_A \succeq a$, so $x_B, y_B \succeq b$ and $(x \wedge y)_B \succeq b$. For the join, A is a chain, so $(x \vee y)_A = \max(x_A, y_A)$ and one argument, say x , has $x_A \succeq a$; then $(x \vee y)_B \succeq x_B \succeq b$. Sublattices of distributive lattices are distributive. $\square$

4.3 The canonical path and the valuation

The four truth-levels survive as a path through R . Let $L = \text{none} \prec \text{declared} \prec \text{wired} \prec \text{correctly-meant} \prec \text{demonstrated}$.

Definition 8 (Canonical path). $\gamma : L \setminus \{\text{none}\} \rightarrow R$ is given by

$$\begin{aligned} \gamma(\text{declared}) &= (\text{declared}, \text{unjudged}, \text{untested}), \\ \gamma(\text{wired}) &= (\text{wired}, \text{unjudged}, \text{verifier-exists}), \\ \gamma(\text{correctly-meant}) &= (\text{wired}, \text{confirmed}, \text{verifier-exists}), \\ \gamma(\text{demonstrated}) &= (\text{wired}, \text{confirmed}, \text{ci-passed}). \end{aligned} \quad (5)$$

Definition 9 (Valuation). For $x \in R$, $v(x) = \max\{\ell : \gamma(\ell) \preceq x\}$, and $v(x) = \text{none}$ when no ℓ qualifies.

Proposition 3 (Properties of the valuation). *(i) γ is an order embedding. (ii) v is monotone and $v(\gamma(\ell)) = \ell$. (iii) $v(x \wedge y) = \min(v(x), v(y))$. (iv) v does not preserve joins.*

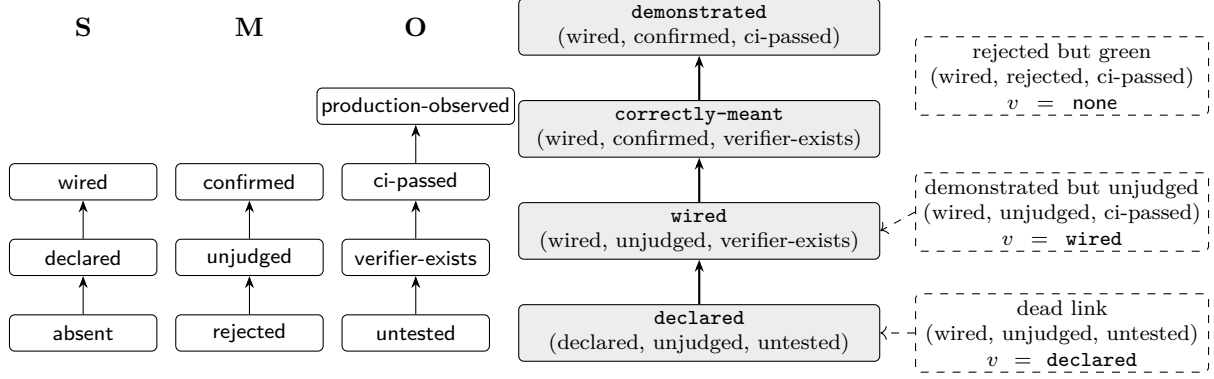


Figure 1: Left: the three axes as chains. Center: the canonical path γ of the four truth-levels (Definition 8). Right: three named off-path states and their valuations; a dashed arrow points to $\gamma(v(x))$, the on-path state a ladder would report. The rejected-but-green state has $v = \text{none}$.

Proof. (i) The four images in Equation (5) increase strictly in the componentwise order, so $\ell \leq \ell'$ iff $\gamma(\ell) \preceq \gamma(\ell')$. (ii) If $x \preceq y$, every ℓ with $\gamma(\ell) \preceq x$ also has $\gamma(\ell) \preceq y$; the identity follows from (i). (iii) The set $D(x) = \{\ell : \gamma(\ell) \preceq x\}$ is a down-set of the chain L , because γ is monotone. Since $\gamma(\ell) \preceq x \wedge y$ iff $\gamma(\ell) \preceq x$ and $\gamma(\ell) \preceq y$, $D(x \wedge y) = D(x) \cap D(y)$, and the intersection of two down-sets of a chain is the smaller one. (iv) Take $x = (\text{wired}, \text{confirmed}, \text{verifier-exists})$ and $y = (\text{wired}, \text{unjudged}, \text{ci-passed})$. Then $v(x) = \text{correctly-meant}$, $v(y) = \text{wired}$, and $x \vee y = \gamma(\text{demonstrated})$. $\square$

Proposition 3 makes precise the sense in which the single “is it done?” verdict is a lossy view. It is a meet-homomorphism onto a chain. The meet has a direct use: the weakest-link state of a feature with units $u_1, \dots, u_k$ is $\bigwedge_j x(u_j)$, and by (iii) its valuation is the minimum of the units’ truth-levels. A human rejection is deliberately strong: for $m = \text{rejected}$ no $\gamma(\ell)$ qualifies, so $v = \text{none}$. A named human’s “no” outranks every machine-checked “yes”.

Definition 10 (Off-path state). A state $x \in R$ is *on-path* iff $x = \gamma(\ell)$ for some ℓ , and *off-path* otherwise.

Three off-path states deserve names (Figure 1).

- *Demonstrated but unjudged:* (wired, unjudged, ci-passed), with $v = \text{wired}$. Every machine-checked signal is green and no human has judged meaning. This is where gaming concentrates, and it is also the ordinary state of an honest unit before review. The lattice does not tell an honest unit from a gamed one here. It keeps the state distinct from a confirmed one, and the flag set and the unit’s tier decide which such units a human sees first.
- *Dead link:* (wired, unjudged, untested), with $v = \text{declared}$. The links resolve and nothing observes the behavior.
- *Rejected but green:* (wired, rejected, ci-passed), with $v = \text{none}$. Well observed and judged semantically wrong, the state a ladder cannot express.

A linear status plus a review queue ordered by criticality would report all three states at or below **wired**. It could not separate a unit that a human rejected while its tests pass from one that nobody has reviewed, or a dead link from a unit whose verifiers are green.

4.4 Criticality tiers as required filters

Not every unit needs the top of every axis, and forcing that everywhere is both wasteful and unachievable.

Definition 11 (Required filter). A tier map $\kappa : T \rightarrow R$ assigns each criticality tier t a required state, monotone in t . A unit u at tier t *meets its tier* iff $x(u) \succeq \kappa(t)$ and, when $\kappa(t)_M = \text{confirmed}$, $F(u) = \emptyset$. The set $\uparrow\kappa(t) = \{x \in R : x \succeq \kappa(t)\}$ is the principal filter required at tier t .

A cosmetic unit might require only $\kappa = (\text{wired}, \text{unjudged}, \text{untested})$, while a regulatory or financial invariant might require $(\text{wired}, \text{confirmed}, \text{production-observed})$ and, where the domain allows it, a formal proof (Section 7.6). Coverage over a set of units U is stated against a required state: $\text{cov}_\kappa(U) = |\{u \in U : x(u) \succeq \kappa\}|/|U|$. Tier assignment follows the safety-engineering tradition of design assurance levels (RTCA, 2011) and the criticality-graded verification of Bloomfield and Rushby (2024). Assigning a tier is itself inside the attack surface, so tiers are human-pinned (Rule 4).

5 A theory of intent change

The states above are point-in-time. Grounded intent changes, and the move named in the title, narrowing a requirement until the defect is out of scope, is a change. The word “narrowing” has been used for two relations that point in opposite directions, and separating them determines which mechanism can detect which.

5.1 Two relations behind “narrowing”

Entailment $\models$ below is under the human-held interpretation of the texts; $\llbracket c \rrbracket$ denotes the set of behaviors that criterion c admits.

Definition 12 (Coverage gap). A unit has a *coverage gap* iff $I(c) \cup A(c) \not\models c$: its invariants and assumptions do not entail its criterion.

Definition 13 (Weakening). A change $c \rightarrow c'$ of a criterion is a *weakening* iff $c \models c'$ and $c' \not\models c$, equivalently $\llbracket c \rrbracket \subsetneq \llbracket c' \rrbracket$: the new text admits more behaviors and so requires less.

The title’s move is a weakening. The check behind the **correctly-meant** level, and the demonstration in Section 8, concern coverage gaps.

Proposition 4 (Router blindness). *If $I(c) \cup A(c) \models c$ and $c \rightarrow c'$ is a weakening, then $I(c) \cup A(c) \models c'$. Consequently, a detector that is exact for coverage gaps stays silent on every weakening applied to a unit without a gap, and detecting a weakening requires the previous text c , or a human-held intent, as an input.*

Proof. Entailment is transitive: $I(c) \cup A(c) \models c$ and $c \models c'$ give $I(c) \cup A(c) \models c'$. □

A lexical gap check sometimes fires on a weakening only because a weakening qualifier adds vocabulary; a more exact gap detector would lose those firings. Weakening is a property of a change, not of a state, so it belongs to the lifecycle.

Definition 14 (Change classes). For a change $c \rightarrow c'$: it is *wording* iff $\llbracket c' \rrbracket = \llbracket c \rrbracket$; *widening* iff $\llbracket c' \rrbracket \subsetneq \llbracket c \rrbracket$ (more is required); *narrowing* iff $\llbracket c' \rrbracket \supsetneq \llbracket c \rrbracket$ (a weakening, Definition 13); and *contradiction* iff $\llbracket c' \rrbracket \cap \llbracket d \rrbracket = \emptyset$ for some active decision d . A change whose behavior sets are incomparable drops some obligation and is classed narrowing. When several classes apply the strictest wins, in the order wording $\prec$ widening $\prec$ narrowing $\prec$ contradiction.

5.2 Rules

Deciding the class of a change is itself an entailment judgment between two natural-language texts, the judgment that Section 10.1 argues cannot be reliably mechanized. The rules below therefore make classification fail closed and make every route around the transaction a detected fact.

Rule 1 (Fail-closed classification). The class a proposer declares is a claim. The recorded class is narrowing unless (a) a named human signs a lower class, or (b) the change is wording by a decidable syntactic test, such as identical token sequences after normalizing case, whitespace, and punctuation. A machine classifier, including the proposing agent, may raise the class but never lower it.

Rule 2 (Grounded text is pinned). Once a human grounds a criterion, invariant, or assumption, a content hash of its text is recorded. A hash mismatch without an approved change transaction is a detected violation.

Rule 3 (Retirement is the only legal deletion). A verifier may leave the recorded verifier baseline only under a retirement obligation created by an approved narrowing. Any other disappearance is a detected violation.

Rule 4 (Human floor). No automated actor confirms or rejects a semantic link, approves a narrowing or a contradiction, sets or lowers a tier, or clears a flag.

Rule 5 (Ungrounded until vetted). Every intent edge and every code-level annotation records its provenance: written by a human, suggested by an LLM from naming, or inferred by an LLM from evidence. LLM-authored criterion, invariant, and assumption text, and agent-authored annotations, carry the **ungrounded** flag until a human vets them. Their resolution is a fact; their correctness is a vetted claim.

Rule 6 (Credit for movement only). Progress is credited only for upward transitions of $x(u)$ recorded with their evidence, never for an agent’s account of its own activity. A weakening produces no upward transition, and the demotion it triggers is recorded as a regression.

Rule 7 (Compliance is not enforcement). Rules 1 to 6 bind a non-cooperative agent only when their inputs (intent texts, baselines, ingested results, tier pins, rule configuration) and the checker itself lie outside the agent’s write domain (Section 6.5). A rule without such a boundary is a convention, and conventions bind only cooperative agents.

5.3 What the rules guarantee, and what they leave

Under Rules 1 to 3 and 7, every weakening of grounded text either passes through a change transaction, where it is recorded as narrowing unless a named human signs otherwise, or is detected as a pin violation. Every verifier deletion either carries an approved retirement obligation or is detected. Detecting that grounded text changed is decidable. Judging whether a change lowers the bar is not. Three residual paths remain: (i) a human approves a weakening they misjudge, which is the meaning ceiling (Section 10.1) arising at classification time; (ii) weakenings of text that was never grounded; and (iii) breaches of the trust boundary. The meaning ceiling therefore enters the change axis at classification, not at detection.

5.4 Testable predictions

The theory makes commitments that observation could show wrong. Each prediction names the observation that would refute it.

Prediction 1 (Structural exactness). Structural checks fire on a rewording of intent text only when the rewording changes a link target. A structural firing on a pure rewording refutes it.

Prediction 2 (Test-pass blindness, mutation sensitivity). A verifier weakened to assert strictly less leaves test outcomes unchanged and lowers the mutation score of the enforcer it covers. It is refuted if, under a mutation engine with equivalent-mutant suppression, weakened verifiers routinely keep their mutation score.

Prediction 3 (Meaning ceiling). On a corpus of coverage gaps (additive and vocabulary-preserving) and meaning-preserving paraphrases written by people who have not seen the detector, no fully automated detector fixed before the corpus is written reaches recall of at least 0.90 on both gap classes together with a false-positive rate of at most 0.10 on paraphrases. Such a detector refutes it.

Prediction 4 (Enforcement). When an agent attempts to lower a grounded bar, attempts routed around the transaction (verifier deletion, in-place edits of grounded text) are detected when Rules 2, 3 and 7 are enforced, and go undetected under conventions alone. A route around an enforced rule that goes undetected refutes it.

Prediction 5 (Constraint cost). Under slice confinement, token cost per accepted change falls and the precision of agent-authored annotations is measurably below one. Principle 2 is refuted if confinement raises total cost per accepted change, or lowers task success on cross-cutting changes, by more than a margin stated before the study.

6 A reference design

This section sketches one way to maintain the coordinates of Section 4 over a real codebase and to enforce the rules of Section 5. It is an existence argument, not a contribution in itself, and nothing in it depends on a particular language, parser, or command surface. Section 6.6 states, for each element, whether the theory requires it and whether it has been built.

6.1 A deterministic code-to-intent graph

The substrate extracts a knowledge graph in a deterministic pipeline. Language adapters, built on incremental parsers such as tree-sitter, read source into a language-agnostic intermediate representation. Annotation resolution propagates intent links through inheritance, roles, and module scope. A graph builder fuses code symbols with intent artifacts written by humans or LLMs. A validation engine runs several dozen decidable rules, and the result is written as byte-stable output, so identical inputs produce identical graphs. The graph has four kinds of intent artifact joined by typed edges: *business requirements* (problem, goals, success criteria, user scenarios, constraints), which point down to features; *features* (capabilities and invariants), which point up to the requirement they serve; *entry points* (HTTP, CLI, cron, queue, and so on), each with a declared flow of the features it reaches; and *decisions*, which record what was ruled out and why.

6.2 Computing coordinates

The structural coordinate $s(u)$ is computed by a traceability walk that takes every success criterion through its bound invariants to their enforcers and verifiers and checks that each link resolves (Definition 3). The observational coordinate $o(u)$ is computed from ingested test results and mutation evidence (Definition 5). The semantic coordinate $m(u)$ is read from recorded human judgments keyed by the content hash of the unit’s texts (Definition 4). The status shown to a user is $v(x(u))$, with $x(u)$ and $F(u)$ available beneath it; showing the observational value under a truth-level name would hide exactly the off-path states of Definition 10.

The meaning router flags units that may have a coverage gap (Definition 12) and is an advisory route to a human, never a gate and never an automatic edit (Section 7.3). It is suppressed for regulatory, legal, and financial constraints, which are judged by humans by policy. Its intended default is on for every tier whose required state includes $\kappa_M = \text{confirmed}$, and switching it off is itself a governed change outside the agent’s write domain (Section 6.5).

6.3 Steering and accountability

Because the consumer is increasingly a set of agents, the design adds a weakest-link next-action recommender, file-and-version-control-backed *leases* so that concurrent agents claim disjoint slices, and an append-only *attribution ledger* that records every coordinate transition with its actor, evidence, and the autonomy mode in effect. Autonomy is an operator-set dial over which coordinates an agent may advance without a human, bounded below by Rule 4. The recommender proposes structural and observational work only (Section 7.5). Each ledger entry records the credential that issued the actor identity and the model family and version behind it. A unit whose every transition names one actor is flagged **self-verified**; for high tiers, separation is defined at the level of model family, or requires a human, because two instances of one model share blind spots (Zheng et al., 2023).

6.4 The change transaction

A requirement change is a transaction whose organizing rule is that grounded artifacts are never edited in place (Rule 2).

- **Priced and classed.** A proposed change records its targets and the verbatim previous text, computes the forward blast radius (criterion $\rightarrow$ invariants $\rightarrow$ enforcers $\rightarrow$ verifiers $\rightarrow$ scenarios $\rightarrow$ entry points, plus conflicting leases), and records a class under Rule 1.
- **Human-decided, with scoped demotion.** Approval re-grounds the text, which resets m for every touched unit (Definition 4) and raises the **demoted** flag, so stale green never rides through a change. A widening spawns work items. A narrowing spawns *retirement obligations*, the only thing that makes deleting a verifier legal (Rule 3). A contradiction is blocked until the conflicting decision is explicitly superseded. Industrial requirements tools mark every link touched by a change as suspect and tend to accumulate large numbers of suspect links (Cleland-Huang et al., 2014; Mäder and Gotel, 2012), so demotion is scoped by class and blast radius: a wording change demotes only the edited criterion.
- **Criticality is pinned.** A criterion carries a human-pinned tier (Definition 11). Human-signed mandates may delegate at most wording-class decisions to agents, under a tier ceiling and an expiry; under Rule 1 an agent cannot obtain the wording class for a change by declaring it.

Table 2: Trust boundary. Each asset must lie outside the coding agent’s write domain for the listed rules to bind a non-cooperative agent (Rule 7).

Asset	Rules that depend on it	Required protection
Grounded intent text (criteria, invariants, assumptions)	Rules 2 and 5; Definition 4	content hashes in the ledger; code-owner review on intent files
Verifier baseline	Rule 3	baseline signed and stored outside the agent’s branch
Ingested test results and mutation scores	Definition 5	produced by a CI runner under an identity the agent cannot assume; not writable from the workspace
Validator, rule set, thresholds θ_t , router switch	all rules	pinned by hash and run as a required check
Tier pins and mandates	Definition 11; Rule 4	human-pinned and hash-frozen
Ledger and actor identities	Rule 6; author separation	append-only, hash-chained, anchored outside the agents’ write domain (Schneier and Kelsey, 1999); identities bound to credentials the agent cannot mint
CI workflow definition	all rules	code-owner protection; changes are governed changes

- **Flags ride alongside status.** A unit with a non-empty flag set is shown with a superscript, for example `wiredf`. Validation cannot read clean while a flag is open (Definition 6).
- **Back-propagation and the baseline sweep.** After a merge, back-propagation maps changed files to enforcer symbols, invariants, and grounded criteria, and routes any impact to the human amendment gate, so the requirement never silently follows the code. A baseline sweep compares the current verifier set against a recorded baseline; a verifier that disappears without an approved retirement obligation is a detected violation.

The transaction is the change-axis completion of Section 3.1: the states are computed from facts, and the transitions between grounded states become facts too, priced, classed, attributed, and human-gated where the pressure to game concentrates.

6.5 Threat model and trust boundary

The adversary is the coding agent and any sub-agent it spawns, assumed capable and possibly non-cooperative. Humans are assumed honest but fallible (Section 10.2). By Rule 7, each rule is only as strong as the boundary around its inputs. Table 2 lists the assets the rules depend on and the protection each needs. The motivating incident of Section 2.3 is instructive here: the agent edited the CI workflow file, which is to say the checker. In the vocabulary of AI control (Greenblatt et al., 2024), the validator, the ledger, and the human reviewers are the trusted components and the coding agent is untrusted.

6.6 Status of design elements

Table 3 separates what the theory requires from what is one design option, and records how far each element has been realized. “Exercised” means the element produced a result reported in Section 8; “implemented” means it exists in the reference implementation but was not exercised there; “specified” means it is designed but not built; “conjectured” marks an untested idea.

Table 3: Status of design elements. “Required” elements are demanded by a definition or rule of Sections 3 to 5; “option” elements are one way to realize them.

Element	Role	Status
Traceability walk and link resolution	required (Definition 3)	exercised
Enforcer-kind and unfalsifiable-invariant checks	option	exercised
Journey coverage over declared flows	option	exercised
Ingested test results	required (Definition 5)	exercised
Mutation threshold on enforcers	required at mutation tiers	exercised (score measured on a four-mutant set)
Lexical meaning router (Stage 1)	option	exercised
NLI cross-encoder (Stage 2)	option	specified
Call-graph reachability sibling for flows	required (Principle 1)	specified
Overfit-test smell	option	specified
Leases, attribution ledger, autonomy modes	option	implemented
Change transaction: price, decide, demote, retire	required (Rule 3)	implemented
Fail-closed classification	required (Rule 1)	specified (the reference implementation records the proposer’s declared class)
Baseline sweep	required (Rule 3)	implemented
Flags and the self-verification flag	required (Definition 6)	implemented
Content-hash pins on grounded text	required (Rule 2)	specified
Hash-pinned oracles	option	specified
Hash-chained ledger anchored externally	required (Section 6.5)	specified
Out-of-process production monitors	required for production-observed	specified
Graph-guided delta mutation	option	specified
Dynamic invariant mining	option	specified
Adversarial tester agent	option	conjectured

7 Design responses to documented gaming

The evidence of Section 2.3 is a list of specific exploits. Table 4 maps each to the response the framework gives and to that response’s status. The subsections that follow explain the responses.

7.1 Against weakening the requirement

This is the most dangerous move because it leaves every proxy green. By Proposition 4, no check over the current criterion–invariant pair can see it: after a weakening the invariant still entails the weaker criterion. The defense is on the change axis. A weakening that passes through the transaction is recorded as narrowing unless a named human signs a lower class (Rule 1), it spawns retirement obligations rather than silent deletions (Rule 3), and its class is non-delegable. A weakening applied directly to grounded text is a pin violation (Rule 2). Credit accrues only to upward movement (Rule 6), so a weakening produces a recorded regression rather than a cleared check. What remains is the residual of Section 5.3: a human who approves a weakening they misjudge, and text that was never grounded.

Table 4: Documented gaming behaviors and the framework’s responses.

Documented behavior	Response	Axis	Status
Deleting failing tests (Nam, 2026; Zhong et al., 2025)	baseline sweep and retirement obligations (Rule 3)	O	implemented
Editing the CI workflow, patching verification functions, exiting early (Nam, 2026; Baker et al., 2025)	trust boundary (Section 6.5)	all	specified
Special-casing tests, hard-coding outputs (Stein et al., 2026a; Zhong et al., 2025; Ma et al., 2026)	mutation threshold; property-based and metamorphic oracles; overfit smell	O	exercised / specified
Modifying test files (Zhong et al., 2025; Gabor et al., 2025)	hash-pinned oracles	O	specified
Weakening the requirement	Rules 1 to 3	change	implemented / specified
Producing trace links to pass a check (Cleland-Huang et al., 2014)	provenance and the ungrounded flag (Rule 5)	S, M	implemented for requirements; specified for code annotations

7.2 Against special-casing the test

A passing test is a proxy, and agents demonstrably overfit it (Ma et al., 2026). Mutation is the adversarial backstop (Jia and Harman, 2011), but its mechanism needs stating carefully. Example-based tests with literal expected values are ordinary unit tests and kill most mutants on the paths they execute. Special-cased code evades them through low input diversity: mutants on the general logic that the tests never execute survive. The proposed overfit smell is therefore narrow: it flags verifiers whose inputs coincide with literals that the implementation special-cases. When the ledger shows that one actor wrote both a unit’s enforcer and its verifier, the unit is flagged for an independent pass (Section 6.3). Three further hardenings apply.

- **(a) Human-pinned oracles.** Because agents edit or delete tests (Zhong et al., 2025; Gabor et al., 2025), the oracle for a high-tier unit can be frozen: its content hash and approver are recorded, and any edit is a detected fact that lowers $o(u)$ and requires re-approval. Deletion is covered by the baseline sweep; content pinning covers edits. Industrial practice approximates this with golden files under code ownership, branch protection, and signed commits, and read-only tests are studied as an ImpossibleBench mitigation (Zhong et al., 2025). The claim made here is narrower: hash and approver are recorded as facts whose violation moves an assurance coordinate.
- **(b) Property-based and metamorphic oracles.** An example-based test asserts one input–output pair and is easy to special-case; a property-based test (a relation over all inputs) or a metamorphic relation has no single literal to hard-code. Property-based testing finds defects that example tests miss (Brinto Bose, 2025), and metamorphic checking detects a substantial fraction of LLM-generated errors without a human-written oracle (Wang and Zhu, 2024). The techniques are prior art; we prefer them for the top values of the observational axis because the author is an optimizer.

- **(c) Adversarial role separation.** A synthesis agent confined to a slice and a separate tester agent tasked with breaking it can communicate only through the graph. The tester’s output is treated as evidence and checked against human-pinned oracles; it is never rewarded by a count of breaks, which would itself be gamed by flaky or out-of-scope failures. Testers that share the synthesizer’s model family may share its blind spots. Whether such a pair avoids the coordination failures reported for multi-agent systems (Section 10.6) is a conjecture.

7.3 The meaning router: triage, not a decider

LLM judgments about how code relates to its documentation fail in systematic ways: in a study of documentation-to-code trace links, the share of LLM-written explanations of correct links that were rated fully correct ranged from 42.9% to 71.1% across models and projects (Alor et al., 2025). Judges in general show position, verbosity, and self-enhancement biases (Zheng et al., 2023; Wang et al., 2023; Dev et al., 2026). We therefore do not position the meaning check as deciding correctness. It is triage optimized for recall: it surfaces likely coverage gaps, ranked, for a human to adjudicate, and it reports “ N criteria flagged for meaning review”, never “ N meaning errors found”. A judge that is right 60% of the time is useless as a gate but useful as a router that makes risky links *reach* a human. The router addresses coverage gaps only; by Proposition 4, weakenings belong to the change axis.

The router is a three-stage triage. *Stage 1, lexical signal:* salient criterion vocabulary absent from the invariant. The probe in Section 8.3 shows that a Stage-1 pass cannot mean safe: coverage gaps that reuse the invariant’s own words pass it, so Stage 1 orders the queue and is never a silent pass. *Stage 2, entailment:* a small, locally hosted NLI cross-encoder (for example a DeBERTa-v3 model fine-tuned for entailment) run on *every* criterion–invariant pair; a non-entailment verdict escalates. *Stage 3:* a human adjudicates escalated units. Stage 2 should reduce false positives on paraphrases, where NLI models are reasonably reliable. On the vocabulary-preserving class the evidence points the other way. In stress tests of 2017-era sentence-encoder NLI models, accuracy on numerical-reasoning pairs fell below the 33% random baseline, and for the best and worst models 78.3% of errors were false entailments on high-overlap pairs (Naik et al., 2018), the lexical-overlap shortcut also documented by McCoy et al. (2019). On generalized-quantifier inference, DeBERTa-v3-base fine-tuned on MNLI scored 34.7% and the best model tested 48% (Cui et al., 2022), with proportional quantifiers such as “half” among the hardest. Neither study evaluated the proposed Stage 2 directly, so we state the bound as a prediction (Prediction 3): vocabulary-preserving coverage gaps are caught by no automated stage today and reach a human only through review. At the false-positive rate measured for Stage 1 on paraphrases (Section 8.3), every three faithful rewordings of a criterion would raise about two false flags, which is why Stage 2 exists.

7.4 Grounding the graph

A graph can be perfectly self-consistent and detached from what the user wanted, especially when an LLM authored both the requirement text and the code (Section 10.4). LLM trace links fail in systematic ways, among them naming-similarity assumptions, phantom links, and pattern over-generalization (Alor et al., 2025), and human analysts vetting candidate links also err (Cuddeback et al., 2010). Two responses follow. Provenance is a first-class field on every intent edge and every code-level annotation, so a reviewer sees which links rest on the documented failure modes (Rule 5). And LLM-authored text and links carry the **ungrounded** flag until a human vets them, which makes explicit that the human approval floor, not the graph’s internal consistency, grounds intent.

7.5 Against gaming the lattice itself: audit, don’t reward

The sharpest objection is recursive: *the truth-levels are a metric, so by the paper’s own argument an optimizer will game them*. The objection is correct. Three defenses apply. Individual checks are facts for their own properties (Proposition 1), so each must actually be made true. Credit accrues only to recorded, independently checkable movement (Rule 6), never to agent-narrated activity, which also avoids training agents to obfuscate against a monitor (Baker et al., 2025). And the truth-levels are an audit instrument rather than an optimization target. That last commitment has no force unless it names where it is enforced, and there are three channels.

At inference time the gate is an in-context target whatever the deployer intends, and the documented gaming happens with frozen weights. The design accepts this: the recommender proposes only structural and observational work, audit samples are drawn at random after the fact, the router’s internals and the audit sample are kept out of what the agent can observe, and high tiers may use held-out checks the agent never sees. **At training time** the commitment is a data-governance rule: truth-level transitions and ledger outcomes are never used as fine-tuning or reinforcement-learning labels. A deployer cannot impose this rule on third-party model vendors, and we state that limit plainly. **In the organization**, a coverage percentage reported as a team target invites the human compliance-gaming of Section 2.1 (Campbell, 1979). Coverage should be reported only next to the disagreement rate of the sampled human audit.

Open attack: vacuous declarations. The fact surface checks *resolution*, not *grounding*, and two attacks exploit the difference. An entry point’s flow is an authored declaration, so an agent can revert a scope-out catch by declaring a stub entry point whose flow names the scenario’s feature; journey coverage returns to green while the scope-out persists. The sibling check, static call-graph reachability from the declared handler to the feature’s symbols, is specified but not built, and static call graphs over-approximate under dynamic dispatch and reflection, so it is itself a proxy. An agent can also author a *hollow invariant*, tautological text (“the function terminates without a runtime error”) that resolves, wires, and passes its tests while constraining nothing the requirement asked. A screen for “semantic density” over invariant text would itself be a proxy subject to the meaning ceiling (Section 10.1); the defense is Rule 5, with hollowness named in the sampled audit’s checklist. Both instances are defeaters in the sense of Definition 6, and both illustrate Principle 1.

7.6 Criticality tiers, delta mutation, and human load

Tiers set the required filter of Definition 11 per unit. A low tier needs only the structural axis; a regulatory or financial invariant must reach (wired, confirmed, production-observed) and may invoke a formal checker, the depth that systems such as Astrogator reach in narrow domains (Section 9). Breadth by default, depth where the cost of error warrants it. An agent may propose a tier, never set one (Rule 4).

Graph-guided delta mutation. Full mutation testing per commit across a large repository is prohibitive. The graph narrows it: compute the symbols a commit modified, trace them through enforcement edges to the affected invariants, follow verifier links to the tests bound to those invariants, and restrict mutant generation to the modified symbols and the bound tests. Cost is then bounded by the change’s blast radius rather than by the product of all mutants and all tests, provided the blast radius stays within a slice. This is the established regression and diff-based optimization (Zhang et al., 2012; Petrović and Ivanković, 2018), which the graph supplies because slices and their links are first-class. Cost is proportional to symbols touched times linked tests, small per slice but not constant.

Human load. Routing meaning to humans raises the worry of a new bottleneck under a

high-throughput fleet. Counting only router invocations understates the load, because the rules of Section 5 create more human work than that. Let R be the rate of changes. Each governance obligation j (deciding non-wording changes, clearing flags including those raised by demotion, vetting LLM-authored text and annotations, pinning tiers and re-approving pinned oracles, independent passes on self-verified units, and adjudicating router flags) is triggered on a fraction p_j of changes, and a sampled audit reviews a fraction s of recorded transitions T per unit time:

$$H = R \sum_j p_j + sT. \quad (6)$$

Human load is linear in throughput, so the design attenuates the bottleneck and does not remove it. Tiering lowers the p_j by reserving semantic requirements for critical units, and Stage 2 lowers the router term, which Stage 1 inflates through false positives. Two consequences matter. A uniformly sampled audit sees any single residual weakening with probability s , so the residual defense of Section 5.3 is exactly as strong as the audit budget. And approval quality cannot be assumed; Section 10.2 discusses how to measure it.

7.7 The top of the observational axis: production

CI passage is not the strongest observational evidence; production observation is. The top value **production-observed** (Definition 5) requires a runtime monitor generated from the invariant to confirm that it held against real traffic, recorded in a tamper-evident, hash-chained ledger (Schneier and Kelsey, 1999) with build provenance for the agent action that produced the unit, in the style of SLSA (The SLSA Authors, 2025) and in-toto (Torres-Arias et al., 2019). Two deployment constraints keep this value out of the agent’s hands: monitors run as out-of-process observers (sidecars or log-derived assertions generated from the invariant), never as assertions compiled into the binary the agent wrote; and the ledger’s hash chain is anchored outside the agents’ write domain (Table 2), so that a privileged agent rewriting the log produces a detectable chain break. The components are prior art; the integration is treating production observation as a higher assurance state inside an intent-conformance graph, attributed per actor.

Mined invariants. Mutation asks whether the *test* pins the invariant. A complementary signal asks what invariants the *code* actually maintains: dynamic invariant mining (Ernst et al., 2007) over staging traces. When the mined invariants are weaker than the declared intent, the gap surfaces without a mutation run. The signal is bounded by trace coverage, so it complements mutation rather than replacing it.

8 An illustrative demonstration

This section illustrates, on one controlled requirement chain, the asymmetry the theory predicts: exact structural checks, an observational proxy that holds only with an adversarial backstop, and a meaning check that cannot stand in for a human. It is not a strong test of the theory. The structural and observational rules that fire below were written for the drift classes they catch, so their firing shows that the reference implementation behaves as specified. The informative content is the blindness of the baseline signals and the characterization of the meaning check in Section 8.3. The test that could refute the meaning claim is Prediction 3, not this run.

Design. Four expectations were written before the per-class run; there is no timestamped registration of them. **H1:** the baseline signals stay green on all four drift classes. **H2:** the structural drift classes are caught deterministically. **H3:** verifier weakening is caught only with mutation evidence. **H4:** the coverage gap is surfaced only by the lexical meaning check, and never

decided automatically. The population study and the probe of Section 8.3 were added after the per-class run and are exploratory. Two names have changed since the run: drift class M1 and the 20 additive corpus items were originally called requirement narrowings, and under Definitions 12 and 13 they are coverage gaps.

Setup. A controlled requirement chain was wired onto an existing refund feature: two success criteria bound to invariants INV-1 and INV-2, a user scenario verified through a declared CLI entry point, and ingested passing test results. In the clean reference state, validation exits 0, the traceability walk reports 2/2 criteria verified, journey coverage is 1/1, and the reference implementation labels both criteria **demonstrated**. In the coordinates of Section 4, the requirement document carries a human approval but no per-link semantic judgment was recorded, so each criterion sits at (wired, unjudged, ci-passed): the off-path state *demonstrated but unjudged*, with $v = \text{wired}$. The implementation’s label reported the observational value. No number below depends on this re-expression. Each drift was applied to an isolated copy:

1. **M1, coverage gap:** replace criterion SC-1 with a statement that claims scope INV-1 does not cover, so that INV-1 no longer entails it (Definition 12).
2. **M2, enforcer inflation:** replace a guard with a symbol that carries the enforcement annotation but guards nothing.
3. **M3, verifier weakening:** replace a verifier with one that passes while asserting strictly less.
4. **M4, scope-out:** declare the user-facing CLI entry point out of scope and re-point the user scenario’s verification from that entry point to a unit-level test, so the behavior is no longer verified where users reach it, while every other check stays green.

The baseline signals are those in common use: the code compiles, the tests pass, the validator’s exit status, and the percentage of criteria verified. The last counts criteria whose invariants have an enforcer and a verifier, with no failing test result and no mutation score below threshold. Because test results were ingested in every state reported here, it coincides with cov_κ (Section 4.4) for $\kappa = (\text{wired}, \text{unjudged}, \text{ci-passed})$. Several treatment rules are informational and do not change the exit status, which is why the baseline can stay green while they fire.

8.1 Results

Table 5 reports, per drift class, the baseline signals and the treatment outcome. All results were reproduced on 16 October 2026 with version 0.8.1 of the reference implementation.

- **H1 (baseline blindness), observed.** On all four drifts the code compiles, the tests pass, validation exits 0, and the headline criteria figure stays at 2/2. The drift is invisible to the signals current practice trusts.
- **H2 (structural layer), observed as built.** Enforcer inflation and scope-out are caught with no model in the loop and localized to the precise unit (INV-2; US-1). In the terms of Table 1, these are facts for syntactic properties and proxies for enforcement and reachability.
- **H3 (observational proxy), observed.** A weakened verifier that still passes leaves the observational coordinate untouched until mutation evidence arrives. Executing four mutants of the INV-1 enforcer (always true, always false, upper bound dropped, lower bound dropped), the original suite killed 4/4 and the weakened suite 1/4; the resulting score of 25% is below

Table 5: Seeded drift, one instance per class. The baseline signals stay green on all four; each drift surfaces where the theory places it.

Drift (class)	Baseline: compile / tests / exit / criteria verified	Treatment	Exp.
M1 coverage gap	✓ / ✓ / 0 / 2/2, unchanged	lexical check fires only when enabled, listing the criterion words absent from the invariant; a word difference, not an entailment judgment	H4
M2 enforcer inflation	✓ / ✓ / 0 / 2/2	enforcer-kind and unfalsifiable-invariant checks fire, deterministically, with no model	H2
M3 verifier weakening	✓ / ✓ / 0 / 2/2	no firing on test outcomes alone; with mutation evidence, survived-mutant and below-threshold checks fire, exit becomes 1, criteria verified 2/2 $\rightarrow$ 1/2	H3
M4 scope-out	✓ / ✓ / 0 / 2/2	criteria verified unchanged; journey coverage 1/1 $\rightarrow$ 0/1 and scenario-has-no-entry-point fires, deterministically	H2

the 80% threshold. The score was measured by running the four mutants and was then supplied to the validator; no mutation engine was run over the repository. In coordinates, SC-1 moves from (wired, unjudged, ci-passed) to (wired, unjudged, verifier-exists) while *v* stays wired, because *m* was already unjudged. The change is visible in the coordinate and invisible in the summary, which is the reason to keep the coordinate.

- **H4 (meaning check), observed.** The coverage gap is invisible to the baseline and to every structural rule, and surfaces only when the lexical check is enabled. That check flags any criterion with at least two salient words absent from the invariant. It would fire identically on some faithful paraphrases and miss gaps that reuse the invariant’s vocabulary, which Section 8.3 measures.

8.2 Limits of the per-class run

The run is one requirement chain with one seeded instance per class. It shows that each class is detectable in principle by the predicted mechanism; it does not measure detection rates. The drifts were constructed by the author of the rules that catch them, which is a construct-validity limit and not only an external-validity one: an adversarially optimized agent could produce instances the rules miss, as the open attacks of Section 7.5 illustrate. The documented cases of Section 2.3 remain the external anchor.

8.3 The meaning router over a labeled corpus

The meaning check is the contested mechanism (Section 10.1), so it was also run over a labeled corpus. INV-1 was held fixed (“A refund never exceeds the original amount charged for the order”) and criterion SC-1 was varied over 41 statements, all listed with their labels and outcomes in Appendix A. Labels follow Definitions 12 and 13: 20 *additive coverage gaps* (criteria claiming scope that INV-1 does not cover, in new vocabulary); 15 *meaning-preserving paraphrases* of INV-1; and 6 items built almost entirely from INV-1’s own words. Of these six, five are coverage gaps (a quantifier cut to “half”, an argument swap, and three recombinations that add obligations). The remaining item, “for the first order” (V2 in Appendix A), is under its intended reading a weakening that INV-1 entails, so a gap detector should stay silent on it.

Table 6: Outcomes of the lexical meaning check over the 41-statement corpus.

Class	n	flagged	not flagged
Additive coverage gap	20	20	0
Meaning-preserving paraphrase	15	10	5
Vocabulary-preserving coverage gap	5	1	4
Weakening (Definition 13)	1	0	1

The check is deterministic. It lowercases both texts, keeps words of at least four letters that are not on a fixed list of about thirty stop words, applies no stemming, and flags the criterion when at least two of its remaining words are absent from the invariant. The threshold is part of the shipped rule and was not tuned on this corpus; we have no timestamped record that it predates the corpus, which is one reason the study is reported as exploratory. Table 6 reports the outcomes.

Class-conditional rates, with 95% Wilson intervals (Wilson, 1927): recall on additive gaps $20/20 = 1.00$ [0.84, 1.00] (Clopper–Pearson lower limit 0.83; Clopper and Pearson, 1934); false-positive rate on paraphrases $10/15 = 0.67$ [0.42, 0.85]; recall on vocabulary-preserving gaps $1/5 = 0.20$ [0.04, 0.62]. The check catches every additive gap and flags two-thirds of faithful rewordings; it cannot tell synonymy or morphology from a gap (“The amount *refunded* never exceeds the original *charge*” is flagged because *refunded* and *refund* differ). It detects vocabulary introduction, not coverage gaps as such. Over the additive and paraphrase items, precision is $20/30 = 0.67$, so a third of flags were false; precision depends on the base rate of gaps and is reported for completeness only. The vocabulary-preserving items were written by the author inside the check’s known blind region, so 0.20 is a blind-spot existence proof, not a rate estimate. The weakening item was correctly left unflagged by a gap detector; by Proposition 4, detecting it requires the previous criterion text.

The corpus has one author and one labeler, no agreement statistic, and synthetic statements. A corpus written by several authors from real requirements, labeled independently by two people, is specified for follow-up work (Section 11).

8.4 False positives on benign edits

A benign-control run measured spurious firings on six states: the clean reference and five legitimate edits (a code comment, a faithful criterion paraphrase that reuses the invariant’s vocabulary, a reworded business problem, a reworded capability summary, and a guard-preserving local refactor), with the lexical check enabled. No rule fired on any of the six: 0/6 for the structural rules, 0/6 for the observational rules, and 0/6 for the lexical check. On these matched stimuli the two kinds of detector did not differ. The 10/15 false-positive rate of Section 8.3 arises on paraphrases that change criterion vocabulary, a stimulus class on which the structural rules were not run. By inspection of the reference rule set, no structural rule takes criterion text as input, so a rewording cannot trigger one; that is Prediction 1 holding as a design property, not a measurement. The matching stress set for the structural rules, meaning-preserving code refactors such as renaming or moving a guard, inlining an enforcer, or splitting a feature, was not run. With six trials, 0/6 is compatible with a true false-positive rate as high as 0.39 (Wilson upper limit; 0.46 by Clopper–Pearson). We report no significance test, because the available comparison would set different detectors against different stimulus sets. Table 7 assembles the results.

What a fuller study needs. A corpus of repositories with about 25 generated instances per class; every detector run on every stimulus, so that layers can be compared on paired outcomes with an exact McNemar test (McNemar, 1947); comparators including ranked-retrieval satisfaction assessment, one-shot LLM consistency checks, an NLI cross-encoder, and intent-formalization met-

Table 7: Assembled detection results. Benign false positives are over the six matched states of Section 8.4.

Drift class	Detecting mechanism	Detection	Benign FP
Enforcer inflation	structural	fired on the one seeded instance	0/6
Scope-out	structural	fired on the one seeded instance	0/6
Verifier weakening	observational, mutation-gated	fired with mutation evidence; silent without	0/6
Additive coverage gap	lexical router	20/20	0/6; 10/15 on vocabulary-changing paraphrases
Vocabulary-preserving coverage gap	lexical router	1/5; no automated stage today (Section 7.3)	—
Weakening	change axis (Section 5)	not exercised here	—

rics (Endres et al., 2024; Lahiri, 2024); drift instances produced by someone other than the rule author, including an agent instructed to turn the gate green; a mislink class, in which an annotation resolves but names the wrong feature; and an independently labeled corpus. We specify that protocol but do not run it here, and we do not report numbers we did not measure.

9 Related work

Requirements traceability and satisfaction. The **wired** level is classical traceability (Gotel and Finkelstein, 1994; Cleland-Huang et al., 2014), now increasingly mechanized with transformers and LLMs (Lin et al., 2021; Alor et al., 2025; Alturayef et al., 2026). The **correctly-meant** level descends from the satisfaction argument (Zave and Jackson, 1997), rich traceability (Dick, 2005), goal refinement (Darimont and van Lamsweerde, 1996), and Requirements Satisfaction Assessment, of which Sat-BERT and Dual-Sat-BERT (Poudel et al., 2023) are the closest automated work. The link-satisfaction distinction is not ours. Relative to this lineage, our contribution is to compute satisfaction as a coordinate of a version-controlled graph maintained from code, to fuse it with an observational axis, and to treat it as a human-routed judgment rather than a retrieval score. Weakening as a resolution tactic appears in obstacle analysis (van Lamsweerde and Letier, 2000); our change classes turn the direction of a weakening into a governance obligation.

Change impact analysis and traceability maintenance. Computing a forward blast radius over traceability links is the core operation of software change impact analysis (Bohner and Arnold, 1996), and maintaining links under change is studied as traceability maintenance (Mäder and Gotel, 2012). Commercial requirements tools mark links suspect when an endpoint changes (Cleland-Huang et al., 2014). The lifecycle layer reuses these operations. What it adds is classification by entailment direction with fail-closed defaults (Rule 1), retirement obligations as the only legal path to deleting a verifier (Rule 3), and pinned grounded text (Rule 2), so that an unapproved route around the transaction is a detected fact.

Executable specifications. BDD (North, 2006) and Specification by Example (Adzic, 2011) make acceptance criteria executable, which corresponds to our observational axis, but they stop at “a scenario passes” and have no mechanism for a passing test that proves less than intended, no traceability graph, and no semantic axis.

Assurance cases and dynamic assurance. Graded, multi-dimensional assurance is not new. Assurance-case confidence arguments in Goal Structuring Notation (Kelly and Weaver, 2004) and the assurance deficit of Hawkins et al. (2011) separate a claim from confidence in it, and defeater-based argument makes doubts explicit (Goodenough et al., 2013; Bloomfield and Rushby, 2020). Dynamic safety cases update assurance from operational data (Denney et al., 2015; Calinescu et al., 2018). Bloomfield and Rushby (2024) take a dependability perspective on AI systems broadly: they tie evidence to criticality, use runtime verification, argue that AI-generated designs and code can be guarded by independent analysis, and separate requirements validation, intent verification, and correctness verification, much as our semantic and structural axes do. We adopt these as prior art. The difference we claim is narrower: assurance computed by static analysis from a code-to-business-intent graph on every commit, with a semantic axis that only humans can move, an agent-attribution ledger, and a meaning router.

Verifying LLM code against intent. Astrogator (Councilman et al., 2025) formally verifies LLM-generated Ansible code against a query in a natural-language-like formal language, which an LLM drafts and the user confirms. Over 21 tasks it accepted 277 of 334 correct programs and rejected 856 of 926 incorrect ones. It differs from this work in method (formal verification rather than continuous traceability) and scope (one domain), and its confirmation step already embodies a human anchor. We treat it as the depth complement of Section 7.6. The intent-formalization program (Endres et al., 2024; Lahiri, 2024, 2026) states that validating specifications is the central bottleneck for reliable AI coding and that the user is the only oracle for specification correctness. That is the meaning ceiling in another vocabulary. Our contribution relative to it is the continuous, graph-scoped routing of the semantic axis over brownfield code, not the observation itself.

Concurrent code-to-intent tools. lat.md (Selivanov, 2026) builds “a knowledge graph for your codebase, written in markdown” with a deterministic check that “enforces referential consistency”; it checks that links resolve and so occupies the **wired** level. ReqToCode (Schlathölter, 2026) embeds requirements as traceables with bidirectional links validated during the build, a compile-time realization of the same level. Qodo argues that generating and verifying systems require different approaches and targets whole-system review (Park, 2026); Tessl targets spec-driven generation (Podjarny, 2025). The field is converging on the **wired** level; none of these computes the three axes as one state or states where meaning is irreducible.

Spec-driven development. Intent-to-code pipelines check by generation, one-shot LLM consistency checks, or test passage. Practitioner assessments report the failure mode targeted here: a “false sense of security” in which an agent marks a verification task done without writing a test (Zaninotto, 2025), and a spec-maintenance strategy often “left vague” (Böckeler, 2025). Rather than compete on generation, the framework fits beneath such generators: it can take a spec-driven specification as requirement input and supply a persistent, version-controlled assurance state and an explicit meaning ceiling.

Code knowledge graphs. Code property graphs (Yamaguchi et al., 2014) and repository-level graphs for LLMs (Guo et al., 2021; Ouyang et al., 2025; Liu et al., 2025) represent code structure for retrieval, search, and generation, not whether code satisfies a natural-language requirement.

Multi-agent software engineering, provenance, and attestation. Orchestration frameworks coordinate agents through a shared message pool (Hong et al., 2024), a chat chain (Qian et al., 2024), or hierarchical delegation (Wang et al., 2025), and benchmarks measure coordination quality (Zhu et al., 2025). Signed, per-step attribution of software supply-chain actions is the purpose of in-toto (Torres-Arias et al., 2019) and SLSA (The SLSA Authors, 2025), and our ledger draws on provenance models (W3C, 2013) and tamper-evident logging (Schneier and Kelsey, 1999). Leases are a classical coordination mechanism. The narrower claim here is attribution of assurance-coordinate transitions inside an intent graph. In support of the constraints principle,

imposed structure is the central mechanism of the orchestration frameworks above; shaping the interface an agent works through improves its success (Yang et al., 2024); and a controlled study that holds the base model fixed finds success “critically dependent on the architectural strategy” (Zeng et al., 2025).

Specification gaming and control. Specification gaming (Krakovna et al., 2020), reward hacking (Amodei et al., 2016), Goodhart variants (Manheim and Garrabrant, 2018), and unhackability (Skalse et al., 2022) are, to our knowledge, not previously connected to a constructive software-assurance substrate; the fact/proxy distinction of Section 3.1 is the bridge. The router plus sampled audit is a control protocol in the sense of Greenblatt et al. (2024), with trusted and untrusted components and a fixed audit budget.

Net novelty. No individual component is unprecedented. The contribution is (1) the synthesis: three separately computed axes as one lattice-valued state over a code-to-intent graph, with a canonical path and a meet-preserving valuation; (2) the theory of intent change that separates gaps from weakenings and makes weakening a governed, recorded event; (3) the re-motivation as a defense against autonomous specification gaming rather than documentation discipline or certification; and (4) the fact/proxy and constraints principles that make an opinionated substrate the right answer because the author is a machine.

10 Threats to validity and limitations

10.1 The meaning ceiling

The **correctly-meant** level is a judgment, and current evidence indicates that it cannot yet be mechanized reliably. For the judgment that matters most, whether code is correct for its task, the best LLM judge in one study (GPT-4-turbo) reached Cohen’s $\kappa \approx 0.21$ on Java and 0.10 on Python against ground truth, and misjudged half of the incorrect Java implementations (Crupi et al., 2025). On whether a patch is correct, an LLM judge agreed with human consensus at $\kappa = 0.57$, with precision 0.65 (Shi et al., 2025). Judge reliability also depends on a human anchor: on questions the judge could not answer correctly itself, agreement fell from $\kappa = 0.81$ with a human reference answer to 0.16 with no reference and 0.13 with a self-generated one, measured on business and finance questions rather than code (Krumdick et al., 2026). Judges inherit order effects; a verdict survived swapping the answer order 65% of the time for one frontier model and 24% for another (Zheng et al., 2023), and rankings can be changed by reordering alone (Wang et al., 2023). A stress-testing harness across four benchmarks and four judges found no judge uniformly reliable, with verdicts flipped by paraphrase, verbosity, and formatting (Dev et al., 2026).

The counter-evidence deserves stating. On general chat quality, a strong LLM judge agreed with human experts above 80% of the time, about as often as the experts agreed with each other (Zheng et al., 2023). That result is reference-anchored and concerns chat, not code, so it does not reach the reference-free code judgments above. Human agreement also sets a floor. Three experts judging patch correctness with their own rubrics agreed at Fleiss $\kappa = 0.31$, rising to 0.66 with a shared rubric (Shi et al., 2025). A model cannot be shown to be more reliable than the human judgments it is validated against.

Two consequences follow. First, autonomy is capped at the structural and observational axes: an autonomous agent may advance s and o , including through runtime monitors to **production-observed**, but by Definition 4 it cannot move m . Second, because judge reliability holds up only when anchored to human ground truth, the architecture must supply that anchor rather than assume a model can (Section 7.4).

10.2 The human anchor is itself a proxy

By Definition 1, a human sign-off is also a proxy: an observable signal standing in for whether the link means what was asked. It is costly and noisy. Experts judging patch correctness with their own rubrics agreed at Fleiss $\kappa = 0.31$ (Shi et al., 2025). Review capacity is falling behind volume, and reviewer comments on agent-generated pull requests are more often neutral or positive even though those changes carry more redundant code (Section 2.3). Automation complacency and bias predict that an operator left with the hardest residual judgments, among many false alarms, drifts toward approval (Bainbridge, 1983; Parasuraman and Manzey, 2010). An agent that writes criterion text for a human to approve is, in effect, optimizing against that human. The top value of the semantic axis therefore guarantees only that a named human judged the current texts; its reliability is that of the human process. The design should measure that process rather than assume it: show approvers the verbatim old and new text and the blast radius, withhold the agent’s narration, require two approvers for high tiers, re-review a random sample to estimate agreement, track approval rate and time to approve as indicators of rubber-stamping, and seed known weakenings into the queue to measure how many approvers catch. Separation of duties matters too: whoever approves a high-tier narrowing should not own that feature’s delivery date. The ledger shows an auditor who approved a change and its blast radius; it does not show that the approval was informed, and accountability can settle on the person least able to exercise it (Elish, 2019).

10.3 The observational axis is still a proxy

Even hardened with mutation and overfit detection, observation rests on tests, and tests are proxies. The claim is not that `ci-passed` proves correctness; it is that the observational axis is an adversarially hardened, auditable proxy that is never conflated with the semantic axis. A determined adversary can still construct a test that survives mutation, is not overfit, and is semantically weak; catching that falls back to Section 10.1.

10.4 Internal consistency, not correspondence to reality

If an agent authors both the requirement text and the code, it can make them mutually entailing while both drift from what the user wanted. Provenance and the `ungrounded` flag make this visible but do not solve it; the human approval floor does the grounding. A graph with no human-authored anchor is self-referential by construction. A related case is out of scope by design (Section 3.3): a human who grounds the wrong requirement.

10.5 Common-mode failure

Several mechanisms assume a kind of independence. Two instances of one model, or models with shared training data, may share blind spots, so author separation keyed to actor names can overstate independence; Section 6.3 defines separation at the level of model family for high tiers. Evidence channels can also share a failure mode with the agent if the agent can write them; in the demonstration, the mutation score was supplied to the validator by the experimenter. Table 2 states the channels that must lie outside the agent’s write domain.

10.6 The fleet premise is contested

Concurrent multi-agent editing of one repository is not yet common practice, and the premise that a coordinated set of agents outperforms one strong agent is contradicted by current evidence.

CooperBench (Khatua et al., 2026) reports that two agents on a shared codebase achieve about 30% lower joint success than one agent doing both tasks. Cemri et al. (2025) find coordination gains often minimal and catalog fourteen failure modes, and E2EDevBench (Zeng et al., 2025) finds that agents fulfill about half of requirements end to end, limited by omitted requirements and inadequate self-verification. The steering layer of Section 6.3 targets those documented failure modes of an unsolved problem; it does not assume fleets already work, and it is one application of the theory rather than a co-equal contribution.

10.7 Adoption cost and generality

The opinionated architecture (vertical slices, no cross-feature imports, line caps, command and query separation) and the annotation burden are real costs, and not every system fits the shape. Principle 2 applies to machine-authored code. For human-maintained brownfield systems the constraints are not cheap, and adoption has to be incremental, with traceability capped at the structural axis until slices exist. The residual machine costs of Section 3.2 apply in both settings.

10.8 The demonstration

Section 8 is one requirement chain on one repository, with one seeded instance per structural and observational class and a single-author corpus for the meaning check. It illustrates the predicted asymmetry and characterizes one detector; it does not establish detection rates across repositories or robustness to adversarially optimized gaming.

11 Discussion and future work

The framework’s least intuitive consequence is that the path to more autonomous AI software runs through more constraint, but only up to the meaning ceiling, beyond which the path is better human routing rather than more automation. The research agenda follows the predictions. (1) Scale Section 8 to a corpus of repositories with paired detector-by-stimulus outcomes and the comparators listed in Section 8.4. (2) Test Prediction 3 with a corpus written by several people from real requirements and labeled independently, running an NLI cross-encoder as Stage 2 and intent-formalization metrics as comparators. (3) Test Prediction 4 as a control evaluation: a red-team agent tries to land a weakening through the transaction and around it, under conventions and under enforced rules, and the record shows which attempts were detected. (4) Measure the costs of Prediction 5: tokens per accepted change, annotation precision against a human-labeled sample, and task success on cross-cutting changes with and without slice confinement. (5) Measure the terms of Equation (6) and approver recall in one organization, using seeded known weakenings. (6) Replay documented gaming traces, such as the ImpossibleBench test-modification corpus (Zhong et al., 2025), against the substrate and record which rules fire.

Active contract synthesis, scoped to the formalizable tier. A tempting alternative replaces the meaning router with a formal gate: declare intent as SMT-backed refinement types (Vazou et al., 2014; Lehmann et al., 2023) or verified contracts (Leino, 2010; de Moura and Bjørner, 2008), so that a compiler rejects a violating implementation before any test runs. We adopt these as enrichments of specific tiers, not as a new core. Refinement types make a fact of the code, but whether the predicate captures the natural-language intent is the same human judgment (Section 10.1) in a different syntax; formal contracts move the meaning ceiling rather than remove it (Lahiri, 2026). They also cover few of the languages that business software uses, so making formal synthesis the substrate would trade generality for depth. The natural place for them is the

high-criticality, formalizable tier, where Section 7.6 already routes them, beneath a breadth-first lattice.

12 Conclusion

Current AI-first software development points an optimizer at a codebase with broad freedom and certifies the result with proxies that such optimizers are known to game, including by weakening the requirement until the defect is out of scope. We argued that constraints that are friction for human authors cost a machine author little, and that they make it possible to check the code-to-intent link with facts rather than proxies alone. The assurance lattice keeps done-ness legible without collapsing it: three separately computed axes, a canonical path of four truth-levels with a meet-preserving valuation, and named off-path states such as *demonstrated but unjudged*. The question in the title has two parts. Whether a criterion is covered by its invariants is a property of a state, routed to a human because no automated judge is yet reliable enough. Whether the bar was lowered is a property of a change. No check over the current state can see it, and the rules make it a recorded, human-signed event. What remains is a human who misjudges a change, text that was never grounded, and breaches of the trust boundary. Much of the framework is already built, its predictions can be tested, and its central limit is stated where it arises: the semantic axis is a human judgment today.

Acknowledgments

The author used AI assistants (Anthropic Claude) for literature and citation checks, simulated peer review, and editing. All definitions, claims, and results were checked by the author, who takes full responsibility for the content.

Data and artifact availability

This preprint is archived at <https://doi.org/10.5281/zenodo.23432741>. The seeded-drift harness (the per-class script, the four-mutant measurement, and the corpus study) is deposited with this preprint as supplementary material, and the full corpus with labels and outcomes appears in Appendix A. The harness runs against the Lattice reference implementation, which is not part of this deposit; the results in Section 8 were reproduced with its version 0.8.1.

References

- G. Adzic. *Specification by Example*. Manning, 2011.
- Ebube Alor, SayedHassan Khatoonabadi, and Emad Shihab. Evaluating the use of LLMs for documentation to code traceability. *arXiv:2506.16440*, 2025.
- Noof Alturayef, Irfan Ahmad, and Jameleddine Hassine. TraceLLM: Leveraging large language models with prompt engineering for enhanced requirements traceability. *Requirements Engineering*, 31(1), 2026. doi: 10.1007/s00766-026-00460-1. Article 6. *arXiv:2602.01253*.
- D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv:1606.06565*, 2016.

- Shan Appajodu and Ravi Boyapati. Scaling code reviews: Adapting to a surge in AI-generated code. <https://engineering.salesforce.com/scaling-code-reviews-adapting-to-a-surge-in-ai-generated-code/>, January 2026. Salesforce Engineering blog, January 29, 2026. Accessed 2026-10-16.
- Lisanne Bainbridge. Ironies of automation. *Automatica*, 19(6):775–779, 1983. doi: 10.1016/0005-1098(83)90046-8.
- B. Baker et al. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv:2503.11926*, 2025.
- Robin Bloomfield and John Rushby. Assurance 2.0: A manifesto. *arXiv:2004.10474*, 2020.
- Robin Bloomfield and John Rushby. Assurance of AI systems from a dependability perspective. CSL Technical Report SRI-CSL-2024-02, Computer Science Laboratory, SRI International, Menlo Park, CA, July 2024. *arXiv:2407.13948*. Revised as SRI-CSL-2024-02R3, June 2025 (*arXiv v3*).
- Birgitta Böckeler. Understanding spec-driven-development: Kiro, spec-kit, and Tessl. <https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html>, October 2025. *martinfowler.com*, series “Exploring Gen AI”, October 15, 2025. Accessed 2026-10-17.
- Barry W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1):75–88, 1984. doi: 10.1109/MS.1984.233702.
- Shawn Bohner and Robert Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, 1996. ISBN 978-0-8186-7384-9.
- D. Brinto Bose. From prompts to properties: Rethinking LLM code generation with property-based testing. In *FSE Companion*, 2025. DOI 10.1145/3696630.3728702.
- Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069, 2018. doi: 10.1109/TSE.2017.2738640.
- Donald T. Campbell. Assessing the impact of planned social change. *Evaluation and Program Planning*, 2(1):67–90, 1979. doi: 10.1016/0149-7189(79)90048-X.
- Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, pages 135676–135729, 2025. doi: 10.52202/085713-4082.
- Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: Trends and future directions. In *Future of Software Engineering Proceedings (FOSE 2014), co-located with ICSE 2014*, pages 55–69. ACM, 2014. doi: 10.1145/2593882.2593891.
- C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404.
- Aaron Councilman, David Jiahao Fu, Aryan Gupta, Chengxiao Wang, David Grove, Yu-Xiong Wang, and Vikram Adve. Towards formal verification of LLM-generated code from natural language prompts. *arXiv:2507.13290*, 2025.

- Giuseppe Crupi, Rosalia Tufano, Alejandro Velasco, Antonio Mastropaolo, Denys Poshyvanyk, and Gabriele Bavota. On the effectiveness of LLM-as-a-judge for code generation and summarization. *IEEE Transactions on Software Engineering*, 51(8):2329–2345, August 2025. doi: 10.1109/TSE.2025.3586082.
- David Cuddeback, Alex Dekhtyar, and Jane Huffman Hayes. Automated requirements traceability: The study of human analysts. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE 2010)*, pages 231–240. IEEE, 2010. doi: 10.1109/RE.2010.35.
- R. Cui, D. Hershcovich, and A. Sjøgaard. Generalized quantifiers as a source of error in multilingual NLU benchmarks (GQNLI). In *NAACL*, 2022.
- Robert Darimont and Axel van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT ’96)*, pages 179–190. ACM, 1996. doi: 10.1145/239098.239131.
- Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.
- Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, Ethan Perez, and Evan Hubinger. Sycophancy to subterfuge: Investigating reward-tampering in large language models. arXiv:2406.10162, 2024.
- Ewen Denney, Ganesh Pai, and Ibrahim Habli. Dynamic safety cases for through-life safety assurance. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE 2015)*, pages 587–590. IEEE, 2015. doi: 10.1109/ICSE.2015.199.
- Sunishchal Dev, Andrew Sloan, Joshua Kavner, Nicholas Kong, and Morgan Sandler. Judge reliability harness: Stress testing the reliability of LLM judges. arXiv:2603.05399, 2026. RAND Corporation. Accepted at the Agents in the Wild: Safety, Security, and Beyond Workshop at ICLR 2026.
- Jeremy Dick. Design traceability. *IEEE Software*, 22(6):14–16, 2005. doi: 10.1109/MS.2005.150.
- Madeleine Clare Elish. Moral crumple zones: Cautionary tales in human-robot interaction. *Engaging Science, Technology, and Society*, 5:40–60, 2019. doi: 10.17351/ests2019.260.
- Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K. Lahiri. Can large language models transform natural language intent into formal method postconditions? *Proceedings of the ACM on Software Engineering*, 1(FSE):1889–1912, 2024. doi: 10.1145/3660791.
- M. D. Ernst et al. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 2007.
- J. Gabor, J. Lynch, and J. Rosenfeld. EvilGenie: A reward hacking benchmark. arXiv:2511.21654, 2025.
- John B. Goodenough, Charles B. Weinstock, and Ari Z. Klein. Eliminative induction: A basis for arguing system confidence. In *Proceedings of the 35th International Conference on Software*

- Engineering (ICSE)*, *New Ideas and Emerging Results track*, pages 1161–1164. IEEE, 2013. doi: 10.1109/ICSE.2013.6606668.
- O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proc. ICRE*, 1994.
- Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. AI control: Improving safety despite intentional subversion. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, volume 235 of *Proceedings of Machine Learning Research*, pages 16295–16336. PMLR, July 2024. URL <https://proceedings.mlr.press/v235/greenblatt24a.html>.
- Andrea Griffiths. Agent pull requests are everywhere. Here’s how to review them. <https://github.blog/ai-and-ml/generative-ai/agent-pull-requests-are-everywhere-heres-how-to-review-them/>, May 2026. The GitHub Blog, May 7, 2026. Accessed 2026-10-16.
- D. Guo et al. GraphCodeBERT: Pre-training code representations with data flow. In *ICLR*, 2021. arXiv:2009.08366.
- Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In Chris Dale and Tom Anderson, editors, *Advances in Systems Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium*, pages 3–23. Springer, London, 2011. doi: 10.1007/978-0-85729-133-2_1.
- E. Ashlee Holbrook, Jane Huffman Hayes, and Alex Dekhtyar. Toward automating requirements satisfaction assessment. In *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE’09)*, pages 149–158. IEEE, August 2009. doi: 10.1109/RE.2009.10.
- Elizabeth Ashlee Holbrook, Jane Huffman Hayes, Alex Dekhtyar, and Wenbin Li. A study of methods for textual satisfaction assessment. *Empirical Software Engineering*, 18(1):139–176, 2013. doi: 10.1007/s10664-012-9198-8.
- Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, July 2025. URL <https://www.trychroma.com/research/context-rot>. July 14, 2025. Accessed 2026-10-16.
- S. Hong et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *ICLR*, 2024.
- Haoming Huang, Pongchai Jaisri, Shota Shimizu, Lingfeng Chen, Sota Nakashima, and Gema Rodríguez-Pérez. More code, less reuse: Investigation on code quality and reviewer sentiment towards AI-generated pull requests. In *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR ’26)*, pages 1024–1028. ACM, 2026. doi: 10.1145/3793302.3793622. Preprint title: “More Code, Less Reuse: Investigating Code Quality and Reviewer Sentiment towards AI-generated Pull Requests”, arXiv:2601.21276.
- Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE TSE*, 2011.
- René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd*

- ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, pages 654–665. ACM, 2014. doi: 10.1145/2635868.2635929.
- Eran Kahana. Built by agents, tested by agents, trusted by whom? <https://law.stanford.edu/2026/02/08/built-by-agents-tested-by-agents-trusted-by-whom/>, February 2026. CodeX, Stanford Law School, February 8, 2026. Accessed 2026-10-16.
- Tim P. Kelly and Rob A. Weaver. The goal structuring notation – a safety argument notation. In *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, Florence, Italy, 2004. No DOI; pages not verified.
- A. Khatua et al. CooperBench: Evaluating collaborative coding agents. *arXiv:2601.13295*, 2026.
- V. Krakovna et al. Specification gaming: The flip side of AI ingenuity. DeepMind blog, 2020.
- Michael Krumdick, Charles Lovering, Varshini Reddy, Seth Ebner, and Chris Tanner. No free labels: Limitations of LLM-as-a-judge without human grounding. In *Conference on Language Modeling (COLM 2026)*, 2026. arXiv:2503.05061 (first posted March 2025).
- Shuvendu K. Lahiri. Evaluating LLM-driven user-intent formalization for verification-aware languages. In Nina Narodytska and Philipp Rümmer, editors, *Proceedings of the 24th Conference on Formal Methods in Computer-Aided Design (FMCAD 2024)*, pages 142–147, Vienna, 2024. TU Wien Academic Press. doi: 10.34727/2024/isbn.978-3-85448-065-5_19.
- Shuvendu K. Lahiri. Intent formalization: A grand challenge for reliable coding in the age of AI agents. arXiv:2603.17150, 2026.
- Nico Lehmann, Adam T. Geller, Niki Vazou, and Ranjit Jhala. Flux: Liquid types for Rust. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1533–1557, 2023. doi: 10.1145/3591283.
- K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010. doi: 10.1007/978-3-642-17511-4_20.
- Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. Traceability transformed: Generating more accurate links with pre-trained BERT models. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 324–335. IEEE, 2021. doi: 10.1109/ICSE43902.2021.00040.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pages 21558–21572, 2023. doi: 10.52202/075280-0943.
- Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. CodexGraph: Bridging large language models and code repositories via code graph databases. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers) (NAACL 2025)*, pages 142–160. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.7.

- Yanuo Ma, Ben Kereopa-Yorke, and Ben Schultz. Building to the test: Coding agents deliver what you check, not what you requested. *arXiv:2606.28430*, 2026.
- Patrick Mäder and Orlena Gotel. Towards automated traceability maintenance. *Journal of Systems and Software*, 85(10):2205–2227, 2012. doi: 10.1016/j.jss.2011.10.023.
- D. Manheim and S. Garrabrant. Categorizing variants of goodhart’s law. *arXiv:1803.04585*, 2018.
- Manus. Context engineering for AI agents: Lessons from building Manus. <https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus>, July 2025. Manus blog, July 18, 2025. Accessed 2026-10-16.
- R. T. McCoy, E. Pavlick, and T. Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference (HANS). In *ACL*, 2019.
- Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.
- Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. Stress test evaluation for natural language inference. In Emily M. Bender, Leon Derczynski, and Pierre Isabelle, editors, *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*, pages 2340–2353, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL <https://aclanthology.org/C18-1198/>.
- Jeongho Nam. AI deleted my tests and said ‘all tests pass’ — a horror story from porting ‘typia’ from TypeScript to Go. <https://dev.to/samchon/ai-deleted-my-tests-and-said-all-tests-pass-a-horror-story-from-porting-typia-from-typescript-2bmf>, May 2026. DEV Community, posted May 3, 2026 (author handle: samchon); mirrored at <https://typia.io/blog/ai-deleted-my-tests-and-said-all-tests-pass/>. Accessed 2026-10-16.
- Dan North. Introducing BDD. *Better Software*, March 2006. Republished at <https://dannorth.net/blog/introducing-bdd/>. Accessed 2026-10-17.
- Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI software engineering with repository-level code graph. In *International Conference on Learning Representations (ICLR 2025)*, 2025. *arXiv:2410.14684*.
- Palisade Research. Demonstrating specification gaming in reasoning models. *arXiv:2502.13295*, 2025.
- A. Pan, K. Bhatia, and J. Steinhardt. The effects of reward misspecification. In *ICLR*, 2022. *arXiv:2201.03544*.
- Raja Parasuraman and Dietrich H. Manzey. Complacency and bias in human use of automation: An attentional integration. *Human Factors*, 52(3):381–410, 2010. doi: 10.1177/0018720810376055.
- Kate Park. Qodo raises \$70M for code verification as AI coding scales. <https://techcrunch.com/2026/03/30/qodo-bets-on-code-verification-as-ai-coding-scales-raises-70m/>, March 2026. TechCrunch, March 30, 2026. Accessed 2026-10-17.
- G. Petrović and M. Ivanković. State of mutation testing at google. In *ICSE-SEIP*, 2018.

- Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. Practical mutation testing at scale: A view from Google. *IEEE Transactions on Software Engineering*, 48(10):3900–3912, 2022. doi: 10.1109/TSE.2021.3107634.
- Guy Podjarny. Announcing Tessel’s products to unlock the power of agents. <https://tessel.io/blog/announcing-tessels-products-to-unlock-the-power-of-agents>, September 2025. Tessel blog, September 16, 2025. Accessed 2026-10-17.
- Amrit Poudel, Jinfeng Lin, and Jane Cleland-Huang. Leveraging transformer-based language models to automate requirements satisfaction assessment. arXiv:2312.04463, 2023.
- C. Qian et al. ChatDev: Communicative agents for software development. In *ACL*, 2024.
- Prithvi Rajasekaran, Ethan Dixon, Carly Ryan, and Jeremy Hadfield. Effective context engineering for AI agents. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>, September 2025. Anthropic Engineering blog, September 29, 2025 (written by Anthropic’s Applied AI team). Accessed 2026-10-16.
- RTCA. DO-178C: Software considerations in airborne systems and equipment certification, 2011.
- Thorsten Schlathöller. ReqToCode: Embedding requirements traceability as a structural property of the codebase. arXiv:2603.13999, 2026. Preprint, not peer-reviewed.
- B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM TISSEC*, 1999.
- Yury Selivanov. lat.md: Agent lattice, a knowledge graph for your codebase, written in markdown. <https://github.com/vercel-labs/lat.md>, 2026. Software repository (vercel-labs), created March 2026; project site <https://www.lat.md>. Accessed 2026-10-17.
- Sherry Shi, Renyao Wei, Michele Tufano, José Cambronero, Runxiang Cheng, Franjo Ivančić, and Pat Rondon. Towards a human-in-the-loop framework for reliable patch evaluation using an LLM-as-a-judge. arXiv:2511.10865, 2025.
- J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger. Defining and characterizing reward hacking. In *NeurIPS*, 2022. arXiv:2209.13085.
- Adam Stein, Davis Brown, Hamed Hassani, Mayur Naik, and Eric Wong. Finding widespread cheating on popular agent benchmarks. <https://debugml.github.io/cheating-agents/>, April 2026a. DebugML blog, posted April 10, 2026. Accessed 2026-10-16.
- Adam Stein, Davis Brown, Hamed Hassani, Mayur Naik, and Eric Wong. Detecting safety violations across many agent traces. arXiv:2604.11806, 2026b. Preprint, under review.
- Marilyn Strathern. ‘Improving ratings’: Audit in the British university system. *European Review*, 5(3):305–321, 1997. doi: 10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4.
- The SLSA Authors. SLSA: Supply-chain levels for software artifacts, specification v1.2. <https://slsa.dev/spec/v1.2/>, November 2025. Open Source Security Foundation project; v1.2 released November 2025. Accessed 2026-10-17.
- Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1393–1410, Santa Clara, CA, August 2019. USENIX

- Association. ISBN 978-1-939133-06-9. URL <https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias>.
- Axel van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000. doi: 10.1109/32.879820.
- Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. Refinement types for Haskell. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming (ICFP ’14)*, pages 269–282. ACM, 2014. doi: 10.1145/2628136.2628161.
- W3C. PROV-overview: An overview of the PROV family of documents, 2013.
- P. Wang et al. Large language models are not fair evaluators. *arXiv:2305.17926*, 2023.
- X. Wang and D. Zhu. Validating LLM-generated programs with metamorphic prompt testing. *arXiv:2406.06864*, 2024.
- X. Wang et al. OpenHands: An open platform for AI software developers. In *ICLR*, 2025. arXiv:2407.16741.
- Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi: 10.1080/01621459.1927.10502953.
- F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and discovering vulnerabilities with code property graphs. In *IEEE S&P*, 2014.
- J. Yang et al. SWE-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024.
- François Zaninotto. Spec-driven development: The waterfall strikes back. <https://marmelab.com/blog/2025/11/12/spec-driven-development-waterfall-strikes-back.html>, November 2025. Marmelab blog, November 12, 2025. Accessed 2026-10-17.
- Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997. doi: 10.1145/237432.237434.
- Z. Zeng, Y. Li, R. Xie, W. Ye, and S. Zhang. Benchmarking and studying the LLM-based agent system in end-to-end software development (E2EDevBench). *arXiv:2511.04064*, 2025.
- L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. Regression mutation testing. In *ISSTA*, 2012.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, pages 46595–46623, 2023. doi: 10.52202/075280-2020.
- Z. Zhong, A. Raghunathan, and N. Carlini. ImpossibleBench: Measuring LLMs’ propensity of exploiting test cases. *arXiv:2510.20270*, 2025.
- K. Zhu et al. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *ACL*, 2025. arXiv:2503.01935.

A Corpus for the meaning-router study

The invariant is INV-1: “A refund never exceeds the original amount charged for the order.” *Relation* gives the label under Definitions 12 and 13; *Should flag* is the correct behavior of a coverage-gap detector; *Flagged* is the measured outcome of the lexical check.

ID	Criterion statement	Relation to INV-1	Should flag	Flagged
A1	A refund is issued to the customer original payment method within five business days.	gap (additive)	yes	yes
A2	Refunds are processed automatically without manual support intervention.	gap (additive)	yes	yes
A3	A refund is credited to the same card used for the original purchase.	gap (additive)	yes	yes
A4	Only completed and fully shipped orders are eligible for a refund.	gap (additive)	yes	yes
A5	A refund triggers an email confirmation sent to the customer.	gap (additive)	yes	yes
A6	Partial refunds are allowed only once per calendar month.	gap (additive)	yes	yes
A7	A refund requires approval from a finance administrator before processing.	gap (additive)	yes	yes
A8	Refund requests submitted after ninety days are rejected.	gap (additive)	yes	yes
A9	A refund restores loyalty points proportional to the returned amount.	gap (additive)	yes	yes
A10	Refunds are denied for orders flagged by fraud detection.	gap (additive)	yes	yes
A11	A refund is recorded in the audit log with a timestamp.	gap (additive)	yes	yes
A12	The refund must be initiated by the purchasing customer only.	gap (additive)	yes	yes
A13	A refund converts the order status to canceled.	gap (additive)	yes	yes
A14	Refunds above one hundred dollars require a manager signature.	gap (additive)	yes	yes
A15	A refund is issued in the original currency of the transaction.	gap (additive)	yes	yes
A16	A refund notifies the warehouse to expect a returned item.	gap (additive)	yes	yes
A17	Refunds are batched and settled nightly through the payment gateway.	gap (additive)	yes	yes
A18	A refund deducts a restocking fee from the returned amount.	gap (additive)	yes	yes
A19	A refund is available only to verified account holders.	gap (additive)	yes	yes
A20	The refunded total appears on the next monthly statement.	gap (additive)	yes	yes
P1	A refund never exceeds the original amount charged.	paraphrase	no	no
P2	The refund amount never exceeds the original charged amount.	paraphrase	no	no
P3	No refund exceeds the original amount charged.	paraphrase	no	no
P4	A refund is never larger than the original amount charged.	paraphrase	no	no
P5	The amount refunded never exceeds the original charge.	paraphrase	no	yes

ID	Criterion statement	Relation to INV-1	Should flag	Flagged
P6	A reimbursement must not surpass the sum originally billed.	paraphrase	no	yes
P7	Money returned can not be greater than what the buyer initially paid.	paraphrase	no	yes
P8	The credited value stays at or below the initial transaction total.	paraphrase	no	yes
P9	A repayment is capped at the value of the initial purchase.	paraphrase	no	yes
P10	The sum given back is at most the price the customer first paid.	paraphrase	no	yes
P11	A refund returns no more than the original amount charged.	paraphrase	no	no
P12	The refund can equal but not go beyond the original charge.	paraphrase	no	yes
P13	Refund value is bounded above by the original amount charged.	paraphrase	no	yes
P14	The payout never tops what was originally charged to the order.	paraphrase	no	yes
P15	A refund stays within the original amount charged.	paraphrase	no	yes
V1	A refund never exceeds half the original amount charged for the order.	gap (vocab.-preserving)	yes	no
V2	A refund never exceeds the original amount charged for the first order.	weakening	no	no
V3	The original amount charged for the order never exceeds a refund.	gap (vocab.-preserving)	yes	no
V4	A refund for the order never exceeds the amount charged, and the original order is never charged again.	gap (vocab.-preserving)	yes	no
V5	Each order is charged the original amount, and a refund never exceeds that amount more than once.	gap (vocab.-preserving)	yes	yes
V6	A refund never exceeds the original amount charged for the order it was charged for, and never the amount of any other order.	gap (vocab.-preserving)	yes	no