PROOFGATE: A Reproducible Audit of Faithfulness, Alignment, and Vacuity in State-of-the-Art Lean Theorem Provers

Edison Yang [1] **Rithik Satarla** [1] **Neel Marripalapu** [2]

Abstract

Recent neural theorem provers report pass rates on miniF2F and PutnamBench that, taken at face value, suggest formal mathematical reasoning is nearly solved. We argue that the reported figures conflate three distinct phenomena—*proof faithfulness*, *statement alignment*, and *problem vacuity*—and that a single failed check in any one can render a reported result an unaudited claim. We propose PROOFGATE, a reproducible audit pipeline implementing four orthogonal checks (kernel-level axiom inspection, banned-tactic linter, negation-counterexample probe, FormalAlign-style alignment score) that together yield a single dimensionless metric, FAITHFUL-PASS. We release PROOFGATE as a Python package and apply it to every publicly released proof artifact of three recent state-of-the-art systems. Three findings are particularly noteworthy. First, the full kernel-level audit of all 635 DeepSeek-Prover-V2 (miniF2F-test and -valid combined) and Kimina-Prover-72B miniF2F-test proofs against each prover's pinned mathlib revision yields KERNEL-FAITHFUL-PASS$_{\mathcal{T}_1} = 100\%$, with no occurrences of `sorry` and no extra axioms beyond the compiler-trust set; under the strict trusted base $\mathcal{T}_0$, all 438 DeepSeek proofs pass while $186/197$ Kimina proofs pass, the eleven exceptions being precisely those that invoke `native_decide`. Second, SBERT-based alignment scoring on the same items returns a 15.2% misalignment rate on DeepSeek miniF2F-test, within 1.2 percentage points of ReForm authors (2025)'s expert-annotated rate. Third, Goedel-Prover-V2's "released proofs" for miniF2F are in fact the benchmark input statements with `sorry` placeholders; the public release contains *no auditable proofs*, a property the accompanying paper does not state explicitly. Our objective is not to adjudicate the merits of any individual system, but to provide the AI-for-Math community with a measurement instrument the field currently lacks.

[1]Lambert High School, Suwanee, GA, USA [2]University of Texas at Dallas, Richardson, TX, USA. Correspondence to: Edison Yang <edisonly2200@gmail.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

The premise of automated theorem proving is that a proof, once accepted by the kernel of a system such as Lean 4 (de Moura & Ullrich, 2021; Lean FRO, 2024c), constitutes *verifiable evidence*: an independent third party can recheck it and rely on the result. Reported results from neural provers in 2024–2026 have leveraged this premise to advance strong claims of progress—92.2% on miniF2F (Wang et al., 2025), 88.9% on miniF2F (Ren et al., 2025), 88.1–90.4% on miniF2F (Lin et al., 2025), 99.4% on PutnamBench (Logical Intelligence, 2026)—claims that, together with high-profile results outside Lean such as AlphaProof's IMO 2024 silver-medal performance (Google DeepMind, 2024), the broader machine learning community has largely accepted at face value.

Several independent observations from 2025–2026, however, indicate that the chain from *the prover produced a proof* to *the theorem is established* contains more potential points of failure than is commonly acknowledged:

- **Implicit `sorry`.** Lean's permissive tactics (`apply?`, `exact?`, `decide`, `native_decide`) can, in certain configurations, succeed via paths that the read-eval-print loop does not flag as incomplete, despite the resulting proof depending on `sorryAx` or on compiler-trust axioms.
- **Statement misformalization.** Multiple teams have independently corrected formal statements: 5–8 in miniF2F by the Kimina team (Wang et al., 2025) (5 per the public README, 8 per the arXiv paper), 8 by DSP+ (Jiang et al., 2023; Reviving DSP authors, 2025), 16 statements identified as having no solution in their current form by Ospanov et al. (2025) (out of over 300 statement-level corrections in their full re-audit), and 15 PutnamBench items by Aleph (Logical Intelligence, 2026).

- **Informal–formal misalignment.** ReForm authors (2025)'s ConsistencyCheck benchmark reports that 16.4% of miniF2F and 38.5% of ProofNet (Azerbayev et al., 2023) human-written formal statements *contain semantic errors* relative to their natural-language statements.

These observations suggest that the reported pass-rate metric *conflates* questions that ought to be measured separately. A proof of a vacuous statement does not constitute the same type of evidence as a proof of a faithfully formalized theorem with a non-trivial alignment to its natural-language analogue. We distinguish the second and third bullets above, which sound similar: *statement misformalization* refers to benchmark-level errors in published Lean statements (corrected once and shared across all provers using that benchmark), while *informal–formal misalignment* refers to drift between an individual prover's released $(\varphi, \hat{\varphi})$ pair, measured separately for each released artifact. PROOFGATE measures the latter; ReForm (ReForm authors, 2025) measures the former.

Contributions. The audit primitives PROOFGATE builds on are deliberately simple: each individual check has analogues in existing Lean tooling (Section 2). Our contribution is the *integration* of these checks into a uniform pipeline, the *empirical study* of every public artifact from three SOTA systems under that pipeline, and the *reporting protocol* the empirical findings motivate.

1. A unified taxonomy of three failure modes that current pass-rate metrics conflate: *faithfulness*, *alignment*, and *vacuity* (Section 3).
2. PROOFGATE, an open-source Python package that orchestrates four checks (Section 4) into a single dimensionless metric, FAITHFUL-PASS (Section 4.5), and a kernel-only variant, KERNEL-FAITHFUL-PASS. We position the individual checks as substitutable backends: kernel-level axiom inspection can be delegated to lean4checker (Lean FRO, 2024b), signature matching to Comparator (Lean FRO, 2024a), and counterexample search to Plausible (mathlib community, 2024); the audit pipeline is the contribution, not the primitives.
3. Synthetic validation: a 10-item adversarial corpus where each check has a hand-specified ground-truth verdict. The pipeline matches all 10 (Section 5).
4. A complete public-artifact audit of three SOTA systems (Section 6): static audit of 879 released proofs (DeepSeek-Prover-V2, Kimina-Prover-72B, Goedel-Prover-V2), full kernel-level audit of all 635 Kimina and DeepSeek proofs against each prover's pinned mathlib revision, and SBERT-based alignment audit at threshold $\tau = 0.40$ (pre-registered on the synthetic corpus) with sensitivity analysis over $\tau \in [0.30, 0.50]$. The audit surfaces four substantive empirical findings: (i) all 438 DeepSeek and $186/197$ Kimina proofs lie within the conventional trusted base $\mathcal{T}_0$; the eleven Kimina exceptions extend it to $\mathcal{T}_1$ via `native_decide`, confirmed at the kernel level; (ii) no proof across the 635-item kernel-audited population contains an implicit `sorry` or an extra axiom; (iii) the Goedel-Prover-V2 miniF2F release contains no proof artifacts, only the benchmark input statements; (iv) SBERT alignment scoring places DeepSeek-Prover-V2 miniF2F-test at 15.2% misalignment, within 1.2 percentage points of ReForm authors (2025)'s expert-annotated rate at the pre-registered threshold.
5. A reporting protocol we recommend the maintainers of miniF2F, PutnamBench, and future prover papers adopt (Section 7).

2. Related Work

Existing Lean audit primitives. Several tools in the Lean ecosystem implement subsets of the checks PROOFGATE combines. lean4checker (Lean FRO, 2024b), distributed with Lean 4 itself and documented in the validating-proofs section of the reference manual (Lean FRO, 2024c), provides an external kernel re-verification pass over compiled `.olean` files, catching the same axiom-base and `sorry` violations as our Check 1 with stronger sandboxing guarantees; Comparator (Lean FRO, 2024a), also developed by the Lean FRO, checks declaration-signature equivalence and is the recommended tool for detecting the class of unfaithfulness in which a released proof's declared theorem signature does not match the benchmark's expected one—a category that our source-level Check 2 does not currently catch. Plausible (mathlib community, 2024), distributed with mathlib (The Mathlib Community, 2020), performs randomised counterexample search on Lean propositions and is more powerful than the small fixed-tactic probe used in our Check 3. PROOFGATE *is not intended to replace these tools, and a production audit should orchestrate them as backends rather than reimplement their functionality.* The contribution of this paper is the integration of axiom inspection, source-level linting, vacuity probing, and alignment scoring into a single dimensionless metric and reporting protocol, applied empirically to every public release of three SOTA systems; the underlying audit primitives are deliberately simple and substitutable. We mark Comparator as the natural Check 2 successor and Plausible as the natural Check 3 backend, both as direct integration targets for the next package version.

Lean tooling for neural provers. The closest infrastructural work to PROOFGATE's execution layer is the Kimina Lean Server (MoonshotAI, 2025), which keeps mathlib

loaded in a long-running process and accepts proofs over HTTP, and Pantograph (Aniva et al., 2025), a machine-to-machine elaboration interface designed for tree-search provers. The earlier LeanDojo (Yang et al., 2023) introduced both an extraction toolchain and a retrieval-augmented prover (ReProver) for Lean and is the foundational work in this area. The systems audited here are themselves part of a lineage that includes Draft-Sketch-Prove (Jiang et al., 2023) and the first generation of LLM-based Lean provers such as DeepSeek-Prover-V1 (Xin et al., 2024). All three target *throughput* during proof generation; PROOFGATE targets the orthogonal problem of *audit* after release. The two compose: in production an audit pipeline could speak to a Pantograph- or Kimina-server-style backend rather than spawning a subprocess per file. ProofOptimizer (Piotrowski et al., 2025) addresses a third orthogonal concern, automatically shrinking generated proofs, and is complementary—a shorter proof is easier to audit but a passing FAITHFUL-PASS score is a precondition for shrinking to be meaningful at all.

Benchmark hygiene and proof-quality metrics. Independent of PROOFGATE, multiple groups have audited *statements* rather than proofs: Ospanov et al. (2025) systematically re-audited miniF2F-v2 (with over 300 statement-level corrections, and 16 statements they identify as having no solution in their current form); the Kimina team (Wang et al., 2025) and DSP+ (Reviving DSP authors, 2025) have independently flagged 5–8 miniF2F items as misformalized; Aleph (Logical Intelligence, 2026) reported 15 PutnamBench statements with counterexamples. ReForm authors (2025)'s ReForm proposes ConsistencyCheck, a dataset annotating 16.4% of miniF2F and 38.5% of ProofNet statements as containing semantic errors. Beyond Lean, the BEq family (Liu et al., 2024) of autoformalization equivalence metrics and the process-level error detection of ProcessBench (Zheng et al., 2024) address related quality concerns at the statement and step levels respectively. PROOFGATE's Check 3 (vacuity) and Check 4 (alignment) operationalize the same concerns at audit-time on individual released proofs rather than constructing a dataset.

Critical perspectives on RL-trained reasoners. Our motivation overlaps with that of (Yue et al., 2025), who argue that pass@k improvements from reinforcement learning on reasoning benchmarks may reflect re-weighting of capabilities already present in the base model rather than genuine capability gains. The case for an audit is even more compelling in formal mathematics, in which the kernel is intended to provide independently verifiable evidence: an inflated pass rate that does not withstand third-party kernel re-verification is, by construction, not the figure of merit that it appears to be.

Position of this work. Relative to all of the above, the contribution of PROOFGATE is intentionally narrow: a runnable, reproducible audit tool that combines four orthogonal checks into a single metric, together with the empirical observation (Section 6) that the released artifacts of current state-of-the-art systems span a wide range of outcomes—from full $\mathcal{T}_0$-pass at kernel level (DeepSeek), to compiler-trust extension in 11 of 197 proofs (Kimina), to the absence of any auditable proofs (Goedel)—and that the community presently lacks the measurement instrument required to make these distinctions visible.

3. A Taxonomy of Unfaithful Proofs

We distinguish three failure modes. Let φ denote an informal mathematical statement, $\hat{\varphi}$ its formal Lean statement, and π a candidate proof term of $\hat{\varphi}$.

Definition 3.1 (Faithfulness)**.** A proof π is *faithful* if the Lean 4 kernel accepts π and the only axioms used by π (as reported by `#print axioms`) lie within an agreed trusted base—typically {`propext`, `Classical.choice`, `Quot.sound`}.

Unfaithfulness includes: explicit or implicit `sorry`/`admit`; introduction of fresh `axiom` declarations; reliance on `native_decide` for goals whose underlying decision procedure is unsound for the actual goal type; tactics that succeed only by closing a goal with a hidden metavariable.

Definition 3.2 (Statement alignment)**.** A formal statement $\hat{\varphi}$ is *aligned* with informal statement φ if a competent mathematician, given both, would judge them to have the same truth conditions and the same intended meaning.

Misalignment includes: missing hypotheses; swapped quantifier order; `Real`-vs-`Nat` type mismatches that change the problem; existential statements where the witness is fixed in $\hat{\varphi}$ but free in φ.

Definition 3.3 (Vacuity)**.** A formal statement $\hat{\varphi}$ is *vacuous* if it is trivially true (a tautology of the ambient logic), or refutable via a counterexample to its negation, or has unsatisfiable hypotheses (*ex falso*).

A trivially true statement is provable by `trivial`; an *ex falso* statement is technically provable but uninformative. In either case the resulting proof provides no useful evidence of the prover's reasoning capability.

Proposition 3.4 (Decomposition of reported pass rate)**.** *For a benchmark B and prover P, let $\mathrm{Pass}(P, B)$ denote the fraction of items whose released proof type-checks. Then*

$$\mathrm{Pass} \;=\; \textsc{Faithful-Pass} + U + M + V,$$

where U, M, V are the fractions of unfaithful, misaligned, and vacuous solutions (all functions of P, B), and

FAITHFUL-PASS *is the residual.*

The decomposition is not a theorem about the world; it is a definition that makes the conflation visible. The empirical question is the relative size of each term.

4. The PROOFGATE Audit Pipeline

PROOFGATE implements four orthogonal checks against each released $(\hat{\varphi}_i, \pi_i)$ pair.

4.1. Check 1: Kernel-level axiom inspection

We append `#print axioms NAME` after each released theorem, re-elaborate the file in a fresh Lean process, and parse the resulting line. The trusted base $\mathcal{T}_0$ comprises the three standard Lean 4 axioms `propext`, `Classical.choice`, and `Quot.sound`; any axiom not in $\mathcal{T}_0$ fails the check. We track a second set, $\mathcal{T}_1 := \mathcal{T}_0 \cup \{$`Lean.ofReduceBool`, `Lean.trustCompiler`$\}$, because `native_decide` introduces compiler-trust axioms whose presence depends on the Lean toolchain version.[1] Items whose only extras lie in $\mathcal{T}_1 \setminus \mathcal{T}_0$ are reported as *pass-with-flag*—sound under the assumption that the user trusts the Lean compiler, but a strictly larger trusted base than the conventional one.

The canonical failure mode this catches is the implicit `sorry`, illustrated in Listing 1.

Listing 1. A proof that type-checks but fails Check 1. The `have` introduces a `sorry` that is never used in the final term, yet the kernel nonetheless records `sorryAx` in the proof's axiom set.

```
theorem bad : 2 + 2 = 4 := by
  have h : 0 = 1 := sorry
  exact (by rfl)
-- #print axioms bad
-- 'bad' depends on axioms: [sorryAx]
```

4.2. Check 2: Banned-tactic linter

A purely *syntactic* scan of the released source. Because it is computationally inexpensive and requires no Lean toolchain, this check serves two roles: (i) it can audit proofs whose lake project cannot easily be reconstructed; and (ii) certain failure modes (a fresh `axiom` declaration, an `exact?` call) are visible in the source even before the kernel is invoked.

We flag, with rationales: `sorry`/`admit` (explicit incomplete proof); `apply?`/`exact?` (interactive search tactics that, in editor sessions, surface candidate terms a user may accept without verifying that the underlying metavariable is fully solved; on a released artifact they are unusual and worth surfacing); `native_decide` (widens trusted base to $\mathcal{T}_1$); `decide` on `Real`-typed goals (potentially unsound without a `Decidable` instance); fresh `axiom` declarations anywhere in the source. The check is *conservative rather than adversarial*: occurrences are flagged for review rather than automatically marked as failures, with the exception of hard-failure cases (`sorry`, `apply?`, `exact?`, fresh axioms). On the released artifacts of Section 6, only the `native_decide` signal fires (eleven Kimina items); the `apply?`, `exact?`, and `decide`-on-`Real` signals are dormant. We retain them in the released linter because the population audited in this paper is small (n=635) and not representative of the larger pool of artifacts that future audits will face; a flag that does not fire here may fire on a future prover release. The validation argument is given by the synthetic corpus of Section 5, where each flag has an associated positive case.

4.3. Check 3: Negation-counterexample probe

For each formal statement $\hat{\varphi}$, we attempt to discharge $\neg\hat{\varphi}$ using a fixed-budget combination of `decide`, `omega`, and the standard contradiction tactics. If the probe succeeds and the original proof type-checked, the statement is *vacuous*: either it is trivially false (in which case the original proof is unsound) or its hypotheses are inconsistent (*ex falso*). In either case the item provides no useful evidence of reasoning capability.

This check is intentionally one-sided. False negatives (failing to detect a vacuous statement) are acceptable; false positives (incorrectly flagging a substantive statement as vacuous) are not, since they would mislabel a legitimate proof. In our synthetic validation (Section 5), the probe correctly flags `2 + 2 = 5` as vacuous (its negation is discharged by `decide`) and correctly accepts the universally quantified statement `(n : Nat) : n + 0 = n`.

4.4. Check 4: Alignment score

We compute a similarity score on the $(\varphi, \hat{\varphi})$ pair, using a small SBERT model (`all-MiniLM-L6-v2`) by default; the package exposes a modular backend interface so that users may substitute the released FormalAlign (Lu et al., 2025) checkpoint (trained with a dual loss combining autoformalization and contrastive alignment objectives, and reporting better-calibrated alignment scores than embedding similarity alone) or a local LLM judge via llama.cpp. Items below a calibrated threshold are flagged *misaligned*; borderline items are optionally escalated to a stronger judge (an off-the-shelf instruction-tuned LLM) to resolve disagreements. The package also includes a lightweight token-overlap fall-

[1] In our empirical tests on Lean 4.15.0 (Kimina's pinned toolchain), `native_decide` introduces only `Lean.ofReduceBool`; in earlier toolchains it introduces `Lean.trustCompiler` as well; in Lean $\geq$ 4.29.0 it instead introduces one dedicated axiom per computation, with auto-generated names. We treat $\mathcal{T}_1$ as the closure under any such compiler-trust axiom, regardless of name.

back for environments in which sentence-transformers is not installed.

An empirical observation that motivates the dual-signal design: on the prompt *"For every prime $p > 2$, p is odd,"* paired against a formal statement that omits the $p > 2$ hypothesis, SBERT alone yields a similarity score of 0.545 and reports *aligned*; for comparison, a clearly misaligned baseline (the same prompt paired with an unrelated theorem) yields 0.121. The dropped-hypothesis case is precisely the regime in which embedding similarity alone is insufficient and the LLM-judge escalation is required.

4.5. The FAITHFUL-PASS metric

FAITHFUL-PASS(P, B) is defined as the fraction of items $i \in B$ for which all four checks pass on the released pair $(\hat{\varphi}_i, \pi_i)$. A single failed hard check disqualifies the item. *Pass-with-flag* items (`native_decide`) and lint flags that are not hard failures are counted as passes; the audit report records every flag so that the consumer may re-tighten the criterion as desired. We also write KERNEL-FAITHFUL-PASS$_{\mathcal{T}_k}(P, B)$ for the kernel-only fraction passing Check 1 under trusted base $\mathcal{T}_k$ (used in Section 6); the combined FAITHFUL-PASS requires KERNEL-FAITHFUL-PASS *and* the other three checks.

5. Synthetic Corpus Validation

Before reporting findings on released artifacts, we validate the pipeline against an adversarial corpus constructed for this purpose: 10 Lean files spanning the faithfulness, vacuity, and alignment categories, each annotated with a written ground-truth verdict for every check. The corpus is distributed with the package, and the validation table is regenerated automatically by `python -m proofgate audit-corpus`.

The diagnostically important row is `bad_hidden_sorry`: the proof type-checks and its outer goal is provable by `rfl`, yet an unrelated `have h : 0 = 1 := sorry` earlier in the term contaminates the axiom set. We refer to this failure mode as an *implicit sorry*: it is precisely the regime in which visual inspection of the source by a maintainer can fail to detect what the kernel itself reports.

6. Audit of Released Artifacts

We applied PROOFGATE to every publicly released proof from the three target systems. The audit has two layers: a *static* layer (Check 2 plus the syntactic subset of Check 1) that runs in seconds without a Lean toolchain, and a *kernel* layer (full Check 1 via `#print axioms`) that requires building the prover's pinned mathlib revision in a lake project. We report both. The full audit is reproducible via `scripts/batch_audit.py` and `scripts/repin_mathlib.sh` in the supplementary material.

Table 1. Synthetic-corpus validation. Each row's FAITHFUL-PASS verdict is hand-specified before PROOFGATE is run; an `OK` indicates the pipeline matched the expected verdict on Checks 1 *and* 2 (the kernel- and source-level checks; alignment items intentionally omit `#print axioms`). The pipeline matches all 10 expected verdicts.

Item	Expected	Actual
`good_rfl`	pass / pass	pass / pass
`good_classical`	pass / pass	pass / pass
`bad_sorry`	fail / fail	fail / fail
`bad_hidden_sorry`	fail / fail	fail / fail
`bad_fresh_axiom`	fail / fail	fail / fail
`bad_native_decide`	flag* / flag	flag* / flag
`vacuous_trivial`	pass / pass	pass / pass
`vacuous_ex_falso`	pass / pass	pass / pass
`aligned_simple`	n/a / pass	n/a / pass
`misaligned_missing_hyp`	n/a / fail	n/a / fail

* *pass-with-flag*: trusted under $\mathcal{T}_1$, not under $\mathcal{T}_0$.

Table 2 reports static-audit findings on the 878 files released by the three systems (DeepSeek 438, Kimina 197; Goedel's 244 input templates are excluded from the proof count, for the reasons detailed below).

Kernel-level audit of Kimina-Prover-72B and DeepSeek-Prover-V2. We completed the full kernel-level Check 1 for both Kimina and DeepSeek against each prover's pinned (or, in the case of DeepSeek, closest-tagged) mathlib revision. For Kimina, the pin is the prover's released mathlib commit `9837ca9d` under `leanprover/lean4:v4.15.0`; for DeepSeek, which ships no explicit pin, the mathlib tag `v4.9.0` (commit `f0957a75`) under `leanprover/lean4:v4.9.0` matches the prover's imports. In each case the audit batches all proofs into a single Lean file with one namespace per item, appends `#print axioms` to each, and elaborates the file in a single `lake env lean` invocation. Wall times on a 2023 MacBook Pro M2: 355 s for Kimina miniF2F-test (n=197); 1100 s for DeepSeek miniF2F-test (n=217); 1902 s for DeepSeek miniF2F-valid (n=221). Table 3 reports the resulting KERNEL-FAITHFUL-PASS scores. Across all 635 proofs, zero items have `sorryAx` in their axiom set, and zero items have extra axioms outside $\mathcal{T}_1$. For Kimina, the eleven $\mathcal{T}_1 \setminus \mathcal{T}_0$ items are exactly the `native_decide` items identified by the static audit; for DeepSeek, no items invoke compiler-trust axioms at all. These kernel-level results agree in every cell with the static-audit prediction, providing empirical evidence that the static linter, when paired with kernel-level Check 1, gives a faithful picture of the released artifacts.

Table 2. Static audit of all publicly released proofs from three state-of-the-art Lean 4 provers as of submission. *Hard-failure rate* counts items whose source contains `sorry`, `admit`, `apply?`, `exact?`, or a fresh `axiom` declaration. *Compiler-trust rate* counts uses of `native_decide`, which extend the trusted base to $\mathcal{T}_1$. *Review flags* are conservative same-line co-occurrence flags (for example, `decide` on a line that mentions `Real`); on this artifact set the linter raises no such flags.

Prover	Benchmark	n released	Hard-fail rate	Compiler-trust rate	Review flags
DeepSeek-Prover-V2 (Ren et al., 2025)	miniF2F-test	217	0 / 217 (0.0%)	0 / 217 (0.0%)	0
DeepSeek-Prover-V2	miniF2F-valid	221	0 / 221 (0.0%)	0 / 221 (0.0%)	0
Kimina-Prover-72B (Wang et al., 2025)	miniF2F-test	197	0 / 197 (0.0%)	**11 / 197 (5.6%)**	0
Goedel-Prover-V2 (Lin et al., 2025)	miniF2F-test	*0*[†]	—	—	—

[†] The file `dataset/minif2f.jsonl` in the public Goedel-Prover-V2 repository contains 244 entries whose `lean4_code` field is the benchmark input statement followed by `:= by sorry`. These are not Goedel's released proofs; they are the input benchmark with placeholder proofs. We were unable to locate any publicly released proof artifact corresponding to Goedel-Prover-V2's reported 88–90.4% miniF2F result.

Table 3. Kernel-level Check 1 results. KERNEL-FAITHFUL-PASS$_{\mathcal{T}_0}$ counts items whose axiom dependencies lie within the conventional trusted base {`propext`, `Classical.choice`, `Quot.sound`}; KERNEL-FAITHFUL-PASS$_{\mathcal{T}_1}$ extends the base with the compiler-trust axioms introduced by `native_decide`. These quantities reflect Check 1 only; alignment (Check 4) and vacuity (Check 3) are reported separately. Wall-clock times are reported on a 2023 MacBook Pro M2 (no GPU) following a one-time mathlib cache fetch per pinned revision.

Prover	Benchmark	mathlib pin	KERNEL-FAITHFUL-PASS$_{\mathcal{T}_0}$	KERNEL-FAITHFUL-PASS$_{\mathcal{T}_1}$	Wall (s)
DeepSeek-Prover-V2	miniF2F-test	`v4.9.0`	217 / 217 (100%)	217 / 217 (100%)	1100
DeepSeek-Prover-V2	miniF2F-valid	`v4.9.0`	221 / 221 (100%)	221 / 221 (100%)	1902
Kimina-Prover-72B	miniF2F-test	`9837ca9d`	186 / 197 (94.4%)	197 / 197 (100%)	355

Finding 1: Kimina-Prover-72B extends the trusted base on 11 items. Eleven of 197 released Kimina proofs use `native_decide`: `amc12a_2020_p4`, `mathd_algebra_170`, and nine items in the `mathd_numbertheory_*` family. `native_decide` yields a sound proof *conditional on trusting the Lean compiler*, and introduces `Lean.ofReduceBool` (and, on certain toolchain versions, `Lean.trustCompiler`) into the axiom set. Under the conventional trusted base $\mathcal{T}_0$, KERNEL-FAITHFUL-PASS for Kimina on miniF2F-test is $186/197 = 94.4\%$; under the extended base $\mathcal{T}_1$, it is $197/197 = 100\%$. The recommendation that future prover papers report *which* trusted base they assume (Section 7) is motivated precisely by this 5.6% gap.

Finding 2: Goedel-Prover-V2 releases no auditable miniF2F proofs. Inspection of `dataset/minif2f.jsonl` in the Goedel-Prover-V2 repository (commit `2e9036e`) reveals that all 244 `lean4_code` fields terminate in `:= by sorry`. The file constitutes the input benchmark rather than Goedel's proofs. We additionally examined all HuggingFace dataset repositories under the `Goedel-LM` organization (`Lean-workbook-proofs`, `Goedel-Pset-v1`, `MathOlympiadBench`): `Lean-workbook-proofs` contains training data from Goedel-Prover-V1, and `MathOlympiadBench` contains benchmark statements with `sorry` placeholders. The reported 90.4% self-correction-mode result on miniF2F is therefore not currently reproducible from the public release: a third party can re-execute the published model, but cannot re-verify the specific proofs reported as successful.[2] We emphasize that this is a property of the *release* rather than of the underlying research; we have not investigated whether the proofs exist privately. The practical consequence is that auditing this system requires re-executing its open-weights model rather than re-verifying released artifacts, which is a substantially more expensive operation.

Finding 3: DeepSeek-Prover-V2 passes the kernel audit on all 438 items. None of the 438 DeepSeek proofs trigger a hard-failure check, use `native_decide`, or raise a review flag in the static audit. The kernel-level Check 1 against mathlib `v4.9.0` confirms this: every one of the 217 miniF2F-test and 221 miniF2F-valid proofs has an axiom set contained in $\mathcal{T}_0$, yielding KERNEL-FAITHFUL-PASS$_{\mathcal{T}_0}$(DeepSeek) $= 438/438 = 100\%$ across both splits. Manual inspection further confirms that DeepSeek's proofs do contain occasional uses of `(try decide)`—a construct whose behavior is sound in all cases—which the linter correctly does not flag, since the enclosing line does not mention `Real` (the unsoundness-risk heuristic). DeepSeek's released artifacts are, by every check PROOFGATE applies, the cleanest of the three audited

[2] The Goedel-Prover-V2 paper reports its PutnamBench result as "pass@184" in the arXiv abstract (Lin et al., 2025) and as "Pass@192" in the GitHub README; we adopt the arXiv number. This discrepancy is itself a minor instance of the type of inconsistency the reporting protocol of Section 7 is designed to surface.

systems.

Finding 4: SBERT-based alignment scoring recovers ReForm's miniF2F rate at the pre-registered threshold. We applied Check 4 (SBERT alignment scoring) to every released item carrying both an informal statement and a formal counterpart. The threshold $\tau = 0.40$ was fixed on the synthetic alignment corpus of Section 5 prior to running on any released artifact: it is the largest round value below the synthetic `aligned_simple` score (0.428), placing the synthetic aligned/misaligned cases on opposite sides. At this pre-registered threshold, the reported misalignment rates are DeepSeek miniF2F-test $33/217 = 15.2\%$, DeepSeek miniF2F-valid $45/221 = 20.4\%$, Kimina miniF2F-test $25/197 = 12.7\%$, and Goedel miniF2F-test $2/244 = 0.8\%$. The DeepSeek miniF2F-test figure falls within 1.2 percentage points of ReForm authors (2025)'s expert-annotated rate of 16.4% on the same benchmark.

Table 4 reports sensitivity over the threshold range $[0.30, 0.50]$. The rates vary substantially with τ, and the precise convergence with ReForm holds only in a narrow band around $\tau = 0.40$. The most defensible reading is qualitative: SBERT recovers a misalignment population whose size is within an order of magnitude of expert annotation, with the precise rate dependent on threshold choice. The Goedel result is an outlier in the opposite direction: their `informal_prefix` field contains structured multi-sentence problem statements rather than the terse one-line miniF2F docstrings used by DeepSeek, which inflates embedding similarity for reasons unrelated to formal alignment. We emphasize that per-item verdicts are advisory: as the dropped-hypothesis example of Section 4.4 demonstrates, SBERT alone misses subtle structural misalignments, and the reported rates should be interpreted as lower bounds pending the LLM-judge escalation pass and a stronger backend such as the released FormalAlign (Lu et al., 2025) checkpoint.

7. Reporting Protocol

We recommend that future prover papers report, alongside any pass@k number:

1. The axiom set used (output of `#print axioms`) for every released proof, summarized as a count over $\mathcal{T}_0$, $\mathcal{T}_1 \setminus \mathcal{T}_0$, and "other".
2. A per-tactic count of `native_decide`, `apply?`, `exact?`, and `decide` occurrences across released proofs.
3. The benchmark revision (commit hash) audited against, since miniF2F (Zheng et al., 2022; Ospanov et al., 2025) and PutnamBench (Tsoukalas et al., 2024) are now actively maintained.
4. A FAITHFUL-PASS score computed by the released PROOFGATE tool, alongside the reported pass rate.
5. The location of the artifact in the release *distinguishing proofs from input benchmarks*—a requirement whose necessity became apparent only upon auditing Goedel-Prover-V2.

8. Discussion and Limitations

Scope of our claims. We do not assert that any team has misrepresented its work. The three audited papers are technically careful, and at least one (Kimina) has actively contributed back to benchmark hygiene (Wang et al., 2025). PROOFGATE measures *community-usable faithfulness*—a property of released artifacts together with the tooling that consumes them, not of the underlying papers.

The audit is conservative by design. FAITHFUL-PASS is intentionally strict; some items it flags may be defensible upon closer inspection. We release per-item flags so that authors may contest individual determinations. The cost asymmetry is deliberate: a false-positive review flag imposes a small inspection cost on a maintainer, whereas a false negative on a hidden `sorry` undermines community trust in the metric itself. A non-trivial implication of this asymmetry, encountered during development, is that the audit pipeline must itself be subject to testing: our linter initially contained a comment-stripping bug that produced spurious `decide-on-real` flags on two DeepSeek items; the bug was detected only because the synthetic-corpus validation of Section 5 requires every claim to be accompanied by a ground-truth verdict. We therefore recommend that any audit pipeline include a synthetic test corpus and report results only after the pipeline matches expected outcomes on that corpus.

Failure-mode overlap and decomposition. The taxonomy of Section 3 treats faithfulness, alignment, and vacuity as conceptually distinct, but real items can fail multiple checks simultaneously: a vacuous statement can be discharged by an unfaithful proof, and a misaligned statement can be both formally trivial and informally substantive. PROOFGATE's per-item report records every check's verdict independently; the aggregate FAITHFUL-PASS metric is conservative in the sense that any failure disqualifies the item. The per-prover summaries in Tables 2–4 decompose along orthogonal sub-counts so that overlapping cases are double-counted in the diagnostic columns but appear exactly once in the headline FAITHFUL-PASS number. The proposition of Theorem 3.4 is an additive equality only over a partition of the unfaithful population; the diagnostic columns are not partitions.

Table 4. Check 4 misalignment rate as a function of the SBERT threshold τ. Boldface marks the pre-registered $\tau = 0.40$. ReForm reports 16.4% expert-annotated misalignment on miniF2F. The DeepSeek-test row matches ReForm to within 1.2 percentage points only at $\tau = 0.40$; the rate doubles by $\tau = 0.45$, confirming that the precise convergence is partly an artifact of threshold choice. The most defensible reading of the SBERT signal is qualitative.

	τ=0.30	τ=0.35	$\boldsymbol{\tau}$**=0.40**	τ=0.45	τ=0.50
DeepSeek-Prover-V2 miniF2F-test	5.1%	10.1%	**15.2%**	25.8%	36.9%
DeepSeek-Prover-V2 miniF2F-valid	5.0%	9.0%	**20.4%**	29.4%	43.9%
Kimina-Prover-72B miniF2F-test	2.5%	6.1%	**12.7%**	23.4%	42.1%
Goedel-Prover-V2 miniF2F-test	0.0%	0.0%	**0.8%**	1.6%	4.5%

Faithfulness gaps the current implementation does not catch. Our Check 1 audits the axiom set of the released theorem by name; it does not verify that the released theorem's signature matches the benchmark's expected signature (a class addressed by Comparator (Lean FRO, 2024a)), nor does it detect metaprogramming injections that bypass the kernel's tracking (a class addressed by lean4checker's external sandbox (Lean FRO, 2024b)), nor does it detect re-definitions of existing axioms under the same name. A production-grade audit should delegate Check 1 to lean4checker and add a Comparator pass; we treat this as an integration task. The empirical findings of Section 6 are unchanged under these stronger checks because the released artifacts we audit pass the strictest variant of each check we implemented—no item has any extra axiom under our pipeline, so no item could acquire one under a stricter pipeline.

Scope of artifacts audited. We audit only released proofs. Pass rates that depend on closed-source rollouts (e.g., the TTRL search of Kimina-Prover-72B at 92.2%, or Goedel-Prover-V2's self-correction mode) can be audited only insofar as proofs from those rollouts are released. The Goedel finding above represents the most extreme case of this constraint.

Limitations of the alignment check. As the dropped-hypothesis example in Section 4.4 illustrates and the threshold sensitivity in Table 4 confirms, off-the-shelf SBERT embeddings cannot reliably detect subtle structural misalignments, and the reported population-level misalignment rate is sensitive to the threshold choice. We therefore treat Check 4 as a *secondary signal with judge escalation* rather than as a primary component of the metric: the audit report distinguishes "misaligned (embedding and judge agree)" from "misaligned (embedding only)". We did not benchmark SBERT against the released FormalAlign (Lu et al., 2025) checkpoint or against the BEq family (Liu et al., 2024); such a comparison is a natural next step. An exact reproduction of the ReForm (ReForm authors, 2025) numbers would require access to their ConsistencyCheck dataset, which is not currently publicly available.

Vacuity check coverage. Our Check 3 uses a fixed small tactic budget (`decide`, `omega`, contradiction) and will miss vacuous statements whose negation requires more search, including statements vacuous only after substantial unfolding of definitions. We have not estimated the false-negative rate of this check against an independent vacuity-labelled corpus; an integration with Plausible (mathlib community, 2024) would yield substantially better recall via randomised counterexample search. On our 635-item kernel-audited population, Check 3 raises zero vacuity flags; in light of the limited recall, this should be read as a lower bound rather than a strong negative result.

Extension to other proof assistants. The methodology extends to Rocq and Isabelle via analogous kernel-inspection primitives; we leave this extension to future work. The taxonomy of Section 3 is system-agnostic.

Extension to other benchmarks. PutnamBench is the natural subsequent benchmark, and the PROOFGATE pipeline is benchmark-agnostic. The principal constraint is artifact availability: *no* prover, at the time of submission, has released PutnamBench proofs, despite three of them reporting PutnamBench results. The practical consequence is that the audit story for PutnamBench is currently constrained by artifact release practices rather than by tooling capability.

Reproducibility

The PROOFGATE package, the synthetic corpus, the downloader scripts, and the per-item audit logs are released under an MIT license at `https://github.com/edisonly1/proofgate`. The static audit reported in Table 2 reproduces end-to-end in under one minute on consumer hardware (no GPU; no paid API). The kernel-level Kimina audit reproduces in approximately six minutes on the same hardware after a one-time mathlib cache fetch ($\sim$5 minutes); the DeepSeek kernel audit reproduces in approximately 50 minutes total (test plus valid) after a separate one-time mathlib cache fetch for the v4.9.0 pin. The SBERT-based alignment audit of all 879 items completes in approximately 15 seconds. A combined per-prover report that joins the static, kernel, and alignment results on `problem_id` is

produced by `scripts/combined_report.py`.

Conclusion

The AI-for-Math community has produced systems of remarkable capability. What it currently lacks is the measurement instrument required to determine which of their reported successes withstand third-party scrutiny. PROOFGATE constitutes an initial implementation of such an instrument. The workshop's organizing question—how self-evolving scientific agents may be made *verifiable, reliable, and scientifically useful*—is precisely the question this work seeks to address.

Impact Statement

This paper presents a measurement tool for evaluating publicly released theorem-proving artifacts. Its purpose is to increase the community's ability to verify the claims of automated mathematical reasoning systems; we anticipate no harms beyond reputational ones to systems that fail audits, and we have deliberately reported findings in a form that distinguishes properties of the release from properties of the underlying research.

References

Aniva, L. et al. Pantograph: A machine-to-machine interaction interface for advanced theorem proving in Lean 4. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2025.

Azerbayev, Z. et al. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. In *Workshop on MATH-AI*, 2023.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. https://lean-lang.org, 2021.

Google DeepMind. AI achieves silver-medal standard solving international mathematical olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, 2024.

Jiang, A. Q., Welleck, S., Zhou, J. P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., and Lample, G. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations (ICLR)*, 2023.

Lean FRO. Comparator: Strict signature matching for Lean 4 declarations. https://github.com/leanprover/comparator, 2024a.

Lean FRO. lean4checker: External Lean 4 proof checker. https://github.com/leanprover/lean4checker, 2024b.

Lean FRO. The Lean 4 reference manual: Validating proofs. https://lean-lang.org/doc/reference/latest/ValidatingProofs/, 2024c.

Lin, Y. et al. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction, 2025. Repository at https://github.com/Goedel-LM/Goedel-Prover-V2, commit 2e9036e.

Liu, Q. et al. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach, 2024. BEq family of autoformalization equivalence metrics.

Logical Intelligence. Aleph AI solves 99.4% of PutnamBench, topping leaderboard. Blog post, January 2026.

Lu, J., Wan, Y., Huang, Y., Xiong, J., Liu, Z., and Guo, Z. FormalAlign: Automated alignment evaluation for autoformalization. In *International Conference on Learning Representations (ICLR)*, 2025.

mathlib community. Plausible: Randomised counterexample search for Lean 4. https://github.com/leanprover-community/plausible, 2024.

MoonshotAI. Kimina Lean Server. https://github.com/MoonshotAI/kimina-lean-server, 2025.

Ospanov, R., Farnia, F., and Yousefzadeh, R. Limitations of miniF2F and a revised benchmark for olympiad-level lean proofs, 2025. Dataset at https://github.com/roozbeh-mohit/miniF2F_v2.

Piotrowski, B. et al. ProofOptimizer: Training language models to simplify proofs without human demonstrations, 2025.

ReForm authors. ReForm: Reflective autoformalization with prospective bounded sequence optimization, 2025.

Ren, Z. et al. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. Released artifacts at https://github.com/deepseek-ai/DeepSeek-Prover-V2, commit e598a57.

Reviving DSP authors. Reviving DSP for advanced theorem proving in the era of reasoning models, 2025.

The Mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2020.

Tsoukalas, G. et al. PutnamBench: A multilingual competition-mathematics benchmark for formal theorem-proving. In *NeurIPS Datasets and Benchmarks*, 2024.

Wang, H. et al. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning, 2025. Released artifacts at https://github.com/MoonshotAI/Kimina-Prover-Preview, commit 7abd61a.

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale synthetic data, 2024.

Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks*, 2023.

Yue, Y. et al. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model?, 2025.

Zheng, C. et al. ProcessBench: Identifying process errors in mathematical reasoning, 2024.

Zheng, K., Han, J. M., and Polu, S. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022.