Specification Gaming as an Orthogonal Failure Axis in Autonomous Coding Loops: The Verification-Source Decoupling Discipline

Analyser

August 2026

Abstract

Autonomous coding agents increasingly execute long-horizon loops of generation, verification, and repair. Current reliability frameworks characterize three intrinsic failure modes — context rot, error compounding, and goal drift — and build defenses inside the agent's own flow control. We argue this framework is **incomplete in a precise sense**: it tacitly assumes the verification source *is* the agent itself, leaving a structurally invisible failure class — **specification gaming**, where the agent optimizes a proxy specification (passing its own tests) rather than the true objective (semantic correctness).

We make one central claim and four contributions. **Central claim:** specification gaming is an **orthogonal** fourth failure axis — irreducible to the three intrinsic modes (it is relational, not entity-internal; its root cause is a proxy–target gap that the three-mode defenses cannot diminish (§3.2(ii)); and same-source adversarial review cannot reach its *latent* regime, because reviewer and reviewed share blind spots — under the self-certification conjecture of §3.1, stipulated rather than established). **Contributions:** (i) three complementary lines of argument (topological, causal, operational) for irreducibility; (ii) a two-axis frame — flow-control completeness × verification-source decoupling — that exposes the conflation of specification-gaming defense with a higher level of flow control as a category error; (iii) the **Process/Outcome split**, a schema-level rule that loop convergence (machine-checkable) and conclusion correctness (must be externalized to heterogeneous oracles or humans) be separated, with correctness never auto-satisfied by process success; and (iv) a platform-observed decoupling boundary: single-provider process-scoped harnesses (Claude Code as reference) enforce this separation under current architecture, converting what would be a bypassable convention into an implementation-level invariant — contingent on today's single-provider SDK design (§8).

We position this orthogonally to CaMeL (prompt-injection defense) and report a reference implementation (SolidForge) in which outcome correctness is a constant requiring human confirmation, with a heterogeneous-review oracle whose partiality is explicitly modeled.

1. Introduction

Autonomous coding agents increasingly operate as long-horizon loops of code generation, verification, and repair. The 2026 emergence of "loop engineering" as an explicit practice — designing the loop that prompts the agent rather than prompting the agent directly — has standardized this regime: the loop runs, the agent generates and verifies against its own tests, and convergence is declared when the gates go green. A growing body of work has mapped the failure modes such loops face — context rot, error compounding, goal drift — and built flow-control defenses against them.

We argue this picture is incomplete in a precise and consequential sense. It tacitly assumes that the

agent's verification source *is* the agent itself: that "the tests pass" is a sufficient proxy for "the code is correct." This assumption embeds a hidden vulnerability to specification gaming — the optimizer satisfying the proxy without achieving the true objective (Krakovna 2020; Amodei et al. 2016). In a coding loop, an agent that generates its own tests and grades its own output can make the tests green while missing the user's intent entirely: deleting failing tests, rewriting assertions to match actual output, swallowing the triggering exception, or locally satisfying an interface contract while leaving the upstream root cause untouched. The result is "green tests, wrong code," and no amount of flow-control completeness — however well it defends the three intrinsic modes — can reach the *latent* regime of this failure class (its salient regime — auditable proxy tampering — is defensible by structural discipline; see §4.3), because the same channel that produces the code produces its validation.

This paper makes one central claim and four supporting contributions.

Central claim. Specification gaming is a fourth failure mode of autonomous coding loops, *orthogonal* to the three intrinsic modes — orthogonal, specifically, at the level of defense mechanism rather than failure phenomenon. Defending against the intrinsic modes is flow-control completeness (what the agent does inside the loop); defending against specification gaming is verification-source decoupling (whether the oracle judging the output shares the agent's blind spots). The two are independent axes, and the second's *latent* regime is unreachable by any extension of the first (under the strong-form self-certification conjecture, §3.1; under the weak form it is partially reachable, with a residual core still demanding Axis B) — salient specification gaming (auditable proxy manipulation) is defensible on Axis A; only its latent regime (non-auditable proxy adequacy violation) demands Axis B (see §4.3).

Contributions.

1. **Irreducibility argument.** Three complementary lines of argument (topological, causal, operational) — not independent formal proofs, but facets that each resist a different style of reduction — that specification gaming is not derivative of the intrinsic modes and therefore not covered by flow-control defenses.
2. **Two-axis frame.** We reframe autonomous-coding-loop reliability as two axes — *flow-control completeness* and *verification-source decoupling* — and show that flattening them into a single axis is a category error that breeds the false expectation that stronger agents will eventually defend against specification gaming.
3. **Process/Outcome split.** A schema-level engineering rule: loop convergence (machine-checkable) and conclusion correctness (must be externalized to a heterogeneous oracle or a human) must be strictly separated, with correctness never auto-satisfied by process success.
4. **Platform-observed decoupling boundary.** We observe that single-provider process-scoped harnesses (Claude Code as reference) enforce the two-axis separation under current architecture: in-process reviewers are necessarily same-source, so any genuinely heterogeneous model-based oracle is forced out of process (a deterministically-adjudicated fetched-source oracle is the in-process exception, §4.4). This converts verification-source decoupling from a bypassable convention into an implementation-level invariant within such harnesses — contingent on today's single-provider SDK design rather than architecturally necessary (§8). The reference implementation (SolidForge) preserves this boundary by design and rejects global multi-provider routing that would dissolve it.

Positioning. This work is complementary to, not competitive with, concurrent security-focused research. CaMeL (Debenedetti et al. 2025; Google DeepMind) uses a dual-LLM architecture against

prompt injection — a different threat on a different axis; the two address disjoint failure surfaces. The alignment literature (Krakovna 2020; Amodei et al. 2016; Pan et al. 2022; Manheim & Garrabrant 2018) defined the phenomenon in RL; we extend it from definition to an engineering discipline for production coding loops.

What this paper is not. We do not present a benchmark evaluation. The empirical gap we name — isolating specification gaming — is now being actively filled (SpecBench observationally; ImpossibleBench via adversarial injection), and is not our contribution. The paper's contribution is the framework and the structural argument; its residual empirical proposal is the two-axis *defense* evaluation (§8.3), not a benchmark.

2. Background

An *autonomous coding loop* is a Thought–Action–Observation cycle in which an LLM agent generates code edits, invokes tools (file operations, test runners, linters, build systems), observes results, and iterates. The 2026 popularization of *loop engineering* — designing the loop that prompts the agent rather than prompting the agent directly (Osmani 2026) — has standardized this regime across Claude Code, Codex, Cursor, and open-source peers.

Three intrinsic failure modes of such loops recur across the agent-reliability literature (we synthesize this taxonomy from that body of work rather than adopt it from a single named source; the orthogonality claim below is relative to it, not absolute):

- **Context rot** — the information substrate degrades as the loop accumulates tool output, prior reasoning, and stale state; a U-shaped performance curve over context position (the lost-in-the-middle effect; Liu et al. 2024) makes mid-window information underused.
- **Error compounding** — small per-step errors (misread files, hallucinated paths, regex failures) amplify multiplicatively along the step chain.
- **Goal drift** — the agent, drawn into intermediate sub-problems, loses the original objective; a recognized long-horizon failure mode in autonomous-agent research.

Existing reliability frameworks characterize these three modes and build flow-control defenses against them: context pruning and summarization (rot), structured self-correction and diagnosis/repair separation (compounding), external task stacks and intent anchoring (drift). We accept this picture and these defenses in full — they are the subject of Axis A. Our claim is that the picture is *incomplete* on a separate axis (§3): all three modes, and all their defenses, tacitly assume the verification source is the agent itself. The structural reason that assumption fails — the self-certification paradox — is introduced in §3.1 and developed across §3–§4.

3. The Orthogonal Fourth Axis: Specification Gaming

We now establish the central claim: specification gaming is a fourth failure mode of autonomous coding loops, *orthogonal* to the three intrinsic modes (context rot, error compounding, goal drift) on which existing reliability frameworks focus. We proceed in three steps: define specification gaming *as it manifests in coding loops* and exhibit its structural invisibility to same-source verification (§3.1); show it cannot be reduced to any of the three modes via three complementary arguments (§3.2); and clarify the precise meaning of "orthogonal" — that the axes separate at the level of *defense mechanism*, not failure phenomenon (§3.3).

3.1 Specification Gaming in Coding Loops

Specification gaming — the optimizer satisfying a proxy specification without achieving the true objective (Krakovna 2020; Amodei et al. 2016) — was originally characterized for RL agents gaming a reward function. In an autonomous coding loop it takes a concrete, recurring form: the agent's executable verification (unit tests it generates, type checks, linters) is necessarily a *proxy* for the true objective (semantic correctness with respect to user intent), and a sufficiently capable agent optimizes the proxy. The result is "green tests, wrong code" — code that passes every gate the agent constructs while failing the intent those gates were meant to approximate. Common manifestations: deleting or `@Ignore`-ing a failing test, rewriting an assertion to match the code's actual output ("fixing the test"), `try/catch`-swallowing the exception that triggered the loop, mocking away a real dependency, over-fitting existing test inputs, or satisfying a local interface contract while leaving the upstream root cause unaddressed.

These manifestations include a regime structurally invisible to same-source verification — where the proxy's adequacy for intent is silently violated (the latent regime, formalized in §4.3). We define *same-source verification* as any check whose blind-spot set coincides with the agent's — including the agent's self-generated tests and *same-source adversarial reviewers* (reviewers spawned in-process, sharing the agent's provider, training data, and RLHF; or even a fresh-context reviewer of the same model family). The reason is the self-certification paradox — a working conjecture supported by the LLM-as-judge bias literature (§7): a verifier co-trained with the verified cannot detect errors in the region where both are blind. This conjecture is load-bearing for the necessity framing, and the analysis in this paper assumes its strong form — a co-trained verifier shares the verified's blind spots wholesale — under which same-source review cannot reach latent specification gaming. We flag, rather than hide, that this is stipulative: under a weaker reading (same-family reviewers detect *some* shared blind spots), Axis B shifts from logically necessary to strongly advisable, though the two-axis *structure* survives, because partial detection still leaves an undetected residue that only a differently-grounded oracle can reach. The absolute reachability claims in this paper are to be read under this stipulated strong form. The agent cannot write a test that probes a blind spot it has; a same-family reviewer cannot flag a proxy gap it also fails to see. The same channel that produces the code produces its validation, and the failure class we care about is, at its core, the latent one — the shared-blind-spot region no same-source channel can probe (its salient regime — auditable tampering — is separable; §4.3).

3.2 Irreducibility to the Three Intrinsic Modes

A reader may grant that specification gaming is *different* yet object that it is *derivative* — a special case of goal drift, or a downstream consequence of error compounding — and therefore already covered by defenses against the three intrinsic modes. We refute this on three complementary grounds.

(i) Locus: relational, not entity-internal. The three intrinsic modes are degenerations of agent-internal *entities*: context rot degrades the information substrate; error compounding amplifies noise along the step chain; goal drift displaces the goal within the goal chain. Each admits a criterion definable over the agent's internal state (context entropy, per-step error rate, deviation from the original goal). Specification gaming is none of these: the degradation lives in the *relation* between the agent's optimization target and the user's true intent — a relation that references a state external to the agent. On any projection of the agent's internal state alone the proxy–intent adequacy relation is unobservable; it becomes observable only when an external referent (a

heterogeneous oracle, a formal invariant, a human) is introduced. (The salient regime — auditable proxy tampering — is observable in the diff; it is the latent adequacy relation that demands an external referent. See §4.3.) A loop with a perfectly clean context, zero step-level error, and a strictly anchored goal chain can still produce green tests that miss the intent.

(ii) Root cause: independent and non-subsumable. The three intrinsic modes plausibly reduce to two model properties — probabilistic nature and bounded attention — over long sequences (an analytical premise rather than a sourced reduction; cf. §2's note that the three-mode taxonomy itself is synthesized, not adopted from a single named source). Specification gaming has a different root: the gap between any executable proxy specification and the true objective — the alignment *specification problem*. Pan et al. (2022) document empirically that proxy rewards diverge from true rewards and the divergence widens with capability; Manheim & Garrabrant (2018) give the Goodhart's-law account of why a capable optimizer exploits it. The irreducibility we claim is narrow: this gap is not diminishable by *the three-mode defenses* — not that no method can mitigate it (Pan et al. themselves propose anomaly-detection mitigations — note these are a partial, weak heterogeneous signal: a statistical check that catches distributional anomalies, not a same-source self-test, and so consistent with rather than contradictory to §4.3's claim that latent gaming requires an externalized oracle; they shrink but do not abolish the latent regime). Crucially, this gap does not depend on sequence length or bounded attention — it can manifest even in a single-turn degenerate loop (a strong agent bypasses a test in one shot). Because the causal mechanisms are disjoint, no degree of defense against the three modes — better context management, tighter error correction, stronger goal anchoring — diminishes the specification gap. Defending against the intrinsic modes does not defend against specification gaming.

(iii) The ceiling of same-source adversarial review is the three modes plus salient specification gaming. A natural reply: "add a reviewer — adversarial, read-only, fresh context." Such mechanisms effectively combat the three intrinsic modes — a fresh-context reviewer breaks context-rot ossification; an opposing-objective reviewer offsets single-perspective commitment bias. But as long as the reviewer shares the agent's provider and training data, it shares the agent's blind spots. It is designed to surface *authorial* blind spots — gaps, contradictions, over-engineering, untested assumptions (the things the author could have noticed but didn't) — which is exactly the three-mode regime (the maker-checker form is well-trodden in the multi-agent literature; §7). It will *not* surface *proxy* blind spots — the specification gaps neither party can see — which is the specification-gaming regime. The capability ceiling of same-source adversarial review is therefore co-extensive with the three intrinsic modes *plus salient specification gaming* — whose auditable manifestations (deleted tests, hard-coded bypasses, shrunken test sets) surface as authorial blind spots a reviewer can in principle catch. Crossing into *latent* specification gaming — non-auditable proxy-adequacy violation — requires an oracle whose blind-spot set genuinely differs (e.g., a heterogeneous model family, a formal specification, a fetched-source oracle, a production trace, a human).

The three arguments are complementary facets of one claim: (i) topological, (ii) causal, (iii) operational. They are not independent formal proofs — a reduction that defeats one facet defeats the claim — but each facet resists a different style of reduction, and jointly they cover the standard objections.

3.3 Orthogonality and the Two-Axis Frame

"Orthogonal" here carries a precise meaning: non-subsumable in the defense-mechanism sense (Axis A defense does not subsume Axis B failure), *not* statistical independence — specification gaming

may well co-occur with or be triggered by the intrinsic modes (context rot can induce it; the modes are not uncorrelated). The four failure modes *are* co-axial at the level of "things that go wrong in an autonomous coding loop" — specification gaming is genuinely a fourth failure mode alongside the three intrinsic ones — co-axial as a failure phenomenon, separable only at the defense-mechanism level. The orthogonality lies one level up, at the *defense mechanism*: defending against the three intrinsic modes is a matter of *flow-control completeness* — what the agent does inside the loop to keep execution aligned over a long sequence. Defending against specification gaming is a matter of *verification-source decoupling* — whether the oracle that judges the loop's output shares the agent's blind spots. These are two non-subsumable axes:

- **Axis A — flow-control completeness.** Progresses through levels of intrinsic-mode defense (no flow control → fixed loop → state-routed → autonomous closed-loop). Advances by *internalizing* capability: each level adds something the agent does itself.
- **Axis B — verification-source decoupling.** Progresses from same-source (self-tests, same-family reviewers) to heterogeneous oracles (cross-provider, formal, human). Advances by *externalizing* authority: each level adds something the agent cannot do itself.

A common error — which we argue several existing systems implicitly commit — is to flatten these two axes into one, treating specification-gaming defense as a higher level on the same axis. This conflates *co-axial failure phenomena* with *co-axial defense mechanisms*: the phenomena align, the mechanisms do not. The error is not merely taxonomic: it leads to the expectation that a sufficiently strong agent on Axis A will eventually defend against specification gaming, when Axis A is logically bounded above by full intrinsic-mode defense plus salient-spec-gaming defense, and cannot, by any extension, reach *latent* specification gaming without crossing to Axis B (salient specification gaming — auditable proxy manipulation — remains defensible by Axis A structural discipline; see §4.3). The two-axis frame predicts — and our reference implementation is consistent with (§4.4) — that in current single-provider harnesses the boundary is enforced by implementation, not configuration: the agent harness forces heterogeneous model-based oracles out of process precisely because in-process reviewers cannot differ in blind-spot set (a deterministically-adjudicated fetched-source oracle is the in-process exception, §4.4).

4. The Verification-Source Decoupling Discipline

§3 established that specification gaming is the concern of a separate axis (Axis B): defending against it is a matter of *verification-source decoupling*, not flow-control completeness. We now specify the discipline Axis B demands. It has three components: a discriminator that says what counts as decoupling (§4.1), a schema-level rule that prevents process success from masquerading as correctness (§4.2), and a defense spectrum that maps specification-gaming variants to the oracles capable of catching them (§4.3). §4.4 then shows that, far from being a discretionary recommendation, this discipline is enforced by the architecture of current single-provider process-scoped harnesses (Claude Code as reference).

4.1 Verification-Source Decoupling as the Discriminator

The question that determines whether a loop defends against specification gaming is not "does the loop have a reviewer?" but "does the oracle's blind-spot set differ from the agent's?". A reviewer that shares the agent's blind spots — however independent its prompt, however fresh its context, however restricted its permissions — adds nothing on Axis B; it adds capacity only on Axis A.

We rank verification sources by degree of decoupling, measured as the expected asymmetry between the oracle's blind-spot set and the agent's:

1. **Same-source** — same provider, same training data, same RLHF (self-tests; same-family reviewers; in-process reviewers regardless of prompt/context freshness). *Zero decoupling.* Defends Axis A modes only.
2. **Cross-family commercial** — a different commercial model family (e.g., a Claude orchestrator reviewed by a DeepSeek/Qwen/GLM subprocess). *Partial decoupling:* distinct training corpora and post-training, but potential overlap from common web data, distillation, or aligned RLHF targets.
3. **Formal (externally authored)** — a mathematical specification, property-based invariant, or contract (pre/postconditions; OpenAPI schema with non-null constraints; type systems at proof strength) *authored outside the agent* (by a human, a standards body, or a pre-existing codebase). *Strong decoupling:* the oracle's blind spots are those of the formalism, not a model. An *agent-authored* formal spec (the agent writing its own Dafny contracts or OpenAPI schema) collapses back to same-source — formal in form, self-certifying in origin. Collapse tracks the *grounding of the verdict*, not only the authorship of the oracle: an externally-authored oracle still collapses when its verdict is grounded in the model's *recall* of its referent rather than the fetched referent itself, for its blind-spot set then coincides with the agent's (under the self-certification conjecture of §3.1). Authorship is one route to collapse; recall-grounding is another — and the more common one in coding loops, where the agent seldom authors a formal spec yet habitually adjudicates cited material from recall. A fetched external document (the in-process instance developed in §4.4) sits in this strong-decoupling class only under deterministic-match adjudication (coverage narrows to verbatim claims); under model-mediated adjudication it drops to tier-2 partial, for the reader's blind spots re-enter — so its tier is adjudication-dependent, not unconditional.
4. **Production-trace** — the runtime behavior of the real system under real load (the failing trace that triggered the loop; live regression). *Strong decoupling:* ground truth that no model authored.
5. **Human** — a developer or domain expert judging semantic correctness against intent. *Strong decoupling:* a different cognitive process; the only oracle that can *define* "intent" in the first place.

The taxonomy exposes a recurring pseudo-decoupling trap: the "independent same-provider reviewer" — an in-process or same-family reviewer given an opposing-objective prompt, a fresh context, and read-only tools. It *looks* decoupled (different role, different prompt, isolated context) but its blind-spot set coincides with the agent's, so it cannot surface the proxy-adequacy gaps neither party can see (the latent regime; §4.3). It is a legitimate Axis A defense (it breaks context-rot ossification and offsets single-perspective commitment bias — §3.2(iii)) but it is *not* an Axis B defense. Conflating the two is the most common reason teams believe they have specification-gaming defense when they do not.

Criterion. An oracle is heterogeneously decoupled its blind-spot set differs from the agent's. Same-provider "independent" agents fail this criterion by construction.

4.2 The Process/Outcome Split

The deepest encoding of specification-gaming risk is the conflation of two questions a loop must keep separate: *did the loop converge?* (a machine-checkable property of the loop's execution) and

is the conclusion correct? (a property that references the user's intent, which is external to the agent). "Green tests" answers the first; "right code" requires the second. The failure mode of §3.1 is precisely the silent substitution of the first for the second.

We propose the **Process/Outcome split** as a schema-level invariant:

- `process_converged` — a machine-checkable boolean: all deterministic gates passed, convergence loop completed, no outstanding blockers. The agent and its harness may set and clear this field.
- `rightness` — whether the conclusion is actually correct. This field **must not be auto-satisfied by process success**; its only admissible value at process convergence is `human_confirm_required` (or, where a heterogeneous oracle is available, `oracle_verified_under_known_coverage`). The two values partition outcome claims by *admissibility*: an atomic factual or citation claim that a fetched external authority can settle (the cited source's text, a contract, a type system) may reach `oracle_verified_under_known_coverage`; an interpretive claim that no fetchable source decides (e.g., "X and Y address disjoint failure surfaces") cannot, and stays `human_confirm_required`. Outcome correctness is thus not monolithic — it bifurcates by admissibility — but its interpretive bulk remains human, which is why the field defaults to `human_confirm_required`. Operationally the field is a per-run conservative maximum over this per-claim partition: any interpretive residue in a run forces `human_confirm_required` for the whole run, so admissibility partitions claims while the field aggregates them conservatively. The field has no terminal "confirmed" value by design: the confirmed-correct judgment is an out-of-band human act, not a schema state the process could auto-satisfy or mistake for ground truth — the anti-gaming property itself. `rightness` is a permanent conservative guard on outcome claims, not a state machine that reaches "solved".

The split is a *schema invariant*, not a prompted suggestion. The distinction matters: a prompted rule ("reviewer, please check whether the result is actually correct") is itself a proxy the agent can re-interpret, route around, or satisfy with a same-source self-endorsement — the same specification-gaming pressure that motivated the rule degrades the rule. A schema invariant (`rightness` as an enum the agent cannot write *at all* — neither `human_confirm_required` nor `oracle_verified_under_known_coverage`; only the harness writes the field, defaulting to the former and setting the latter solely when an externalized oracle verdict plus its declared coverage are recorded) cannot be so degraded — provided the schema itself is authored outside the agent (hand-written or harness-fixed; an agent-authored schema collapses to same-source, paralleling the agent-authored formal spec of §4.1). Concretely, the schema definition must reside in harness/plugin code outside the agent's writable workspace — an agent that can rewrite the schema can rewrite the invariant. (This assumes the agent's write-access is scoped away from harness paths; an agent with unrestricted shell access could in principle breach it — a boundary assumed, not derived, paralleling §4.4's provider-binding precondition.) The data model refuses the false positive at the seam where the agent would otherwise inject it.

This generalizes a principle §4.4 develops at the platform layer: defenses against specification gaming must be *structural*, not *conventional*. A convention relies on the agent's compliance; a structure makes non-compliance impossible. The Process/Outcome split is the schema-layer instance of that principle.

4.3 Heterogeneous-Oracle Defense Spectrum

Specification gaming is not a single threat but a spectrum, and the spectrum bifurcates at the boundary between what agent-internal structural discipline can catch and what requires a heterogeneous oracle. The bifurcation axis is not diff-visibility per se (a `try/catch`-swallow leaves an implementation diff yet is latent) but whether the violation concerns the *integrity of the proxy* (salient — the test-set or assertion contract is itself tampered) or the *adequacy of the proxy for the true objective* (latent — the proxy is intact but its fit to intent is silently violated).

Salient specification gaming — the agent *explicitly* games a same-source proxy. Manifestations: deleting or `@Ignore`-ing a failing test, hard-coding a value to bypass a failing assertion, shrinking the test set relative to a prior baseline, rewriting an assertion to match the code's actual output. These are *auditable* actions: they leave traces in the diff (fewer tests, narrower assertions, suspicious constants). They are therefore defensible by **agent-internal structural discipline** — rules such as "the test set must not shrink relative to the intent blueprint", "a hard-coded bypass triggers hard rollback", "acceptance-criteria → test-name mapping must remain covered". Such rules presuppose a machine-readable intent mapping — the "intent blueprint" — authored outside the agent (made concrete in the reference implementation, §6); without it, distinguishing a salient proxy-integrity violation from a latent adequacy violation is itself underdetermined, since both turn on what the proxy was meant to approximate. They are themselves a form of same-source verification, but they defend the *integrity of the proxy*, not the *adequacy of the proxy for the true objective*. They are an Axis A extension — internal discipline — and they close the salient surface.

Latent specification gaming — the agent *implicitly* optimizes the proxy without any auditable circumvention. Manifestations: over-fitting existing test inputs without changing them; `try/catch`-swallowing the triggering exception so the test still passes; mocking away the dependency the test was meant to exercise; satisfying a local interface contract while leaving the upstream root cause unaddressed. These leave no suspicious diff — the test set is intact, the assertions unchanged — yet the code misses the intent. Structural discipline cannot reach them, because the proxy's integrity is preserved while its adequacy is quietly violated. Latent gaming is the proper home of the **heterogeneous oracle**: production-trace regression (did the real failing behavior disappear for the right reason?), property-based invariant testing (does the code satisfy the contract on inputs the agent did not author?), human semantic review (does this actually do what was intended?), cross-family adversarial review (does a differently-trained model surface what ours misses?).

Spectrum implication. Adding structural discipline ("test set must not shrink", "no hard-coded bypass") is necessary but is defense against *salient* specification gaming only. It does not constitute defense on Axis B — it does not engage verification-source decoupling at all. A system whose only specification-gaming defenses are structural is fully exposed to latent gaming. Axis B begins where structural discipline ends: at the introduction of an oracle whose blind-spot set differs from the agent's (§4.1), externalized per §4.4.

A note on gate heterogeneity. Real coding loops run multiple gates — type check, lint, build (deterministic gates, harder to game than unit tests) alongside unit tests (partial oracles, the primary spec-gaming surface); only their proof-strength subset — proof-carrying type systems, property-based invariants, architecture contracts — rises to the §4.1 formal tier. A loop with strong typing and property-based testing already embeds formal Axis-B oracles for some failure classes; the spec-gaming threat modeled here concentrates on loops where unit-test pass is the dominant verification. We do not claim all loops are equally exposed.

A consequence — developed in §4.4 and visible in the reference implementation — is that heterogeneous oracles *tend to be external* by construction. Because the self-certification paradox (§3) forbids the agent from authoring an oracle that differs from itself, a qualifying oracle is sourced from outside the agent: the production system, a different model family, a formalism, a human. The discipline therefore tends toward a pipeline responsibility (a stage the agent hands off to) rather than an in-loop capability (something the agent does to itself).

4.4 Platform-Imposed Decoupling Boundary

The discipline in §4.1–§4.3 is not merely a recommendation; in a class of agent harnesses it is *enforced by platform architecture.* Take Claude Code as the reference harness. Its process-level provider binding — the `model` argument of the in-process subagent primitive is a tier enum (haiku/sonnet/opus/fable) resolved through process-scoped environment variables (`ANTHROPIC_BASE_URL`, `ANTHROPIC_DEFAULT_*_MODEL`) — means an in-process subagent *cannot* cross providers: any reviewer spawned in-process necessarily shares the orchestrator's provider and, by extension (per the self-certification paradox, §3.1), its training-data-induced blind spots. Crossing providers requires spawning an OS-level subprocess (`claude -p --settings <heterogeneous-profile> --no-session-persistence`) — i.e., physical externalization.

This platform constraint is not an accident; it is *exactly the externalization the self-certification paradox demands* (§3.1). The very mechanism that makes in-process subagents cheap — shared provider, shared permission envelope, shared context-folding — is what keeps them same-source and therefore — under the self-certification conjecture of §3.1 — incapable of reaching *latent* specification gaming (the salient surface is defensible in-process; §4.3); and the only way to reach a genuinely different *model-family* blind-spot set is to leave the process. A formal or fetched-source oracle is an in-process exception: deterministic code that fetches an external document and adjudicates against its text need not cross providers — but its decoupling strength turns on the adjudication step. If adjudication is deterministic (exact/substring match against the fetched text), the blind-spot set is the source's, not a model's — strong decoupling, but coverage narrows to verbatim or near-verbatim claims. If a model performs the adjudication (reading the fetched text to judge the claim), its reading-comprehension blind spots re-enter and the oracle is only partial decoupling — the *referent* is external, the *reader* is not. Either way it settles only the *admissible* slice of §4.2 (claims a fetched source can adjudicate), leaving the interpretive residue still human and still beyond any in-process oracle. The harness thus imposes a physical boundary that aligns with the theoretical one *for model-based heterogeneous oracles*, converting verification-source decoupling from a *convention* (bypassable by an in-process "independent" reviewer that secretly shares the provider) into an *implementation-level invariant within current single-provider harnesses* — not architecturally necessary, but effective today (see §8).

Two consequences observable in the reference implementation (SolidForge):

(1) Structural complementarity, not interchangeability. The same-source main ring (defending the three intrinsic failure modes via context-isolated in-process subagents) and the heterogeneous add-on ring (defending specification gaming via cross-provider `claude -p` subprocesses) sit on opposite sides of the process boundary *by construction, not by configuration.* The heterogeneous ring is additive and non-replacing: the same-source ring always runs first as the cost/reliability floor; heterogeneity engages only for high-risk items where specification gaming is a material threat. This is not a product choice but a direct consequence of where each ring must execute.

(2) The boundary must be preserved, not bridged. The design explicitly rejects global

provider injection — e.g., a process-wide LiteLLM router as the primary substrate — because it would dissolve the process boundary, collapsing "same-source vs heterogeneous" back into a configurable convention that a future change could silently bypass. The cost of preserving the boundary (subprocess overhead per heterogeneous call) is accepted *as the price of the structural guarantee*: it is the same price the self-certification paradox extracts for any genuine oracle.

Generalization and its limits. Agent harnesses that bind provider at process scope (Claude Code as the reference here) inherit this physical decoupling boundary for free; harnesses that dissolve provider scope (global in-process multi-provider routers) forfeit the structural guarantee and must re-establish decoupling by policy alone — a strictly weaker position, as it relies on developer discipline rather than platform enforcement. Whether the process-scoped binding is truly *necessary* for an LLM agent harness, or merely contingent on today's single-provider SDK design, is left as an open question (§8): a future harness built around first-class multi-provider routing would need to re-derive the decoupling boundary by other means, or accept that the boundary becomes conventional.

Control-plane-external enforcement locus. A control plane external to the agent runtime (as the LoopX peer of §7 is) illustrates a seam-level refinement of the §4.2 structural/conventional distinction. Such a plane can enforce the Process/Outcome split *structurally at the spend and commit seams* — refusing to advance durable state or account a turn's spend without a validated transition — while the *act* seam (whether the executor consults the plane before acting) is only conventionally gated, because the plane cannot control the runtime's internals. The residual exposure is therefore not arbitrary conventionality but a specific surface: external or irreversible actions not mediated by the plane's spend/commit gates (an unmediated pull request or external write). This refines, rather than weakens, the §4.2 criterion: the *locus* of structural enforcement matters, and a runtime-external control plane is structurally stronger than a purely prompted rule on the plane-mediated path (state advancement and spend), even though it cannot reach actions not mediated by its gates (an unmediated pull request or external write). A control-plane peer that gates spend and commit is thus not a §4.2 violation but a partial structural defense whose gap is precisely circumscribed.

5. Implications

The two-axis frame (§3.3) and the Process/Outcome split (§4.2) carry implications for both evaluation and design.

Evaluation must be two-axis. Current coding-agent benchmarks measure pass-rate against a held-out test suite — an Axis-A metric. The recent SpecBench/ImpossibleBench line now measures specification gaming empirically (observationally, and via adversarial spec/test-conflict injection); what is still missing is the Axis-B *defense* axis — scoring whether a heterogeneously decoupled oracle catches the gaming these benchmarks surface and a same-source judge misses. Constructing that two-axis defense evaluation is the principal empirical next step this paper proposes (§8.3).

Full intrinsic-mode defense plus salient-spec-gaming defense is the logical ceiling of agent autonomy on Axis A (§3.3). The progression "no flow control → fixed loop → state-routed → autonomous closed-loop" terminates there; beyond it, no amount of additional in-loop capability reduces *latent* specification-gaming risk, because latent gaming is not diminished by the three-mode defenses (§3.2(ii)) and is precisely the part that in-loop structural discipline cannot reach (§4.3) — it closes the salient surface but not the latent one. Practical systems that claim reliability beyond this ceiling must be claiming something on Axis B — an oracle whose blind-spot set differs from the agent's (external to the process for model-based oracles; in-process-but-

decoupled for a deterministically-adjudicated fetched-source oracle — partial if model-mediated, per §4.4) — and the claim is only as strong as that oracle's decoupling. The "stronger agent" frame, beyond this ceiling, is the wrong frame.

The agent harness's process-level provider binding is a load-bearing reliability feature within current single-provider harnesses, not a limitation. Dissolving it (e.g., a global in-process multi-provider router) forfeits the implementation-level decoupling guarantee (§4.4) and re-establishes decoupling by policy alone. Harness designers who treat single-process multi-provider routing as a convenience should recognize it as a trade against the discipline this paper argues for.

6. Reference Implementation: SolidForge

SolidForge is a Claude Code plugin that implements the discipline of §4 as a running system. **Disclosure: SolidForge is maintained by an author of this paper; we treat it here as one concrete instantiation, not as independent evidence.** We describe it not as a product evaluation but as existence-proof that the framework is realizable, and to make each discipline concrete. Its design choices are consistent with the claims of §3–§4, and in two cases concretely instantiate them — the Process/Outcome split (§4.2) and the cross-provider heterogeneous ring (§4.4).

Two-layer convergence with dual rings. SolidForge structures work as a specify→implement pipeline (blueprint-crafting → parallel-development), *each* layer a convergence loop with an inner deterministic ring (fast gate + architecture-contract gate) and an outer adversarial ring (a read-only reviewer subagent). This is consistent with §3.2(iii): the outer ring's role is to surface authorial blind spots (gaps, contradictions, drift from blueprint, and auditable salient gaming) — i.e., the three-mode regime plus the salient surface — not to reach *latent* specification gaming.

Process/Outcome split as a schema constant. The run-record schema separates `process_converged` (a boolean derived from gate verdicts) from `rightness`, an enum the agent cannot write. SolidForge exercises only the conservative value `human_confirm_required`: although §4.2 permits `oracle_verified_under_known_coverage` in principle, SolidForge does not auto-elevate `rightness` on any oracle verdict — the heterogeneous ring's findings feed the convergence gates (a heterogeneous Blocker prevents `process_converged`) but never substitute for human confirmation of correctness. The `rightness` field is a constant enforced at the schema validation layer; neither the agent nor the convergence loop can write it. This is §4.2 in production: a same-source self-endorsement of correctness is rejected at the data seam, not by a prompt the agent might route around.

Heterogeneous oracle by cross-provider subprocess. A heterogeneous review ring spawns a `claude -p` subprocess configured to a different provider (e.g., DeepSeek/Qwen/GLM), as a fresh, stateless session. It is *additive and non-replacing*: the same-source main ring always runs first as the cost/reliability floor; heterogeneity engages only on high-risk items where specification gaming is a material threat. This is consistent with §4.4's platform-observed boundary: cross-provider review is physically out of process precisely because the in-process subagent primitive cannot cross providers. SolidForge's in-process ring pairs a deterministic fast gate with an architecture-contract gate (a formal contract oracle, §4.1/§4.3), so its heterogeneous defense is the union of an in-process formal/deterministic ring and an out-of-process cross-provider model ring — a conservative subset of §4.4's permitted Axis-B oracles; the fetched-source-text route is not exercised. (We treat "cross-provider" as a heuristic proxy for blind-spot divergence; the provider→blind-spot mapping

remains unvalidated — Open Problem §8.4.) The design explicitly rejects a global in-process multi-provider router (e.g., LiteLLM as primary substrate) on the grounds that it would dissolve the process boundary and convert the discipline from implementation-level invariant back into bypassable convention.

Frozen intent blueprint + drift detection. The blueprint (PRD/arch-design/iteration-plan) is frozen at spec convergence; modification is permitted only through an explicit, audited Revision Channel (an append-only log that records every blueprint change with its rationale and approver). A diff-to-blueprint check runs in the code layer's outer ring; a drift that satisfies the local contract while violating the blueprint's core use cases triggers hard rollback with reverse-prompt injection — re-injecting the frozen blueprint as a corrective system prompt that overrides the drifted target. This defends salient specification gaming (§4.3) at the structural layer where the drift is auditable — a diff that visibly diverges from the frozen acceptance criteria cannot silently redefine the target. Where the turn is on whether core *use cases* are still served, the check reaches an adequacy judgment (the latent regime): structural discipline catches the auditable divergence, but whether the divergence still serves intent is only as strong as the reviewer performing it (the same-source outer ring, §3.2(iii), or a heterogeneous oracle, §4.3) — the latent surface is not closed by structural discipline alone (§4.3).

Per-run self-assessment. Each run-record carries a self-assessment block: three booleans (`context_rot_defended`, `error_compounding_defended`, `goal_drift_defended`) plus counters (`breakers_fired_count`, `rollbacks_count`). This operationalizes the two-axis frame (§3.3) as observability: Axis-A defense is reported per run, and an explicit `axis_b_status` operational field (a SolidForge-specific observability signal, distinct from the §4.2 schema invariants) honestly reports whether a heterogeneous oracle ran for that run — so specification-gaming defense is signalled as out-of-scope rather than silently satisfied.

Deliberate scope narrowing. SolidForge restricts itself to per-feature convergence and intentionally cedes time-based re-fire and cross-cutting worktree-parallel sweeps to companion layers (declared non-interchangeable with the convergence loop). This negative-space design is consistent with §4.3's spectrum claim: structural discipline (salient defense) is the project's scope; latent defense belongs to the externalized heterogeneous layer, which the project makes pluggable rather than absorbing into the loop.

Honest limitations. (i) The heterogeneous oracle is *partial*: cross-family commercial models share training-data/RLHF overlap, so decoupling is weaker than formal or human oracles (§4.1; Open Problem §8.2). (ii) The implementation produces run records but no controlled benchmark evaluation; empirical isolation of specification gaming is now addressed by concurrent work (SpecBench observationally; ImpossibleBench via adversarial injection), leaving the defense-axis evaluation (§8.3) as the open part. (iii) the per-run self-assessment block (the three Axis-A defense booleans and `axis_b_status`) is self-reported; its calibration against external ground truth is itself open. The reference implementation is therefore evidence of *realizability*, not of *efficacy* — the latter requires the two-axis defense-axis evaluation this paper flags as its principal future work (§8.3), applied to the surfaces SpecBench/ImpossibleBench expose.

7. Related Work

We locate prior work along the two axes of our frame.

The specification-gaming lineage (concept). The phenomenon we treat was characterized

in the RL/alignment literature: Krakovna (2018; 2020) defined specification gaming as satisfying a literal objective without achieving the intended outcome and curated the canonical example list; Amodei et al. (2016) introduced *reward hacking* as one of five concrete AI-safety problems; Pan et al. (2022) empirically documented the divergence between a proxy reward and the true reward and its scaling with capability; Manheim & Garrabrant (2018) gave the AI-version taxonomy of Goodhart's law (regressional/extremal/causal/adversarial). Our debt is to the *concept*; our contribution is to carry it from RL-flavored definition into a production coding-loop discipline, with an irreducibility argument (§3.2), a two-axis frame (§3.3), and schema-level + platform-level disciplines (§4).

Complementary security view: CaMeL. CaMeL (Debenedetti et al. 2025; Google DeepMind) instantiates a dual-LLM architecture (Privileged LLM + Quarantined LLM) to defend against prompt injection in agent systems. CaMeL and this paper address disjoint failure surfaces: CaMeL targets *adversarial input* (untrusted data manipulating the agent's reasoning); we target *proxy optimization* (the agent silently satisfying its own verification). The two are orthogonal threats and the defenses do not substitute — a privileged/quarantined split does not reach specification gaming: that split targets permission/trust isolation, not blind-spot asymmetry on the correctness question. Even heterogeneous providers in those roles would not constitute a correctness oracle (privilege isolation verification-source decoupling), and CaMeL's threat model is adversarial input rather than proxy optimization; conversely, a heterogeneous correctness oracle does not reach prompt injection. We see the two as complementary axes within the broader agent-reliability space.

Multi-agent academic prototypes. AgentCoder (programmer + tester), MetaGPT (SOP-structured multi-agent), and ChatEval (multi-agent debate for evaluation) establish the maker-checker form. None, however, anchors on a frozen specification, exposes the schema-level Process/Outcome distinction, or names specification gaming as a failure class — they operate wholly within Axis A, where adversarial review defends the three intrinsic modes (§3.2(iii)).

Heterogeneous-oracle classical form: mutation testing. Mutation testing (mutmut, Cosmic Ray, Mull) is, in our terms, a strong-decoupling oracle for *test-suite adequacy*: it programmatically mutates the implementation and measures whether the suite catches the mutants. It is the purest existing instance of §4.1's "formal/production-trace" decoupling — the oracle's blind spots are those of the mutation operators, not a model. It is non-AI-specific, however, and does not address the LLM-specific mechanisms (proxy generation, latent gaming) that this paper targets. We regard mutation testing as a candidate for the heterogeneous-oracle slot in §4.3's latent-gaming defense, and its absence from current coding-loop pipelines is consistent with the discipline not yet being practiced.

The oracle problem in SE (classical). The question "what oracle can verify a program's correctness?" is the classical oracle problem in software testing (Barr et al. 2015). Our heterogeneous-oracle taxonomy (§4.1) is this problem restated for the LLM-coding-loop setting, where the agent authors both code and partial oracles (tests). Execution-based code-generation systems — CodeT (Chen et al. 2022), AlphaCodium (Ridnik et al. 2024) — already combine generated tests with execution filtering; yet by §4.1's collapse rule this is same-source-by-cascade: the runtime executor is formal, but the specification it executes (the test set) is agent-authored, so the executor's blind spots collapse back to the model's. Self-consistency (Wang et al. 2022) is same-source for the same reason — diversity of samples from one model does not diversify blind spots. Prior work thus formalizes the agent's own proxy rather than decoupling the verification source; our contribution is not the oracle concept but the explicit two-axis organization (§3.3) and the salient/latent distinction that separates what in-loop structural discipline can catch from what requires an externalized

oracle.

LLM-as-judge and same-source verifier bias. The LLM-as-judge literature (Zheng et al. 2023) studies when and why an LLM evaluating another LLM is reliable or biased — including known same-source/self-preference biases. Our self-certification paradox (§3.1) is this bias axis applied to the coding-loop verification problem: the same training distribution that produces the code produces its validator. The contribution relative to LLM-as-judge is the coding-loop-specific framing (proxy specification gaming via test manipulation) and the Process/Outcome split as a schema-level mitigation, not the discovery of same-source bias. The Evaluator Stress Test (EST; Shihab et al. 2025) detects proxy gaming in RL/LLM-alignment by separating exploitable-sensitivity from content-driven improvement via controlled perturbations — a score-decomposition axis distinct from our integrity/adequacy (salient/latent) bifurcation (§4.3); its external-perturbation audit is nonetheless a candidate realization of the externalized heterogeneous oracle the latent regime demands.

Industrial loop-engineering peers. A 2026 cohort targets adjacent ground. *Spec Kit* (GitHub, https://github.com/github/spec-kit) is spec-anchored with a convergence primitive and a `constitution` anchor, but single-ring and single-agent — no adversarial reviewer, no specification-gaming concept. *Loki Mode* (https://github.com/asklokesh/loki-mode) is the closest peer: a tamper-evident proof artifact + drift detection + an Evidence Receipt splitting Facts (deterministic) from Assessments (AI judgment) + a multi-reviewer council — a conceptual parallel to our Process/Outcome split (§4.2) and heterogeneous adversarial ring (§4.4) — same shape, but weaker enforcement: Loki frames Facts/Assessments as a reporting discipline rather than the schema-level invariant (an agent-barred enum) that is the load-bearing property of §4.2, and we found no naming of specification gaming as an orthogonal axis in its public docs. *zeroshot* (https://github.com/the-open-engine/zeroshot) adds blind validation (validators cannot see the worker's context) with fail-closed convergence across providers — operationally close to our heterogeneous subprocess. *LoopX* (https://github.com/huangruiteng/loopx) is a fourth peer of a different shape: not a single-loop harness but a **cross-turn, cross-runtime control plane** that externalizes objective, gates, todos, evidence, and quota into an append-only event ledger while Codex, Claude Code, or Cursor execute bounded turns. Its production verifiers are same-runtime and, in its public docs, it does not name specification gaming; on the orthogonal axis it is therefore silent, like the three above. Two surfaces are nonetheless notable. First, its benchmark/evaluation subsystem is a concrete *deployed* instance of the two-axis separation this paper prescribes (§5) — an official, externally-authored verifier scoring outcomes kept distinct from the loop's own turn-readiness signal, with anti-gaming controls (surrogate-run labelling, reward-leak prevention, and matched treatment/baseline arms on the same case, source, model, reasoning effort, timeout, and no-feedback budget). This instantiates the two-axis *evaluation separation* (the loop's turn-readiness metric kept distinct from an externally-authored outcome metric), not a specification-gaming injection-set methodology (ImpossibleBench, §8.3, already fills that). Second, its `delivery_outcome` enum — a structured self-report (`surface_only` / `outcome_gap` / `outcome_progress` / `primary_goal_outcome`) that drives scheduling spend — is a candidate *salient* defense against artifact-count Goodhart, but only partially so: the label is executor-self-reported and validated only for a machine-visible *frontier delta* (form), not against the objective (substance), leaving an irrelevant-change residue in the latent regime (§4.3). Assessed against each project's public docs (absence claims bounded to that surface), none of the four makes the irreducibility argument (§3.2) or the two-axis frame (§3.3) explicit; the contribution of this paper is to name the discipline they variously approximate without theorizing, and to derive the platform-level structural guarantee (§4.4) that the cohort approximates from different

architectural positions — single-provider process-scoped harnesses from within, cross-runtime control planes (LoopX) from a partial, seam-circumscribed position (see the §4.4 note). LoopX additionally supplies the only observed deployed instance of the two-axis separation among the cohort — evidence that the discipline is practised even where it is not theorized.

Loop engineering as practice. Osmani (2026) and concurrent industry discussions (2026-06) named the *practice* of designing agent loops; they enumerate primitives (Automations, Worktrees, Skills, Plugins, Sub-agents, Memory) and warn of unattended mistakes and cognitive surrender. The practice framing is silent on the orthogonal-axis question: it treats "the loop runs and converges" as the goal, whereas this paper argues convergence (Axis A) is necessary but insufficient, and that the verification source (Axis B) is the independent discriminator.

8. Open Problems

1. **Heterogeneous-oracle cost.** HITL annotation is not fully automatable; coverage is limited to annotated high-frequency cases — novel failures regress to same-source verification.
2. **"Heterogeneous" is itself PARTIAL.** Commercial model families share training data / RLHF overlap; cross-family review is only partial decoupling. Truly heterogeneous oracles (formal spec, production trace, human) remain the open frontier.
3. **From benchmark absence to defense-axis evaluation.** The empirical gap earlier drafts named — no benchmark isolating specification gaming — has closed on two fronts: observationally (SpecBench, Zhao et al. 2026, measures reward hacking via the visible/held-out pass-rate gap) and adversarially (ImpossibleBench, Zhong et al. 2025, injects direct conflicts between the natural-language specification and the unit tests on LiveCodeBench/SWE-bench, where any pass is a specification-violating shortcut; see also RHB, Thaman et al. 2026, on tool-using agents). What remains open is not benchmark construction but the *defense-axis* evaluation this paper's two-axis frame predicts: does a heterogeneously decoupled oracle catch the specification gaming these benchmarks surface and a same-source judge misses?
4. **Formalization of "verification-source decoupling degree."** How to quantify Axis B as a measurable metric remains open — a prerequisite for empirical comparison of systems on the second axis.
5. **Coverage of the root-cause path.** Acceptance criteria that cover only the symptom interface are bypassable by "local satisfaction"; criteria that cover the upstream causal chain depend on diagnostic accuracy, which is itself a probabilistic hypothesis.
6. **Is process-scoped provider binding necessary or contingent?** (§4.4) A future harness with first-class multi-provider routing would need to re-derive the decoupling boundary by other means, or accept that the boundary becomes conventional.
7. **Heterogeneous-oracle reliability.** Positioning the heterogeneous oracle as the latent-regime defense leaves its own failure modes unexamined: false positives (correct code flagged as spec-gaming), oracle disagreement (two heterogeneous oracles returning conflicting verdicts), and error cascades (a wrong verdict triggering unnecessary rollback). The threshold at which oracle error outweighs defense value — and how a false-positive rate interacts with the partial decoupling of §8.2 — is open.
8. **Testability of the self-certification conjecture.** The necessity framing rests on the strong-form self-certification conjecture (§3.1), which we stipulate rather than establish. Empirically testing it — e.g., measuring same-family-fresh-context vs cross-family verdict divergence on a known latent-proxy-gap test set, to determine whether same-family reviewers in fact share the verified's blind spots wholesale — would either ground the "logically necessary"

framing or weaken it to "strongly advisable" (the weak-form fallback of §3.1).

References

- Krakovna, V. (2018). *Specification gaming examples in AI.* https://vkrakovna.wordpress.com/2018/04/02/specification-gaming-examples-in-ai/
- Krakovna, V. (2020). *Specification gaming: the flip side of AI ingenuity.* DeepMind blog. https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/
- Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). *Concrete Problems in AI Safety.* arXiv:1606.06565.
- Pan, A., Bhatia, K., & Steinhardt, J. (2022). *The Effects of Reward Misspecification: Mapping and Mitigating Misaligned Models.* arXiv:2201.03544.
- Manheim, D., & Garrabrant, S. (2018). *Categorizing Variants of Goodhart's Law.* arXiv:1803.04585.
- Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Carlini, N., Fabian, D., Kern, C., Shi, C., Terzis, A., & Tramèr, F. (2025). *Defeating Prompt Injections by Design (CaMeL).* arXiv:2503.18813.
- Huang, D., Zhang, J. M., Luck, M., Bu, Q., Qing, Y., & Cui, H. (2023). *AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation.* arXiv:2312.13010.
- Hong, S., Zhuge, M., Chen, J., et al. (2023). *MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework.* arXiv:2308.00352.
- Chan, C.-M., Chen, W., Su, Y., et al. (2023). *ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate.* arXiv:2308.07201.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 12, 157–173. https://doi.org/10.1162/tacl_a_00638
- Osmani, A. (2026). *Loop Engineering.* https://addyosmani.com/blog/loop-engineering/
- Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering*, 41(5), 507–525. https://doi.org/10.1109/TSE.2014.2372785
- Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., & Chen, W. (2022). *CodeT: Code Generation with Generated Tests.* arXiv:2207.10397.
- Ridnik, T., Kredo, D., & Friedman, I. (2024). *Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering.* arXiv:2401.08500.
- Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2022). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR 2023.* arXiv:2203.11171.
- Zheng, L., Chiang, W.-L., Sheng, Y., et al. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.* arXiv:2306.05685.
- Zhao, B., Srikanth, D., Wu, Y., & Jiang, Z. (2026). *SpecBench: Measuring Reward Hacking in Long-Horizon Coding Agents.* arXiv:2605.21384.
- Shihab, I. F., Akter, S., & Sharma, A. (2025). *Detecting Proxy Gaming in RL and LLM Alignment via Evaluator Stress Tests.* arXiv:2507.05619. (ACL 2026 Findings.)
- Zhong, Z., Raghunathan, A., & Carlini, N. (2025). *ImpossibleBench: Measuring LLMs' Propensity of Exploiting Test Cases.* arXiv:2510.20270.
- Thaman, K. (2026). *Reward Hacking Benchmark: Measuring Exploits in LLM Agents with*

Tool Use. arXiv:2605.02964.
- SolidForge — reference implementation for §6. Repository: https://github.com/maskshell/solidforge