BEYOND LLM JUDGE: MECHANICAL VERIFICATION FOR AGENTIC AUTOFORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a novel method for verifying Lean formalizations that goes beyond LLM-judged faithfulness. Unlike most verification methods, which rely on LLMs or trained models, our method combines LLM-based semantic review with mechanical numerical and vacuity checks, providing checkable feedback and detecting errors missed by LLM judges. We apply this method, along with our Extract-Verify-Proof autoformalization pipeline, to construct the Formalized Library of Mathematical Functions (FLMF), a partial formalization of the NIST Digital Library of Mathematical Functions (DLMF). On three labeled datasets, our method achieves the lowest accepted-error rate among the evaluated methods, measured as the fraction of incorrect statements accepted by the verification method. Expert review of the results also identifies translation and source defects in existing datasets. We will release these findings, along with the FLMF dataset, with this paper.

With the rapid development of agentic coding assistants, the formalization of mathematics has become very accessible to researchers and has received a lot of attention. However, to this day, the most trusted and widely used library is still Mathlib (The mathlib Community, 2020), whose correctness is guaranteed by careful human review and which is therefore trusted by its downstream users. For the same reason, it also faces a slow review process and limited coverage of existing mathematics, blocking the formalization of more advanced mathematical work. Efforts have been made to resolve this problem by building agentic systems to formalize textbooks, research blueprints, and dependency-guided libraries at increasing scale (Gloeckle et al., 2026; Rammal et al., 2026; Wang et al., 2026; Cabral et al., 2026; Milikic et al., 2026; Douglas et al., 2026).

In contrast to Mathlib, autoformalizations of this kind face severe challenges to their reliability. An autoformalizer can change a domain, index, sign, or branch convention, or introduce contradictory hypotheses, and still produce a provable statement. Audits of formal benchmarks document such defects (An et al., 2026).

We therefore propose a verification method that combines numerical verification, an LLM check, and a vacuity check, along with an agentic autoformalization pipeline, to allow the creation of trustworthy reusable formal libraries. We chose the Digital Library of Mathematical Functions (DLMF) as our testing ground, and propose the Formalized Library of Mathematical Functions (FLMF) as the product of our pipeline and verification method.

This paper makes three contributions:

- **Numerical and vacuity verification method** (Section 2). We introduce numerical and vacuity checks on Lean statements, which are mechanical and provide checkable feedback on points of failure. As indicated by their names, a numerical check verifies numerical validity of a Lean statement, and a vacuity check looks for vacuous formalization, for example a tautological definition. So far, all verification in autoformalization relies on LLM judgements, and our method complements the ambiguity and arbitrariness of an LLM judge's verdicts. Its feedback is tractable and informative, and passing these checks adds another layer of certainty about the correctness of a Lean statement.
- **Autoformalization pipeline** (Section 3). Our pipeline divides the formalization process into three stages: extract, verify, and prove. We emphasize human gating at each stage, requiring every item either to pass a gate or to be documented with a reason for failure. We then use

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

an auditor agent to revise those documents against written guidelines. We also keep track of all statement corrections to allow future studies of them.

- **Corrections to existing datasets** (Section 4). Expert review of our verification flags identifies multiple defects in existing datasets. Among the statements that are labeled correct, we found 4 defects among ProofNet items labeled faithful by Ground False, 61 defects among correct-labeled Beyond Compilation items, and 224 defects in RLM25 (Table 3). We will release corrected statements for confirmed defects.

1 Related Work

Evaluating formalization faithfulness. Some recent works use reference-based metrics to evaluate the faithfulness of formalization: BEq and its variants check bidirectional provable equivalence (Liu et al., 2025c; Poiroux et al., 2025a), GTED compares operator trees (Liu et al., 2025b), and ASSESS combines semantic and structural similarity (Liu et al., 2026). These metrics need a reference formal statement, which is a limitation when it comes to the formalization of natural language statements. On the other hand, reference-free methods score the informal–formal pair directly. FormalAlign trains a model to predict alignment from sequence certainty and representation similarity (Lu et al., 2025a). LeanScorer breaks a statement into components and has an LLM judge each one (Xuejun et al., 2025). Beyond Compilation evaluates LLM judges for faithful natural-language-to-Lean translation (Zhang et al., 2026). FormalMATH uses back-translation followed by LLM judgments to filter formalizations that do not preserve the original problem (Yu et al., 2025). As a result, Poiroux et al. show that automatic metrics disagree with humans on research-level statements and release RLM25 (Poiroux et al., 2025a). Ground False audits formal benchmarks for kernel-checked but wrong targets, classifies each ProofNet statement's relation to its informal source, and releases both that taxonomy and a corrected ProofNet-Verified (An et al., 2026). In this paper, we compare our verification method against them using formal–informal pair datasets.

Agentic formalization. Agentic formalization extends single statement translation to the construction of formal libraries. Gloeckle et al. coordinate parallel agents on a shared code base to formalize a graduate textbook (Gloeckle et al., 2026), while AutoformBot combines dependency-aware scheduling and collaborative version control to build libraries across multiple textbooks (Rammal et al., 2026). At the workflow level, LeanFlow studies how proof workflows and toolsets affect the completion and efficiency of formalizing entire papers (Milikic et al., 2026); Requena Pozo et al. combine iterative proof refinement, library search, and context management in a minimal agent (Requena Pozo et al., 2026); and OpenGauss provides an open-source Lean autoformalization harness (Math, Inc., 2026). FormalScience incorporates domain-expert feedback into an agentic pipeline for formalizing scientific reasoning (Meadows et al., 2026). These systems motivate our emphasis on statement verification and human gates within a larger formalization workflow.

2 Faithfulness Verification

Our verification method consists of three components: a semantic (LLM) check, a numerical check, and a vacuity check. The first two evaluate the Lean theorem statement without the proof body. The vacuity check runs after the proof is constructed and determines if the proof was vacuous. No single check establishes faithfulness, and we treat verification as a stack in which each layer has a distinct failure model and any substantive mismatch blocks automatic acceptance. The following discussion is based on our formalization of DLMF, but a large part of it is applicable to other large-scale formalizations of natural language math. For each theorem, we attached a verification tag, which is `unverified` by default and turns to `verified` after passing the checks.

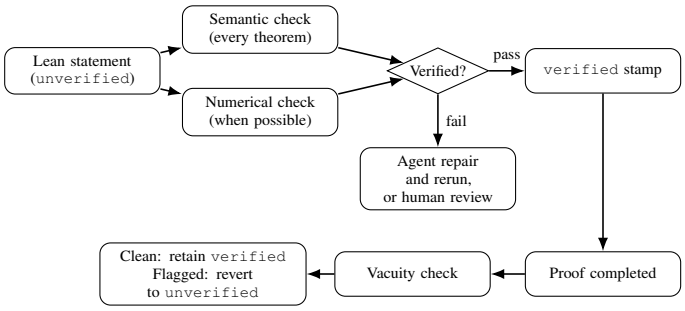


Figure 1: The verification workflow. A theorem is `verified` if it receives a semantic `FAITHFUL` and no numerical or vacuity `FAIL`.

2.1 Semantic Check

The semantic checks consist of a direct LLM inspection and a back-translation check. The direct inspection passes both the Lean statement and the informal statement to an LLM judge, and the judge returns a verdict about the faithfulness of the Lean translation.

The back-translation check consists of two steps. In the first step, an agent (translator) reads the Lean theorem statement, along with all definitions involved, *without* comments, and translates it into plain English. Masking the comments reduces the risk of hallucination on the agent's side, forcing it to adhere to the exact Lean code. The translation is formatted to include the statement, variables, constraints, and symbols.

As an example, the translation of §DLMF5.5.3 $\Gamma(s)\Gamma(1-s) = \pi/\sin(\pi s)$ would be:

> STATEMENT: For any complex number z that is not an integer, $\Gamma(z)\Gamma(1 - z) = \pi/\sin(\pi z)$.
>
> VARIABLES: z: A complex number, $z \in \mathbb{C}$.
>
> CONSTRAINTS: $z \neq m$ for every $m \in \mathbb{Z}$.
>
> SYMBOLS: Gamma - the complex Gamma function. Complex.sim - the complex sine function. $\uparrow\pi$ - the mathematical constant π, cast to a complex number.

A second agent (judge) reads this English rendering and compares it with the equation extracted from the DLMF source.[1] In addition, we supply a STANDING_CONSTRAINTS.md document to capture the constraints stated earlier in the source that may be missed by the equation parser.

The direct inspection may be the most intuitive thing to do, but having a back-translation check guarantees that the Lean statement is accurate enough to preserve its meaning after a back-translation. Having two separate agents working independently also avoids the failure mode in which an agent recognizes an inconsistency but chooses to ignore it. The verdicts from the judges fall into three categories: `FAITHFUL`, `DIVERGE`, or `UNCERTAIN`. For the latter two, the judge also documents why it considers the translation unfaithful and waits for further action. When a theorem obtains a `FAITHFUL` verdict from both the direct inspection and the back-translation check, its verification status can change to `verified`.

2.2 Numerical Check

When possible[2], a stated theorem undergoes a numerical check in which a mechanical extractor translates the Lean theorem into evaluations in *mpmath*x, an open-source Python library for arbitrary-precision numerical computation and special functions. A special function is bound to its mpmath counterpart with the appropriate convention. The checker samples points from the stated domain and compares the function values on both sides of the identity. Exact identities use residual gates;

[1] We designed an extractor to pull information from DLMF metadata, which involves an XML file along with the website display, allowing more reliable extraction of equation content and constraints. A sample extracted JSON file is available in Appendix C.

[2] Exceptions include definitions, limits, infinite sums, etc., that cannot be converted into a numerical check.

asymptotics use ratio or rate gates. Failure of the numerical check vetoes the `verified` status granted by the back-translation check.

One natural question is how to guarantee the correctness of the Lean-mpmath binding. This binding is trustworthy because we require that every special function is only defined once, and the binding is attached to the definition. When performing the numerical check, the program traces the definitions used in the Lean statement and uses the corresponding bindings. We can also observe how many times each definition/binding has been used independently for numerical checks, further confirming its reliability.

There are also limitations to the numerical check. As we are working with special functions in DLMF, we can only sample points in the domain and evaluate the functions at those points. A common failure is the inability to identify an incorrect domain for a functional equality. For example, the Lean statement may have a smaller domain than the actual statement, or it may fail to exclude isolated points from the domain. If those points are not selected by the numerical checker, it will fail to flag the error. Therefore, the numerical check should only be used as a partial check for correctness, not a complete faithfulness check.

2.3 Vacuity Check

A vacuity check scans the theorem statement and proof body to detect empty statements. Simple checks include testing whether trivial proof methods such as `rfl` or `simp` close the goal. In addition, we have a checker that scans for goal-independent proofs, which works by replacing the target goal with an opaque conclusion `VacQ` and checking whether the proof still compiles. If it does, then the proof is goal-independent, and the most likely cause is a set of contradictory hypotheses from which one can derive `False`[3]. If the checker flags a vacuous proof, the `verified` status is also reverted to `unverified`.

We would like to emphasize that none of these checks is perfect; they all have different domains of effectiveness and limitations, and the stack is undergoing active development as the project progresses. For example, the case that motivated the vacuity check was the original formalization of DLMF 7.13.5: its back-translation was judged faithful, yet proof review found that its completeness and ordering hypotheses for the Fresnel zeros were jointly contradictory. We also made these checks strict on purpose, so we would rather have them flag a correct statement than miss an incorrect one. (Section 5 gives more detail on this particular case.)

3 Autoformalization Pipeline

The DLMF is organized in chapters and equations. Distinct chapters in the DLMF focus on different types of functions and rarely depend on each other; therefore, parallelization is quite feasible in FLMF. There are also special "methods" chapters, which describe mathematical techniques used to establish some of the identities. In these cases, we need to formalize, or at least state, the content of the methods before we can prove the related identities. These dependency relations are captured in a DAG (Directed Acyclic Graph) and are dynamically updated as formalization progresses. When the proving agent attempts to prove a theorem, it first looks it up in the DAG and reports whether infrastructure needed to construct the proof is missing.

This strategy is not unique to DLMF. Indeed, related works have used similar approaches (Wang et al., 2026; Cabral et al., 2026), and it plays a crucial role in the autoformalization of more sophisticated math objects that have complicated dependencies on each other.

In FLMF, we proceed with formalization in three stages: Extract, Verify, and Prove.

[3] Any conclusion can be derived from `False`.

Stage	Agent work	Gate on every theorem / equation
DAG	Enumerate equations and dependencies	Open-world gap report
Extract	Write typed definitions and theorem statements	(i) Formalized with appropriate tag or (ii) documented in `DISPOSITIONS.md`
Verify	Run numerical and back-translation fidelity checks	(i) Obtained a `verified` stamp or (ii) documented in `HUMAN_REVIEW.md`
Prove	Fill frozen theorem bodies in waves	(i) Proved with clean build and 0 `sorry` or (ii) documented in `UNPROVEN.md`

Table 1: Autoformalization pipeline for FLMF.

3.1 EXTRACT

At this stage, an agent extracts all listed equations in a chapter by their equation numbers. The agent then writes each equation in the DAG and establishes its dependencies on existing nodes. The agent identifies all *definitions* needed to state the extracted equations, for example, the definitions of Bessel functions or theta functions. The definitions are only written once to guarantee consistency in downstream usage. After that, parallel agents are deployed to write Lean statements using these definitions. A complete proof is not required. Each extracted theorem carries 4 tags: DLMF equation number, Fidelity check method, Verification status, and its corresponding DAG node. The Extract workflow is gated on completion: every listed equation in the DLMF has to either (i) appear in FLMF as a theorem with appropriate tags or (ii) be documented in `DISPOSITIONS.md` with an explanation of why it is not stated. This is an example of a formalized statement:

```
/- **DLMF 5.5.3** -- Γ(s)Γ(1 − s) = π/ sin(πs) (complex)
(DLMF-faithful restriction of 'complex_reflection' : s ≠ 0, ±1, ±2, ...).
DAG: E-5.5.3.
Fidelity:  numeric PASS
Verified:  verified
-/
theorem complex_reflection_dlmf
  (s : ℂ)
  (_hs : ∀ m : ℤ, s ≠ (m : ℂ)) :
  Complex.Gamma s * Complex.Gamma (1 - s) =
    (Real.pi : ℂ) /
    Complex.sin ((Real.pi : ℂ) * s) := by
  sorry
```

3.2 VERIFY

When a theorem is first stated, its verification status is `unverified` by default. A successful verification check, which is the combination of semantic, numerical, and vacuity checks discussed in Section 2, allows a stamper to set this tag to `verified`. If a theorem fails a verification check, an agent analyzes the reason for failure. For the easy fixes, e.g., missing premises, a wrong sign, or incompatible conventions, an agent fixes the theorem and reruns the verification check; otherwise it is directed to human review. This stage is gated by requiring every equation to have either (i) a `verified` stamp or (ii) a documented entry in `HUMAN_REVIEW.md`. If a statement does get corrected, it will be logged in an ERRATA file, keeping track of the mistakes made by agents and allowing further error analysis.

3.3 PROVE

After the Extract and Verify stages, we have the full list of theorems that are faithful but await to be proved. An agent surveys all open `sorry`s and proposes a proof plan. It contains a methods component, consisting of general proof strategies that apply to many identities, and waves of proofs that resolve the remaining theorems in dependency order. This step also observes and modifies the DAG dependency structure. The proving stage is gated on either (i) a `sorry`-free proof body or (ii) documentation in `UNPROVEN.md`.

3.4 AUDIT

Beyond the pipeline, we have auditor agents to inspect the human review documents at each stage (dispositions, human_review, unproven, etc.). Very often, these documents contain a mix of simple mistakes, such as a wrong convention, and serious issues, such as a hidden assumption that is not stated in the informal text. The auditors are prompted with the human evaluation guidelines and make simple fixes while providing analysis and suggestions for the harder issues that really need a human decision.

We would like to emphasize the need for gating and human interaction at each stage. Asking the agent to work alone risks two failure modes: the agent spends many tokens working on a single problem and gets stuck, or it secretly skips the work and does not report its true status. Human review is also crucial for design choices, such as definition conventions, or special cases in which implicit constraints or specifications are conveyed but not listed.

Having the auditor significantly reduces the work of a human expert. The human expert can focus on the hard and subtle cases in which we do not readily trust the model's judgement, while the rest are taken care of autonomously. Having an auditor does not reduce the reliability of the output, since the edited statement will have to go through the verification check again before it can be stamped `verified`. On very special occasions, for example, one that involves a source misprint, a human can stamp a theorem `verified` despite it not passing the verification check, but that is only possible after careful review by the expert, and no autonomous agents can do that.

Combined with our verification method, the workflow allows autoformalization of large-scale libraries without losing credibility. As part of this work in progress, we were able to complete the formalization of about 10% of DLMF, comprising around 1,000 entries, in 2 man-months. Further details are available in Appendix A.

4 EXPERIMENTS

In the following experiments, we compare our verification method with other baseline methods across different datasets. We demonstrate that ours has the lowest rate of accepting an incorrect formalization, which is the goal of building a reliable autoformalization verification. We also report runs of our method on external datasets, presenting potential defects in them.

4.1 SETUP

Datasets. The experiment contains 1,810 informal–formal pairs from four datasets, and the datasets provide a correctness evaluation for each pair. **FLMF** is a selected set of 471 statements from the ongoing DLMF formalization, combining library statements, mutations[4], historical errors, and source-misprint pairs. This dataset focuses on special function equalities. **ProofNet** uses the original release (Azerbayev et al., 2023), with Ground False's faithfulness verdicts as labels (An et al., 2026). It contains 371 undergraduate math problems from analysis, algebra, and topology. **Beyond Compilation** is a set of 400 graduate-level math problems with semantic evaluations (Zhang et al., 2026). **RLM25** extracts 617 informal–formal statement pairs from six research-level formalization projects across number theory, harmonic analysis, combinatorics, etc[5](Poiroux et al., 2025a). Appendix D gives a more detailed breakdown of dataset construction process.

Methods and models. *Ours* is the union of the three checks as discussed in Section 2, where direct inspection uses GPT-5.5, and back-translation uses Gemini 3.1 Pro as the translator and GPT-5.5 as the judge. The baseline methods are GPT-5.5 and Opus 4.6 as direct judges, each called once per pair and once in joint batches of 16 ("joint"); LeanScorer (Xuejun et al., 2025), which uses LLMs (GPT-5.5) to assess how faithfully each component of a statement is formalized; and FormalAlign (Lu et al., 2025a), a trained model that evaluates alignment between informal and formal statements, at a certainty threshold of 0.3 (Appendix E gives more detail). Every method sees the same informal statement and formal context.

[4]We included some mutations that are intended to be incorrect, mutation mechanisms are extracted from historical errors

[5]RLM25 doesn't contain any evaluation component, so we assume all statements are labeled "correct".

Method	FLMF (182 C / 289 I)		Beyond Compilation (245 C / 106 I)		ProofNet (163 C / 204 I)	
	Incorrect / accepted (rate)	Flagged / correct	Incorrect / accepted (rate)	Flagged / correct	Incorrect / accepted (rate)	Flagged / correct
Ours	**5/105 (4.8%)**	45.1%	**3/154 (1.9%)**	38.4%	**14/153 (9.2%)**	14.7%
GPT-5.5	28/178 (15.7%)	17.6%	10/227 (4.4%)	11.4%	26/176 (14.8%)	8.0%
GPT-5.5 (joint)	45/203 (22.2%)	13.2%	13/232 (5.6%)	10.6%	38/194 (19.6%)	4.3%
Opus 4.6	72/232 (31.0%)	12.1%	24/261 (9.2%)	3.3%	44/206 (21.4%)	0.6%
Opus 4.6 (joint)	88/250 (35.2%)	11.0%	42/267 (15.7%)	8.2%	56/215 (26.0%)	2.5%
LeanScorer	10/93 (10.8%)	54.4%	7/175 (4.0%)	31.4%	17/135 (12.6%)	27.6%
FormalAlign (≤ 0.3)	32/76 (42.1%)	75.8%	52/148 (35.1%)	60.8%	23/30 (76.7%)	95.7%

Table 2: Accepted-error rate by method on the three labeled datasets. *Incorrect / accepted* counts the incorrect items among those a method did not flag. A lower accepted-error rate means fewer incorrect Lean formalizations after the check. *Flagged / correct* is the percentage of all items the method flags among those labeled "correct". The counts under each dataset are its correct (C) and incorrect (I) items.

Method coverage. The semantic checks, direct LLM judges, LeanScorer, and FormalAlign apply to every informal–formal pair. However, the goal-independent part of the vacuity check requires a completed proof and has no coverage on the original ProofNet or Beyond Compilation. In these cases, the vacuity check only looks for trivial proof methods. Similarly, the numerical check requires a statement to be numerically evaluable, which is not the case for, for example, infinite sums statements.

Metric. We evaluate acceptance by a verification method against each dataset's labels, which are categorized into "correct" and "incorrect". Our primary metric is the *accepted-error rate*: among the items a verification method accepts (does not flag), the fraction that are labeled incorrect. This metric aligns with our goal of ensuring that a Lean statement passing the gate is correct. We also report the percentage of total statements flagged.

4.2 Main results

Table 2 reports the results of running all methods on the FLMF, Beyond Compilation, and ProofNet datasets. Our method's accepted-error rates are the lowest reported rates on all three datasets. Admittedly, our flagging rate is towards the higher end among all methods, but false alarms are generally acceptable given the help of auditing agents. We note that the "incorrect" labels from the datasets are not always accurate. The 3 items labeled incorrect that our method accepted in Beyond Compilation turn out to all have source ambiguity or incorrect labeling, and we would actually label 2 of the 3 "correct" after expert review.

In Table 3, we present an audit using our verification method on the Beyond Compilation, ProofNet, and RLM25 datasets. The flags raised by the verification pipeline then go through expert review to identify translation defects, source defects, or false alarms. In each of them, we were able to identify a number of defective statements that were labeled "correct". We will release the corrected dataset with this paper.

Notice that the flags raised by a numerical check have a very high rate of identifying real defects, and many of those defects were not identified by a semantic LLM check. In Beyond Compilation, all 41 statements flagged by the numerical check yet labeled "correct" turn out to contain real defects. In RLM25, 8 out of the 12 flagged statements contain real defects. The four RLM25 numerical false alarms are attributed to binding errors in adapting the checker to the project's data structures.

5 Case Studies

We present two case studies on formalization defects identified by our method. One is identified by the vacuity checker alone and not by any other method, which is a subtle formalization defect involving contradictory hypotheses. The second is a false Taylor bound from the *Beyond Compilation* dataset that was faithfully translated to Lean and flagged only by our numerical check and none others. We therefore argue that even if we can make sure an LLM translates an informal statement

Dataset	Check	Flagged	Labeled incorrect	Translation defects	Source defects	False alarm
Beyond Compilation	Numerical	76	35	8	37	0
	Semantic	151	101	15	12	25
	Vacuity	23	12	1	0	10
	Combined	195	102	18	43	35
ProofNet	Numerical	85	85	0	0	0
	Semantic	203	183	3	1	16
	Vacuity	4	1	0	0	3
	Combined	208	186	3	1	18
RLM25	Numerical	12	n/a	8	0	4
	Semantic	367	n/a	203	21	143
	Vacuity	12	n/a	3	1	8
	Combined	373	n/a	203	21	149

Table 3: Flags across different datasets and reviewed defect types. *Flagged* counts all alerts by each method. *Labeled incorrect* counts alerts on items the dataset labels incorrect. *Translation defects* and *Source defects* count review findings among flagged items that the dataset labels correct. RLM25 has no correctness labels; therefore, all statements are considered "correct" by the dataset.

seemingly well, the statement itself can contain subtleties or defects that are not captured by the model, resulting in an incorrect formalization.

5.1 Case 1: vacuous statement in FLMF.

DLMF §7.13(iii) In the first quadrant of $\mathbb{C}$, the Fresnel integral $C(z)$ has an infinite set of zeros $z_n = x_n + iy_n$, $n = 1, 2, 3, \ldots$ arranged in order of increasing absolute value. As $n \to \infty$, the x_n and y_n corresponding to the zeros of $C(z)$ satisfy

$$x_n \sim \lambda + \frac{\alpha(\alpha\pi - 4)}{8\pi\lambda^3} + \cdots, \qquad y_n \sim \frac{\alpha}{2\lambda} + \cdots, \tag{DLMF §7.13.5}$$

with

$$\lambda = \sqrt{4n - 1}, \qquad \alpha = (2/\pi)\ln(\pi\lambda). \tag{DLMF §7.13.6}$$

The extracted Lean statement was (definitions stripped)

```
theorem erf_7_13_5 (z : ℕ → ℂ)
  (hzero : ∀ n, cfresnelC (z n) = 0)
  (hquad : ∀ n, 0 < (z n).re ∧ 0 < (z n).im)
  (hmono : StrictMono (fun n ↦ (z n).re))
  (hall : ∀ w : ℂ, cfresnelC w = 0 →
        0 < w.re → 0 < w.im → ∃ n, z n = w) :
  (fun n ↦ (z n).re - fresnelCZeroReApprox n)
    =o[atTop] (fun n ↦ fresnelCZeroLambda n ^ (-3 : ℤ)) ∧
  (fun n ↦ (z n).im - fresnelCZeroImApprox n)
    =o[atTop] (fun n ↦ fresnelCZeroLambda n ^ (-1 : ℤ)) := by
```

This Lean theorem assumes that a sequence z_n enumerates all first-quadrant zeros of $C(z)$ while their **real parts** are strictly increasing (`hmono`). However, the informal source explicitly orders the zeros by increasing **absolute value**. The assumptions in the Lean statement are contradictory because the large-index zeros will approach the real axis, and a mirror symmetry transforms them into first-quadrant zeros with arbitrarily small positive real parts. Therefore, there must be a first-quadrant zero whose real part is smaller than that of z_0, the first entry of the sequence, violating the monotonicity imposed by `hmono`. The resulting contradiction among the hypotheses proves `False`, from which the stated asymptotic conclusion can be obtained vacuously by `False.elim`.

In verification, this Lean statement obtained a `FAITHFUL` verdict from the semantic checks because the checker was not able to catch this subtle mistake. Indeed, all other baseline methods accepted it as a correct translation, and the proof compiled without error. Only the vacuity check was able to spot the problem.

5.2 CASE 2: A DEFECTIVE INFORMAL SOURCE IN BEYOND COMPILATION.

The informal statement says that if f is holomorphic on a neighborhood of the closed unit disk and $f(0) \neq 0$, then there exists a constant $C \in \mathbb{R}$ such that

$$\left| 1 - \sum_{k=0}^{m} \frac{f^{(k)}(0)}{k!\, f(0)} z^k \right| \leq C|z|^{m+1} \qquad (m \in \mathbb{N},\ |z| \leq 1).$$

The corresponding formalization, with its theorem name abbreviated to `X`, is

```
theorem X {f : ℂ → ℂ}
  (hf : ∃ U : Set ℂ, IsOpen U ∧ Metric.closedBall (0 : ℂ) 1 ⊆ U ∧
    DifferentiableOn ℂ f U)
  (h0 : f 0 ≠ 0) :
  ∃ C : ℝ, ∀ m : ℕ, ∀ z : ℂ, ‖z‖ ≤ 1 →
    ‖(1 : ℂ) - (∑ k ∈ Finset.range (m + 1),
       ((deriv^[k] f) 0) / ((k.factorial : ℕ) * f 0) * z ^ k)‖
     ≤ C * ‖z‖ ^ (m + 1)
```

This is a faithful translation of the informal statement. However, the numerical check finds the admissible counter example $f(z) = e^z$. With $m = 1$, for real $0 < z \leq 1$ the claimed bound becomes

$$|1 - (1 + z)| = |z| \leq C|z|^2.$$

Dividing both sides by $|z|^2$, we get $C \geq 1/|z|$, which no constant C can satisfy as $z \to 0$.

In the experiment, this correct-labeled entry is flagged only by the numerical check among all methods. Back-translation, GPT-5.5 and Opus 4.6 in both single-pair and joint modes return `FAITHFUL`; LeanScorer accepts it with a score of 0.67 at a threshold of 0.6; FormalAlign accepts it with a certainty score of 0.394 at a threshold of 0.3. None of the stored semantic explanations identifies the impossible uniform bound.

6 CONCLUSION

In this paper, we introduced our verification method, conducted experiments across different benchmarks, and demonstrated that our method yields the lowest accepted-error rate, therefore yielding the most trustworthy outcome. We integrate these methods into an Extract–Verify–Prove pipeline with human gates and audits, ensuring completeness and faithfulness of the formalization results. FLMF provides the development setting and a preliminary evaluation panel, and its broader dataset remains under construction and will be presented in a separate paper.

We highlight the numerical and vacuity checks in our verification, which are complementary to the verdicts of an LLM or model-based judge, the most common approach used across all other formalization work. According to our experiments, these checks are able to spot mistakes ignored by all LLM judges, including subtle translation defects or informal source errors. The checks are also informative in pointing out the exact numerical misalignment or proof inconsistency. A theorem passing these checks can be trusted to be at least numerically consistent and not trivially vacuous. Combined with our autoformalization pipeline, we can then obtain a promising Lean formalization of the informal source.

There are certain limitations to our current methods. Numerical and vacuity checks are only applicable to numerically evaluable theorems or complete proofs. There could be defects in these checks as well, for example, a wrong binding in the numerical check or a subtly vacuous proof that does not fall within the detection range of our vacuity detector. Passing these checks doesn't guarantee correctness either, for example a narrower domain in the Lean statement compared to the informal source is a common failure mode.

Future work will extend the numerical and vacuity checks to handle additional corner cases encountered during formalization, while guaranteeing faithfulness of the checks. We aim to reduce vacuity false alarms by distinguishing contradictory assumptions and missing mathematical content from legitimate definitional identities and routine proofs. We will also explore additional mechanical checks that complement the existing methods and provide independently checkable evidence.

AI USAGE STATEMENT

AI tools were used as part of the research methodology described in this paper, including for autoformalization, semantic verification, auditing, and theorem proving. The specific models used in the reported experiments are documented in the corresponding sections. AI tools were also used to assist with language editing and polishing of the manuscript. The authors reviewed and take full responsibility for the final text, results, and claims presented in this paper.

REFERENCES

NIST Digital Library of Mathematical Functions, Release 1.2.7, 2026. https://dlmf.nist.gov/.

L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. In *CADE-28*, 2021.

The mathlib Community. The Lean mathematical library. In *CPP*, 2020.

M. R. Douglas, S. Hoback, A. Mei, and R. Nissim. Formalization of QFT. arXiv:2603.15770, 2026.

H. Wang, R. Xie, Y. Wang, G. Gao, X. Yu, and B. Dong. Aria: an agent for retrieval and iterative auto-formalization via dependency graph. In *ICLR*, 2026.

R. Cabral, T. Manh, X. Yu, W. M. Tai, Z. Feng, and X. Shen. ProofFlow: a dependency graph approach to faithful proof autoformalization. In *ICLR*, 2026.

F. Gloeckle, A. Rammal, C. Arnal, R. Munos, V. Cabannes, G. Synnaeve, and A. Hayat. Automatic textbook formalization. arXiv:2604.03071, 2026.

A. Rammal, N. Patel, F. Gloeckle, A. Hayat, J. Kempe, R. Munos, C. Arnal, and V. Cabannes. Formalizing mathematics at scale. arXiv:2605.29955, 2026. https://github.com/facebookresearch/atlas-lean.

L. Milikic, S. Guilloud, K. Nguyen, and V. Kuncak. LeanFlow: a case study in workflow-driven Lean autoformalization. In *ICML AI for Math Workshop*, 2026.

B. Requena Pozo, A. Letson, K. Nowakowski, I. Beltran Ferreiro, and L. Sarra. A minimal agent for automated theorem proving. arXiv:2602.24273, 2026.

Math, Inc. OpenGauss: an open-source autoformalization harness for Lean. Software, 2026. https://github.com/math-inc/OpenGauss.

O. An, M. J. Min, X. Si, and O. Bastani. Ground False: dissecting errors in formal mathematics benchmarks. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026. https://openreview.net/forum?id=5c0RSYyIWW. ProofNet-Verified release: https://github.com/marcusm117/ProofNet-Verified.

M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions*. National Bureau of Standards, 1964.

E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis*. Cambridge University Press, 4th edition, 1927.

A. Erdelyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi. *Higher Transcendental Functions*, volumes 1–3. McGraw–Hill, 1953–1955.

A. Cuyt, V. B. Petersen, B. Verdonk, H. Waadeland, and W. B. Jones. *Handbook of Continued Fractions for Special Functions*. Springer, 2008.

H. S. Wall. *Analytic Theory of Continued Fractions*. Van Nostrand, 1948.

W. B. Jones and W. J. Thron. *Continued Fractions: Analytic Theory and Applications*. Addison–Wesley, 1980.

F. Oberhettinger and L. Badii. *Tables of Laplace Transforms*. Springer, 1973.

A. C. Dixon. On a certain double integral. *Proceedings of the London Mathematical Society*, s2-2:8–15, 1905.

G. W. Anderson. A short proof of Selberg's generalized beta formula. *Forum Mathematicum*, 3:415–417, 1991.

P. J. Forrester and S. O. Warnaar. The importance of the Selberg integral. *Bulletin of the American Mathematical Society*, 45:489–534, 2008. arXiv:0710.3981.

A. Selberg. Bemerkninger om et multipelt integral. *Norsk Matematisk Tidsskrift*, 26:71–78, 1944.

M. L. Mehta. *Random Matrices*. Elsevier, 3rd edition, 2004.

F. J. Dyson. Statistical theory of the energy levels of complex systems. I. *Journal of Mathematical Physics*, 3:140–156, 1962.

A.-L. Cauchy. *Exercices d'analyse et de physique mathematique*. Bachelier, 1841.

J. A. Wilson. Some hypergeometric orthogonal polynomials. *SIAM Journal on Mathematical Analysis*, 11:690–701, 1980.

G. E. Andrews, R. Askey, and R. Roy. *Special Functions*. Cambridge University Press, 1999.

K. G. C. von Staudt. *De numeris Bernoullianis*, 1845; T. Clausen, related denominator theorem, *Astronomische Nachrichten*, 1840.

N. E. Norlund. *Vorlesungen uber Differenzenrechnung*. Springer, 1924.

E. E. Kummer. Beitrag zur Theorie der Function $\Gamma(x)$. *Journal fur die reine und angewandte Mathematik*, 35, 1847.

L. Euler. Early correspondence on the interpolation of factorials, 1729.

A.-M. Legendre. Recherches sur diverses sortes d'integrales definies. *Memoires de la classe des sciences mathematiques et physiques*, 1809.

J. L. Raabe. Zur Theorie der Eulerschen Integrale. *Journal fur die reine und angewandte Mathematik*, 20:146–159, 1840.

R. Bellman. *A Brief Introduction to Theta Functions*. Holt, Rinehart and Winston, 1961.

G. Polya. Remarks on computing the probability integral in one and two dimensions. In *Proceedings of the First Berkeley Symposium on Mathematical Statistics and Probability*, 1949.

J. M. Blair, C. A. Edwards, and J. H. Johnson. Rational Chebyshev approximations for the inverse of the error function. *Mathematics of Computation*, 30, 1976.

B. C. Carlson. *Special Functions of Applied Mathematics*. Academic Press, 1977.

B. C. Carlson. Numerical computation of real or complex elliptic integrals. *Numerical Algorithms*, 10:13–26, 1995.

J. V. Armitage and W. F. Eberlein. *Elliptic Functions*. Cambridge University Press, 2006.

A. Tauber. Ein Satz aus der Theorie der unendlichen Reihen. *Monatshefte fur Mathematik und Physik*, 8:273–277, 1897.

H.-J. Runckel. On the zeros of the hypergeometric function. *Mathematische Annalen*, 191:53–58, 1971.

M. Hairer. Φ^4_3 is orthogonal to GFF. Note, 2022. https://hairer.org/Phi4.pdf.

M. Hairer, S. Kusuoka, and H. Nagoji. Singularity of solutions to singular SPDEs. arXiv:2409.10037, 2024.

N. Ebel, T. Kennedy, and S. Rychkov. Tensor renormalization group meets computer assistance. arXiv:2506.03247, 2025.

A. Ospanov, F. Farnia, and R. Yousefzadeh. APOLLO: automated LLM and Lean collaboration for advanced formal reasoning. arXiv:2505.05758, 2025.

J. Lu, K. Emond, K. Yang, S. Chaudhuri, W. Sun, and W. Chen. Lean Finder: semantic search for Mathlib that understands user intents. arXiv:2510.15940, 2025.

F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic*, version 1.3.0, 2023.

Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad. ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics. arXiv:2302.12433, 2023. https://arxiv.org/abs/2302.12433.

A. Poiroux, G. Weiss, V. Kunčak, and A. Bosselut. Reliable Evaluation and Benchmarks for Statement Autoformalization. In *EMNLP*, pp. 17947–17969, 2025. https://aclanthology.org/2025.emnlp-main.907/.

A. Poiroux, A. Bosselut, and V. Kunčak. RLMEval: Evaluating Research-Level Neural Theorem Proving. In *Findings of EMNLP*, 2025. https://aclanthology.org/2025.findings-emnlp.581/.

M. J. Min, Y. Gao, W. Sy, Z. Li, X. Si, and O. Bastani. Divide and Abstract: Autoformalization via Decomposition and Abstraction Learning. In *ICLR*, 2026. https://openreview.net/forum?id=NjgaeXNit3.

K. Zhang, P. Gallardo Candela, S. Murthy, Y. Xie, Z. Wang, and M. Raissi. Beyond Compilation: Evaluating Faithful Natural-Language-to-Lean Statement Formalization. arXiv:2606.31002, version 2, 2026. https://arxiv.org/abs/2606.31002v2.

Yu Xuejun, J. Zhong, Z. Feng, P. Zhai, R. Yousefzadeh, W. C. Ng, H. Liu, Z. Shou, J. Xiong, Y. Zhou, C. B. Ong, A. J. Sugiarto, Y. Zhang, W. M. Tai, H. Cao, D. Lu, J. Sun, Q. Xu, Shen Xin, and Z. Li. Mathesis: Towards Formal Theorem Proving from Natural Languages. arXiv:2506.07047, 2025. https://arxiv.org/abs/2506.07047. LeanScorer implementation: https://github.com/Huawei-AI4Math/Mathesis.

J. Lu, Y. Wan, Y. Huang, J. Xiong, Z. Liu, and Z. Guo. FormalAlign: Automated Alignment Evaluation for Autoformalization. In *ICLR*, 2025. https://arxiv.org/abs/2410.10135. Data and code: https://github.com/rookie-joe/FormalAlign.

Q. Liu, X. Zheng, X. Lu, Q. Cao, and J. Yan. Rethinking and Improving Autoformalization: Towards a Faithful Metric and a Dependency Retrieval-Based Approach. In *ICLR*, 2025. https://openreview.net/forum?id=hUb2At2DsQ.

Y. Liu, T. Zhu, X. Liu, Y. Chen, Z. Liu, Q. Guo, J. Zhang, K. Bao, and T. Luo. Generalized Tree Edit Distance (GTED): A Faithful Evaluation Metric for Statement Autoformalization. In *ICML AI for Math Workshop*, 2025. https://arxiv.org/abs/2507.07399v2.

X. Liu, T. Zhu, Z. Dong, Y. Liu, Q. Guo, Z. Liu, Y. Chen, and T. Luo. ASSESS: A Semantic and Structural Evaluation Framework for Statement Similarity. In *ICLR*, 2026. https://arxiv.org/abs/2509.22246v2.

J. Kim, H. Han, and S.-W. Hwang. Benchmarking Testing in Automated Theorem Proving. In *ACL Industry Track*, pp. 2241–2260, 2026. https://aclanthology.org/2026.acl-industry.150/.

J. Meadows, L. Zhang, and A. Freitas. FormalScience: Scalable Human-in-the-Loop Autoformalisation of Science with Agentic Code Generation in Lean. In *ACL*, 2026. https://arxiv.org/abs/2604.23002. FormalPhysics dataset: https://huggingface.co/datasets/jmeadows17/FormalPhysics.

J. Liu, X. Lin, J. Bayer, Y. Dillies, W. Jiang, X. Liang, R. Soletskyi, H. Wang, Y. Xie, B. Xiong, Z. Yang, J. Zhang, L. Zhi, J. Li, and Z. Liu. CombiBench: Benchmarking LLM Capability for Combinatorial Mathematics. arXiv:2505.03171, 2025. https://arxiv.org/abs/2505.03171.

J. Jiang, W. He, Y. Wang, G. Gao, Y. Hu, J. Wang, N. Guan, P. Wu, C. Dai, L. Xiao, and B. Dong. FATE: A Formal Benchmark Series for Frontier Algebra of Multiple Difficulty Levels. arXiv:2511.02872, 2025. https://arxiv.org/abs/2511.02872.

Z. Yu, R. Peng, K. Ding, Y. Li, Z. Peng, M. Liu, Y. Zhang, Z. Yuan, H. Xin, W. Huang, Y. Wen, G. Zhang, and W. Liu. FormalMATH: Benchmarking Formal Mathematical Reasoning of Large Language Models. arXiv:2505.02735, 2025. https://arxiv.org/abs/2505.02735.

K. Zheng, J. M. Han, and S. Polu. MiniF2F: A Cross-System Benchmark for Formal Olympiad-Level Mathematics. In *ICLR*, 2022. https://arxiv.org/abs/2109.00110.

G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri. PutnamBench: Evaluating Neural Theorem-Provers on the Putnam Mathematical Competition. In *NeurIPS Datasets and Benchmarks Track*, 2024. https://arxiv.org/abs/2407.11214.

K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *NeurIPS*, 2023. https://github.com/lean-dojo/LeanDojo.

L. Zhang, M. Valentino, and A. Freitas. Autoformalization in the Wild: Assessing LLMs on Real-World Mathematical Definitions. In *EMNLP*, 2025. https://arxiv.org/abs/2502.12065.

A CHAPTER-LEVEL PIPELINE PROGRESS

Table 4 reports progress of FLMF on all 36 chapters. Of the 8,833 numbered DLMF equations, 1,326 (15%) are stated in FLMF, 641 (7%) are verified, and 1,142 have a `sorry`-free proof. Chapter 5, 7, and 20 are essentially complete. Chapters 6 and 15 are fully extracted but less fully verified (56 of 97 and 52 of 198).

Table 4: Complete DLMF chapter progress, reported as counts.

Ch.	Chapter	Targets	Extract	Verified	Proved
1	Algebraic and Analytic Methods	705	88	38	75
2	Asymptotic Approximations	349	2	1	2
3	Numerical Methods	297	2	0	2
4	Elementary Functions	518	26	14	24
5	Gamma Function	164	162	123	137
6	Exponential, Logarithmic, Sine and Cosine Integrals	100	97	56	77
7	Error Functions, Dawson's and Fresnel Integrals	137	137	136	137
8	Incomplete Gamma and Related Functions	241	15	1	15
9	Airy and Related Functions	230	13	4	11
10	Bessel Functions	698	52	10	40
11	Struve and Related Functions	149	11	6	11
12	Parabolic Cylinder Functions	177	4	1	4
13	Confluent Hypergeometric Functions	381	17	3	17
14	Legendre and Related Functions	284	14	4	11
15	Hypergeometric Function ${}_2F_1$	202	198	52	157
16	Generalized Hypergeometric and Meijer G	99	5	2	5
17	q-Hypergeometric and Related Functions	182	7	1	7
18	Orthogonal Polynomials	677	56	10	43
19	Elliptic Integrals	464	17	3	17
20	Theta Functions	113	113	88	112
21	Multidimensional Theta Functions	59	0	n/a	n/a
22	Jacobian Elliptic Functions	258	22	10	18
23	Weierstrass Elliptic and Modular Functions	167	21	6	21
24	Bernoulli and Euler Polynomials	188	22	7	20
25	Zeta and Related Functions	174	24	10	23
26	Combinatorial Analysis	201	16	9	16
27	Functions of Number Theory	132	33	20	29
28	Mathieu Functions and Hill's Equation	353	15	1	15
29	Lamé Functions	205	1	0	1
30	Spheroidal Wave Functions	114	3	0	2
31	Heun Functions	120	103	25	64
32	Painlevé Transcendents	286	6	0	6
33	Coulomb Functions	146	0	n/a	n/a
34	$3j$, $6j$, $9j$ Symbols	64	18	0	17
35	Functions of Matrix Argument	62	0	n/a	n/a
36	Integrals with Coalescing Saddles	137	6	0	6
	Total	**8,833**	**1,326**	**641**	**1,142**

B DLMF-SIDE DISCREPANCIES IDENTIFIED TO DATE

The following potential misprints in the DLMF had been identified while formalizing and proving the corresponding FLMF statements. They have been confirmed by the DLMF committee and corrections are updated.

DLMF	Printed defect and correction	Independent evidence
20.10.4	The display uses $(1+\beta)$ where scale covariance and the theta shift require $(\ell+\beta)$. The θ_1 transform has the opposite printed sign, and the θ_1 and shifted θ_2 integrals are negatives of each other, not equal.	Theta quasi-periodicity already makes the printed chain inconsistent at $\ell = 1$; termwise Laplace transforms and independent 25-digit quadrature recover the corrected form.
20.10.5	The display uses $(1+\beta)$ where scale covariance and the theta shift require $(\ell+\beta)$.	Termwise Laplace transforms and independent 25-digit quadrature recover the corrected form.
20.13.5	The alternating heat-kernel sum is printed as θ_4; Poisson summation gives θ_2.	At $(\alpha, t, z) = (0.7, 0.3, 0.4)$, the sum is $1.5982\ldots$ and matches θ_2 to 1.6×10^{-26}, whereas the printed θ_4 side is $0.3972\ldots$; complex samples agree.

Table 5: DLMF misprint findings.

C DLMF EXTRACTION EXAMPLE

The following is the presentation MathML (pMML) source for DLMF §5.5.3.

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML"
  altimg="E3.png" altimg-height="99px" altimg-valign="-7px"
  altimg-width="149px"
  alttext="\Gamma\left(z\right)\Gamma\left(1-z\right)=\pi/\sin\left(\pi z\right),"
  display="block">
  <m:mrow>
    <m:mrow>
      <m:mrow>
        <m:mrow>
          <m:mi href="DLMF:/5.2#E1" mathvariant="normal"
            title="gamma function">&#x393;</m:mi>
          <m:mo>&#x2061;</m:mo>
          <m:mrow><m:mo>(</m:mo>
            <m:mi href="DLMF:/5.1#t1.r4"
              title="complex variable">z</m:mi>
            <m:mo>)</m:mo></m:mrow>
        </m:mrow>
        <m:mo>&#x2062;</m:mo>
        <m:mrow>
          <m:mi href="DLMF:/5.2#E1" mathvariant="normal"
            title="gamma function">&#x393;</m:mi>
          <m:mo>&#x2061;</m:mo>
          <m:mrow><m:mo>(</m:mo><m:mrow><m:mn>1</m:mn>
            <m:mo>&#x2212;</m:mo>
            <m:mi href="DLMF:/5.1#t1.r4"
              title="complex variable">z</m:mi>
          </m:mrow><m:mo>)</m:mo></m:mrow>
        </m:mrow>
      </m:mrow>
      <m:mo>=</m:mo>
      <m:mrow>
        <m:mi href="DLMF:/3.12#E1"
          title="the ratio of the circumference of a circle to its diameter">
          &#x3C0;</m:mi>
        <m:mo>/</m:mo>
        <m:mrow>
          <m:mi href="DLMF:/4.14#E1" title="sine function">sin</m:mi>
          <m:mo>&#x2061;</m:mo>
          <m:mrow><m:mo>(</m:mo><m:mrow>
            <m:mi href="DLMF:/3.12#E1"
              title="the ratio of the circumference of a circle to its diameter">
              &#x3C0;</m:mi>
            <m:mo>&#x2062;</m:mo>
            <m:mi href="DLMF:/5.1#t1.r4"
              title="complex variable">z</m:mi>
          </m:mrow><m:mo>)</m:mo></m:mrow>
        </m:mrow>
      </m:mrow>
    </m:mrow>
    <m:mo>,</m:mo>
  </m:mrow>
</m:math>
```

Because the standalone pMML omits the side condition, the extractor reads the equation's container in the full DLMF section page. Its normalized JSON output is:

```
{
 "tag": "5.5.3",
 "url": "https://dlmf.nist.gov/5.5#E3",
 "kind": "equation",
 "latex": [
  "\\Gamma\\left(z\\right)\\Gamma\\left(1-z\\right)=\\pi/\\sin\\left(\\pi z\\right),",
  "\\Gamma\\left(\\NVar{z}\\right)",
  "z\\neq 0,\\pm 1,\\dots",
  "\\sin\\NVar{z}",
  "\\pi",
  "z"
 ],
 "eqs": [],
 "section_eqs": [
  [
   "5.5.1",
   "\\Gamma\\left(z+1\\right)=z\\Gamma\\left(z\\right),"
  ],
  [
   "5.5.2",
```

```
    "\\psi\\left(z+1\\right)=\\psi\\left(z\\right)+\\frac{1}{z}."
   ],
   [
    "5.5.3",
    "\\Gamma\\left(z\\right)\\Gamma\\left(1-z\\right)=\\pi/\\sin\\left(\\pi z\\right),"
   ],
   [
    "5.5.4",
    "\\psi\\left(z\\right)-\\psi\\left(1-z\\right)=-\\pi/\\tan\\left(\\pi z\\right),"
   ],
   [
    "5.5.5",
    "\\Gamma\\left(2z\\right)=\\pi^{-1/2}2^{2z-1}\\Gamma\\left(z\\right)\\Gamma\\left(z+\\tfra
    ↪  c{1}{2}\\right)."
   ],
   [
    "5.5.6",
    "\\Gamma\\left(nz\\right)=(2\\pi)^{(1-n)/2}n^{nz-(1/2)}\\prod_{k=0}^{n-1}\\Gamma\\left(z+\
    ↪  \frac{k}{n}\\right)."
   ],
   [
    "5.5.7",
    "\\prod_{k=1}^{n-1}\\Gamma\\left(\\frac{k}{n}\\right)=(2\\pi)^{(n-1)/2}n^{-1/2}."
   ],
   [
    "5.5.8",
    "\\psi\\left(2z\\right)=\\tfrac{1}{2}\\left(\\psi\\left(z\\right)+\\psi\\left(z+\\tfrac{1}
    ↪  {2}\\right)\\right)+\\ln 2,"
   ],
   [
    "5.5.9",
    "\\psi\\left(nz\\right)=\\frac{1}{n}\\sum_{k=0}^{n-1}\\psi\\left(z+\\frac{k}{n}\\right)+\\
    ↪  ln n."
   ]
  ],
  "context": ", pMML , png See also: Annotations for \u00a75.5(i) , \u00a75.5 and Ch.5
  ↪  \u00a75.5(ii) Reflection \u24d8 Keywords: gamma function , psi function , reflection
  ↪  formula Notes: See Olver ( 1997b , pp. 35 and 39) . Permalink:
  ↪  http://dlmf.nist.gov/5.5.ii See also: Annotations for \u00a75.5 and Ch.5 5.5.3 \u0393
  ↪  \u2061 ( z ) \u2062 \u0393 \u2061 ( 1 \u2212 z ) = \u03c0 / sin \u2061 ( \u03c0 \u2062 z
  ↪  ) , z \u2260 0 , \u00b1 1 , \u2026 , \u24d8 Symbols: \u0393 \u2061 ( z ) : gamma function
  ↪  , \u03c0 : the ratio of the circumference of a circle to its diameter , sin \u2061 z :
  ↪  sine function and z : complex variable A&S Ref: 6.1.17 (without the condition on z .)
  ↪  Referenced by: \u00a710.19(i) , \u00a715.4(ii) , \u00a715.8(ii) , (25.9.2) , \u00a75.21 ,
  ↪  (9.10.18) , (9.12.15) Permalink: http://dlmf.nist.gov/5.5.E3 Encodings: TeX , pMML , png
  ↪  See also: Annotations for \u00a75.5(ii) , \u00a75.5 and Ch.5 <table",
  "xref_context": [],
  "constraints": [
   "z\\neq 0,\\pm 1,\\dots"
  ],
  "symbols": [
   {
    "token": "\u0393",
    "href": "5.2#E1",
    "gloss": "gamma function"
   },
   {
    "token": "z",
    "href": "5.1#t1.r4",
    "gloss": "complex variable"
   },
   {
    "token": "\u03c0",
    "href": "3.12#E1",
    "gloss": "the ratio of the circumference of a circle to its diameter"
   },
   {
    "token": "sin",
    "href": "4.14#E1",
    "gloss": "sine function"
   }
  ],
  "info": {
   "referenced_by": "\u00a710.19(i) , \u00a715.4(ii) , \u00a715.8(ii) , (25.9.2) , \u00a75.21 ,
   ↪  (9.10.18) , (9.12.15)",
   "as_ref": "6.1.17 (without the condition on z .)",
   "proof_sketch": "",
   "sources": "",
   "keywords": "",
   "notes": "",
   "defines": "",
```

```
  "symbols_text": "\\Gamma\\left(\\NVar{z}\\right) : gamma function , \\pi : the ratio of the
  ↪  circumference of a circle to its diameter , \\sin\\NVar{z} : sine function and z :
  ↪  complex variable",
  "changes": []
 },
 "parser_version": 2
}
```

D Dataset construction, panel composition, and detection

D.1 Dataset construction and labels

The evaluation contains 1,810 informal–formal pairs from four datasets. Table 6 gives their composition. Each dataset supplies its own criteria for labeling a pair correct or incorrect; these labels do not represent a common judgment of faithfulness or mathematical validity.

FLMF (182 correct / 289 incorrect). The FLMF dataset contains 471 statements in eight components: 40 library seeds, 40 copies with bound variables renamed, 7 human-approved corrections to DLMF misprints, 160 intentional mutations of the seeds, 17 historical FLMF errors, 7 printed forms of DLMF misprints, 100 sampled verified theorems, and 100 statements from the project's errata log. The seeds and renamings each contain 38 correct and 2 incorrect items, and the sampled verified theorems contain 99 correct and 1 incorrect item. Corrected misprints are labeled correct. Mutations, historical errors, printed misprints, and errata items are labeled incorrect.

The mutations target eight mechanisms (M1–M8): impossible hypotheses, wrong objects, branches, totalization at poles, quantifier strength, indices and coercions, normalization, and lost source content. There are 12–20 mutations per mechanism and 24 simpler sanity mutations (Table 7). We exclude 20 independent control nominations. We also exclude two Fresnel coefficient pairs, comprising four items, because their stronger asymptotic claims are unresolved. Historical Fresnel errors remain included.

The 100 sampled verified theorems satisfy the same eligibility criteria as the seeds: a semantic `verified` stamp at the pinned commit, a single numbered DLMF equation, and a completed proof. Selection is deterministic and stratified by library area, with a cap of 20 items per stratum. This component is disjoint from the seeds, renamings, mutations, historical artifacts, and misprint pairs by both declaration name and DLMF equation.

Each of the 100 errata items is a declaration recovered from git history in its form immediately before correction. Of these, 76 correspond to errors found by the numerical or vacuity check and 24 to errors found by other methods. They form 65 erratum clusters, and 99 of the 100 statements elaborate. Together with the mutations, historical errors, printed misprints, and five library entries rejected in human review, they account for the 289 incorrect items. The 129 incorrect items other than mutations provide examples of errors arising in formalization or in the source.

ProofNet (163 correct / 204 incorrect) (Azerbayev et al., 2023). We use 367 original ProofNet statements from the Lean 4.28 port distributed with ProofNet-Verified. The formal statements are those of ProofNet, and the labels are the Ground False verdicts (An et al., 2026). The 163 items labeled *faithful* are correct. The incorrect items comprise 75 labeled *stronger*, 55 *weaker*, 64 *incomparable*, and 10 whose informal statement is ambiguous (9) or wrong (1).

Beyond Compilation (245 correct / 106 incorrect) (Zhang et al., 2026). We use 351 of the 400 released sources. The exclusions are 42 reported compilation failures, 6 entries from which no single statement can be separated from the proof, and 1 informal source that states no claim. A pair is labeled correct if the release's GPT-5.2 judge assigns a score of at least 9. Human annotations are available for 156 items whose recorded judge score matches the candidate's released score, and we use them as secondary labels. The annotations are predominantly acceptances and concern translations selected from model acceptances, so they do not independently assess the items labeled incorrect by the judge.

RLM25 (621 reference pairs) (Poiroux et al., 2025a). The reconstructed dataset contains reference pairs from six projects: Carleson (111), FLT (56), FLT3 (84), PFR (146), PrimeNumberTheoremAnd (99), and TestingLowerBounds (125). The published release contains 619 pairs. Reconstruction from public sources yields 621, including one extra eligible node in PFR and one in TestingLowerBounds, so membership does not exactly match the release manifest. RLM25 provides no correctness labels. We treat all reference pairs as correct and report their flag rate.

The formal context includes project definitions referenced by each statement, including definitions declared in other files and those introduced with `irreducible_def` or `noncomputable abbrev`. For 310 of the 621 items, this supplies definitions absent from the source prefix in the

statement's own file. Statement extraction accounts for local `let` expressions, which is necessary to preserve two statements in full. FormalAlign's released adapter does not receive formal context.

Proof availability. The goal-independent part of the vacuity check requires a completed proof. The 100 sampled verified FLMF theorems have completed proofs. For errata items, coverage depends on the availability of a replayable proof of the statement as written before correction. Beyond Compilation and the original ProofNet statements provide no proofs, so the vacuity check on these datasets is limited to trivial proof methods, as described in Section 4.1.

D.2 Panel composition and detection

Table 6 gives the composition of the four datasets. Table 7 reports flags on the 271 FLMF seeds, renamings, mutations, historical artifacts, and misprint items.

Dataset	Component	Correct	Incorrect
FLMF	Seeds	38	2
	Bound-variable renamings of the seeds	38	2
	Human-approved corrected forms of DLMF misprints	7	
	Mutations of the seeds (M1–M8 + sanity)		160
	Historical FLMF errors		17
	Misprinted DLMF equations		7
	Sampled verified theorems after human review	99	1
	Errata statements		100
	FLMF total (471 items)	**182**	**289**
ProofNet (verdict by Ground False)	faithful	163	
	stronger		75
	weaker		55
	incomparable		64
	informal source ambiguous (9) or wrong (1)		10
Beyond Compilation	Full release (label: released judge score ≥ 9)	245	106
RLM25	Reconstructed reference pairs from all six projects, with project definitions	621	
Total		**1,211**	**599**

Table 6: Dataset composition (1,810 items).

FLMF subset	n	Ours	GPT-5.5	GPT-5.5 (joint)	Opus 4.6	LeanScorer	FormalAlign	N	B	V
Seeds, renamings, and corrected misprints										
Seeds	40	12	8	4	3	22	37	0	9	3
Renamed seeds	40	15	7	4	4	21	36	0	12	3
Corrected misprints	7	7	7	7	6	3	1	0	7	0
Mutations, historical artifacts, and printed misprints										
M1 impossible hypotheses	20	20	20	20	19	20	18	0	20	0
M2 wrong object	16	16	16	16	15	16	16	15	16	0
M3 branches/phases	20	20	20	20	20	20	19	18	20	0
M4 totalization/poles	20	18	18	18	13	19	20	13	18	1
M5 quantifiers/asymptotics	16	16	14	14	14	16	14	9	15	0
M6 indices/coercions	16	16	16	16	16	16	16	14	16	0
M7 normalization/scaling	16	16	15	16	15	16	16	11	16	0
M8 lost source content	12	11	12	12	12	12	12	10	7	0
Sanity mutations	24	24	24	24	24	24	23	21	24	0
Historical artifacts	17	15	15	14	10	14	12	6	13	1
Printed misprints	7	7	6	2	4	4	1	5	6	0

Table 7: Flags on the 271 FLMF seeds, renamings, mutations, historical artifacts, and misprint items.

E FormalAlign threshold

FormalAlign returns a certainty score, and we flag a statement when its score is at most the chosen threshold. Table 8 reports the threshold sweep on the three labeled datasets. At the comparison threshold of 0.3, the accepted-error rates are 42.1% on FLMF, 35.3% on Beyond Compilation, and 80.0% on ProofNet. Raising the threshold reduces the number of accepted statements but does not consistently reduce the fraction labeled incorrect.

	FLMF (182 C / 289 I)		Beyond Compilation (245 C / 106 I)		ProofNet (163 C / 204 I)	
Threshold	Incorrect / accepted	Flagged / correct	Incorrect / accepted	Flagged / correct	Incorrect / accepted	Flagged / correct
0.1	243 / 409	8.8%	100 / 338	2.9%	185 / 312	22.1%
0.2	111 / 199	51.6%	76 / 248	29.8%	81 / 120	76.1%
0.3	**32 / 76**	**75.8%**	**49 / 139**	**63.3%**	**16 / 20**	**97.5%**
0.4	12 / 22	94.5%	21 / 45	90.2%	2 / 2	100.0%
0.5	1 / 1	100.0%	3 / 8	98.0%	0 / 0	100.0%

Table 8: FormalAlign certainty-threshold sweep on 1,189 labeled pairs. A statement is flagged when its certainty score is at most the threshold. C and I denote the dataset's correct and incorrect labels. *Incorrect / accepted* counts incorrect-labeled items among accepted statements. *Flagged / correct* is the percentage of correct-labeled items that are flagged. The threshold of 0.3 used in the main comparison is shown in bold. The 0 / 0 entry indicates that no statements are accepted. RLM25 reference pairs are not included in this sweep.

F Cost and runtime

Channel	Items	API calls	Cost (USD)	Cost / item (USD)	Time / item (s)
Numerical check	1,539	3,961	314.18	0.204	77.0
Back-translation	1,539	3,080	99.57	0.065	29.0
Vacuity check	1,539	0	0.00	0.000	–
GPT-5.5 direct	918	920	33.01	0.036	13.5
Opus 4.6 direct	918	922	20.96	0.023	14.6
LeanScorer	918	1,840	66.65	0.073	20.5
GPT-5.5 joint	918	61	12.02	0.013	4.2
Opus 4.6 joint	918	61	5.43	0.006	3.1

Table 9: API cost and recorded runtime. The 918-item set comprises 200 FLMF, 351 Beyond Compilation, and 367 ProofNet statements. The 1,539-item set also includes 621 RLM25 reference pairs. Costs include retained reservations totaling $9.64 for 25 requests whose charges remain unresolved. Time per item is the sum of final recorded job durations divided by the item count, with joint-batch time amortized across items; it excludes superseded attempts and is not elapsed wall-clock time. Vacuity runtime is omitted because 1,512 records report result-reuse overhead rather than the original checking time. FormalAlign scored 918 items locally on CPU, with 32.1 minutes of recorded inference time (2.1 seconds per item), excluding model loading.