From Agents to Axioms: Verifier-Gated Lean Formalization for Statistical Learning Theory

Robert Sneiderman [1]

Abstract

This paper presents a verifier-gated acceptance protocol for Lean formalization, exercised with coding-agent assistance, demonstrated on *FormalSLT*, a nontrivial Lean 4 / Mathlib statistical-learning-theory library. The contribution is not a new statistical-learning-theory theorem or a model benchmark; it is a reproducible PR-level protocol in which machine-written formalization edits are treated as untrusted proposals and admitted only through Lean build, proof-hygiene gates, and a blocking `#print axioms` audit.

The submitted artifact contains 45 Lean modules, 20,080 source lines including `FormalSLT.lean`, 412 theorem/lemma declarations, and 107 showcase axiom traces covering VC, PAC-Bayes, Rademacher, Azuma, covering-number, and algorithmic-stability components. Over a 61h39m May 2026 run, theorem-scoped targets were dispatched through isolated worktrees and pull requests; the ledger records 67 opened PRs, 60 merged, and 7 closed or superseded. Of the 67 ledger PRs, 9 carry an explicit coding-agent label; the remainder are operator-dispatched branches whose anonymized provenance cannot separate agent-written from operator-written proof terms. We therefore present the ledger as feasibility evidence for the acceptance protocol, not as an agent-productivity or agent-vs-operator result. The accepted showcase theorems audit to {`propext`, `Classical.choice`, `Quot.sound`}. The result is a concrete acceptance layer for reliable scientific agents: agents may propose formalization state transitions, but the Lean kernel, CI, and axiom audit decide what lands.

[1]Independent. Correspondence to: Robert Sneiderman <robbysneiderman@gmail.com>.

Proceedings of the ICML 2026 Workshop on AI for Math, Seoul, South Korea, 2026. Copyright 2026 by the author(s).

1. Introduction

Statistical learning theory is a useful stress test for agent-assisted formalization because the named results are familiar but unforgiving. A Sauer–Shelah or PAC-Bayes statement fixes a class shape, a boundedness condition, a measurability hypothesis, a probability space, and constants that vary across references. In a proof assistant, each of these assumptions has to become an explicit Lean hypothesis.

That makes the domain a good place to ask a practical question: can coding agents help build a real formal library while the review process remains mechanical? Our answer is not to trust the model. It is to require every accepted edit to pass the same checks: Lean typechecking, proof-hygiene checks, and a blocking axiom audit.

This does not close the full self-evolution loop: the proposing agents are not updated by the verifier signal. It does isolate the acceptance layer such a loop would need. Each PR is a proposed state transition, CI supplies the selection rule, the trunk is the accumulated state, and the showcase axiom audit is a verifier-grounded acceptance criterion.

The artifact is *FormalSLT*, a Lean 4 / Mathlib library (de Moura & Ullrich, 2021; The mathlib Community, 2020) spanning finite-class, VC, PAC-Bayes, Rademacher, Azuma, covering-number, and stability components, publicly available at `https://github.com/Robby955/FormalSLT`. The human role in the loop is statement adequacy: maintaining the informal-to-formal target contract, checking dependency discipline, and rejecting formal statements that do not match the intended mathematical claim.

The paper's contribution is the workflow around the artifact:

- a worktree-based loop for theorem-scoped Lean targets;
- a four-gate CI protocol with a blocking showcase axiom audit;
- an anonymized PR ledger reporting throughput and failure modes;
- a claim-to-signature adequacy table stating what each showcase theorem does and does not prove;

- a nontrivial statistical-learning-theory library as evidence that the loop was not only exercised on toy examples.

We do not claim model self-improvement, theorem-discovery novelty, or a Claude-vs-Codex benchmark. We claim a reproducible acceptance protocol for agent-written formalization work. We report declarations as artifact scale, not proof difficulty, and use the ledger as feasibility evidence rather than an estimate of agent productivity, marginal speedup, or human labor saved. Table 1 records the counts and where each one is verified.

Section 2 gives the workflow. Section 3 reports the Lean artifact. Section 4 positions the paper against SLT formalizations and agentic theorem-proving systems. Section 5 gives the failure modes, limitations, and reproducibility recipe.

2. Workflow

Lane decomposition. The library is split by theorem family. The main namespace trees are `Azuma.*`, `Rademacher.*`, `VC.*`, `PACBayes*`, `Stability/` / `AlgorithmicStability`, and `Covering/`. A shared `Risk`/`ERM` spine sits below them. Each lane has a target list of named theorems, informal statements, desired Lean signatures, and named Mathlib dependencies. Agents pick one target at a time.

Per-lane git worktrees. Each in-flight target gets a fresh git worktree from the current `release-candidate` tip. The agent edits that tree, runs `lake build`, and never touches sibling lanes. The orchestration host used separate worktrees for in-flight targets during the submission window; merge-time conflicts were serialized by pull request review.

Parallel agent pool. Claude and Codex coding-agent CLIs were both available in the pool. The anonymized ledger preserves only partial provenance: 9 of 67 PRs carry an explicit Claude/Codex prefix, while 58 rows are anonymized as operator- or human-driven branches that may still include agent assistance. We therefore report aggregate throughput and failure modes, not a Claude-vs-Codex comparison.

Continuous integration as verification oracle. The CI workflow (`.github/workflows/ci.yml`) runs four blocking gates on every pull request.

1. *Build.* `lake exe cache get && lake build FormalSLT` compiles every cited theorem against pinned Mathlib. The Lake configuration treats Lean warnings as errors, so a proof term depending on `sorryAx` cannot land silently.
2. *Examples.* CI runs every file in `examples/`, including the showcase theorem manifest.
3. *Proof-hygiene grep.* Regex gates reject `sorry`, `admit`, custom `axiom`, custom `constant`, `opaque`, and `unsafe` escape hatches in project source. These regexes are defense in depth, not the soundness proof.
4. *Blocking axiom audit.* `scripts/axiom_audit.py` runs over `examples/CheckShowcaseTheorems.lean` and fails unless every printed axiom set is a subset of {`propext`, `Classical.choice`, `Quot.sound`}.

The merge rule is deliberately blunt. Either Lean typechecks the term, the hygiene gates pass, and the blocking axiom audit reports only canonical Mathlib axioms, or the PR is rejected. Human review focuses on the part Lean cannot decide: whether the formal statement is the intended theorem.

What is standard versus novel here. Three of the four gates are standard Mathlib practice: Mathlib's own CI builds against a pinned toolchain, treats warnings as errors so that `sorryAx` cannot land silently, and relies on human review to judge whether a statement is the intended one. Gates 1–3 simply port that discipline into a small standalone repository. The non-standard element is Gate 4: a *blocking* `#print axioms` audit run over an explicit showcase manifest (`examples/CheckShowcaseTheorems.lean`), which fails the build unless every headline declaration and its named dependencies reduce to {`propext`, `Classical.choice`, `Quot.sound`}. Mathlib audits axioms but does not gate merges on a per-claim manifest. The second non-standard element is packaging: the manifest, the audit script, and the four-gate `ci.yml` ship as a drop-in acceptance layer that a fresh Lean repository can adopt unchanged. The contribution is this Gate-4-plus-packaging delta, not the re-use of Mathlib's existing build and hygiene discipline.

PR-based composition. An agent or operator that finishes a target opens a pull request against `release-candidate`. CI runs; the human reviewer reads the diff, checks the Lean signature against the target list, verifies lane-locality and dependency discipline, and merges only after the mechanical gates are green. The anonymized ledger records 67 PRs over a 61h39m window: 60 merged and 7 closed or superseded without merge. Median PR lifetime in Table 1 starts when a PR is opened after local iteration and pre-PR checks; it is not total theorem-development time or total human-review time. The same ledger gives a concrete parallelism measure: at peak, 8 PRs were open simultaneously, and active

Table 1. Workflow evidence. Counts are generated from the submitted artifact snapshot and anonymized PR ledger. "Theorem/lemma declarations" means comment-skipped line-leading Lean declarations introduced with `theorem` or `lemma`; it excludes `def`, `instance`, `example`, and prose inside comments. Window: 61h39m in May 2026. All counts are feasibility evidence for the acceptance protocol; none is a productivity or speedup measurement.

Evidence item	Count	Where verified
Lean modules	45	file tree
Lean source lines	20,080	snapshot script
Theorem/lemma declarations	412	snapshot script
Showcase axiom traces	107	axiom audit
Total PRs opened	67	PR ledger
PRs merged	60	PR ledger
PRs closed/superseded	7	PR ledger
Opened-PR merge rate	60/67 ($\approx 89.6\%$)	computed
Median merged-PR lifetime	6 min	PR ledger
Max simultaneously open PRs	8	PR ledger
Mean open PRs over window	2.96	computed
Agent-family provenance	partial	branch/commit prefixes
Artifact checks	passing	supplement quickstart

PR-minutes divided by wall-clock window gives 2.96 mean open PRs. Because branch names were anonymized and squash commits can combine operator activity with agent assistance, we do not use the ledger for a Claude-vs-Codex performance table.

Prompting pattern. Every prompt names one theorem target: an informal statement, a desired Lean signature, and one source file the agent should import and reuse. The hard contract is explicit: no `sorry`, no `admit`, no custom `axiom`; the proof must close or the PR is rejected. We kept targets small enough for one Lean file, usually one theorem plus a few supporting lemmas. The most important prompt field was the compositional-source pointer. Without it, agents tended to reprove already-formalized lemmas instead of importing them.

3. Verified Artifact

The submitted snapshot contains 45 Lean library modules, 20,080 source lines including `FormalSLT.lean`, and 412 theorem/lemma declarations outside comments.

The artifact is not a collection of isolated lemmas. A supplemental static source-structure scan finds 463 textual local-reference edges among the 412 theorem/lemma declarations, with 257 declarations referencing at least one uniquely named local theorem or lemma; we use this only as evidence of compositional source structure, not proof difficulty or mathematical novelty. Three proof chains share the `Risk`/`ERM` spine and close to named bounds with explicit constants: (i) a PAC–VC ERM excess-risk tail through Sauer–Shelah, Rademacher symmetrization, Massart's finite-class bound, and a binary VC bridge; (ii) a PAC-Bayes finite-grid McAllester confidence bound through Catoni's exponential-moment template and a formalized Donsker–Varadhan variational inequality; and (iii) a uniform-stability expected generalization-gap bound through Bousquet–Elisseeff's coordinate-swap argument plus a McDiarmid-style bounded-differences step. These are standard statistical-learning-theory and concentration ingredients (Shalev-Shwartz & Ben-David, 2014; Boucheron et al., 2013). Table 2 indexes the eight headline theorems.

The closing theorem of the PAC–VC chain has the Lean signature shown in Listing 1, representative of the boundary statements the workflow targeted.

Every theorem listed in `examples/CheckShowcaseTheorems.lean` (the showcase manifest covering the eight headline theorems of Table 2 plus their named dependencies) reduces under `#print axioms` to the canonical Mathlib triple {`propext`, `Classical.choice`, `Quot.sound`}. The submitted audit checks 107 showcase traces. The CI build is 0-`sorry` under warning-as-error and contains no custom `axiom` or `constant` declarations in our source. The showcase manifest is theorem-facing: it audits the public claims used in the headline theorem chains, while auxiliary lemmas remain under build and hygiene gates and enter the paper's claims through audited endpoints. Theorems outside the showcase manifest are subject to the build and hygiene gates but not the showcase axiom audit; we do not claim the canonical-triple property for them without audit. Because statement adequacy is the residual human burden, Table 4 is part of the artifact evidence: it records the scope, constants, and non-claims for each headline theorem.

The 13-line `donsker_varadhan` proof body is short because the analytic work is factored upstream in the same file. It composes `klDiv_nonneg`, `klDiv_gibbs_eq`, and four `gibbsPosterior_*` helper lemmas with `linarith`; those upstream lemmas account for roughly

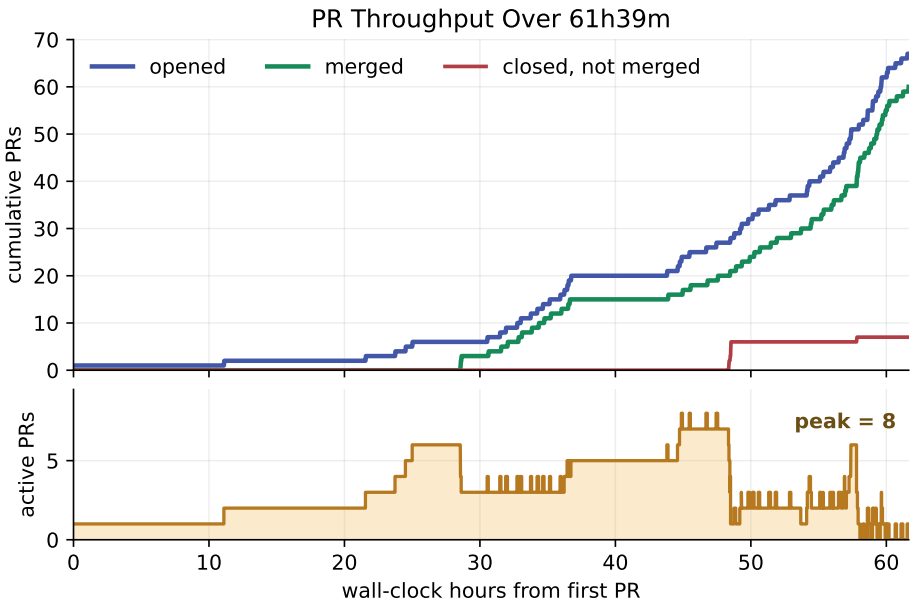

Figure 1. **Ledger-derived throughput curve.** The anonymized PR ledger records 67 opened PRs over 61h39m, of which 60 merged and 7 closed or were superseded. The shaded curve shows instantaneous open PRs, peaking at 8. This plot is evidence of feasibility and parallel queue pressure, not a causal speedup measurement.

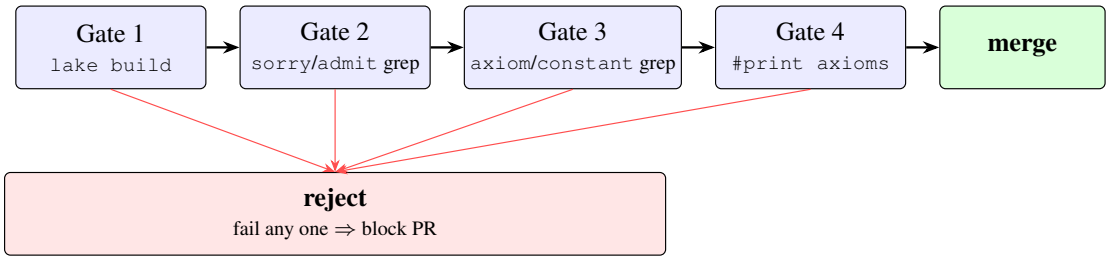

Figure 2. **Four-gate CI as verification oracle.** A PR clears all four gates and merges, or fails one and is rejected. No human heuristic about proof correctness intervenes: either the kernel typechecks the term and the showcase axiom set is canonical, or the PR is blocked.

100 source lines in `FormalSLT/PACBayesKL.lean`.

4. Comparison and Related Work

The closest contemporary SLT effort is Zhang et al. (2026), which formalizes the continuous empirical-process route in Lean 4 (Gaussian Lipschitz concentration, Dudley's entropy integral, a sparse least-squares rate). Sonoda et al. (2025) formalize Rademacher-style generalization bounds in Lean, with a Dudley entropy-integral extension in the Feb. 2026 version. Both target continuous-process or Rademacher toolkits and report human-AI collaboration; ours targets finite PAC–VC, PAC-Bayes, and Bousquet–Elisseeff stability chains. Our contribution sits on a different axis: we specify PR acceptance under a fixed CI contract, including a blocking `#print axioms` audit and a machine-readable showcase manifest.

The agentic theorem-proving literature mostly evaluates a different unit of work: whether systems can solve formal theorem tasks. We instead ask how machine-written edits can be admitted into a growing library without trusting the writer. Prover Agent (Baba et al., 2025), MA-LoT (Wang et al., 2025b), Ax-Prover (Breen et al., 2025), DeepSeek-Prover-V2 (DeepSeek-AI, 2025), Kimina-Prover (Wang et al., 2025a), and Goedel-Prover-V2 (Lin et al., 2025) optimize proof search or theorem success on benchmark-style tasks. FormalML (Yang et al., 2025) studies Lean 4 subgoal completion in machine-learning theory, drawing on FoML and Optlib-style developments (Li et al., 2024). We do not claim stronger isolated-theorem pass rates. Our unit is the pull request: how agent-written edits enter, fail, or land in a coherent Lean statistical-learning-theory library under fixed CI and axiom-audit rules.

Recent autonomous-discovery reports further show why the acceptance layer matters. The OpenAI unit-distance note (OpenAI, 2026) reports autonomous generation of a can-

```
theorem vc_erm_excessRisk_tail
  [MeasurableSpace Z] [StandardBorelSpace Z]
  {mu : Measure Z} [IsProbabilityMeasure mu]
  {l : iota -> Z -> R} {B : R} (hB : 0 < B)
  (hl_meas : forall i, Measurable (l i))
  (hl_bdd  : forall i z, |l i z| <= B)
  {n d : N} (hn : 0 < n) (hd : 0 < d) (hdn : d <= n)
  (hGrowth : ...)  -- effective-class <= sum binomials
  (hERM : forall S, IsERM (empiricalRisk S l) (hhat S))
  (i_star : iota) {eps : R} (heps : 0 <= eps) :
  (piMeasure mu n).real
    { S | risk mu l i_star
        + 4 * B * Real.sqrt (2*d*Real.log (Real.exp 1*n/d)/n)
        + 2 * eps <= risk mu l (hhat S) }
  <= 2 * Real.exp (- eps^2 * n / (8 * B^2))
```

Listing 1. VC ERM excess-risk tail (`FormalSLT/VC/SampleComplexity.lean`).

Table 2. Eight headline theorems produced by the workflow, with proof-block length (Ln). Pinned to {`propext`, `Classical.choice`, `Quot.sound`} via `examples/CheckShowcaseTheorems.lean`.

Chain	Result (informal)	Lean lemma (module:line)	Ln
PAC–VC	VC growth bound $(en/d)^d$	`sauerShelah_polynomial_bound` (`VC/SauerShelah.lean:44`)	102
PAC–VC	Rademacher–VC bridge to Massart	`empiricalRademacherComplexity_le_massart_effective` (`VC/Rademacher.lean:85`)	52
PAC–VC	ERM excess-risk tail	`vc_erm_excessRisk_tail` (`VC/SampleComplexity.lean:334`)	89
PAC-Bayes	Donsker–Varadhan KL variational	`donsker_varadhan` (`PACBayesKL.lean:235`)	13
PAC-Bayes	Catoni fixed-λ bound, $[0, 1]$ loss	`finiteCatoni_badEventMass_le_delta` (`PACBayesBoundedLoss.lean:408`)	76
PAC-Bayes	McAllester finite-grid peeling	`finiteMcAllesterGridPeeling_badEventMass_le_delta` (`PACBayesBoundedLoss.lean:766`)	29
Stability	Bousquet–Elisseeff expected gap	`expectedFiniteGeneralizationGap_le_uniformStability_finiteProduct` (`AlgorithmicStability.lean:1604`)	17
Stability	McDiarmid bounded-differences tail	`mcdiarmid_inequality_iid_const_width` (`Stability/BousquetElisseeff.lean:118`)	49

didate solution to a classical discrete-geometry problem followed by human and external expert review. FormalSLT makes a smaller but more mechanical claim: no new theorem is asserted, but every accepted formalization transition is kernel-checked, CI-gated, and axiom-audited.

Prior formalizations of single learning-theory chains report multi-month person-effort budgets where effort is specified. Table 3 summarizes the reported profiles. None of these works use coding-agent automation. Our claim is narrower than a productivity theorem: in 61h39m, the workflow closed three main chains plus cross-track Azuma scaffolding, and the accepted showcase theorems typecheck with the canonical axiom triple.

Table 4 is the guardrail against overclaiming. It maps each headline theorem to the assumptions, constants, and non-claims in the Lean statement. The main human-review risk is not proof validity; Lean decides that. The risk is statement adequacy: whether the formal theorem is the intended informal theorem. The source tarball includes `THEOREM_MANIFEST.md` with the 107 showcase declaration names; Table 4 gives the paper-level hypothesis and non-claim summary for the eight headline theorems.

5. Lessons and Reproducibility

5.1. What worked

Three workflow patterns drove the opened-PR merge rate. (i) *CI as the verification oracle.* The four gates above are mechanical, fast, and binary: a green PR typechecks against pinned Mathlib, passes proof-hygiene checks, and has a canonical showcase axiom trace. This is what makes the loop tolerable to low-supervision agent input. (ii) *Per-lane worktrees.* Pinning each in-flight target to its own working tree eliminated cross-lane edit conflicts; sibling agents only met at PR merge time, which the human reviewer serialized. (iii) *Single-target, single-source-file prompts.* Targets that fit in one Lean file with one named compositional source were easiest to review and merge; broader cross-module refactors were more often narrowed, split, or closed before merge. Together, these patterns implement verifier-gated acceptance: agents propose state transitions, mechanical gates reject invalid edits, and the trunk accumulates only changes admitted by Lean and the axiom audit.

Table 3. Cost profile across recent learning-theory formalizations. "Not specified" indicates that the cited writeup does not report a comparable wall-clock or person-effort figure. Wall-clock figures for our work are from the anonymized workflow ledger and artifact snapshot.

Effort (system)	Scope	Wall-clock	Person-effort
Tassarotti et al. (2021) (Lean 3)	Decision-stump PAC	not specified	multi-month
Bagnall & Stewart (2019) (Coq)	MLCert generalization guarantees	not specified	not specified
Karayel & Tan (2023) (Isabelle)	Concentration tooling	not specified	not specified
Sonoda et al. (2025) (Lean 4)	Rademacher / Hoeffding / McDiarmid	not specified	not specified
Zhang et al. (2026) (Lean 4)	Continuous empirical-process	not specified	not specified
This work (Lean 4)	PAC–VC, PAC-Bayes, Stability, Azuma	61h39m, 60 merged PRs	not specified

Table 4. Claim-to-Lean-signature adequacy for the showcase theorems. "Scope" indicates whether the formalized statement is over a finite hypothesis class or extends to general measurable classes. "Not proved" lists the natural extension that the Lean signature explicitly does not close.

Informal theorem	Lean theorem (module:line)	Scope	Constants	What is not proved
VC growth bound	`sauerShelah_polynomial_bound` (`VC/SauerShelah.lean:44`)	finite	explicit $(en/d)^d$	continuous-class lifts
Rademacher–VC bridge	`empiricalRademacherComplexity_le_massart_effective` (`VC/Rademacher.lean:85`)	finite, B-bounded loss	explicit $B\sqrt{2\log\lvert\mathcal{H}\rvert/n}$	non-binary loss
ERM excess-risk tail	`vc_erm_excessRisk_tail` (`VC/SampleComplexity.lean:334`)	finite class, bounded loss, `StandardBorelSpace`	explicit $4B\sqrt{2d\log(en/d)/n}$	continuous-class generalization
Donsker–Varadhan KL	`donsker_varadhan` (`PACBayesKL.lean:235`)	finite-measure setting	explicit	continuous DV under non-finite priors
Catoni fixed-λ	`finiteCatoni_badEventMass_le_delta` (`PACBayesBoundedLoss.lean:408`)	finite class, [0, 1] loss	explicit λ-grid	adaptive λ
McAllester finite-grid peeling	`finiteMcAllesterGridPeeling_badEventMass_le_delta` (`PACBayesBoundedLoss.lean:766`)	finite class, [0, 1] loss	explicit grid budget	continuous-grid peeling
Bousquet–Elisseeff expected gap	`expectedFiniteGeneralizationGap_le_uniformStability_finiteProduct` (`AlgorithmicStability.lean:1604`)	finite product measure	explicit	high-prob version
McDiarmid bounded-differences	`mcdiarmid_inequality_iid_const_width` (`Stability/BousquetElisseeff.lean:118`)	i.i.d. with constant width	explicit Hoeffding-style	non-i.i.d. extensions

5.2. What failed and why

Five recurring failure classes shaped the orchestration. Table 5 lists the primary detector for each class; the anonymized ledger records aggregate closed-without-merge PRs, not gate-specific rejection counts. The 7 closed-without-merge PRs were superseded or rejected before landing; failed local attempts not opened as PRs are outside the ledger count.

The reviewer load was concentrated on statement adequacy and integration hygiene: does the Lean signature match the informal target, does the diff stay in its lane, and does the PR preserve the import structure? Proof correctness and showcase axiom discipline are checked mechanically.

5.3. Statement-adequacy near-misses

The gates accept any well-typed term, so a formally valid theorem can still be the wrong theorem. Three concrete cases from the development show the shape of the human-caught residual.

Vacuous parameter regime. The approximate-ERM predicate `IsApproxERM Rhat optError` is sound for `optError` ≥ 0, but with `optError` < 0 it becomes strictly stronger than exact `IsERM` and is typically vacuous on a finite class: any downstream excess-risk corollary built on it would typecheck while quantifying over an empty premise. Lean accepts the definition; the human contract pins `optError` $= 0$ for the exact-ERM corollary and documents the boundary in source. The Lean signature alone does not flag the degenerate regime.

Tail bound that could have been vacuously satisfied. The sharp-McDiarmid headline theorem bounds an upper-tail probability. A bound of the form $\Pr[\cdot] \leq c$ is uninformative if the event is empty or the bound exceeds 1. The Lean term typechecks regardless. Statement adequacy here required a separate non-vacuity witness: a concrete Bernoulli-product instance for which the bound is strictly below 1, scales as ε^2, and controls a nonempty positive-probability event. Without that witness the theorem would have been formally true and substantively empty.

Table 5. Failure-mode catalog and primary detector. The table is qualitative; the anonymized ledger does not include gate-specific rejection counts. The blocking axiom audit protects the showcase manifest from noncanonical dependencies.

Failure mode	Description	Caught by
Module fabrication	Agent imports a nonexistent Mathlib path	**Gate 1**
Stale-base rebases	PR opened against pre-rename trunk; build red on post-merge rebuild	**Gate 1**
Mathlib cache races	Concurrent `lake exe cache get` on `.lake/build`	**Gate 1** (deterministic fail, retry clears)
Subtle axiom leak	Showcase theorem depends on a noncanonical axiom set	**Gate 4**
Overstated completion	Diff matches the informal target imperfectly (weaker variant)	Reviewer (post-CI)

Missing-hypothesis pattern (illustrative). A finite-class Rademacher or Massart bound stated without the per-coordinate boundedness or measurability hypothesis can still typecheck on a restricted instantiation, yielding a statement that reads like the textbook bound but does not entail it for unbounded or non-measurable losses. The intended theorem requires those hypotheses explicitly; a draft that drops them is accepted by Lean and rejected on human review.

These are why the artifact ships the claim-adequacy table and the per-theorem non-claim column: they are the audit surface for exactly the failure that the kernel cannot catch.

5.4. Reproducibility recipe

The supplement (https://github.com/Robby955/FormalSLT) is meant to be runnable, not just archival. It is designed to let readers reproduce the artifact and reuse the acceptance checklist in another Lean repository:

1. *Lean toolchain.* `elan` for Lean version management; pin to the toolchain in `lean-toolchain` (the file is the source of truth); `lake` for build.
2. *Pinned Mathlib commit.* The exact commit hash lives in `lake-manifest.json`; reproducing the build requires the same hash because Mathlib API churns.
3. *Four-gate CI workflow.* `.github/workflows/ci.yml` contains the four gates verbatim. Drop into a fresh GitHub repository and the gates apply unchanged.
4. *Prompting contract.* `AGENTS.md` and `PROMPT_TEMPLATE.md` record the operational proof contract: scoped theorem-family work, local build before review, no `sorry` or `admit`, no custom axiom or constant, and explicit claim boundaries.
5. *Target schema.* `PROMPT_TEMPLATE.md` gives the target schema used for theorem-scoped dispatch: target family, Lean declaration name, informal statement, desired Lean location, and compositional source file.
6. *Showcase verifier.* `examples/CheckShowcaseTheorems.lean` runs `#print axioms` for every public theorem and serves as the Gate-4 fixture.
7. *Agent CLI.* Any coding-agent CLI that writes files and runs shell commands. We used Claude and Codex; one is sufficient for a serial loop, two enable the parallel pool.
8. *Orchestration pattern.* The workflow uses a small shell pattern: create a fresh worktree, dispatch one prompt-scoped target, run the local checks, and open a PR only after a green local build. The submitted supplement reproduces the artifact and acceptance gates; it does not rely on an orchestration script for reviewer verification.

Reviewer quickstart. The supplement includes `REVIEWER_QUICKSTART.md` with copy-paste commands. The minimal path is: clone the repository, run `lake exe cache get`, run `lake build FormalSLT`, then run `scripts/axiom_audit.py` on `examples/CheckShowcaseTheorems.lean` with allowed axioms {`propext`, `Classical.choice`, `Quot.sound`}. The two Python scripts in `scripts/` regenerate the snapshot and ledger summaries.

5.5. What we did not try, and why

The cleanest robustness check would replay the same target queue with one agent. We did not run that experiment, so the paper reports throughput and failure-mode evidence rather than causal attribution to parallelism. A non-Lean comparison (Coq, Isabelle, Rocq) is also out of scope.

5.6. Public artifact and ledger

The public/anonymized artifact release is squashed and does not preserve full internal operator history. The supplementary ledger records PR IDs, target family, timestamps, merge decision, and final commit hash. That is enough for throughput and failure-mode reporting, but not enough for a Claude-vs-Codex performance comparison.

5.7. Relation to prior work on the same artifact

An earlier non-archival draft about the same artifact emphasizes the formalized theorems rather than the workflow. This submission is distinct in scope: it evaluates the PR-level orchestration, CI contract, and axiom-audited acceptance rule. We include this note for transparency.

5.8. Limitations

The limits are straightforward. The parallel-agent result is observational; there is no single-agent replay on the same queue. The workflow is Lean/Mathlib-specific; other proof assistants will expose different bottlenecks. CI checks proof correctness and axiom discipline, but not mathematical intent. Statement adequacy remains a human-reviewed contract, which is why the artifact includes showcase manifests, claim-to-signature tables, and exact reproducibility scripts. Finally, the theorem count is not comparable to MiniF2F or PutnamBench pass rates: FormalSLT is a library with cooperating named lemmas, not a set of independent contest targets. We did not instrument reviewer wall-clock, so human effort is reported only structurally: one reviewer serialized all merges, review load concentrated on statement adequacy and lane-locality, and the ledger's median merged-PR lifetime bounds post-open latency but not total human-review time.

References

Baba, K., Liu, C., Kurita, S., and Sannai, A. Prover agent: An agent-based framework for formal mathematical proofs. *arXiv:2506.19923*, 2025. 2nd AI for Math Workshop @ ICML 2025.

Bagnall, A. and Stewart, G. Certifying the true error: Machine learning in Coq with verified generalization guarantees. In *AAAI*, volume 33, pp. 2662–2669, 2019.

Boucheron, S., Lugosi, G., and Massart, P. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.

Breen, B., Del Tredici, M., et al. Ax-prover: A deep reasoning agentic framework for theorem proving in mathematics and quantum physics. *arXiv:2510.12787*, 2025.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. Conference on Automated Deduction (CADE), 2021.

DeepSeek-AI. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv:2504.21801*, 2025.

Karayel, E. and Tan, Y. K. Concentration inequalities. *Archive of Formal Proofs*, 2023.

Li, C., Wang, Z., He, W., Wu, Y., Xu, S., and Wen, Z. Formalization of complexity analysis of the first-order algorithms for convex optimization. *arXiv:2403.11437*, 2024. Companion library: github.com/optsuite/optlib.

Lin, Y. et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv:2508.03613*, 2025.

OpenAI. Planar point sets with many unit distances. Technical note, 2026. https://cdn.openai.com/pdf/74c24085-19b0-4534-9c90-465b8e29ad73/unit-distance-proof.pdf.

Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

Sonoda, S., Kasaura, K., Mizuno, T., Tsukamoto, M., and Onda, A. Lean formalization of generalization error bound by Rademacher complexity and Dudley's entropy integral. *arXiv:2503.19605*, 2025. v1 Mar 2025; v4 Feb 2026 adds the Dudley title clause.

Tassarotti, J., Vajjha, K., Banerjee, A., and Tristan, J.-B. A formal proof of PAC learnability for decision stumps. In *CPP 2021*, pp. 5–17. ACM, 2021.

The mathlib Community. The Lean mathematical library. Conference on Certified Programs and Proofs, 2020.

Wang, H., Unsal, M., Lin, X., et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv:2504.11354*, 2025a.

Wang, R. et al. Ma-lot: Model-collaboration lean-based long chain-of-thought reasoning enhances formal theorem proving. *arXiv:2503.03205*, 2025b.

Yang, X.-W. et al. Formalml: A benchmark for evaluating formal subgoal completion in machine learning theory. *arXiv:2510.02335*, 2025.

Zhang, Y., Lee, J. D., and Liu, F. Statistical learning theory in Lean 4: Empirical processes from scratch. *arXiv:2602.02285*, 2026.