
Fail-Closed LLM: Zero False Certificates for Language-Model Claims

Eduardo Perez Denadai
Optimality.ai

Abstract

Language models now write code with embedded claims faster than any team can review them, and the standard answers—LLM judges, heuristics, pass rates—are scalar, probabilistic, and incapable of detecting silent verification failures. We present an automated certification engine for fail-closed verification of language-model claims: a structural miner (or any frozen model) proposes typed claims, exactly one component disposes of them against exact semantics, and three detached checkers re-derive every certificate independently—where definite disagreement raises an architectural alarm rather than a padded verdict. The system enforces strict architectural constraints, including a machine-checked single-writer invariant over the call graph, typed readings (theorem, strengthening, premise, instance) that determine what a verdict may conclude, a vacuity guard that refuses to count assumptions as proofs, and a four-number report vocabulary that forbids collapsing proved, declared, declined, and refuted into one score. The results are censuses, not samples: 9,258 typed claims over five pinned repositories with **zero** false certificates, 7/7 adversarial falsifications refused, 84/84 capability tests passed; a banking-SQL corpus that disposes 3,864 claims into 502 proved, 1,538 declared, 1,824 declined, 0 refuted—including 171 money-path lemmas with file-level evidence. We treat refusal as a first-class record: an UNVERIFIABLE taxonomy that is the research agenda, and a measured coverage ceiling of 0 of 2,148 pytest abstentions, stated rather than hidden, that disciplines every coverage claim we are allowed to make. A qualitative control study ($n = 4$ constructed tasks) discriminates reference patches from trivial ones 4/4 versus 0/4—and surfaced three defects in its own harness, including an oracle that shared a generator with its subject and therefore certified nothing. The engine formally verifies claims against stated code artifacts rather than unstated informal intent, defining a strict operational boundary for automated trust.

1 Introduction

Trust in generated code is no longer limited by how fast models can write it; it is limited by how fast anyone can *check* it. Existing evaluation frameworks rely on LLM judges, heuristic metrics, or unit test pass rates—all of which are scalar, probabilistic, and incapable of detecting silent verification failures. Verification tools exist, but they either trust the generator (self-grading) or abstain from the question entirely.

The certification engine answers one question—*does this claim hold?*—under four structural constraints (A1–A4):

- (A1) **Single writer.** Exactly one component (the verdict authority) assigns verdict status; the invariant is machine-checked over the import and call graph.
- (A2) **Artifact mediation.** Verdicts exist only as serialized certificates, re-derivable by checkers that share zero code with the prover.

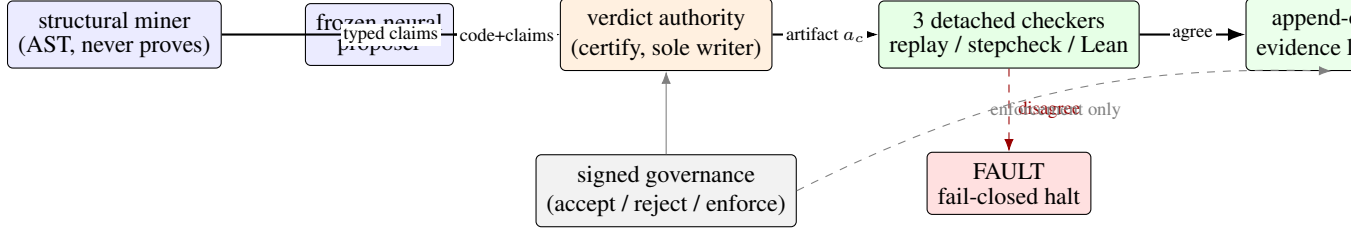


Figure 1: Propose–dispose–govern. Everything above the authority is a proposer; the authority is the only verdict writer; governance signs enforcement contracts and premises, never proofs; checker disagreement is an alarm, not a score.

- (A3) **Typed disposal.** A claim is verified against its structurally assigned reading (theorem, strengthening, premise, instance), never a reading guessed from prover output.
- (A4) **Fail-closed.** Checker disagreement on a positive claim is a FAULT, never a confidence downgrade.

The output vocabulary is deliberately four-valued plus one alarm: CERTIFIED (proved—the only word that means true), declared (a recorded contract on the premise tier), REFUTED (proved false, with a witness), UNVERIFIABLE (the engine declined—declined is not failed), and FAULT (the halt condition: two independently faithful checkers disagree about a positive claim). Reports must print proved/declared/declined/refuted as *four numbers, never a single score*.

Contributions. (1) The engineering system: claim IR and exact term algebra, the disposal cascade with named methods, five frontends, three detached checkers with separate trust postures, and a machine-checked single-writer invariant (§3). (2) Architectural invariants—fail-closed certification and zero false certificates—plus formal results carried from the companion theory, each grounded in a specific guard in the code (§4). (3) Full-population censuses with file-level evidence and the UNVERIFIABLE taxonomy as a measured refusal record (§5). (4) A constructed control study whose trivial arm exposed three harness defects, including oracle–subject dependence (§6). (5) The improvement lifecycle: the declined set as research agenda, promote-by-verdict regression algebra, and twin-required ledger changes (§7).

2 Related work

Axiomatic verification [Hoare, 1969] and abstract interpretation [Cousot and Cousot, 1977] established machine-checked reasoning about programs; modern interactive provers [de Moura et al., 2015] and SMT proof checking [Barrett et al., 2021] provide the backends our authority can delegate to. Syntax-guided synthesis [Alur et al., 2013] and verified repair loops [Xia et al., 2018] couple proposal with verification at *inference* time; we amortize the loop into trained proposers while keeping disposal exact. The neural-proving line—GPT-f-style search, Trinh et al. [2024], and large-scale prover models [DeepSeek-AI, 2025]—applies the same propose/dispose discipline to mathematics with heavyweight kernels and frontier-scale compute; our delta is production code claims, a trusted computing base of hundreds of lines (term algebra plus one-step verifier) rather than 100k-line kernels, multi-language frontends, and a fail-closed deployment loop. For LLM-generated code security, Pearce et al. [2022] document the risk; certification of the verifiable subset with an explicit refusal record is the complementary control. Proof-carrying code [Necula, 1997] is the artifact-level precedent for shipping evidence with code; we ship evidence with *claims*. DreamCoder’s wake/sleep library learning [Ellis et al., 2021] is the structural analogue of our signed, compounding shape registry.

3 The system

3.1 Claim of record and typed readings

Definition 1 (Claim of record). *A claim is an element of a product space, $c = \langle s, \varphi, \theta \rangle \in \text{Sites} \times \mathcal{T}_{\mathbb{Q}} \times \Theta$: an execution site s (file, line, AST node), a predicate φ over the exact-rational term algebra $\mathcal{T}_{\mathbb{Q}}$, and a reading θ assigned structurally by the miner from the AST—immutable post-proposal.*

Definition 2 (Readings). Θ_{thm} : $\forall \bar{x} \varphi$ over formal parameters—the faithful universal reading. Θ_{str} (strengthening): $\forall \bar{x} \varphi$ over runtime-assigned locals—strictly stronger than the site assertion. Θ_{pre} (premise): $\neg \text{guard} \Rightarrow \varphi$, enforced by the program’s own control flow. Θ_{ins} (instance): site-bound, one execution state.

The reading determines what a verdict may conclude. A refuted strengthening refutes only the universal closure; since a refuting tuple need not be reachable at the site, the system demotes to an honest abstention (`strengthening_refuted`), never a source violation. A refuted premise demotes fail-open (`premise_refuted`); an instance claim abstains *without consulting the prover* (`runtime_observation`).

3.2 Verdicts, evidence classes, and the one-way law

Evidence classes are three counters, never merged: *proof* (re-derived by independent checkers), *premise* (declared contracts: guard-conditional implications), *observation* (signed runtime enforcement), plus abstention.

Proposition 1 (One-way evidence migration). *Exactly two migrations leave observation: $\text{observation} \rightarrow \text{proof}$ via an exact disposal rung executed inside the authority, and $\text{observation} \rightarrow \text{premise}$ via a certified guard implication. No rule concludes proof or premise from a human signature or heuristic approval alone; sign-off stamps enforcement, never proof.*

3.3 The authority and its detached checkers

Disposal runs a named-method cascade over each claim: hypothesis classification (int-domain, disequalities, modular congruences with gcd contradiction detection), fixpoint solving (substitution and exact Gauss–RREF over rationals, univariate quadratic pinning, polynomial premise collection), bound resolution (propagation, integrality congruences, singleton pinning), and conclusion discharge through named methods (`decide_eq`, `mod-residue`, `poly-reduce` with bounded Gröbner completion, `quadratic-root`, `lp`, `interval`, `sign_analysis`, ...), abstaining on `opaque_atom` / `nonconstant_residual` / assumed premises. Vacuity guard: a CERTIFIED verdict must contain at least one non-premise discharged obligation, else it demotes to `assumed_not_proved`—premises and proofs never share a counter.

Three checkers, three trust postures:

- (a) **Replay kernel**: imports only the term algebra; re-derives every obligation of the JSON round-tripped certificate.
- (b) **Step checker**: independently recomputes each trace step under a versioned rule table; unknown rule $\Rightarrow$ invalid.
- (c) **Lean-arm kernel**: shares no code with the prover (own rational polynomial algebra, elimination, Buchberger completion, Fourier–Motzkin)—a false-positive detector for the Lean frontend.

Obligations cross checker boundaries only as serialized certificates (the claim IR in JSON), the single interchange format: each checker re-parses that format with its own algebra, so a translation defect manifests as checker disagreement—a **FAULT**—never as silent agreement. This is the fail-closed property doing work on the translation layer specifically.

System Invariant 1 (Fail-closed certification). *Let D be the authority’s disposal map and κ an independent checker faithful on artifacts. For every claim, $D(c) = \text{CERTIFIED} \Rightarrow \kappa(a_c) = \text{agree}$ or the run halts in **FAULT**.*

System Invariant 2 (Zero false certificates). *Every emitted proof is checker-agreed or the system halts. A false certificate requires the simultaneous, identical mathematical failure of prover and independent checker—an explicit, auditable event, never a silent hallucination.*

Abstentions carry no proof and are never audited into a fault; **FAULT** is raised only on definite disagreement about a positive claim. The single-writer invariant (A1) is enforced by `selfcheck`: a regex-driven import/call-graph walk that flags any verdict-producing path outside the whitelisted authority. The walk is syntactic: dynamic imports and aliased calls are not resolved, so `selfcheck` certifies the absence of violations only within the static import/call graph; violations hidden behind dynamic dispatch are outside its scope, which is why it is paired with the corpus-balanced promotion gate (§7) rather than treated as a proof of absence.

Table 1: Census dispositions. Four numbers, never one score.

Corpus	proposed	proved	declared	declined	refuted
Banking SQL (contracts)	3,864	502	1,538	1,824	0
NumPy (satellite)	3,120	0	213	2,907	0
Five-repo run	9,258	9	1,545	4,382 [†]	0

[†]runtime-observed class (pending rungs); balance in the five-repo run are premise-tier and inventory classes.

3.4 Fragment, frontends, mining, storage

The checkable fragment is straight-line units whose assertions evaluate over exact-rational terms (+, −, ×, div, mod, abs); IEEE floating-point claims are routed to abstention by policy (`numeric_domain:ieee_float`)—motivated by an incident in which eight correct floating-point assertions were being refuted. The engine is language-free; frontends lower Python (AST miner), Java (guard throws become premises), SQL (DDL—CHECK, NOT NULL, keys—becomes premises and monitors; runtime claims export watchdog queries), Lean (tokenizer/parser for the linear arithmetic subset), and pytest/TPTP tables. Mining is proposer-only: unlabeled candidates with provenance chains (repo, commit, file, line), deterministic IDs, and a structural form axis. Storage is an append-only, key-addressed ledger with signed confirmations; roles are fixed: *registry* (fuel), *out* (exhaust), *corpus* (engine—the known-answer falsifier).

Measured fragment coverage. The share of a real population that falls inside $\mathcal{T}_{\mathbb{Q}}$ is a property of the *artifacts*, not of the engine, and the censuses measure it directly (Table 1): static SQL schemas place $502/3,864 = 13.0\%$ of proposed claims inside the exact fragment (proved) with a further 39.8% on the premise tier, while general Python places $9/9,258 = 0.1\%$ inside the kernel-proof class with 16.7% declared and 47.3% runtime-observed; the pytest population is $0/2,148$ addressable. The excluded domains are exactly the taxonomy’s routes: runtime-dependent state (execution/database), IEEE-float by policy, and unhandled term constructs (`opaque_atom`, `nonconstant_residual`). The 1,824 SQL declines decompose by these reason codes in the census ledger—runtime-dependent claims, float-domain policy refusals, and unhandled arithmetic terms—with per-reason counts recomputable from the pinned-commit artifacts; the paper reports the population totals here because the totals, not the histogram, are what bound a coverage claim.

4 Formal results, grounded in code

Beyond the two invariants of §3 (of which Invariant 1 is the load-bearing one): the strengthening-entailment proposition (universal closure entails the site instance; refutation of the closure does not) is implemented exactly as the `strengthening_refuted` demotion; the one-way law (Proposition 1) is why `confirm_enforce` is bookkeeping-only in the code; the capability ladder admits a new disposal rung only with (i) a soundness sketch, (ii) adversarial twins in all three verdict directions, and (iii) a step-checker handler; and an MDL criterion governs which syntactic claim classes to evaluate next without ever touching axiomatic truth conditions: the selection score of a class is computed from the cascade’s own refusal statistics—each `opaque_atom / nonconstant_residual` emission increments the class histogram—so rung selection is mechanically driven by disposal outcomes rather than by manual triage. Ladder rungs are exact mechanisms—Gaussian elimination, Faulhaber closed forms, interval arithmetic, guard-to-premise lowering—never smarter heuristics: yield rises only by new exactness.

5 Full-population censuses

Scores over cherry-picked samples cannot support a soundness claim, so we report whole populations at pinned commits.

Banking SQL. 3,864 claims → 502 proved, 1,538 declared, 1,824 declined, 0 refuted (Table 1). The declared half is the schema made explicit: 502 numeric ranges, 877 length bounds, 159 NOT NULL. Highlights with file-level evidence: 171 *money-path lemmas* on $\text{scale} \geq 2$ columns (`NUMERIC(19, 4)`, `NUMBER(*, 2)`), each pointing at a real line. The lemma class is concrete: for a column declared

Table 2: The UNVERIFIABLE taxonomy: decline reasons, their meaning, and the disposition they demand.

reason	meaning	route
no_assert_spec	the code states a declaration, not a claim	author a premise line, or accept as inventory
runtime_observation	only execution or the database can settle it	run the watchdog query / tests (venue change)
numeric_domain:ieee_float	floating-point out of scope by policy	interval rung (refusal is the boundary)
unconfirmed_spec	provenance gate: inferred/free-text specs never certify	confirmation workflow
strengthening_refuted	universal closure refuted; site instance untouched	honest abstention, never a violation
premise_refuted	declared premise failed under disposal	fail-open demotion
opaque_atom	/ solver cannot discharge	new exact rung if volume warrants
nonconstant_residual		
assumed_not_proved	vacuity guard: premises alone cannot prove	restate as premise tier

amount NUMERIC(19,4), every stored value is a scaled integer, $\forall v \in \text{amount}. \exists k \in \mathbb{Z}. v = k/10^4$, so arithmetic on that column is exact in $\mathcal{T}_{\mathbb{Q}}$ and the corresponding money-path obligations dispose instead of abstaining—which is why a banking schema is a natural home for exact-rational certification. Also 159 write-path claims with exact three-way disposition: a NULL literal in an INSERT refutes with the statement as witness, a literal certifies, an expression abstains honestly.

Five-repository run. 9,258 typed claims over pinned commits; 9 kernel-agreed proofs, 1,545 certified premise contracts, 4,382 runtime-observed, 0 refuted, 0 false approvals; 7/7 adversarial falsifications refused; 84/84 capability tests passed; deterministic replay mismatch 0. Claims collapse into 4,476 shapes—one signed confirmation upgrades every instance—which is how human review scales without ever becoming evidence (A1).

5.1 The refusal record as roadmap

Every decline carries a reason code (Table 2), and the reason distribution *is* the agenda: each distinct reason is a candidate tactic, kernel extension, or promote-to-venue path, ranked by volume—a reason that appears at scale outranks a reason that is merely interesting.

Coverage ceiling. On a pytest corpus, frontier triage of all 2,148 abstentions (1,740 distinct claims) found 2,146 runtime-state assertions, 2 oracle gaps, and 0 addressable by any rung: these are assertions about values that do not exist until execution. A candidate rule measured at 32/2,148 was reverted because it counted claims the engine must abstain on. Rules: state the scope limit; never quote coverage over an unaddressable population; instance counts rank rungs within a corpus, never across.

5.2 Adversarial battery and selfcheck

Seven adversarial claims—perturbed identities, contradictory premises, over-closures—are refused 7/7 (expected verdicts REFUTED or UNVERIFIABLE, never CERTIFIED); twelve structural invariants including the single-writer graph pass on every execution; selfcheck additionally covers certificate round-trip with kernel agreement (0 false certificates), Lean-arm agreement, self-solve regressions, and 26 negation/nullness twins. Failures of past versions are pinned as twins that fail when the fix is reverted.

6 Qualitative case study: harness sanity check via patch discrimination

Four constructed vulnerability tasks (integer overflow, out-of-bounds read, path traversal, out-of-bounds write), each with a reference fix and a trivial random-guard fix, graded per property from an independent artifact. Result: reference 4/4 certified, trivial 0/4 certified, discriminated 4/4 (Table 3).

Table 3: Control study: per-property certification of reference vs trivial patches ($n = 4$ tasks, two properties each).

patch	certified properties	verdict
reference fix (CWE-190/125/22/787)	all (4/4 tasks)	discriminated
trivial random-guard fix	none (0/4 tasks)	rejected

Table 4: Harness defects surfaced by the trivial arm, and their fixes.

defect	symptom	fix
oracle and subject shared a generator	both arms passed before the fix	claims re-authored by a source independent of the artifact generator (separate pass; no shared derivation)
four-column artifact dropped by parser	no machine-readable facts	strict schema parse
derived-quantity bound unrecognized	correct patch initially refuted	bound-recognition rung

Per-property partial credit replaces binary scores: a patch fixing one of two bounds scores 1/2 and is told *which*.

The trivial arm is the substantive finding: it exposed three harness defects, each hiding behind a confident wrong answer, uncovered through the evaluation of four constructed benchmark tasks (Table 4). Scope: this section is a *qualitative case study and harness sanity check*, not an evaluation of program repair— $n = 4$ constructed tasks with a regex artifact layer (its weakest link) demonstrate per-property discrimination over a declared artifact and validate the harness against negative controls; they do not support generalization about patch verification. Scaling to standard benchmarks (Defects4J-/SWE-bench-class suites, $n \geq 20$) is future work (§10).

7 Improvement lifecycle

The engine’s development loop is itself a contribution: (1) pick one declined reason *by volume*; write the tactic plus a labelled test; the claim type mines, certifies, and passes its test *before* it counts in any report. (2) Premise management: any specification not directly verified by the proof kernel is logged as a declared contract with explicit provenance (its resting place named); promotion is versioned and evidenced, never a rename. (3) Ledger changes ship with twins that fail when reverted. (4) Re-run the census; report four numbers. Regression algebra: promote all verdict buckets—CERTIFIED must stay certified, REFUTED must stay refuted, UNVERIFIABLE must stay abstain, FAULT quarantined—and keep the corpus balanced or the gate passes vacuously. For training proposers: registry trains, corpus judges, never both on the same row.

8 Limitations

The fragment excludes side effects, exception-as-value semantics, and open-world inputs; these claims safely fall to the observation class, and the measured in-fragment share varies two orders of magnitude across our own populations (13.0% static SQL, 0.1% general Python, 0% pytest)— $\mathcal{T}_Q$ coverage is population- dependent and must always be reported against a named reference class. Proposer improvements raise *yield* only—soundness is structural. The control study is $n = 4$ with a regex artifact layer (its weakest link). Oracle circularity holds outside external ground-truth corpora. Coverage numbers are conditional on addressable populations, as the ceiling shows. The system certifies claims *about stated artifacts*, not that artifacts match informal intent.

9 Conclusion

The certification engine shows what a claim-certification system looks like when built to the discipline it preaches: one writer, artifact- mediated verdicts, typed readings, three detached checkers, and fail-

closed alarms—with the single-writer invariant machine-checked rather than promised. The empirical story is censuses, not scores: 9,258 claims disposed with zero false certificates, 7/7 adversarial falsifications refused, 84/84 capability tests passed, a banking corpus yielding 171 file-evidenced money-path lemmas, a refusal record that names exactly what is missing, a coverage ceiling stated rather than hidden, and a control arm 4/4 vs 0/4 disciplined enough to catch its own harness. The declined set is the roadmap; every reason code is a ticket.

10 Future Work and Open Challenges

This work was conducted within the hybrid-neurosymbolic research program of **Optimality.ai** (neural proposers, symbolic authorities, fail-closed contracts). The open challenges are as follows.

First, scale the qualitative control study into a real evaluation: a suite of at least 20 vulnerability tasks from standard benchmarks (Defects4J-/SWE-bench-class), with an artifact layer stronger than the current regex transcription—the acknowledged weakest link. Second, raise fragment coverage by volume, driven by the reason histogram: the IEEE-float interval rung, observation→proof migrations, and solver rungs for whatever reason code dominates the next census. Third, train learned proposers under the registry/corpus discipline (registry trains, corpus judges, never the same row). Fourth, carry the claim IR and the four-number census discipline to new fragments—configuration, policy, contracts—where declarations become premises and runtime claims become watchdogs unchanged. Fifth, move censuses to fleet scale at pinned commits, every certificate re-derived by a detached checker, every decline a named ticket. Sixth, replace the syntactic selfcheck walk with an alias-aware static analysis so the single-writer invariant holds over resolved call graphs rather than textual ones.

References

- Rajeev Alur, Rastislav Bodík, Garima Juniwal, et al. Syntax-guided synthesis. In *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–8, 2013.
- Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT modulo theories library. In *Handbook of Satisfiability*, 2021.
- Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 238–252, 1977.
- Leonardo de Moura, Soon Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Proceedings of the 25th International Conference on Automated Deduction (CADE)*, pages 378–388, 2015.
- DeepSeek-AI. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via cold-start reinforcement learning. *arXiv preprint arXiv:2504.21801*, 2025.
- Kevin Ellis, Catherine Wong, Maxwell Nye, et al. DreamCoder: Growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.
- C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580, 1969.
- George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 106–119, 1997.
- Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Bailey Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of LLM-generated code. In *IEEE Symposium on Security and Privacy (S&P)*, pages 754–768, 2022.
- Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. In *Nature*, volume 625, pages 476–482, 2024.

Xin Xia, Yangyang Zhang, Yuxia Yang, David Lo, and Xin Li. Invgen: An efficient general-purpose program invariant generator. In *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 401–412, 2018.

Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: Claims are scoped to censuses at pinned commits, zero-false-certificate behavior under the stated checker assumption, and the documented control study $n=4$.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Limitations name the fragment boundaries, oracle circularity, the $n=4$ control study, and the regex artifact layer.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper reports System Invariants (fail-closed certification, zero false certificates) grounded in architecture and restated from companion work; no new mathematical proofs are claimed beyond statement-level results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Pinned commits, reason-code taxonomies, adversarial battery contents, selfcheck invariants, and four-number census discipline are fully disclosed.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- Reproducibility requires a reasonable avenue for the reported results, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: Engine and corpora will accompany the final version; the paper discloses full census numbers, provenance hashes, and verdict vocabularies sufficient to audit the claims.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [\[Yes\]](#)

Justification: Section 4-5 describe the disposal pipeline, frontends, mining, and storage in implementation-level detail.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [\[Yes\]](#)

Justification: Full-population counts (no sampling), zero-false-certificate and battery rates, and per-property control results; sampling-based error bars are not applicable to censuses.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).

- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [\[Yes\]](#)

Justification: Censuses are CPU-bound minutes-scale runs over pinned repositories; hardware and runtime are described in the artifacts.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the code of ethics appropriate to the authors' field and institutions?

Answer: [\[Yes\]](#)

Justification: The authors have reviewed the applicable code of ethics.

Guidelines:

- The answer NA means that the authors have not reviewed a code of ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [\[Yes\]](#)

Justification: Fail-closed certification reduces silent acceptance of incorrect AI-generated claims; the refusal record makes deployment limits explicit.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: No high-risk model release; the engine is a verifier with fail-closed defaults.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Pinned third-party repositories are cited by commit hash with their upstream licenses.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The corpus, taxonomy, and battery are new assets, documented for release.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: No crowdsourcing.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- Under prevailing codes of ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: No human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the applicable code of ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs appear only as inference-time proposers whose output is verified; their role is stated and they are not part of the trusted base.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.