Counterexample-Carrying FormalRx: Lean-Checkable Witnesses for Autoformalization Misalignment

Joy Zheyun Yang [*1] **Socrates Osorio** [*2]

Abstract

Autoformalization systems can write Lean statements that compile but mistranslate the original mathematics. Existing diagnostic benchmarks like FormalRx ask the model to label and repair the mismatch, but a correction alone does not *prove* that the candidate and intended formalizations differ in meaning. Lean checks the proposition it sees, not whether the English was captured. We propose Counterexample-Carrying FormalRx (FormalRx-CC): every misalignment diagnosis must be accompanied by a small Lean-checkable artifact that mechanically separates the candidate from the intended formalization. We give a task definition, a seven-family witness taxonomy, an adversarial verification protocol that rejects `sorry`, `admit`, and escape hatches, and a 204-example executable artifact with completed 204-example runs on GPT-4.1-mini, GPT-4.1, and Claude Haiku plus one-pass Lean-feedback repair. The results expose a distinction that single-number scoring hides. Lean-feedback repair lifts raw Lean acceptance by 23–61% across the three models, but the joint metric that additionally requires the correct semantic direction is only 13/204 in the strict GPT-4.1-mini funnel: repair fixes proof engineering, not semantic understanding. In the GPT-4.1-mini comparison, full mode passes Lean more often than gold-target mode: 33/204 raw and 48/204 after one repair pass, versus 21/204 in gold mode, suggesting that raw Lean success can conflate witness construction with self-target selection.

[*]Equal contribution [1]University of Oxford, Oxford, United Kingdom [2]University of California, Berkeley, CA, USA. Correspondence to: Joy Zheyun Yang <joy@robots.ox.ac.uk>.

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

1. Introduction

An autoformalization system can write a Lean statement that compiles and yet means the wrong thing. If the human meant "f is strictly monotone" but the model produces `Monotone f`, every constant function satisfies the model's statement. If the human meant $(xy)^2$ but the model writes xy^2, the two expressions already disagree at x=2, y=3. A proof assistant certifies the proposition it sees, not whether the English was captured. The intended formal target has to be trusted independently.

The AI4Math 2026 FormalRx challenge makes this problem concrete. FormalRx-Test contains 7,030 natural-language/Lean 4 statement pairs, including 5,239 misaligned examples, annotated under a 28-category SCI error taxonomy and designed around verdict, categorization, localization, and correction tasks (LARK Lab, 2026; AI4Math Workshop Organizers, 2026). This is a major step beyond opaque binary matching. However, a correction-only diagnosis can still be epistemically weak: it says that the candidate statement should be changed, but does not itself prove that the candidate and intended statements differ in meaning.

We propose *Counterexample-Carrying FormalRx* (FormalRx-CC): a task and benchmark design in which every misalignment diagnosis must carry an executable semantic witness. For the `Monotone` vs. `StrictMono` case above, the system should exhibit a constant function and prove in Lean that it is monotone but not strictly so. For the $(xy)^2$ vs. xy^2 case, it should pick a concrete assignment and prove the expressions differ at that point.

This framing changes the object being evaluated. Standard semantic alignment evaluation asks whether a model can identify and repair a mismatch. FormalRx-CC asks whether the model can also produce a checkable reason that the repair matters. Operationally, the task has three layers: say what went wrong, propose or receive an intended formal target, and provide a small Lean witness showing that the original and intended targets differ at a concrete boundary case. The last layer is the new one. It turns the claim "these two formalizations differ" into an auditable artifact. It does not, by itself, prove that the intended formalization captures

the English.

This distinction is useful because it separates two sources of uncertainty that are otherwise easy to conflate. The first is the natural-language alignment question: what did the problem statement mean? The second is the formal-separation question: once a candidate and intended Lean object are fixed, can we mechanically audit that they differ? FormalRx-CC targets the second question. This is a smaller claim than full semantic faithfulness, but it is a claim that a proof assistant can check and that a reviewer can inspect locally. A failed or ill-typed witness also gives a concrete failure mode: the model may have chosen the wrong semantic relation, produced an unusable correction target, or understood the mismatch but failed to write a Lean proof.

Our contributions are empirical as well as definitional:

- a task definition for counterexample-carrying semantic alignment between candidate and intended Lean formalizations
- a taxonomy of witness families for common autoformalization errors
- a Lean-oriented verification protocol that rejects `sorry`, `admit`, blocked proof escapes, and statement mutation, and optionally performs axiom inspection
- a 204-example executable stress artifact with 25 synthetic seed examples, 26 FormalRx-derived development examples, and 153 expanded coverage examples spanning all taxonomy families
- a reproducible evaluation pipeline with oracle checks, completed 204-example model-facing runs for GPT-4.1-mini, GPT-4.1, and Claude Haiku, joint semantic-proof metrics, one-pass Lean-feedback repair ablations, raw model outputs, Lean logs, score tables, prompts, and scripts.

The anonymized artifact is available at https://anonymous.4open.science/r/ccf-F4C7/.

Scope. FormalRx-CC audits the formal-separation step. It does not certify English-to-Lean faithfulness directly. The certificate is most informative when the intended formalization is independently trusted, the witness is theorem-level, and the diagnostic labels agree with gold. The 204-example artifact is an executable stress artifact for benchmark-construction validation and model-facing smoke runs, not a leaderboard. A frozen hidden test split, independent inter-annotator reliability, a matched head-to-head BEq+ comparison (Poiroux et al., 2025), and specialized-prover baselines (DeepSeek-Prover, Goedel-Prover, Kimina-Prover (Xin et al., 2024; Ren et al., 2025; Lin et al., 2025a;b; Wang et al., 2025)) are the natural next versions, not claims of this paper.

2. Background and Related Work

Lean and formal mathematics benchmarks. FormalRx-CC is implemented in Lean 4 (Moura & Ullrich, 2021) and Mathlib (The mathlib Community, 2020). It follows a line of formal-mathematics benchmarks that make machine-checkable correctness central, including miniF2F for Olympiad-level formal theorem statements (Zheng et al., 2022), ProofNet for undergraduate-level autoformalization and proof generation (Azerbayev et al., 2023), and LeanDojo for retrieval-augmented theorem proving over Lean repositories (Yang et al., 2023). Neural theorem proving has progressed from language-model proof search (Polu & Sutskever, 2020; Han et al., 2022) to retrieval-augmented and compiler-feedback systems (Yang et al., 2023; Thakur et al., 2023). Recent Lean-specialized provers such as DeepSeek-Prover, DeepSeek-Prover-V2, Goedel-Prover, Goedel-Prover-V2, and Kimina-Prover further show that Lean feedback and large synthetic corpora can drive formal proof generation (Xin et al., 2024; Ren et al., 2025; Lin et al., 2025a;b; Wang et al., 2025). Our focus is complementary: rather than proving the target theorem, we ask for an executable witness that a candidate formalization encodes the wrong target.

Autoformalization and semantic evaluation. LLM autoformalization has been studied in Isabelle/HOL and Lean settings (Wu et al., 2022; Jiang et al., 2023b;a). These efforts reinforce that type checking is necessary but insufficient: a well-typed formal statement can still be semantically wrong. FormalAlign studies automated natural/formal alignment evaluation (Lu et al., 2025). BEq+ evaluates statement autoformalization by attempting to establish formal equivalence and correlates better with human judgment than surface metrics (Poiroux et al., 2025). BEq+ and FormalRx-CC answer complementary questions and are designed to stack. BEq+ proves equivalence (or fails to). FormalRx-CC requires a typed object that explains *how* two non-equivalent statements differ. A failed BEq+ proof is an inconclusive negative: it cannot tell a reviewer whether the gap is a constant, a quantifier swap, a domain change, or a boundary case. FormalRx-CC's witness families turn that failure into a localized, inspectable artifact: a numeric instance, expression delta, property boundary, or finite countermodel that Lean checks directly.

Diagnostic autoformalization evaluation. FormalRx reframes semantic alignment as a diagnostic task: decide whether a natural-language/Lean pair is aligned, identify an error category, localize the error, and provide a corrected statement (LARK Lab, 2026). FormalRx-CC is intended as a compatible extension. It does not replace verdicts, categories, locations, or corrections. It only adds an executable witness field that can be checked independently.

Counterexamples and model finding. Counterexamples are a standard way to explain mathematical boundaries. Counterexample-focused LLM benchmarks test whether models can reason from examples that falsify claims (Li et al., 2025). Proof assistants have also long used model finders, such as Nitpick for Isabelle/HOL (Blanchette & Nipkow, 2010) and Nunchaku-style higher-order counterexample generation (Cruanes & Blanchette, 2016). FormalRx-CC differs in its target: the goal is not merely to disprove an isolated mathematical statement, but to exhibit a semantic delta between a candidate autoformalization and an intended one.

3. Task Definition

Each example contains a natural-language statement N, a candidate Lean statement C, and an intended Lean statement I. In gold-correction mode, I is provided by the benchmark. In end-to-end mode, the system must produce a corrected statement, which is then evaluated against gold or accepted alternatives.

The intended use is closer to a proof-carrying diagnostic than to ordinary answer generation. A model should not merely say "the inequality should be strict" or paste a corrected theorem. It should produce a small artifact that would convince a Lean checker that the difference can matter. For a weakened inequality, that artifact may be a boundary value where $x \leq y$ holds but $x < y$ fails. For a quantifier swap, it may be a finite relation in which every input has some witness but no single witness works uniformly. For a local expression error, it may be a concrete assignment under which the two extracted expressions evaluate differently.

The model output extends a FormalRx-style schema:

```
{
  "aligned": "Misaligned",
  "error_type": "...",
  "error_location": "...",
  "corrected_statement": "...",
  "semantic_relation": "candidate_weaker",
  "witness_type": "property_boundary",
  "witness_lean": "...",
  "witness_explanation": "..."
}
```

The semantic relation records the direction of mismatch. The core labels are:

- `candidate_weaker`: the intended statement implies the candidate, but not conversely.
- `candidate_stronger`: the candidate implies the intended, but not conversely.
- `candidate_incomparable`: neither direction holds.
- `candidate_vacuous`: the candidate has impossible or overly strong premises.
- `local_delta`: the whole propositions may both be true, but an extracted relation, expression, or predicate differs.

The final case is important. In mathematics, two different closed theorem statements can both be true, so $C \wedge \neg I$ may be impossible even when C mistranslates N. FormalRx-CC therefore accepts local witnesses over altered subexpressions, predicates, domains, or quantified components when theorem-level separation is not the right target. The local component is benchmark-supplied: the harness provides `Candidate` and `Intended` definitions for the extracted expression or predicate, and the witness must refer to those names. A model is not allowed to introduce its own separated pair. For example, a fake witness that declares `def Candidate : Prop := True` and `def Intended : Prop := False` would be rejected as statement mutation rather than scored as a valid local delta.

The relation labels are therefore not intended as model-ranking trivia. They specify which proof obligation the witness is supposed to support. If the gold relation is `candidate_weaker`, a good witness should exhibit behavior permitted by the candidate but ruled out by the intended statement. If the model instead supplies a compiling proof for an unrelated `local_delta`, Lean success alone is not enough. This motivates the joint metrics reported with the model results: they count a witness only when the executable artifact and the semantic-relation label agree with the benchmark target.

4. Witness Taxonomy

Table 1 summarizes the initial witness families. A single generic "counterexample" field is too coarse for autoformalization errors. Some mismatches are best witnessed by numeric instantiation. Others require a function, finite relation, contradictory-premise proof, or local expression inequality.

Property-boundary witness. For the common error `Monotone` vs. `StrictMono`, a witness can be:

```
def f : Real -> Real := fun _ => 0

example : Monotone f := by
  intro x y hxy
  simp [f]

example : ¬ StrictMono f := by
  intro h
  have h01 : f 0 < f 1 := h (by norm_num)
  simp [f] at h01
```

The artifact is small, executable, and explains why the can-

Table 1. Witness families for counterexample-carrying autoformalization diagnostics.

Witness type	Typical error	Example certificate
Concrete instance	weak vs. strict inequality	$0 \leq 0 \wedge \neg(0 < 0)$
Expression delta	precedence, constants, algebra	$((2 \cdot 3)^2) \neq 2 \cdot 3^2$
Property boundary	nearby definitions	constant function is monotone, not strictly monotone
Quantifier countermodel	$\forall\exists$ vs. $\exists\forall$	equality relation over `Bool`
Vacuity certificate	contradictory premises	prove no x satisfies $x < x$
Finite model	relational or algebraic mismatch	small graph/order/model separates statements
Structural witness	wrong projection/index/component	local proof that extracted terms differ

didate property is weaker.

Expression-delta witness. For an operator-precedence error:

```
example : ((2 : Real) * 3) ^ 2
    <> (2 : Real) * 3 ^ 2 := by
  norm_num
```

This proves that the altered expression is not merely syntactically different but semantically different at concrete values.

Quantifier witness. A frequent autoformalization error swaps the order of a universal and an existential quantifier. The candidate $\exists y, \forall x, R(x, y)$ is strictly stronger than the intended $\forall x, \exists y, R(x, y)$, so a single finite relation separates them. Taking $R(x, y)$ to be equality over `Bool` works: every x has a matching y, but no single y equals both Boolean values. The witness proves the intended statement and refutes the candidate in the same small file:

```
def Candidate : Prop :=
  exists y : Bool, forall x : Bool, x = y
def Intended : Prop :=
  forall x : Bool, exists y : Bool, x = y

example : Intended := fun x => ⟨x, rfl⟩
example : ¬ Candidate := by
  rintro ⟨y, hy⟩
  cases y
  · simpa using hy true
  · simpa using hy false
```

Because `Bool` is finite, the refutation is a short case split rather than an appeal to a deeper theorem. This keeps the witness inside the trusted base the verifier can check directly.

Vacuity witness. Some mistranslations weaken or corrupt a premise until it can never be satisfied. The right object is then not a counterexample to the conclusion but a proof that the candidate premise is unsatisfiable. For a premise that was corrupted from $x \leq x$ to $x < x$, the witness shows that no real number meets it:

```
def CandidatePremise (x : Real) : Prop := x < x

example : ¬ (exists x : Real, CandidatePremise x) :
  rintro ⟨x, hx⟩
  exact lt_irrefl x hx
```

A vacuity certificate is a different proof obligation from a concrete instance, which is why the taxonomy keeps it as a separate family.

Structural witness. When the error is a wrong projection, index, or component rather than a whole-proposition mismatch, the witness is a local inequality between the two extracted terms. For a candidate that reads the first component of a pair where the intended formalization reads the second, a single concrete pair separates them:

```
def Candidate (p : Prod Nat Nat) : Nat := p.1
def Intended (p : Prod Nat Nat) : Nat := p.2

example : Candidate (1, 2) ≠ Intended (1, 2) := by
  norm_num [Candidate, Intended]
```

Here both formalizations may yield true theorems in context, so theorem-level separation is unavailable. The structural witness instead pins down the exact subterm where meaning diverged.

5. Verification Protocol

A verifier constructs a Lean file containing the allowed imports, normalized candidate and intended components, and the model's witness code. Depending on the example, the witness may be theorem-level, instance-level, expression-level, relational, or vacuity-based.

The verifier has two roles. First, it is a compiler harness: it renders the candidate, intended target, and witness into one Lean file and checks whether Lean accepts the result. Second, it is an adversarial filter: it rejects common ways a generated proof can appear successful without certifying the requested boundary. In particular, the model is not trusted

to define the comparison objects, choose arbitrary imports, or leave proof obligations behind using escape hatches.

The verifier should enforce:

- successful Lean compilation under the target Mathlib version
- no `sorry`, `admit`, `axiom`, or `unsafe` escape hatches
- no mutation of benchmark-provided candidate or intended definitions
- allowed imports only
- optional axiom inspection to reject `sorryAx` or unauthorized dependencies
- compiled witness code that refers to the benchmark-provided candidate or intended component.

We distinguish executable witness code from full semantic correctness. A witness may compile while the model's diagnostic labels are wrong, so Lean success is not by itself the headline semantic metric. The main evaluation adds gold relation agreement, and the appendix reports an even stricter relation-plus-witness-family metric. A future benchmark version should synthesize a canonical per-example theorem name and require the model to fill exactly that proof.

This design deliberately keeps the trusted base small. The benchmark may provide normalized `Candidate` and `Intended` definitions, and the checker may provide boilerplate imports, but the generated witness is evaluated in that fixed context. For local-delta cases, the benchmark extraction is part of the annotation: the model may prove that the extracted expressions differ, but it may not choose a new pair of expressions after seeing which one is easier to separate.

6. Benchmark Construction

A practical workshop-scale benchmark should start with a curated subset rather than attempting to witness all FormalRx examples. We propose four phases:

Selection. Sample 100–200 misaligned FormalRx-style examples from categories likely to admit compact witnesses: relation errors, wrong constants, expression corruption, missing constraints, quantifier changes, domain/type changes, and weakened or strengthened properties.

Normalization. For each example, define the candidate and intended proposition or the altered local component. Normalize names and parameters so witnesses can refer to stable objects without modifying the original statements.

Gold witness authoring. Annotators produce at least one Lean-checkable witness and a one-sentence explanation. Witnesses are labeled by type, relation direction, difficulty, and whether they are theorem-level or local.

Splits. The first public artifact exposes seed and FormalRx-derived development examples for debugging, plus a 153-example expanded stress split for coverage validation. A leaderboard version should keep a hidden test split fixed before prompt design. In this paper, the combined 204-example artifact is public and used for executable coverage validation plus the completed model-facing smoke run.

7. Evaluation

The FormalRx-CC scoring framework supports both standard diagnostic metrics and witness-specific metrics. Diagnostic metrics include verdict, category, localization, and correction scores. Witness metrics include compile rate, no-sorry compile rate, direction accuracy, verified separation rate, witness-type accuracy, and repair iterations under Lean feedback. In this pilot, the main tables report exact counts because the model-facing evidence is one 204-example expanded smoke run rather than a hidden leaderboard. Because a Lean-verified witness can still accompany a wrong semantic label, we also report a joint metric: the semantic-relation label must match gold and the rendered witness must pass Lean without static escape hatches. The stricter appendix metric additionally requires the witness-family label to match gold. These label-based counts are provisional: the current gold relation and witness-family labels are author-reviewed but not independently double-annotated.

We evaluate three questions. First, does the artifact itself have executable gold witnesses? Second, if the intended correction is supplied, can a model produce a Lean-checkable separating witness? Third, in the end-to-end setting, can a model produce both a corrected statement and a witness that still checks when rendered against that correction? The second and third questions are reported separately because they isolate witness construction from correction generation.

The resulting counts should be read as a diagnostic funnel rather than a leaderboard. *Static* checks ask whether the output has the right shape and avoids obvious proof escapes. *Rendered* checks ask whether the prediction can be placed into the benchmark's Lean template. *Lean_raw* asks whether the file compiles. *Lean_noesc* applies the stricter no-escape/reference filter. *Joint* then asks whether the compiling witness supports the correct semantic direction. A model can do well on an early stage and fail later for different reasons: formatting noise, missing imports, proof-engineering mistakes, wrong relation labels, or a self-generated correction that is easy to separate from but not faithful to the gold target.

Table 2 reports the current executable artifact status. Static checks enforce schema and forbidden-token constraints.

Table 2. Artifact verification results. These are gold witness checks, not model-generation baselines.

Split	Ex.	Static	Lean
Synthetic seed	25	25/25	25/25
FormalRx dev	26	26/26	26/26
Expanded stress	153	153/153	153/153
Combined expanded	204	204/204	204/204

Lean checks were run on a Slurm compute node with Lean 4.30.0-rc2 and Mathlib via direct `lean` invocations. The seed split contains compact synthetic witnesses spanning concrete instances, expression deltas, property boundaries, quantifier countermodels, vacuity certificates, and structural witnesses. The development split contains 26 FormalRx-derived semantic witnesses from public FormalRx-Test examples. The expanded stress split quadruples the artifact scale and adds finite-model cases that were absent from the seed/development artifact. These numbers should be read as benchmark-construction validation. The central model-evaluation hypothesis remains that systems will classify or repair many errors that they cannot yet support with valid Lean witnesses and correct semantic metadata.

Two patterns in Table 3 drive the rest of the analysis. *First, choosing one's own correction target makes Lean easier.* GPT-4.1-mini accepts only 21/204 witnesses when given the gold intended target (*Gold* mode), but reaches 33/204 raw, and 48/204 with one repair pass, when it picks its own correction (*Full* mode). In this comparison, full-mode Lean success therefore partly measures self-target selection rather than separation skill: a model that picks an easier target compiles more witnesses. *Second, repair fixes proof engineering, not semantics.* Lean-feedback repair lifts raw Lean acceptance by 23–61% across the three models, but the strict GPT-4.1-mini funnel still collapses from 48/204 raw Lean to 13/204 once the semantic-relation label must also match gold, and to 2/204 once the witness-family label must also match (Table B). Repair closes missing splits, unavailable tactics, and malformed namespaces. It rarely changes the witness type or semantic direction.

Table 3 also separates capabilities that are often conflated: relation label, witness family, FormalRx error type, error location, static checks, rendering, and Lean acceptance. Static validity alone is not enough: outputs that avoid forbidden proof escapes still fail Lean from type errors, unknown identifiers, or open goals. Rendered denominators differ because parsing and target-rendering filters reject a small number of outputs before Lean is invoked.

The repair pipeline should be read as a diagnostic decomposition, not a single capability score. In full mode, the model first proposes a corrected target, then writes a witness separating the candidate from that target, then optionally sees Lean feedback on the witness. The corrected-statement exact match for the GPT-4.1-mini repaired run is 53/204. A model can prove separation from its own target without that target being faithful to gold, which is exactly why Lean acceptance alone is not a semantic-faithfulness signal.

The two model orderings under different metrics also illustrate why a single score is misleading: GPT-4.1 has the strongest relation exact count (130/204), while Claude Haiku has the highest post-repair Lean count (108/204). A benchmark that reported only final Lean pass would rank these systems differently from one that also requires the correct relation, correction target, and witness family.

A per-model reading of Table 3 shows that no single system dominates. GPT-4.1 has the strongest relation-exact count (130) and FormalRx error-type count (23), while Claude Haiku leads on witness-family agreement (91) and on both raw and repaired Lean acceptance (88 and 108). GPT-4.1-mini is weaker on relation and witness-family agreement but has the highest reported error-location exact count (53, versus 31 for GPT-4.1 and 52 for Claude Haiku). For GPT-4.1-mini with repair, the strict funnel still drops from 48 raw Lean passes to 44 no-escape passes, 13 joint passes, and 2 joint-plus-type passes, showing that proof closure, relation correctness, and witness-family correctness remain separable.

This decomposition also explains why raw Lean lift overstates progress. Repair raises raw Lean acceptance for all three models (33 to 48 for GPT-4.1-mini, 51 to 82 for GPT-4.1, and 88 to 108 for Claude Haiku), but the gains are concentrated in proof-closing edits such as supplying a missing `split`. We compute the full no-escape and joint funnel only for GPT-4.1-mini, and there the strict counts move far less than the raw count (48 raw to 13 joint), because none of the repair edits we inspected revisited the relation label or the chosen correction target.

Table 4 shows that no single witness family or relation dominates the artifact. The seven families are spread between 19 and 38 examples, and the four theorem- or proposition-level relation buckets plus the local-delta bucket are similarly balanced. This matters for interpreting the joint metric: a model cannot reach a high joint score by mastering one easy family, because every family carries enough weight to move the count. The most common family, concrete instances, still accounts for under a fifth of the artifact, and the rarer structural and quantifier families require genuinely different proof obligations.

Local witnesses are often the *most precise* evidence available: they pinpoint the exact subexpression, predicate boundary, or quantifier domain where meaning changed, which is more diagnostic than a binary "these statements differ" verdict. Annotators used `local_delta` for 53/204

Table 3. Completed 204-example model-facing results. *Gold* mode supplies the intended correction and evaluates witness generation. *Full* mode requires the model to generate both correction and witness, and full-mode repair withholds the gold intended statement and uses only Lean feedback. *ErrT* and *ErrL* are FormalRx error-type and location exact counts. *Rendered* counts files that pass parsing and target rendering, and *Lean_raw* counts rendered files accepted by Lean. All non-rendering counts are out of 204.

Setting	Mode	Rel.	Type	ErrT	ErrL	Static	Rendered / Lean_raw
GPT-4.1-mini	Gold	114	–	–	–	38	204 / 21
GPT-4.1-mini	Full	53	68	18	53	178	193 / 33
GPT-4.1-mini + repair	Full	54	67	18	53	180	193 / 48
GPT-4.1	Full	130	70	23	31	199	199 / 51
GPT-4.1 + repair	Full	130	70	23	31	199	199 / 82
Claude Haiku	Full	95	91	17	52	196	204 / 88
Claude Haiku + repair	Full	95	91	17	52	193	204 / 108

Table 4. Coverage of the 204-example expanded artifact by witness family (left) and gold semantic relation (right). Counts sum to 204.

Witness family	Count	Relation	Count
Concrete instance	38	Candidate weaker	77
Expression delta	37	Local delta	53
Finite model	30	Candidate stronger	25
Property boundary	28	Candidate vacuous	25
Quantifier	27	Candidate incomparable	24
Vacuity	25		
Structural	19		

expanded examples, the cases where theorem-level separation was not the right object. The remaining 151/204 carry theorem- or proposition-level relation labels (77 candidate-weaker, 25 candidate-stronger, 24 incomparable, 25 vacuous). We report theorem-level and local witnesses separately because they answer different audit questions, not because local is inferior.

An archived same-input equivalence probe is included as a plumbing sanity check, not as a comparison: it asks Lean to prove equality between benchmark-provided `Candidate` and `Intended` using only `funext` and simplification. It is intentionally weaker than BEq+, and a matched BEq+ evaluation belongs in the next benchmark version.

A challenge version should still use a hidden test split fixed before prompt design. The current 204-example artifact is public and hand-curated, so it measures executable witness construction under transparent conditions rather than hidden-test generalization.

8. How to Read the Certificate

The most useful way to read a FormalRx-CC score is as a chain of increasingly strong claims. A static pass says that the model produced a well-formed witness object in the benchmark schema and did not use an explicitly forbidden proof escape. A rendered pass says that this object could be embedded into the benchmark Lean harness. A raw Lean pass says that Lean accepted the resulting file. A no-escape Lean pass additionally rules out references that would make the proof depend on benchmark-provided witness lemmas or trusted shortcuts. A joint pass says something stronger and more semantically relevant: the proof compiled and the model named the same relation as the gold annotation. Joint-plus-type adds agreement on the family of witness that explains the mismatch.

These layers are intentionally not collapsed into one number. For example, a model can give a correct counterexample but call the relation `candidate_stronger` instead of `candidate_weaker`. Lean can still verify the arithmetic, but the diagnostic metadata would mislead a user. Conversely, a model can name the right relation and witness family but fail Lean because it omits a `split`, uses a tactic unavailable in the environment, or does not unfold a local definition. The benchmark is designed to expose this distinction rather than hide it inside a single pass/fail label.

Repair, mode, and locality each shift what a passing layer means. A successful repair separates proof-closing errors from deeper diagnostic errors. If the corrected witness differs only in proof structure, the initial failure was Lean fluency. If it changes relation direction, target expression, or witness family, the failure was semantic. Gold and full modes answer different questions. Gold asks whether the model can write executable evidence when the intended target is supplied, while full asks whether it can propose that target and still support the separation claim. Full-mode Lean success should therefore be read next to corrected-statement exact match. Local-delta witnesses are scoped evidence: they isolate the exact point where the formalization changed meaning but do not certify the entire theorem interface, and FormalRx-CC reports local coverage separately for that reason.

A repair that fixes fluency, not meaning. A single full-mode example from the GPT-4.1 run shows what a Lean-feedback pass actually changes. The model had the right idea, a concrete-instance witness that the candidate property holds while the intended one fails, but the raw attempt left

the conjunction open:

```
-- raw witness (rejected: unsolved goals)
example : P 0 ∧ ¬ Q 0 := by
  constructor
  · norm_num [P]
```

Lean reports an unsolved goal for the second conjunct. The repaired witness, written after seeing that feedback, splits the conjunction and discharges both arithmetic goals:

```
-- repaired witness (accepted)
example : P 0 ∧ ¬ Q 0 := by
  refine ⟨?_, ?_⟩
  · norm_num [P]
  · intro h
    norm_num [Q] at h
```

The relation label, witness family, and chosen target are identical before and after. Only the proof structure changed, so this example moves the raw and no-escape Lean counts but not the joint or joint-plus-type counts. It is exactly the kind of case that inflates a raw-Lean leaderboard while leaving semantic understanding untouched.

9. Discussion and Limitations

FormalRx-CC audits a claimed formal mismatch. It does not, by itself, certify that English was captured. The verifier checks separation between formal artifacts, so the human or model that supplies `Intended` still carries the English-to-Lean burden. The certificate is strongest when the intended target is independently trusted, the witness is theorem-level, and the relation label agrees with gold. It is weaker when the target is model-generated, the proof is local rather than theorem-level, or the diagnostic metadata are wrong. These distinctions define what the certificate can and cannot be used for.

Model and annotation scope. The completed runs cover three general-purpose models (GPT-4.1-mini, GPT-4.1, Claude Haiku). Specialized provers (DeepSeek-Prover-V2, Goedel-Prover-V2, Kimina-Prover) and reasoning-tuned models are the natural next baseline. We deliberately exclude harness-mismatched probe runs from the main table rather than report plumbing artifacts as capability findings. Gold relation and witness-family labels are author-reviewed and executable but not double-annotated, which bounds the strength of label-based metrics. Independent annotation is part of the next benchmark version.

Witnessability. Not every mismatch has a short witness. Some formalizations differ only in ways that require deep model construction, domain expertise, or long Mathlib developments. We view witnessability as an empirical property to measure rather than assume: a useful benchmark should include direct witnesses first, then progressively add finite-model and higher-structure cases.

Statistical scope. The model-facing numbers come from a single 204-example run per setting, without resampling, so we report exact counts rather than means with confidence intervals and do not claim that small gaps between adjacent settings are significant. The robust findings are backed by large effects and by mechanism: the raw-to-joint collapse spans tens of examples, the gold-versus-full gap is observed in the GPT-4.1-mini comparison we report, and the repair case studies show what proof feedback does and does not change. A leaderboard version should add repeated runs and report variance before publishing any ranking.

10. Conclusion

Semantic-alignment diagnostics for autoformalization should be proof-carrying when possible. Counterexample-Carrying FormalRx asks the model to do more than label and repair a mismatch: it asks for a Lean-checkable artifact that *demonstrates* the mismatch. The completed 204-example evaluation shows this requirement separates two capabilities that single-number scoring conflates: Lean-feedback repair closes proof-engineering failures (23–61% raw Lean lift) but does not fix the semantic-relation label or correction faithfulness, and in the GPT-4.1-mini comparison the model passes Lean more easily against its own correction target than against the gold target. The anonymized artifact, prompts, scripts, raw outputs, and Lean logs make each layer of the certificate independently auditable. The natural next step is a frozen hidden split, independent annotation, and matched specialized-prover and BEq+ baselines.

Impact Statement

This work aims to improve the reliability and inspectability of AI systems for mathematical formalization. The main positive impact is stronger evidence that two formal artifacts differ in the way a diagnostic claim says they differ. This can support, but does not replace, human judgment about English-to-Lean faithfulness. The main risk is overtrusting local witnesses: a witness can validly separate the benchmark-provided `Candidate` and `Intended` local pair while the extraction itself fails to capture the original English/Lean mismatch, or while the model's diagnostic labels are wrong. We mitigate this by recording source rows, witness type, and verification target, by requiring adversarial mutation checks that reject model-declared local pairs, and by reporting joint label-and-proof metrics, not only Lean pass rates. A second risk is treating a curated artifact as exhaustive. We present it as a stress artifact, not a leaderboard.

References

AI4Math Workshop Organizers. 3rd AI for Math Workshop at ICML 2026. https://ai4math2026.github.io/, 2026.

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Blanchette, J. C. and Nipkow, T. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *Interactive Theorem Proving*. Springer, 2010.

Cruanes, S. and Blanchette, J. C. Extending Nunchaku to dependent type theory. *arXiv preprint arXiv:1606.05945*, 2016.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

Jiang, A. Q., Li, W., and Jamnik, M. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*, 2023a.

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*, 2023b.

LARK Lab. FormalRx-Test. https://huggingface.co/datasets/LARK-Lab/FormalRx-Test, 2026. Dataset card for the AI4Math 2026 semantic alignment evaluation challenge.

Li, Y., Kuang, J., Huang, H., Xu, Z., Liang, X., Yu, Y., Lu, W., Li, Y., Tan, X., Qu, C., Shen, Y., Zheng, H.-T., and Yu, P. S. One example shown, many concepts known! counterexample-driven conceptual reasoning in mathematical large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., and Jin, C. Goedel-Prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025a.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., and Jin, C. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025b.

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*, 2025.

Moura, L. d. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021.

Poiroux, A., Weiss, G., Kunčak, V., and Bosselut, A. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., et al. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Thakur, A., Tsoukalas, G., Wen, Y., Xin, J., and Chaudhuri, S. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*, 2023.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381, 2020.

Wang, H. et al. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*, 2022.

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. *arXiv preprint arXiv:2405.14333*, 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 Datasets and Benchmarks Track*, 2023.

Zheng, K., Han, J. M., and Polu, S. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

A. Artifact And Pipeline

Reproducibility is a primary artifact goal rather than an afterthought. The anonymized repository is available at https://anonymous.4open.science/r/ccf-F4C7/ and contains 25 seed examples, 26 FormalRx-derived development examples, 153 expanded stress examples, frozen dev/smoke slices, generated Lean witness files, prompt templates, verifier and scoring scripts, model-baseline and repair-loop scripts, raw model JSONL outputs, score summaries, Lean logs, and this paper draft.

Run settings. The model-facing runs use the OpenAI models `gpt-4.1` and `gpt-4.1-mini` and the Anthropic model Claude Haiku 4.5 (`claude-haiku-4-5-20251001`; alias `claude-haiku-4-5`), each queried through the scripts in the repository. Decoding uses temperature 0 with a 4096-token completion cap, and the one-pass repair runs reuse the same decoding settings, changing only the prompt to include the Lean error feedback. The Sonnet 4.6 and Goedel-Prover-V2-8B probe runs recorded in the logs are excluded from the main table for the harness reasons given in the submission checklist. Lean checks use `leanprover/lean4:v4.30.0-rc2`. At run time the Lean package tracked Mathlib at the `master` branch rather than a frozen commit, so the archived Lean logs, not a Mathlib hash, are the reproducibility reference for the reported counts. Pinning a specific Mathlib commit and recording it alongside each run is a concrete reproducibility improvement for the next benchmark version.

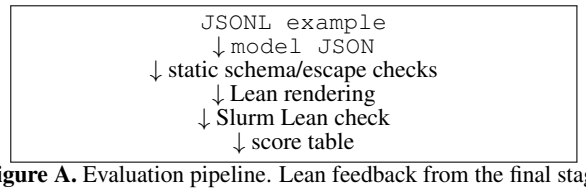

Figure A. Evaluation pipeline. Lean feedback from the final stage is optionally fed into the repair-loop baseline.

B. Coverage Tables

Witness type	Count
Concrete instance	38
Expression delta	37
Finite model	30
Property boundary	28
Quantifier	27
Vacuity	25
Structural	19

Relation	Count
Candidate weaker	77
Local delta	53
Candidate stronger	25
Candidate vacuous	25
Candidate incomparable	24

Table A. Coverage in the 204-example expanded artifact.

C. Appendix Charts

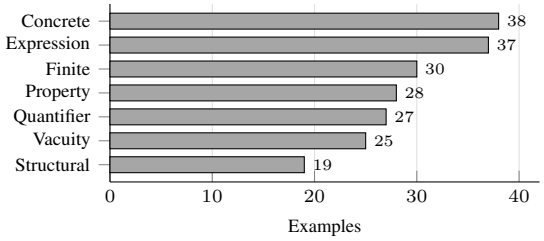

Figure B. Witness-family coverage in the 204-example artifact.

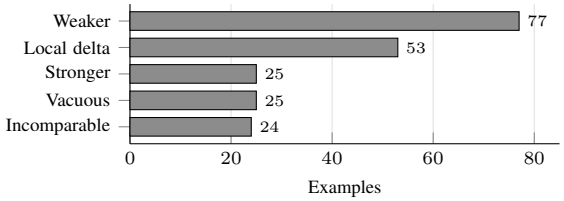

Figure C. Gold semantic-relation distribution in the expanded artifact.

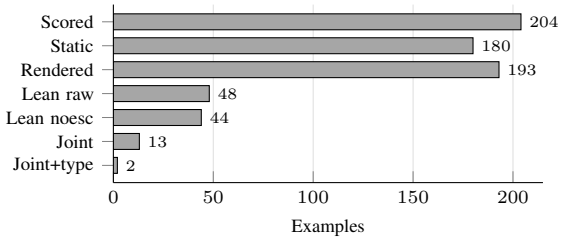

Figure D. GPT-4.1-mini full-mode repair funnel.

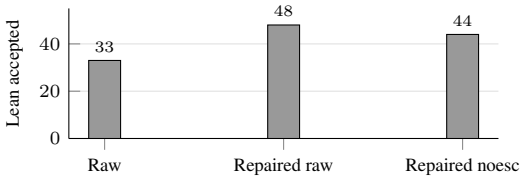

Figure E. Lean-feedback repair lift on the expanded run.

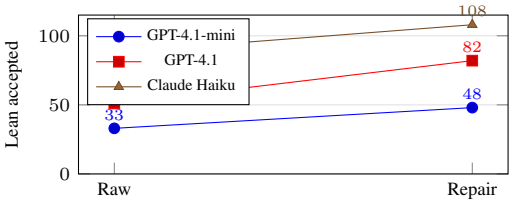

Figure F. Repair trajectories for completed 204-example full-mode runs.

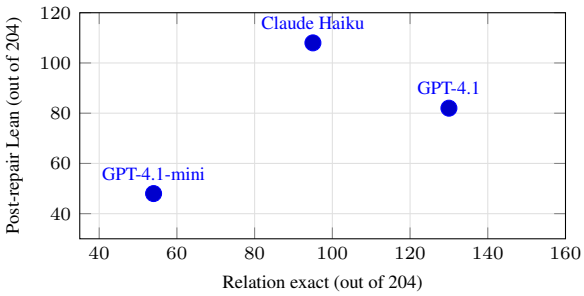

Figure G. Relation accuracy and post-repair Lean acceptance measure different behavior.

D. Model Diagnostics And Failure Modes

The completed expanded run is reported as a diagnostic breakdown rather than a leaderboard. "Lean verified" answers whether the rendered Lean file compiles. Joint counts ask whether the model also named the correct semantic relation and witness family.

Setting	Raw	Noesc	J+type
Mini + repair	48/193	44/204	2/204
GPT-4.1 + repair	82/199	–	–
Haiku + repair	108/204	–	–

Table B. Expanded repaired full-mode diagnostics.

The remaining failures fall into a small set of practical Lean-writing problems: open goals, unavailable tactics, type-class mismatches, using a proposition as a function, or compiling only through a forbidden proof escape. In one GPT-4.1 full-mode development example, the raw witness had the right arithmetic idea but left a conjunction open. The repaired witness unfolded definitions, split the conjunction, and closed both arithmetic goals with `norm_num`.

E. Metric Definitions

Static means the JSON object satisfies the witness schema, avoids blocked proof escapes, and names a supported witness family. *Rendered* means the prediction could be placed into the benchmark Lean template. *Lean_raw* counts rendered files accepted by Lean. *Lean_noesc* removes outputs that compile only by referencing blocked witness constants or helper proofs. *Rel.* is semantic-relation exact match against gold. *Joint* requires relation exact match and no-escape Lean success. *Joint+type* also requires the witness-family label to match gold.

Two final sanity checks guard against misleading wins. Local-delta witnesses must use benchmark-provided local extraction fields rather than model-declared local pairs. Full-mode repair does not see the gold intended statement. It receives only the model output and Lean feedback from that output.

Metric sanity example. The distinction matters in exactly the cases that motivated the benchmark. A witness can compile while the model gives the wrong relation label, because Lean checks the proposition it is given rather than the English diagnostic story. Conversely, a model can give the right relation label and a recognizable witness family while failing Lean on proof details such as an unsplit conjunction or an unavailable tactic. Reporting static, rendered, Lean, no-escape, relation-exact, and joint counts separately keeps those failure modes visible.

F. Submission Checklist

The final submission artifact is intended to be auditable from the repository. The expected reviewer path is: inspect the JSONL examples, run static validation, render witnesses into Lean, submit Lean checks on the cluster, score raw model JSONL outputs, and compare raw Lean, no-escape Lean, relation-exact, and joint metrics. The paper now avoids treating the completed 204-example model runs as a leaderboard: GPT-4.1 has the strongest relation-exact count, Claude Haiku has the strongest post-repair Lean count, and GPT-4.1-mini is the run with the strictest no-escape/joint breakdown currently reported.

The important negative controls are also recorded. Sonnet is omitted from the main table because the completed expanded recheck was dominated by a systematic full-file wrapper/import-formatting failure. Goedel-Prover-V2-8B is not reported as a specialized-prover result because the run used a base chat interface rather than the intended proving harness. The BEq-style probe remains an intentionally weak plumbing check, not a full BEq+ comparison.

The remaining methodological gaps are explicit: no independent inter-annotator agreement, no hidden frozen test split, and no matched DeepSeek-Prover/Kimina/o-series baseline. These gaps define the next version of the benchmark rather than being hidden in the current numbers.

What should reproduce. A clean artifact run should recover 204/204 static-valid gold examples and 204/204 Lean-accepted gold witnesses. Model-output runs should reproduce the JSONL-level score summaries and the rendered/Lean denominators reported in the paper. Differences in exact wall time are expected because Lean checks run on the cluster, but the accepted/failed file counts should match the archived score tables and logs.

What should not be inferred. The current artifact is not a hidden leaderboard and does not establish a general model ranking. Its stronger claim is narrower: once a target formalization is fixed, executable witnesses reveal whether a diagnostic boundary can be audited by Lean and whether the compiled proof still agrees with the semantic metadata.