IndiMathBench: Autoformalizing Mathematical Reasoning Problems with a Human Touch

Param Biyani[†], **Shashank Kirtania, Yasharth Bajpai, Sumit Gulwani, Ashish Tiwari**
Microsoft
parambiyani8@gmail.com
[†]Work done at Microsoft, Corresponding author

Abstract

Reliable autoformalization remains an elusive goal even in the era of large language models (LLMs). Even the best LLMs struggle to translate natural language into formal constructs in languages like Lean. High-quality data for formal theorem proving has been a key bottleneck given the resource costs associated with manual curation and validation of these translations. On these lines, we introduce IndiMathBench, a human-verified benchmark designed to evaluate mathematical theorem proving, curated using an AI-powered human-assisted pipeline for formalizing natural language problems in Lean. IndiMathBench is composed of 312 formal Lean4 theorems paired with their corresponding informal problem statements, sourced from Indian Mathematics Olympiads. Our pipeline uses category-based retrieval and a self-debug loop with feedback from a symbolic checker to generate candidate formalizations. Multiple such formalizations are generated using an ensemble of LLMs. These formulas are presented with a summary to the human through an interactive dashboard. The dashboard enables efficient validation and repair by the human. We analyze the performance of several state-of-the-art models on IndiMathBench, which will facilitate further research on automated theorem proving. IndiMathBench is available at https://github.com/prmbiy/IndiMathBench.

1 Introduction

The formalization of mathematics, expressing informal reasoning in precise, machine-verifiable logic, has been a long-standing goal in computer science and mathematics. A related task, autoformalization, seeks to automatically translate informal mathematical statements and proofs into formal representations (Wu et al., 2022; Agrawal et al., 2022a; Gadgil et al., 2022). Despite advances in large language models and theorem-proving frameworks such as Lean (Moura & Ullrich, 2021), progress remains limited by the scarcity of paired informal–formal data.

Existing benchmarks for formal theorem proving are few and narrow in scope. The largest Lean 4 benchmarks with Olympiad-level problems, miniF2F (Zheng et al., 2022), drawn from AIME, AMC, and IMO exams, and PutnamBench (Tsoukalas et al., 2024), from the Putnam Competition cover only a small fraction of the available competition mathematics, totaling roughly a thousand problems. This limited scale restricts comprehensive evaluation of model generalization and reasoning capabilities.

Let n be a natural number. Prove that:

$$\left\lfloor \frac{n}{1} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor + \cdots + \left\lfloor \frac{n}{n} \right\rfloor + \lfloor \sqrt{n} \rfloor$$

is even.

```
theorem inmo_2014_2 (n : ℕ) :
  Even ((Finset.sum (Finset.range n)
    fun i => ⌊(n : ℝ) / ((i + 1) :
    ℝ)⌋) + ⌊Real.sqrt n⌋) := by
  sorry
```

Figure 1: A sample problem from INMO 2014 with its formalization in Lean 4

Moreover, as these popular sources increasingly appear in large-scale pretraining corpora (Jiang et al., 2024), creating contamination that obscures genuine reasoning ability. Producing new benchmarks is further constrained by the high manual effort required: experts must formalize, annotate, and verify each problem in Lean, a process that is both time-consuming and resource-intensive (Yu et al., 2025). As a result, progress in ATP evaluation depends on developing scalable, high-quality, human-verified formal benchmarks.

To address these challenges, we introduce IndiMathBench , a benchmark for automated theorem proving built from Indian Mathematical Olympiad problems. Our benchmark contains 312 problems spanning diverse mathematical domains, geometry, algebra, number theory, and combinatorics, each paired with human-verified Lean 4 formalizations. A sample problem from IndiMathBench is shown in Figure 1. We conducted systematic human verification of all formalizations, ensuring high-quality ground truth for reliable evaluation.

Our key contributions are:

- A novel Lean 4 benchmark for formal theorem proving, created using LLM-assisted formalization and human verification.
- A Visual Studio Extension to improve human-AI collaboration for Lean annotations.
- A formalization framework employing category-based retrieval and self-debug loop with a Lean validator, and an ensemble of LLMs.
- An evaluation of frontier foundation models on their autoformalization capabilities.

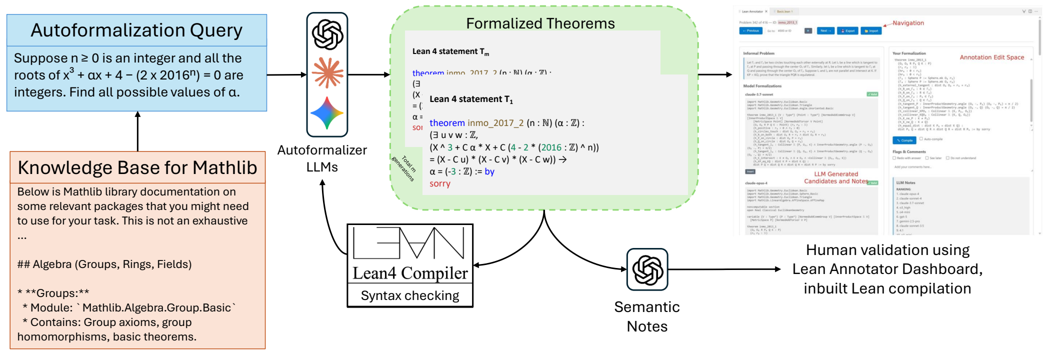
2 Related Work

Formal Theorem Proving Benchmarks. The evaluation of automated theorem proving systems relies critically on curated formal benchmarks, yet existing resources remain limited in scale, diversity, and representativeness. The miniF2F benchmark (Zheng et al., 2022), based on AMC, AIME, and IMO competitions, and PutnamBench (Tsoukalas et al., 2024), derived from the Putnam Competition, have become standard datasets for assessing Olympiad competition-level mathematical reasoning in Lean 4. More recent benchmarks such as ProofNet (Azerbayev et al., 2023) and FormalMath (Yu et al., 2025) focus on undergraduate or textbook-style mathematics, providing complementary but less challenging problem domains. While these resources have advanced systematic evaluation, they remain narrow in both mathematical coverage and cultural scope, largely reflecting Western curricula and competition traditions, where the focus is relatively less on geometry and combinatorics style problems, and more towards analysis, algebra and number theory. Recent efforts like FrontierMath (Collaboration, 2024) extend evaluation toward research-level problems but reveal extremely low model success rates, underscoring the substantial gap between current benchmarks and the diversity of human mathematical reasoning, but doesn't support Lean.

Autoformalization. Autoformalization aims to translate informal mathematical language into machine-verifiable formal logic (like Lean). Recent advances in large language models have substantially improved the ability to generate structured formal statements directly from natural language (Wu et al., 2022; Agrawal et al., 2022b). Despite these advances, fully automated approaches continue to struggle with semantic consistency, complex mathematical reasoning, and the domain gap between natural language and formal specifications (Zheng et al., 2022; Welleck et al., 2022; Kirtania & Iyer, 2025). Current methods struggle to preserve precise semantic details when translating natural-language problems into Lean, particularly in advanced domains, limiting the availability of reliable formal reasoning benchmarks.

Another challenge is the evaluation of these autoformalization systems. Syntactic approaches like BLEU score (Papineni et al., 2002) do not consider the semantic distance between generations. Recent methods like Generalized Tree Edit Distance (GTED) (Liu et al., 2025a) computes structural similarity through operator trees, while Bidirectional Extended Definitional Equivalence (BEq) (Liu et al., 2025b) provides neural-symbolic equivalence checking. Both report high correlation with human evaluations. Other methods include combining certainty and similarity scores through contrastive learning (Lu et al., 2024) or provide standardized quality assessment based on human repair effort on a correction effort scale (0-4) (Jiang et al., 2023). However, scalable evaluation remains challenging, with manual expert verification being costly but necessary for establishing reliable ground truth.

Human-AI Collaboration in Formalization. As human verification remains a necessity, other Lean4 benchmarks like miniF2F, PutnamBench and ProofNet use completely manual annotation


Autoformalization Query
Suppose n ≥ 0 is an integer and all the roots of x^3 + αx + 4 − (2 x 2016^n) = 0 are integers. Find all possible values of α.
Knowledge Base for Mathlib
Below is Mathlib library documentation on some relevant packages that you might need to use for your task. This is not an exhaustive
...
## Algebra (Groups, Rings, Fields)
* **Groups:**
* Module: `Mathlib.Algebra.Group.Basic`
* Contains: Group axioms, group homomorphisms, basic theorems.
Autoformalizer LLMs
Formalized Theorems
Lean 4 statement T_m
Lean 4 statement T_1
Total m generations
Lean4 Compiler
Syntax checking
Semantic Notes
Navigation
Annotation Edit Space
LLM Generated Candidates and Notes
Human validation using Lean Annotator Dashboard, inbuilt Lean compilation

```
theorem inmo_2017_2 (n : ℕ) (α : ℤ) :
(∃ u v w : ℤ,
(X ^ 3 + C α * X + C (4 - 2 * (2016 : ℤ) ^ n))
= (X - C u) * (X - C v) * (X - C w)) →
α = (-3 : ℤ) := by
sorry
```

Figure 2: Overview of approach for creating IndicMath dataset: Human is assisted by a multiple LLM annotators. Each LLM generation is conditioned on the natural language and documentation, and goes through a validation check by Lean. Errors are provided as feedback in subsequent iterations. The final generations from all LLMs are summarized by an LLM in a dashboard to help optimize annotation efficiency.

for robust formalizations, the process is costly and non-scalable. However, the formalization of research-level mathematics increasingly relies on human-AI collaboration that combine automated translation capabilities with human expertise in both competition mathematics and theorem proving languages (Cohen et al., 2023; Yu et al., 2025). Process-supervised approaches use formal system feedback to improve translation quality while reducing annotation requirements. These hybrid methodologies outperform pure automation by combining human expertise for conceptual insights with AI capabilities for pattern recognition and verification.

3 IndiMathBench

IndiMathBench contains 312 formalized problem statements from Indian Olympiad problems. Each problem consists of an informal problem and an informal proof (in English) of an Olympiad math problem, a Lean4 theorem corresponding to that problem, and any numerical solution where applicable. The informal problems are compiled directly from the Regional Mathematical Olympiad (RMO) and Indian National Mathematical Olympiad (INMO) examinations in India. Table 1 gives the number of problems covering various mathematical domains namely geometry, algebra, number theory, and combinatorics. Figure 2 describes the overall human-AI collaborative procedure for formalization.

Category	Count
Geometry	98
Algebra	92
Set Theory & Combinatorics	45
Number Theory	77
Total	**312**

Table 1: Problem distribution by topic domain in IndiMathBench.

Diversity and Breadth. Compared to existing benchmarks like miniF2F (Zheng et al., 2021), which include a wide selection of high-school and early undergraduate mathematics problems (primarily from AMC, AIME, and IMO), the IndiMathBench focuses on problems from the Indian Mathematical Olympiad system. These problems are drawn from national and regional competitions in India, and are restricted to the high-school curriculum, covering algebra, number theory, geometry, and combinatorics. Unlike miniF2F, which incorporates problems involving topics such as inequalities, calculus, or matrix algebra, INMO/RMO problems exclude calculus and typically avoid higher-level abstractions.

Despite this narrower domain scope, INMO problems in particular demonstrate high internal diversity and depth. Many problems involve multi-step reasoning, uncommon constructions, and non-standard techniques. For example, geometric problems often require diagrammatic insight combined with multiple auxiliary constructions, while number-theoretic questions tend to involve clever use of parity, bounding, or invariant arguments.

Problem Domains. IndiMathBench problems are traditionally sourced from a fixed set of topics: algebra, euclidean geometry, elementary number theory, and combinatorics. Calculus, set theory, and linear algebra are not part of the official syllabus. This restriction makes the benchmark more uniform in scope, but allows for deeper exploration of problem-solving within each domain. For example, geometry problems frequently involve classical triangle centers (e.g., orthocenter, centroid) or cyclic quadrilaterals, which require nontrivial formalization in Lean. Geometry problems which are often under-represented by other datasets due to lack of representation in exams and the lack of support in the Lean + Mathlib ecosystem (Song et al., 2025). INDIMATHBENCH tackles these challenges head on by sourcing data geometry problem rich data and formalizing them without using euclidean coordinate representation.

Formalization Effort. The formalization was done over the course of a month by two annotators with prior experience in using Lean for formal proof writing. Every annotation was double checked by the other human annotator. In case of disagreements over the style or correctness of the annotations, the annotators discussed the theorem and came to a final answer. A common code style was followed to make the entire problem be represented within a single theorem statement as possible, and use similar constructions for similar concepts across the benchmark. AI was used extensively throughout the process and we discuss that in depth in Section 4.

Some problems (36%) in our set require solving for a value (rather than proving a statement). For these cases, we rephrase the problems to prove for the particular solution similar to prior works like MINIF2F, i.e. re-frame "solve Question Q for Solution X" as a "prove Question Q iff Solution X". An example for such problem can be seen in Appendix C. Each problem is expressed as a Lean theorem with the proof term replaced by `sorry`. In Lean, `sorry` is a placeholder that allows incomplete proofs to compile. There are other ways to formalize the problem in

4 AUTO-FORMALIZATION APPROACH

We present a scalable approach for using general-purpose LLMs to accelerate human annotation in the formalization of mathematical problems. This approach, which we used to create INDIMATHBENCH, is general-purpose and built for reusability. Algorithm 1 presents the complete pipeline from natural language problem statements to a final human annotation dashboard.

4.1 AUTOMATED FORMALIZATION GENERATION

In our initial evaluations of for autoformalization, the main deficiency observed was the poor quality of formulas written in custom formal languages. Models often hallucinate non-existent imports, mix syntax from other theorem provers (including Lean 3), and produce ill-formed code. Providing access to library snippets and documentation as feedback notably improves formula correctness, especially for geometry problems where mathlib lacks many competition-level constructs. This inability to consistently follow formal syntax motivates a structured process with documentation access and iterative feedback.

Algorithm 1 presents the pseudocode for our overall pipeline from problem to human-evaluable dashboard, comprising three key steps: (a) category-based retrieval, (b) iterative refinement with compiler feedback, and (c) multi-model ensemble generation with comparative analysis. All prompts can be found in Appendix C.3.

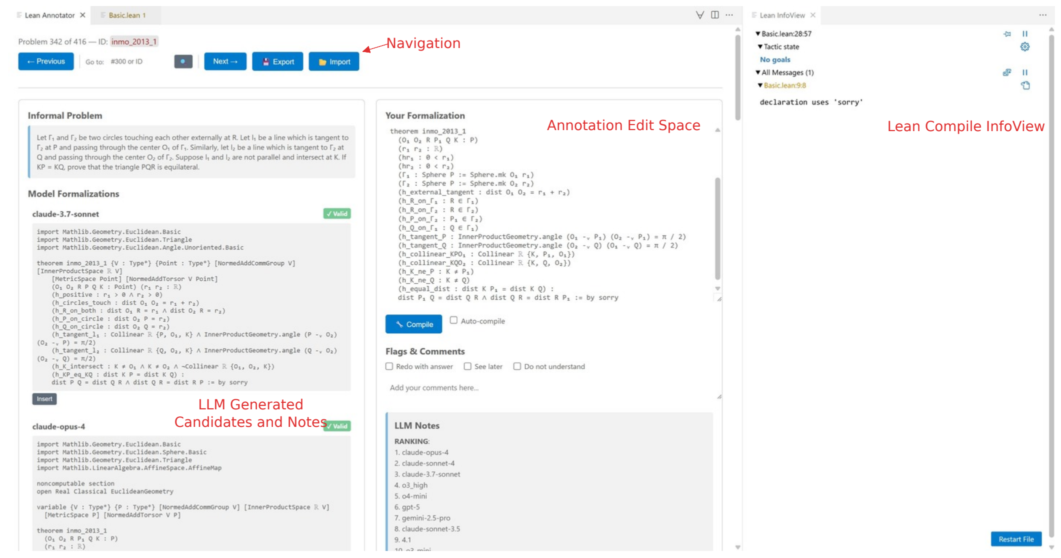
Step 1: Category-based Retrieval. LLMs often fail to import correct modules or use proper notation in Lean. To mitigate this, we augment prompts with automatically curated mathlib documentation. As described in Procedure `PreProcess` (Algorithm 1), each problem in the dataset $\mathcal{P}$ is labeled, using an LLM with human verification, into each one of the four categories listed in Table 1. We sample 25% of problems per category and use a Claude Sonnet 4–based agent with bash access to the mathlib repository. The agent explores relevant library code and extracts key definitions and formulas, forming a *static context* for each category. This context anchors the model with domain knowledge within each category, preventing hallucinations and syntax errors.

Step 2: Iterative Refinement with Error Feedback. Figure 2 illustrates our main formalization loop (Procedure `Formalize`). For each problem p, the LLM generates a formal theorem f conditioned on the corresponding static context. For "solve"-type problems, we also provide the solution and ask the LLM to generate a verifying theorem. The output is then compiled in Lean 4 via `ValidateInLean`; any parse or type errors are extracted and fed back to the model for correction. This refinement continues for up to six iterations, yielding syntactically valid Lean formalizations for 95.3% of problems.

Algorithm 1 LLM-Based Autoformalization of Mathematical Problems

```
1:  procedure FORMALIZE($p$, Model)
2:                                          ▷ $p$: problem description in NL
3:      $ctxt \leftarrow p.Cat.Ctxt$
4:      $f \leftarrow$ MODEL($p, ctxt$)
5:      for $i = 1$ to 6 do
6:          $errors \leftarrow$ VALIDATEINLEAN($f$)
7:          if $errors = \emptyset$ then
8:              break
9:          end if
10:         $feedback \leftarrow$ PARSEERRORS($errors$)
11:         $f \leftarrow$ MODEL($p, ctxt, f, feedback$)
12:     end for
13:     return $f$
14: end procedure
15: procedure MAIN($\mathcal{P}$, Categories)
16:     PREPROCESS($\mathcal{P}$, Categories)
17:     POSTDISPLAY($p$) for $p \in \mathcal{P}$
18: end procedure
```

```
1:  procedure PREPROCESS($\mathcal{P}$, Categories)
2:                                          ▷ $\mathcal{P}$ is a set of problems
3:      for all $p \in \mathcal{P}$ do
4:          $p.Cat \leftarrow$ Label($p$, Categories)
5:      end for
6:      for all $cat \in$ Categories do
7:          $cat.Samples \leftarrow$ Sample($\mathcal{P}, cat$)
8:          $cat.Ctxt \leftarrow$ Retriever($cat.Samples$)
9:      end for
10: end procedure
11: procedure POSTDISPLAY($p$)
12:     $\mathcal{F} \leftarrow \{\}$
13:     for all model $\in$ ModelList do
14:         $\mathcal{F}$.Add(FORMALIZE($p$, model))
15:     end for
16:     summary $\leftarrow$ MODEL($\mathcal{F}, p$)
17:     return $\mathcal{F}$, summary
18: end procedure
```



Problem 342 of 416 — ID: inmo_2013_1

Informal Problem

Let Γ₁ and Γ₂ be two circles touching each other externally at R. Let l₁ be a line which is tangent to Γ₂ at P and passing through the center O₁ of Γ₁. Similarly, let l₂ be a line which is tangent to Γ₂ at Q and passing through the center O₂ of Γ₂. Suppose l₁ and l₂ are not parallel and intersect at K. If KP = KQ, prove that the triangle PQR is equilateral.

Model Formalizations

claude-3.7-sonnet

```
import Mathlib.Geometry.Euclidean.Basic
import Mathlib.Geometry.Euclidean.Triangle
import Mathlib.Geometry.Euclidean.Angle.Unoriented.Basic

theorem inmo_2013_1 {V : Type*} {Point : Type*} [NormedAddCommGroup V]
[InnerProductSpace ℝ V]
    [MetricSpace Point] [NormedAddTorsor V Point]
    (O₁ O₂ R P Q K : Point) (r₁ r₂ : ℝ)
    (h_positive : r₁ > 0 ∧ r₂ > 0)
    (h_circles_touch : dist O₁ O₂ = r₁ + r₂)
    (h_R_on_both : dist O₁ R = r₁ ∧ dist O₂ R = r₂)
    (h_P_on_circle : dist O₂ P = r₂)
    (h_Q_on_circle : dist O₂ Q = r₂)
    (h_tangent_l₁ : Collinear ℝ {P, O₁, K} ∧ InnerProductGeometry.angle (P -ᵥ O₂)
(O₂ -ᵥ P) = π/2)
    (h_tangent_l₂ : Collinear ℝ {Q, O₂, K} ∧ InnerProductGeometry.angle (Q -ᵥ O₂)
(O₂ -ᵥ Q) = π/2)
    (h_K_intersect : K ≠ O₁ ∧ K ≠ O₂ ∧ ¬Collinear ℝ {O₁, O₂, K})
    (h_KP_eq_KQ : dist K P = dist K Q) :
    dist P Q = dist Q R ∧ dist Q R = dist R P := by sorry
```

claude-opus-4

```
import Mathlib.Geometry.Euclidean.Basic
import Mathlib.Geometry.Euclidean.Sphere.Basic
import Mathlib.Geometry.Euclidean.Triangle
import Mathlib.LinearAlgebra.AffineSpace.AffineMap

noncomputable section
open Real Classical EuclideanGeometry

variable {V : Type*} {P : Type*} [NormedAddCommGroup V] [InnerProductSpace ℝ V]
  [MetricSpace P] [NormedAddTorsor V P]

theorem inmo_2013_1
  (O₁ O₂ R P₁ Q K : P)
  (r₁ r₂ : ℝ)
```

Your Formalization

```
theorem inmo_2013_1
  (O₁ O₂ R P₁ Q K : P)
  (r₁ r₂ : ℝ)
  (hr₁ : 0 < r₁)
  (hr₂ : 0 < r₂)
  (Γ₁ : Sphere P := Sphere.mk O₁ r₁)
  (Γ₂ : Sphere P := Sphere.mk O₂ r₂)
  (h_external_tangent : dist O₁ O₂ = r₁ + r₂)
  (h_R_on_Γ₁ : R ∈ Γ₁)
  (h_R_on_Γ₂ : R ∈ Γ₂)
  (h_P_on_Γ₂ : P₁ ∈ Γ₂)
  (h_Q_on_Γ₁ : Q ∈ Γ₁)
  (h_tangent_P : InnerProductGeometry.angle (O₁ -ᵥ P₁) (O₂ -ᵥ P₁) = π / 2)
  (h_tangent_Q : InnerProductGeometry.angle (O₂ -ᵥ Q) (O₁ -ᵥ Q) = π / 2)
  (h_collinear_KPO₁ : Collinear ℝ {K, P₁, O₁})
  (h_collinear_KQO₂ : Collinear ℝ {K, Q, O₂})
  (h_K_ne_P : K ≠ P₁)
  (h_K_ne_Q : K ≠ Q)
  (h_equal_dist : dist K P₁ = dist K Q) :
  dist P₁ Q = dist Q R ∧ dist Q R = dist R P₁ := by sorry
```

Flags & Comments

☐ Redo with answer ☐ See later ☐ Do not understand

Add your comments here...

LLM Notes

RANKING:
1. claude-opus-4
2. claude-sonnet-4
3. claude-3.7-sonnet
4. o3_high
5. o4-mini
6. gpt-5
7. gemini-2.5-pro
8. claude-sonnet-3.5
9. 4.1

▼Basic.lean:28:57
▼Tactic state
No goals
▼All Messages (1)
▼Basic.lean:9:8

```
declaration uses 'sorry'
```

Figure 3: The Annotation Dashboard for the human expert to analyze various ranked options, modify and validate the final entry.

Step 3: Multi-Model Ensemble and Comparative Analysis. Finally, as outlined in Procedure PrepDashboard, each problem p is formalized by 12 state-of-the-art models (GPT, Claude, and reasoning variants). This ensemble provides redundancy against model-specific errors, supports comparative evaluation, and increases the likelihood of a correct formalization. The aggregated outputs are then summarized by GPT-5, which ranks models by correctness, completeness, and faithfulness to the problem statement. While this multi-model setup aids evaluation (Section 5.1), the same process can be used for multiple generations of a single model, analogous to self-consistency (Wang et al., 2023).

4.2 Human-AI Collaborative Annotation Dashboard

We develop a Human–AI collaborative dashboard, as illustrated in Figure 3, that enables efficient expert review of formalizations generated by large language models (LLMs). The interface consolidates all model outputs, validation results, and automated quality assessments into a single interactive workspace, allowing human annotators to efficiently evaluate and refine candidate formalizations.

The dashboard integrates compilation and diagnostic feedback similar to the VS Code Lean extension, allowing experts to compile any candidate directly and view precise error traces. It prioritizes displaying verified results—those that successfully compile and satisfy basic correctness checks—while still providing access to failed attempts for comprehensive analysis.

LLM Summaries. A distinctive feature is the inclusion of AI-generated summaries computed via the `summary` variable in Procedure `PrepDashboard`. These summaries synthesize insights across model generations, highlighting shared errors, missing conditions, and promising candidates. This guidance allows experts to focus their attention on genuinely ambiguous cases rather than routine validation, significantly reducing cognitive load. Annotators can also merge partial outputs—for example, incorporating a missing condition from one formalization into another—to efficiently produce the final, correct formula. Figure 7 shows a direct example of such a speed up.

We release this dashboard as an open-source VS Code extension for broader community use. Beyond our benchmark, it can support both informal-to-formal and formal-to-formal annotation tasks across diverse languages, facilitating scalable and high-quality dataset creation.

Human Guarantee. Although the benchmark is constructed through an elaborate AI pipeline, all 312 theorems in INDIMATHBENCH are manually verified for both correctness and stylistic quality. Every theorem is rechecked by the other annotator. Annotators observed that many generated formalizations, while valid Lean code, often omit important details of the original problems. Most annotations required editing at varying levels. Appendix A.1.3 illustrates the necessity of human oversight in creating robust benchmarks, showing an example where a minor error leads to an incorrect autoformalization.

4.3 ANNOTATOR EFFICIENCY STUDY

We conduct a small controlled study to evaluate the impact of LLM-generated candidates and group comparison summaries on informal to formal annotation efficiency for our case. The study compares three workflow settings:

1. **Full System** provides candidate generations, automated summary notes, and a human-in-the-loop dashboard. This is our proposed system.
2. **Masked Candidates** provides the dashboard with all candidate generations, but model identities are hidden and no summary notes are provided, simulating how much help multiple formalizations can give even without having an LLM group critique parts over a manual workflow.
3. **Manual Formalization** provides no LLM-generated content a-priori.

We asked a Lean formalization expert to formalize three sets of 12 RMO problems each (one set from each of three consecutive years, for a total of 36 problems). This design ensures consistent question styles within each set while providing coverage across problems with similar difficulty. A single annotator formalized three sets across each workflow setting. The annotator is allowed full internet and any AI Assistant access throughout every setting. We record the total time spent and the number of problems successfully formalized for comparison. All 36 formalizations were independently double-checked by a second annotator and verified to be correct.

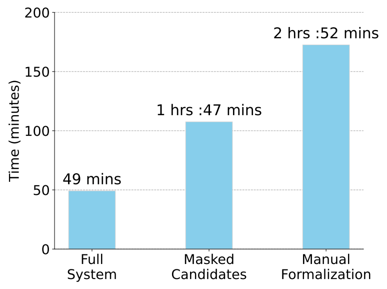


Figure 4: Annotation time for 12 RMO problems under the three workflows

Figure 4 compares annotation times across the three workflow variants. Manual formalization required an average of 14 minutes per problem, comparable to PutnamBench's 25 minutes and MINIF2F's 10 minutes. Using only multiple masked LLM-generated formalizations reduced this to 9 minutes per problem (a 1.6× speed-up), primarily because the annotator no longer needed to construct problem structures from scratch, and only verify and selectively edit partially correct generations. In contrast, much of the manual effort was spent navigating the mathlib library. With the full system, the average time dropped further to 4 minutes per problem—3.5× faster than manual formalization and 2.2× faster than the multi-generation variant. The annotator noted that when the top-ranked generation in

Model	BEq (312)	GTED Mean (over 312)	#GTED >0.9 (312)	Compile Success (312)
Claude Opus 4	**67**	**0.51**	**138**	**243**
Claude Sonnet 4	54	0.42	103	215
Gemini 2.5 Pro	44	0.24	58	151
GPT-5	36	0.48	124	235
o3 (high)	30	0.46	119	205
GPT-4.1	33	0.39	90	120

Table 2: Comparing models across BEq, GTED mean, # of samples with a GTED score >0.9, and compilation validity counts. GTED scores range from 0-1, higher value meaning better similarity.

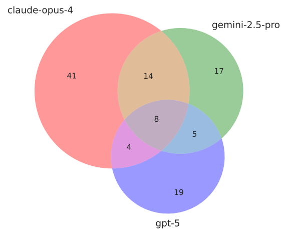


Figure 5: Venn diagram for BEq-passing problems for three different leading models.

the summary contained errors, the summary usually identified them, and the correct fragment was usually found in another generation. Figure 7 illustrates this speed-up in practice using a direct case.

5 Experimental Results

5.1 Autoformalization Evaluation

Setup. As described in Section 4.1, we generate candidate formalizations, given a natural language problem statement, across 12 different general-purpose frontier LLMs. These include Claude Sonnet 4, Claude Opus 4, o3 (high), GPT-5, and Gemini 2.5 Pro, among others. Here, we aim to measure how semantically close the generated formalizations are to the final human annotated formalizations.

Evaluation Metrics. Evaluating autoformalization quality presents unique challenges due to the rigorous logical nature of formal mathematical statements, where seemingly minor syntactic variations can alter meaning. To provide a comprehensive assessment, we employ two complementary evaluation metrics that have demonstrated high inter-annotator agreement with human evaluations.

- **Bidirectional Equivalence (BEq)** (Liu et al., 2025b) evaluates logical equivalence by attempting to prove each theorem using the other. Given two Lean 4 theorems in `sorry`-format, `theorem_A` and `theorem_B`, BEq employs a diverse set of heuristic and LLM-guided tactics to establish proofs in both directions. The formalization is deemed correct only if both directional proofs succeed, ensuring true logical equivalence rather than superficial syntactic similarity.
- **Generalized Tree Edit Distance (GTED)** (Liu et al., 2025a) measures syntactic similarity by representing Lean theorems as operator trees and computing the normalized cost of transforming the candidate theorem into the human-annotated ground truth. Scores range from 0 to 1 per comparison, with higher values indicating greater structural correspondence.

These metrics are contrasting in their approach: BEq captures semantic equivalence through the Lean proof engine's logical reasoning capabilities, while GTED quantifies syntactic structural similarity through tree transformations. This complementary evaluation framework provides both logical validation and structural analysis, offering a complete assessment of autoformalization quality.

For our evaluation, we use Claude Sonnet 4 (temperature 0.7) with pass@1 for BEq calculations, deviating from the original implementation's use of InternLM (Ying et al., 2024). All other implementation details follow the original specifications.

Results. Table 2 depicts the models evaluated against the BEq and GTED metrics. The model Claude Opus 4 does best across metrics. Figure 5 depicts the overlap on the benchmarks that Claude Opus 4, Gemini 2.5 Pro and GPT-5 solve (based on BEq metric). This shows that the models have certain complementary abilities and justifies our ensemble approach. In 160 of the 312 problems, at least one generation passed the BEq check, i.e. for 51.2% of our dataset an LLM had formalized the problem equivalent to ground truth annotation. Another notable detail is that among the three models, Claude Opus 4, Gemini 2.5 Pro and GPT-5, with a cumulative BEq passing for 108 problems, only 12 were from Geometry. This highlights the LLM's difficulty with using Mathlib's lacking

Lean Compilation Pass Rate		
Model	ZS (%)	Doc+FB (%)
Claude Opus 4	4.1	**74.5**
GPT-5	**30.5**	71.4
Claude Sonnet 4	3.6	65.4
o3 (high)	22.1	63.2
Gemini 2.5 Pro	3.8	44.2

Table 3: The percent of Lean-validated formulas generated by the different models in zero-shot (ZS) setting and in the setting with documentation (Doc) and feedback (FB) loops.

Model	Success Rate pass@1		
	Single Turn IMB	10 Turns IMB	10 Turns PB
GPT-4.1	0/312	-	-
o3 (medium)	1/312	-	-
Claude Sonnet 4	1/312	4/312	-
Gemini 2.5 Pro	0/312	12/312	-
GPT-5	1/312	**36**/312	42/660

Table 4: Success Rates, all pass@1, of various frontier models on INDIMATHBENCH. Success Rates here refer to Lean Verifiable proofs submitted by the models. IMB: INDIMATHBENCH, PB: PutnamBench

support for Olympiad style geometry. Appendix B discusses specific insights from different models in greater detail.

5.2 INDIMATHBENCH TECHNIQUE EVALUATION - ABLATIONS

Ablation for Context and Feedback: We compare our `Formalize` procedure (Algorithm 1) against a zero-shot baseline that generates Lean 4 formulas without additional context or iterative feedback. Table 3 reports the percent of problems (312 total) successfully formalized (i.e., passing Lean validation) for both approaches across multiple state-of-the-art language models. We note here that this is only a syntactic notion of theorem correctness, which is necessary, but not sufficient, for full correctness. The results demonstrate that error feedback and documentation retrieval consistently improve success rates across all models. Claude models show particularly notable improvement, starting with low initial compilation success but rapidly refining through available Lean environment knowledge and retrieval context. We use the documentation + feedback generations for our final annotations and evaluations. Additional ablation details are provided in Appendix D.

5.3 AUTOMATED THEOREM PROVING EVALUATION

We measure the difficulty INDIMATHBENCH poses to frontier general purpose LLMs for formal theorem proving, by testing them for completing the theorem statement with a valid proof. We evaluate success rate in single-turn and 10-turn settings with Lean compiler feedback. All results are reported on pass@1 metric. We carry out these evaluations across the best frontier models available to us, Claude Sonnet 4, GPT-5, GPT-4.1, o3 (medium), and Gemini 2.5 Pro, at default parameters. In single turn mode, only one problem was solved (inmo_2015_5, requiring Mathlib's pitot_theorem) by Claude Sonnet 4, GPT-5, and o3 (medium) each, across the 312 strong INDIMATHBENCH. With 10 turns and iterative refinement in a ReAct (Yao et al., 2023) like procedure, GPT-5 achieves 36/312 (11%) on INDIMATHBENCH and 42/660 (7%) on PutnamBench, demonstrating comparable difficulty between the two benchmarks. Notably, no geometry problems were solved by any model. The multi-turn construction used here follows chain-of-thought based practices established in prior systems (Chen et al., 2025; Shen et al., 2025; Ji et al., 2025). Detailed setup and results described in Appendix E.

6 CONCLUSION

We introduce INDIMATHBENCH, a new benchmark of human-verified Lean 4 formalizations for Olympiad-level mathematics problems. The dataset is designed to evaluate the limits of large language models in formal reasoning, where our comparative analysis shows that even frontier models solve only a single problem in single-turn settings and reach just 11% success with iterative refinement. These results highlight the substantial gap between current LLM capabilities and the rigor required for formal mathematics, and the dataset complexity. To support progress in this direction, we further present a structured annotation framework that leverages documentation retrieval, compiler feedback, and multi-model aggregation to streamline manual formalization. We release both the dataset and an integrated VS Code dashboard extension as open resources to advance research in neural theorem proving and autoformalization.

REFERENCES

Ayush Agrawal, Siddhartha Gadgil, Navin Goyal, Ashvni Narayanan, and Anand Tadipatri. Towards a Mathematics Formalisation Assistant using Large Language Models. *arXiv preprint arXiv:2211.07524*, 2022a.

Ayush Agrawal, Abhijeet George, Divy Vyas, and Avijit Balasubramanian. A survey on deep learning approaches for mathematics word problem solving. *arXiv preprint arXiv:2212.10535*, 2022b.

Zhangir Azerbayev, Keiran Paster, Hailey Schoelkopf, Sean Welleck, et al. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Yuchen Wu, Yihang Xia, Huajian Xin, Fan Yang, Huaiyuan Ying, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang, Ge Zhang, Tianyun Zhao, Jianqiu Zhao, Yichi Zhou, and Thomas Hanwen Zhu. Seed-prover: Deep and broad reasoning for automated theorem proving, 2025. URL `https://arxiv.org/abs/2507.23726`.

Nathanael Cohen et al. Towards automated mathematical reasoning. *arXiv preprint arXiv:2304.14565*, 2023.

FrontierMath Collaboration. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024.

Siddhartha Gadgil, Anand Rao Tadipatri, Ayush Agrawal, Ashvni Narayanan, and Navin Goyal. Towards Automating Formalisation of Theorem Statements using Large Language Models. In *International Conference on Neural Information Processing Systems Workshop on MATH-AI*, 2022.

Yichen Huang and Lin F. Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline, 2025. URL `https://arxiv.org/abs/2507.15855`.

Xingguang Ji, Yahui Liu, Qi Wang, Jingyuan Zhang, Yang Yue, Rui Shi, Chenxi Sun, Fuzheng Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover-v2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning, 2025. URL `https://arxiv.org/abs/2507.08649`.

Albert Q Jiang, Wenda Li, and Mateja Jamnik. Evaluating language model autoformalization: A case study in Lean 4. *arXiv preprint arXiv:2311.09101*, 2023.

Minhao Jiang, Ken Ziyu Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. Investigating data contamination for pre-training language models, 2024. URL `https://arxiv.org/abs/2401.06059`.

Shashank Kirtania and Arun Iyer. Steering llms for formal theorem proving, 2025. URL `https://arxiv.org/abs/2502.15507`.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving, 2025. URL `https://arxiv.org/abs/2502.07640`.

Zhenwen Liu, Jingyi Chen, Xuanming Song, Ruocheng Shu, and Jiaqi Yuan. Generalized tree edit distance for mathematical expression comparison. *arXiv preprint arXiv:2501.00001*, 2025a.

Zhenwen Liu, Tianyi Wang, Yifan Li, Xiaofan Zhang, and Zhengying Lu. BEq: Bidirectional extended definitional equivalence for mathematical statement assessment. *arXiv preprint arXiv:2501.00002*, 2025b.

Jianqiao Lu, Zhengying Zhan, Yuhui Xiao, Zhiqing Huang, and Quanshi Gu. FormalAlign: Automated alignment evaluation for autoformalization. *arXiv preprint arXiv:2407.15156*, 2024.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *Proceedings of the International Conference on Automated Deduction*, 2021.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the annual meeting of the Association for Computational Linguistics*, 2002.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025.

Ziju Shen, Naohao Huang, Fanyi Yang, Yutong Wang, Guoxiong Gao, Tianyi Xu, Jiedong Jiang, Wanyi He, Pu Yang, Mengzhou Sun, Haocheng Ju, Peihao Wu, Bryan Dai, and Bin Dong. Realprover: Retrieval augmented lean prover for mathematical reasoning, 2025. URL `https://arxiv.org/abs/2505.20613`.

Chendong Song, Zihan Wang, Frederick Pu, Haiming Wang, Xiaohan Lin, Junqi Liu, Jia Li, and Zhengying Liu. Leangeo: Formalizing competitional geometry problems in lean, 2025.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *arXiv preprint arXiv:2407.11214*, 2024. URL `https://arxiv.org/abs/2407.11214`.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning, 2025. URL `https://arxiv.org/abs/2509.22819`.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. URL `https://arxiv.org/abs/2504.11354`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL `https://arxiv.org/abs/2203.11171`.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. Naturalprover: Grounded mathematical proof generation with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=rhdfTOiXBng`.

Yuhuai Wu, Albert Q Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with Large Language Models. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. Internlm-math: Open math large language models toward verifiable reasoning, 2024. URL `https://arxiv.org/abs/2402.06332`.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. Formalmath: Benchmarking formal mathematical reasoning of large language models, 2025.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=9ZPegFuFTFv`.

A IndiMathBench

Problem Domains. IndiMathBench problems are traditionally sourced from a fixed set of topics: algebra, euclidean geometry, elementary number theory, and combinatorics. Calculus, set theory, and linear algebra are not part of the official syllabus. This restriction makes the benchmark more uniform in scope, while still allowing for deeper exploration of problem-solving within each domain. However, the problems—especially from INMO—demonstrate remarkable internal diversity and depth. For instance, geometry problems frequently involve classical triangle centers (e.g., orthocenter, centroid) or cyclic quadrilaterals, which often demand nontrivial formalization in Lean and combine diagrammatic insight with auxiliary constructions. Similarly, number-theoretic questions tend to hinge on clever use of parity, bounding, or invariant arguments, reflecting the benchmark's emphasis on depth rather than breadth.

Underrepresented Domains. Domains like geometry, combinatorics, and counting are under-formalized even in larger datasets (Zheng et al., 2022; Tsoukalas et al., 2024; Song et al., 2025; Yu et al., 2025; Ren et al., 2025). Another reason for selecting RMO and INMO for formalization was their unique syllabus and domain compared to other exams around covered in literature.

RMO and INMO exams. The Regional Mathematics Olympiad and the Indian National Mathematics Olympiad exams are held annually, and are used to select India's most promising high-school student. Students who pass the RMO qualify to take the INMO, which is a significantly more difficult national-level test. Students typically need to pass another exam before qualifying for RMO. Around 7000 students qualify to write the RMO, and 1000 to write the INMO every year from all over India.

A.1 Example formalizations from IndiMathBench

A.1.1 Geometry problem formalization

Consider the following problem statement:

In a triangle ABC, let D be a point on the segment BC such that AB + BD = AC + CD. Suppose that the points B, C and the centroids of triangles ABD and ACD lie on a circle. Prove that AB = AC.

Its formalization in IndiMathBench:

```
variable {V : Type*} {P : Type*} [NormedAddCommGroup V]
  [InnerProductSpace ℝ V] [MetricSpace P] [NormedAddTorsor V P]

theorem inmo_2014_1 (A B C D : P)
  (hD : ∃ t : ℝ, 0 ≤ t ∧ t ≤ 1 ∧ D = AffineMap.lineMap B C t)
  (h_sum : dist A B + dist B D = dist A C + dist C D)
  (h_concyclic : Concyclic {B, C,
    centroid ℝ {A, B, D} id,
    centroid ℝ {A, C, D} id}) :
  dist A B = dist A C := by sorry
```

A.1.2 Solve type problem formalization

There are other methods of formalization which do not include explicitly mentioning the final answer in case of solve type problems (Tsoukalas et al., 2024), however we only support the question conversion type formalization in this release.

Consider the following problem description.

Suppose $n \geq 0$ is an integer and all the roots of $x^3 + \alpha x + 4 - (2 \cdot 2016^n) = 0$ are integers. Find all possible values of α.

This is a problem that requires a solution. We assume a solution is given. For such cases, the formula states that the given solution is actually correct.

```
import Mathlib.Data.Int.Basic
import Mathlib.Algebra.Polynomial.Basic

open Polynomial

theorem inmo_2017_2 (n : ℕ ) ( α : ℤ ) :
    ( ∃ u v w : ℤ ,
      (X ^ 3 + C α * X + C (4 - 2 * (2016 ^ n))
        = (X - C u) * (X - C v) * (X - C w)) → α = (-3 : ℤ ) := by
  sorry
```

A.1.3 High precision required in formalizations

We list an example formalization where small differences in the theorem statement makes the entire problem unprovable. The left was generated by multiple LLMs, and the right was the human annotation after making the correction of using ℕ+ to define positive natural numbers in Lean.

Natural numbers in the classical (informal) notation maths are denoted by the symbol ℕ. Usually, and in the context of this question, 0 is not included. However in Lean 4, ℕ denotes the set of natural numbers the begin with 0. This distinction plays an important role, since if m = n = 0, condition (b) becomes “0 divides f(0) + f(0)”, which is undefined.

Such an error would not be bought up by the lean compiler and could slip by easily without careful annotations.

Let ℕ denote the set of all natural numbers and let f : ℕ → ℕ be a function such that
(a) f(mn) = f(m)f(n) for all m,n in ℕ;
(b) m + n divides f(m) + f(n) for all m,n in ℕ.
Prove that there exists an odd natural number k such that f(n) = n∧k for all n in ℕ.

```
import Mathlib

theorem inmo_2018_6 :
  ∀ (f : ℕ+ → ℕ+),
  (∀ m n :
    ℕ+, f (m * n) = f m * f n)→
  (∀ m n :
    ℕ+, (m + n) ∣ (f m + f n))→
  (∃ k : ℕ, k % 2 = 1 ∧
  ∀ n : ℕ+, f n = n^k) := by sorry
```

```
import Mathlib

theorem inmo_2018_6 :
  ∀ (f : ℕ → ℕ),
  (∀ m n :
    ℕ, f (m * n) = f m * f n)→
  (∀ m n :
    ℕ, (m + n) ∣ (f m + f n))→
  (∃ k : ℕ, k % 2 = 1 ∧
  ∀ n : ℕ, f n = n^k) := by sorry
```

B Autoformalization Evaluation

Table 5 shows the BEq and GTED scores for all 12 models. Figure 6 shows the heatmap among different models (y-axis) with respect to their GTED scores across the 312 comparisons, with a lighter overall hue of what ratio of comparisons pass various thresholds (x-axis).

B.1 Model failure modes in Autoformalization

Based on empirical evaluation using IndiMathBench, a benchmark of 312 formally verified Lean4 theorems from Indian Mathematical Olympiad problems, we analyze systematic failure patterns across major model families in autoformalization tasks.

B.1.1 Anthropic Claude Model Family

The Claude 4 series demonstrates the strongest overall performance in autoformalization metrics, with Claude Opus 4 achieving the highest BEq score (67/312) and compilation success rate (243/312).

Model	**BEq (312)**	**GTED Mean (over 312)**	**#GTED >0.9 (312)**	**Compile Success (312)**
Claude Opus 4	**37**	**0.51**	**138**	**243**
Claude Sonnet 4	36	0.42	103	215
Gemini 2.5 Pro	35	0.24	58	151
Claude Sonnet 3.7	32	0.29	64	171
Claude Sonnet 3.5	31	0.27	63	162
GPT-5	31	0.48	124	235
o3_high	29	0.46	119	205
o4-mini	29	0.32	73	165
GPT-4.1	25	0.39	90	120
o3-mini	21	0.35	84	142
GPT-4o	10	0.27	59	101
o1-mini	7	0.34	79	44

Table 5: Comparing all 12 models across BEq, GTED mean, # of samples with a GTED score >0.9, and compilation validity counts.

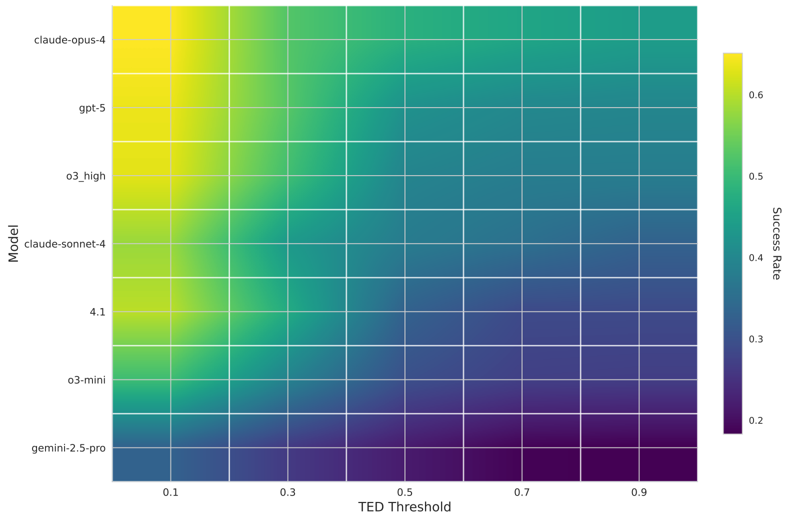


Figure 6: Heatmap for GTED based on thresholds. Better models see a lighter hue.

Claude Opus 4. Claude Opus 4 leads across multiple evaluation metrics with 67 BEq-passing formalizations and 0.512 mean GTED score, representing the current state-of-the-art in mathematical autoformalization. Characteristic strengths and limitations:

- *Superior Semantic Accuracy*: Achieves highest BEq logical equivalence with ground truth formalizations
- *Compilation Reliability*: 77.9% of generations compile successfully in Lean4
- *Iterative Refinement Capability*: Shows dramatic improvement from 17 to 310 successful compilations when provided with error feedback and knowledge base access
- *Domain-Specific Challenges*: Particularly struggles with geometry problems, contributing only 6 of 67 successful formalizations

Sample Claude Opus 4 autoformalization which is also semantically correct:

```
theorem inmo_2014_2 (n : ℕ) :
  Even ((Finset.sum (Finset.range n)
    fun i => ⌊(n : ℝ) / ((i + 1) : ℝ)⌋) + ⌊Real.sqrt n⌋) := by
sorry
```

Claude Sonnet 4. Demonstrates consistent performance with 54 BEq successes and more conservative formalization approaches, showing strong baseline capabilities with room for refinement. Characteristic strengths and limitations:

- *Strong Improvement with Support*: Only 15 successful compilations without assistance, increases to 272 successful compilations with knowledge base and feedback
- *Structural Similarity*: Achieves 0.419 mean GTED score, indicating reasonable syntactic accuracy

B.2 OpenAI GPT Model Family

The GPT series shows varied performance across different model variants, with GPT-5 achieving moderate success but significant challenges in zero-shot scenarios.

GPT-5. Despite being a one of the best general purpose model, GPT-5 achieves only 36 BEq successes, demonstrating the complexity of mathematical formalization tasks. Performance analysis:

- *Strong Zero-Shot Compilation*: 127 successful compilations without assistance, highest among all models
- *Knowledge Base Responsiveness*: Improves to 297 successful compilations with documentation and feedback
- *Semantic Gaps*: Despite compilation success, only 11.5% achieve BEq semantic equivalence with ground truth
- *High Structural Similarity*: 0.475 mean GTED score suggests good syntactic understanding

GPT-4.1. Shows the weakest overall performance among evaluated models, particularly in zero-shot scenarios. Performance analysis:

- *Poor Zero-Shot Performance*: Only 58 successful compilations without assistance
- *Limited Semantic Understanding*: 33 BEq successes despite reasonable compilation rates
- *Lowest Mean GTED*: 0.392 structural similarity score indicates syntactic challenges

B.3 OpenAI o-Series Models

The o3 reasoning model demonstrates interesting patterns, showing strong zero-shot compilation but moderate semantic accuracy.

o3 (High Reasoning). Achieves balanced performance with distinctive reasoning-based approach to formalization. Performance analysis:

- *Strong Zero-Shot Reasoning*: 92 successful compilations without documentation
- *Improvement with Context*: Increases to 263 successful compilations with knowledge base
- *Semantic Accuracy Challenges*: Only 30 BEq successes despite compilation strength
- *Reasoning Overhead*: Sometimes refuses to formalize when uncertain, as evidenced by responses stating inability to find solutions

B.4 Google Gemini

Gemini 2.5 Pro shows the weakest performance among frontier models, despite known for being very good at informal mathematical reasoning (Huang & Yang, 2025; Varambally et al., 2025), particularly struggling with Lean4-specific syntax and mathematical library integration.

Gemini 2.5 Pro Demonstrates significant challenges across all evaluation metrics with only 44 BEq successes. Performance analysis:

- *Poor Zero-Shot Performance*: Only 16 successful compilations without assistance
- *Limited Improvement Capacity*: Increases to only 184 compilations with full support system
- *Lowest Semantic Accuracy*: 14.1% BEq success rate among successful compilations

- *Structural Understanding Gaps*: 0.236 mean GTED score indicates fundamental syntactic challenges

Gemini 2.5 Pro's performance highlights how mathematical reasoning and formal notation in Lean are two different tasks.

B.5 Cross-Family Universal Patterns

B.5.1 Zero-Shot vs. Assisted Performance Gap

All model families show dramatic performance improvements when provided with domain-specific knowledge base and iterative error feedback, with improvement ratios ranging from 2.3x (GPT-5) to 18.2x (Claude Opus 4).

B.5.2 Theorem Proving vs. Formalization Gap

Despite varying formalization capabilities, all models achieve nearly identical theorem proving performance on single turn, with only Claude Sonnet 4, GPT-5, and O3 (medium) successfully proving a single theorem (inmo_2015_5) that relies on existing Mathlib theorems. However, over multiturn, GPT-5 gives performance better than many finetuned (for Lean notation) models over PutnamBench (Lin et al., 2025; Wang et al., 2025).

B.5.3 Geometry Domain Challenges

Across all model families, geometry problems present the greatest formalization challenge. Among 108 successful BEq formalizations from the top three models (Claude Opus 4, Gemini 2.5 Pro, GPT-5), only 12 originated from geometry problems, indicating systematic difficulties with spatial reasoning and coordinate-free geometric formalization.

B.5.4 Compilation vs. Semantic Correctness

A consistent pattern emerges where compilation success significantly exceeds semantic correctness across all families. Claude Opus 4's 77.9% compilation rate yields only 21.5% semantic accuracy, highlighting the challenge of generating syntactically valid but semantically incorrect formalizations. Example of a half incomplete formalization that passes Lean compiler:

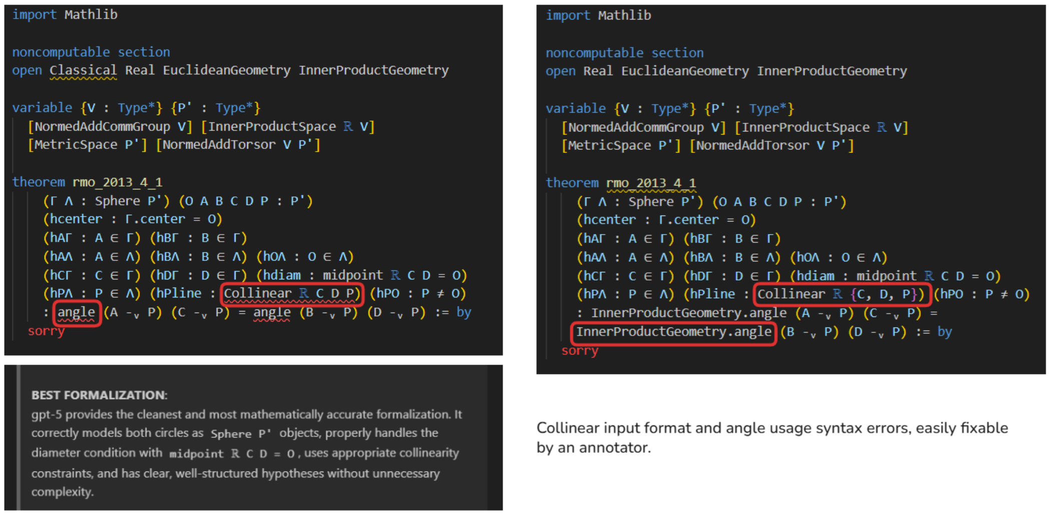
```
-- Typical geometry problem requiring complex formalization:
theorem triangle_centroid_property (A B C D : P)
  (hD : ∃ t : ℝ, 0 ≤ t ∧ t ≤ 1 ∧ D = AffineMap.lineMap B C t)
  (h_sum : dist A B + dist B D = dist A C + dist C D)
  (h_concyclic : Concyclic {B, C,
    centroid ℝ {A, B, D} id,
    centroid ℝ {A, C, D} id}) :
  dist A B = dist A C := by sorry
-- All models struggle with geometric abstractions like centroids and
concyclic points
```

B.6 Implications for Mathematical AI Development

The empirical findings reveal that current autoformalization capabilities remain limited despite advances in general reasoning. The consistent pattern of strong compilation performance coupled with weak semantic accuracy suggests that models can learn Lean4 syntax but struggle with mathematical meaning preservation. The dramatic improvement from iterative feedback indicates that human-AI collaborative approaches may be more promising than fully automated formalization for complex mathematical domains.

C Experimental Setup

All formalizations and experiments were conducted using the Lean theorem prover, version 4.22. We relied on the `mathlib` library at commit



```
import Mathlib

noncomputable section
open Classical Real EuclideanGeometry InnerProductGeometry

variable {V : Type*} {P' : Type*}
  [NormedAddCommGroup V] [InnerProductSpace ℝ V]
  [MetricSpace P'] [NormedAddTorsor V P']

theorem rmo_2013_4_1
    (Γ Λ : Sphere P') (O A B C D P : P')
    (hcenter : Γ.center = O)
    (hAΓ : A ∈ Γ) (hBΓ : B ∈ Γ)
    (hAΛ : A ∈ Λ) (hBΛ : B ∈ Λ) (hOΛ : O ∈ Λ)
    (hCΓ : C ∈ Γ) (hDΓ : D ∈ Γ) (hdiam : midpoint ℝ C D = O)
    (hPΛ : P ∈ Λ) (hPline : Collinear ℝ C D P) (hPO : P ≠ O)
    : angle (A -ᵥ P) (C -ᵥ P) = angle (B -ᵥ P) (D -ᵥ P) := by
  sorry
```

BEST FORMALIZATION:
gpt-5 provides the cleanest and most mathematically accurate formalization. It correctly models both circles as `Sphere P'` objects, properly handles the diameter condition with `midpoint ℝ C D = O`, uses appropriate collinearity constraints, and has clear, well-structured hypotheses without unnecessary complexity.

```
import Mathlib

noncomputable section
open Real EuclideanGeometry InnerProductGeometry

variable {V : Type*} {P' : Type*}
  [NormedAddCommGroup V] [InnerProductSpace ℝ V]
  [MetricSpace P'] [NormedAddTorsor V P']

theorem rmo_2013_4_1
    (Γ Λ : Sphere P') (O A B C D P : P')
    (hcenter : Γ.center = O)
    (hAΓ : A ∈ Γ) (hBΓ : B ∈ Γ)
    (hAΛ : A ∈ Λ) (hBΛ : B ∈ Λ) (hOΛ : O ∈ Λ)
    (hCΓ : C ∈ Γ) (hDΓ : D ∈ Γ) (hdiam : midpoint ℝ C D = O)
    (hPΛ : P ∈ Λ) (hPline : Collinear ℝ {C, D, P}) (hPO : P ≠ O)
    : InnerProductGeometry.angle (A -ᵥ P) (C -ᵥ P) =
    InnerProductGeometry.angle (B -ᵥ P) (D -ᵥ P) := by
  sorry
```

Collinear input format and angle usage syntax errors, easily fixable by an annotator.

Figure 7: Minimal edits needed to fix a broken, but mathematically sound generation, as pointed out by the LLM summary. Reducing human effort by a margin.

`f858fcc3b49c546705ba7d79c58217e85aaa5f0e` to ensure reproducibility and consistency across our proofs and auxiliary results.

Our computational environment consisted of:

- **Hardware:** 8-core CPU, 32 GB RAM
- **OS:** Windows 11 Pro (64-bit)
- **Lean Toolchain:** Installed via `elan`, with `lake` for project management

All proofs were compiled and verified using Lean's native `lake build` system without additional modifications to `mathlib`. The experiment scripts, proof files, and configuration details are provided in the supplementary material to facilitate full reproducibility.

C.1 Impact of LLM Notes

In this section we discuss the impact of LLM generated notes to reduce effort of annotation. While our primary focus is on improving proof generation through activation steering, we observe an additional benefit: the method can substantially reduce the human effort required for data annotation and theorem formalization. Figure 7 illustrates a representative example where steering produces mathematically sound code that requires only minimal corrections to become a valid formal proof.

In this example, the model generates a Lean theorem about geometric relationships in Euclidean space. The initial generation contains two primary issues: incorrect collinear input formatting and improper angle usage syntax. However, the mathematical reasoning underlying the proof is correct, as confirmed by the model's own natural language summary stating that "gpt-5 provides the cleanest and most mathematically accurate formalization." The required fixes are straightforward syntactic corrections that can be applied by domain experts with minimal effort.

This pattern suggests that activation steering not only improves proof success rates but also generates "near-miss" proofs that are closer to correctness than baseline outputs. Rather than producing completely invalid formal statements, steered models tend to generate mathematically coherent structures with localized syntax errors. This property significantly reduces the annotation burden for creating training datasets, as human experts can focus on lightweight editing rather than complete theorem reconstruction.

The implications for scalable dataset creation are substantial. Traditional approaches to building formal mathematics datasets require expert mathematicians to write complete formalizations from scratch—a time-intensive process that limits dataset scale. Our approach enables a more efficient workflow: models generate candidate formalizations that capture the essential mathematical content, while human experts provide targeted corrections to syntax and edge cases. This collaborative paradigm could accelerate the development of large-scale formal mathematics corpora needed to train more capable theorem-proving systems.

We note that this benefit emerges naturally from our steering methodology rather than being explicitly optimized for. The fact that informal reasoning guidance leads to more structured, correctable outputs suggests that the underlying activation patterns encode not just proof search strategies, but also adherence to formal syntax conventions. This observation warrants further investigation in future work focused specifically on human-AI collaborative formalization workflows.

C.2 Knowledge Base Learning Prompt

C.2.1 System Message

```
You are a mathematical documentation agent specializing in the Lean 4
Mathlib library. Your job is to explore the Mathlib repository and create
 a concise, practical documentation summary focused on mathematical
formalization and theorem proving. You have been given access to a yet
unreleased version of this library, which you must go through and pick
out all relevant imports based on the type of problem the user is trying
to solve. The repository contains a comprehensive library of formalized
mathematics for Lean 4.
The repository will have file names and folder names representative of
its content.

Your every response must be a tool call.

The documentation will be used by new lean users, who will use it as a
guide to write all their imports, writing notations and rely solely on it
 to make the correct imports.

WORKFLOW:
1. Use run_bash to explore the repository (ls, cd, cat, grep, find, etc.)
2. Take notes by writing to files in {working_directory}. You are
currently at this directory. Please do not make any changes outside of
this directory, or delete any existing file.
    i.      First read all the given examples, and create a list keywords
, such that each keyword is a concept that appears in any question.
    ii.     Keywords should also include common patterns like how to
express "point lies on line segment", "lines are parallel/perpendicular",
 ratios and divisions of segments.
    iii.    Add these keywords to your notes file, so you can refer to
them for completion later on.
    iv.     Understand the kind of problems the documentation needs to
deal with, and select what goes in accordingly.
3. When you have sufficient information, use final_submit with a complete
 documentation string

EXPLORATION STRATEGY:
- Examine the main mathematical domains asked by the user
- Look for key theorem statements and their dependencies
- Pay attention to naming conventions and mathematical abstractions
- Use the given sample of examples to understand what parts to focus on
- Look for file names, folder names, documentation, examples, source code
 to know their subject
- Focus on user-facing functionality
- Use {working_directory} for any notes (absolute paths since you'll be
changing directories)
```

```
- You decide when you have enough information to create the final
documentation

FINAL DOCUMENTATION FORMAT:
Organize your final output into exactly these 4 sections:

## 1. Installation & Import
- How different imports are situated in the mathlib file hierarchy
- Essential import statements for different mathematical domains
- Any setup requirements, like opening some namespace for certain symbols
, literals, notations or declarations.

## 2. Available Namepaces and Symbols
- Group related functionality together
- Since you will be given a field by the user, focus only on that and
related thing you see in the examples
- Important theorem statements in each subdomain
- Common mathematical objects and their properties
- Exhaustive list of all the functions avaliable for use

## 3. Minimal Usage Example
- Simple theorem statement (with sorry, ignore proofs)
- Basic mathematical definitions
- Make some imports, and open some namespaces and scopes
- All sample codes **must** be complete and well explained, or else it
can confuse the readers on what a complete theorem code looks like
- Do not leave parts of example code as comments
- Give examples for the kind of stuff the reader will be dealing with
when trying to formalize the problem statement
- Lean has difficult type setups, so be sure to explain those with
examples
- Should work out of the box

## 4. Common Pitfalls & Gotchas
- Common mistakes when formalizing mathematics
- Type class resolution issues
- Mathematical notation vs. Lean syntax differences

## 5. Key Files Structure
- An ascii directory tree of all the important/related files and packages

If some concept appears even once in the examples, make sure to cover
that in your documentation. It should be **complete**, don't skip
concepts randomly.
Do not be afraid to make long if it needs to be.

Remember: Your goal is to create a practical cheat sheet that gets
developers productive quickly. It is okay if its long as long as we are
putting relevant information and are correct.
```

C.2.2 User Message

```
Problem Description: I want to understand what all library modules are
available to me for autoformalizing **Set Theory & Combinatorics**
olympiad like problem statements into lean 4. I only care about
autoformalizing the theorem part, so things like tactics and everything
related to solving the problem are unnecessary. Only things relevant to
the theorem statement are useful. I am interested in:
```

```
- All the necessary and relevant imports, their correct paths
- How to open the correct namespace or scope to use particular symbols or
 literals in lean
- Examples of using them
- Other things to note
I'll attach some examples of the type of questions I am trying to write
as a lean theorem.

Examples: Samples of the kind of questions whose autoformalization I'll
be doing:
- All the 7-digit numbers containing each of the digits 1, 2, 3, 4, 5, 6,
 7 exactly once, and not divisible by 5, are arranged in the increasing
order. Find the 2000-th number in this list.
- Prove that the number of triples $(A, B, C)$ where $A, B, C$ are
subsets of $\{1, 2, \ldots , n\}$ such that $A \cap B \cap C = \emptyset$
, $A\cap B \neq \emptyset$, $B\cap C\neq \emptyset$ is $7^n - 2\cdot 6^n
+ 5^n$.
- Let $S = \{1, 2, \ldots , n\}$ and let $T$ be the set of all ordered
triples of subsets of $S$, say $(A1, A2, A3)$, such that $A1 \cup A2 \cup
 A3 = S$. Determine, in terms of $n$, $\sum_{(A1,A2,A3)\in T} |A1 \cap A2
 \cap A3|$ where $|X|$ denotes the number of elements in the set $X$.
- There are 100 countries participating in an olympiad. Suppose \(n\) is
a positive integer such that each of the 100 countries is willing to
communicate in exactly \(n\) languages. If each set of 20 countries can
communicate in at least one common language, and no language is common to
 all 100 countries, what is the minimum possible value of \(n\)?
- A box contains answer 4032 scripts out of which exactly half have odd
number of marks. We choose 2 scripts randomly and, if the scores on both
of them are odd number, we add one mark to one of them, put the script
back in the box and keep the other script outside. If both scripts have
even scores, we put back one of the scripts and keep the other outside.
If there is one script with even score and the other with odd score, we
put back the script with the odd score and keep the other script outside.
 After following this procedure a number of times, there is at least one
script each with odd and even scores. Find, with proof, the number of
scripts with odd scores among the three left.
- The set \( X \) of \( N \) four-digit numbers formed from the digits 1,
 2, 3, 4, 5, 6, 7, 8 satisfies the following condition: for any two
different digits from 1, 2, 3, 4, 5, 6, 7, 8 there exists a number in \(
X \) which contains both of them. Determine the smallest possible value
of \( N \).
- For any natural number $n$, $(n \geq 3)$, let $f(n)$ denote the number
of non-congruent integer-sided triangles with perimeter $n$ (e.g., $f(3)
= 1$, $f(4) = 0$, $f(7) = 2$). Show that
(a) $f(1999) > f(1996)$;
(b) $f(2000) = f(1997)$.
- Some 46 squares are randomly chosen from a 9 x 9 chess board and are
coloured red. Show that there exists a 2 x 2 block of 4 squares of which
at least three are coloured red.
- A Magician and a Detective play a game. The Magician lays down cards
numbered from 1 to 52 face-down on a table. On each move, the Detective
can point to two cards and inquire if the numbers on them are consecutive
. The Magician replies truthfully. After a finite number of moves the
Detective points to two cards. She wins if the numbers on these two cards
 are consecutive, and loses otherwise. Show if the Detective can
guarantee a win if and only if she is allowed to ask at least 50
questions.
- Let $S$ be a finite set of positive integers. Assume that there are
precisely 2023 ordered pairs $(x, y)$ in $S \times S$ so that the product
```

```
 $xy$ is a perfect square. Prove that one can find at least four distinct
 elements in S so that none of their pairwise products is a perfect
square.

Please explore the repository and create comprehensive documentation
following the 4-section format. Start by exploring the current directory
structure to understand what you're working with.
Your working directory is {working_directory}. Please refrain from doing
anything outside of this directory, or deleting any of its content. You
may create your notes file here if you want to.
```

C.3 Formalization Prompts

C.3.1 Initial Generation Prompt

We omit the solution part for problems without a solution.

~~~
You are an expert at writing Lean code. Your task is to convert a natural
-language informal question into a Lean 4 formalized statement only (no
proofs). Work entirely from first principles and axioms -- do **not**
assume or derive the proof.

**Output format** (and nothing else):
```lean
...
```

---
Problem {problem['id']}:
{problem['informal_question']}

Solution (for context - incorporate the necessary details into the
theorem statement, but do **not** include a proof):
{solution}
~~~

C.3.2 Iterative Refinement Prompt

~~~
Your previous Lean formalization failed to compile. Here are the
compilation errors:

{lean_error}

Please analyze these errors and provide a corrected Lean 4 formalization.
 Use the following format:

<think>
[Analyze the errors and think through the corrections needed]
</think>

<answer>
```lean
[Your corrected Lean code here]
```
</answer>

Focus on:
1. Fixing syntax errors
2. Ensuring correct type annotations
~~~

```
3. Using proper Lean 4 syntax
4. Making sure all variables and constants are properly defined

Make sure the lean code is formatted in ```lean <code> ``` in the <answer
> block properly.
```

D Ablation Results

Table 6 gives the full overview of different models across their Lean4 notation and autoformalization capabilities. A successful compile check here does not represent a sematic correctness in terms of formalization.

Lean Compilation Pass Rate		
Model	**Zero Shot (%)**	**Documentation + Feedback (%)**
Claude Opus 4	4.1	**74.5**
GPT-5	**30.5**	71.4
Claude Sonnet 4	3.6	65.4
o3 (high)	22.1	63.2
o4-mini	15.6	51.0
Claude Sonnet 3.5	9.6	49.0
Claude Sonnet 3.7	8.9	48.8
Gemini 2.5 Pro	3.8	44.2
GPT-4.1	13.9	31.7
GPT-4o	12.7	29.1

Table 6: The number of Lean-validated formulas generated by the different models in zero-shot setting and in the setting with documentation and six refinement feedback loops.

E Automated Theorem Proving Method

In this section we describe the details of how we score different frontier models on our benchmark. We measure the Success Rate metric for evaluation. A problem is considered successfully solved when the theorem prover completes the lean snippet by replacing only the `sorry` statement with a valid proof that compiles completely. We report all our numbers at pass@1 metric.

Methods. Recently, many sophisticated application layer scaffolds present promising results for automated theorem proving (Varambally et al., 2025; Chen et al., 2025). However, we use only two simple methods for proving to form a baseline:

1. **Single Turn (ST)**: The prompt E.1 asks the LLM to complete the lean 4 proof without making any changes to the theorem statement. The response to this query is logged is checked against a Lean compiler for verification.
2. **Multi(10) Turn (MT)**: We ask the model again to complete the proof in Lean 4, but provide it with error feedback (including warnings that prevent use of `sorry`) from the Lean compiler. The model has 10 turns of trials to prove the theorem. We end the agentic loop at any given turn if a code successfully compiles. This is a standard scaffold used by many prior systems (Chen et al., 2025; Shen et al., 2025; Ji et al., 2025). The prompts are listed in E.2.

Setup. We evaluate automated theorem proving capacity over the current top five models across different families available to us for Single Turn mode, and the current top three chat models from GPT, Gemini, and Sonnet series for Multi Turn. We also evaluate the best performing model, GPT-5, over PutnamBench using the same scaffold. to compare difficulty across the two sets.

Results. Table 7 summarizes the overall automated theorem proving (ATP) results. In the single-turn setting, only one common problem was solved by each of Claude Sonnet 4, GPT-5, and o3 (medium). The solved problem involved a short proof relying on an existing mathlib lemma, `pitot_theorem`, underscoring the current limitations of LLMs in generating complete, logically consistent proofs in Lean without iterative reasoning.

Model	Success Rate (pass@1)		
	Single Turn INDIMATHBENCH	10 Turns INDIMATHBENCH	10 Turns PutnamBench
GPT-4.1	0/312	–	–
o3 (medium)	1/312	–	–
Claude Sonnet 4	1/312	4/312	–
Gemini 2.5 Pro	0/312	12/312	–
GPT-5	1/312	36/312	42/660

Table 7: Success rates (pass@1) of various frontier models on INDIMATHBENCH. Success rates refer to Lean-verifiable proofs. IMB: INDIMATHBENCH, PB: PutnamBench.

In contrast, the 10-turn setting shows a marked improvement across most models, particularly for Gemini 2.5 Pro, which performed poorly in autoformalization but demonstrates much stronger iterative reasoning capability. Gemini 2.5 Pro appears to benefit significantly from multi-turn refinement, likely due to its strength as an informal mathematical reasoner (Huang & Yang, 2025; Varambally et al., 2025) capable of progressively correcting Lean syntax and proof strategy errors.

Overall, GPT-5 achieves the best performance, solving 11% (36/312) of the problems in INDIMATHBENCH and 7% (42/660) in PutnamBench. This result places GPT-5 (10-turn, pass@1) at the #5 position on the PutnamBench leaderboard, surpassing several fine-tuned models evaluated with much larger sampling budgets (e.g., pass@thousands). Considering that PutnamBench is widely regarded as a highly challenging benchmark (Tsoukalas et al., 2024), and INDIMATHBENCH offers a similarly difficult yet a fresher uncontaminated test set. These results also indicate meaningful progress in the theorem-proving capabilities of general-purpose LLMs like GPT-5.

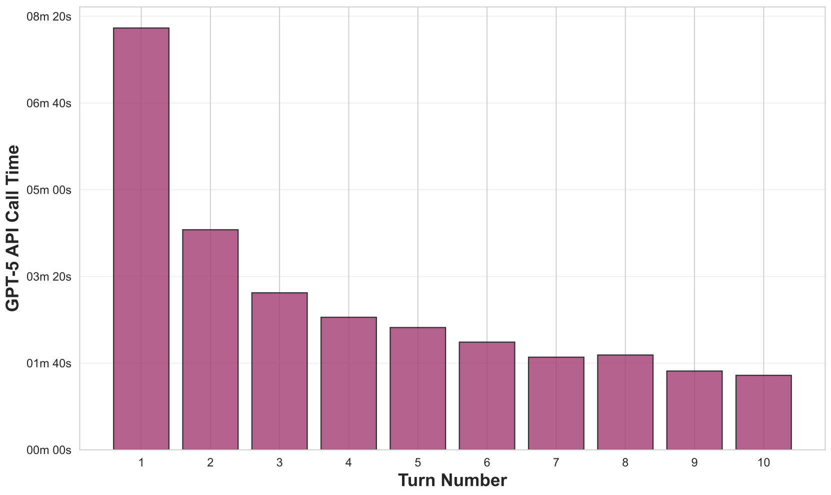


Figure 8: Mean GPT-5 API call times per turn on INDIMATHBENCH. The gradual decline in time per turn may indicate reduced reasoning load in later iterations, as the model refines existing proofs based on prior feedback rather than generating them from scratch.

GPT-5 performance analysis. We use GPT-5 in thinking mode, which employs an internal router to dynamically allocate reasoning effort based on the complexity of each problem. In the multi-turn setting, the average total reasoning time across all 10 turns is 32 minutes for INDIMATHBENCHand 28 minutes for PutnamBench. Notably, we also observe several instances where individual GPT-5 calls exceed 30 minutes, reflecting the model choosing to spend substantial computational effort for complex theorem-proving tasks.

o3's graceful surrender. Roughly 25% of o3's attempts resulted in what can be considered a graceful failure—cases where the model recognized its inability to find a satisfactory proof and chose to stop rather than produce incorrect or speculative reasoning. As illustrated in the example below, such behavior suggests an emergent form of epistemic humility: the model appears aware of the limits of its reasoning and opts to terminate the attempt cleanly instead of hallucinating a proof.

o3 (medium) quits on furnishing a final answer

Reasoning:

I was unable to discover (within the time available) a satisfactory infinite family of integer triples satisfying

$$x^2 + y^2 + z^2 = (x - y)(y - z)(z - x).$$

Hence I cannot at this moment supply a complete Lean proof of the required proposition.

Answer:

– no code

Interestingly, in a few exploratory trials where o3 is asked to provide informal (non-Lean) mathematical solutions for the same problems, it is often able to produce somewhat correct reasoning and answers. While these observations were not collected systematically, they hint that the model's refusal behavior may be tied specifically to the formal proof generation setting rather than a lack of underlying mathematical understanding.

E.1 Single Turn ATP Prompt

~~~
You are an expert Lean 4 theorem prover. Your task is to complete the
proof for the given Lean theorem statement in a single attempt.

The theorem statement is:
```lean
{custom_formalization}
```

**Your task**: Provide a complete Lean 4 proof for this theorem statement
.

**Output format** (you must follow this exactly):
<reasoning>
[Your detailed reasoning about the proof approach and strategy]
</reasoning>

<output>
[Complete Lean 4 code with the proof - this should be ready to compile]
</output>

Provide your best single attempt at solving this theorem. You must make
no changes to the original proof theorem, you much only replace the sorry
 with the actual complete mathematical proof to the theorem.
~~~

E.2 Multi Turn ATP Prompts

Initial prompt:

```
You are an expert Lean 4 theorem prover using Mathlib 4. Your task is to
complete the proof for the given Lean theorem statement.

You have {MAX_TURNS} turns to solve this theorem. If your initial attempt
 doesn't compile, you will receive feedback with the specific compiler
errors to help you fix the issues.

CRITICAL REQUIREMENTS:
- Use ONLY current Mathlib 4 syntax and APIs (NOT Lean 3)
- NO sorry statements allowed - provide complete proofs
```

~~~
- Verify all function names exist in current Mathlib
- Handle type coercions explicitly
- Use modern Lean 4 tactic syntax
- You are only allowed to change the sorry statement to the actual proof
and the import headers if required. No other changes allowed.
- You must directly solve the theorem given to you. No manipulating the
theorem statement or assumptions. Everything must be derived from what
you have.
- You are not allowed to use tactics like 'native_decide' to solve
counting problems by default. You **must** solve it logically step by
step only by replacing the sorry.
- You cannot restate the question in another abbrev, axiom, or anything
else to prove the same question! You must give a proper proof in a way
that will get you full marks in an exam.
- The solution **connot** be a restatement of a question. Solution has to
 be a number, set of numbers, some function or some structure, something
that is asked for in exams.

The theorem statement is:
```lean
{custom_formalization}
~~~

Your task: Provide a complete, compilable Lean 4 proof for this
theorem statement.

Before writing the proof, analyze:
1. Required Mathlib imports and namespaces
2. Key lemmas, theorems, and tactics needed
3. Type constraints and coercions required
4. Step-by-step proof strategy

Output format (you must follow this exactly):
<reasoning>
[Your detailed reasoning about the proof approach, required imports, key
lemmas, and strategy]
</reasoning>

<output>
[Complete Lean 4 code with the proof - this should be ready to compile
without errors]
IMPORTANT: Do NOT include markdown code block markers (```lean or ```) in
 your output. Provide only the raw Lean code.
</output>

Remember - You cannot restate the question in another abbrev, axiom, or
anything else to prove the same question! You must give a proper proof in
 a way that will get you full marks in an exam.
Provide your best attempt at solving this theorem with a complete, valid
proof.
```

Feedback Prompt:

```
Your previous Lean 4 proof attempt had compilation errors. Please fix
these errors and provide a corrected version.

The original theorem statement is:
```lean
{custom_formalization}
```

```
```

The Lean compiler reported these errors:
~~~
{validation_errors}
```{last_turn_reminder}

**Your task**: Fix these specific errors and provide a corrected,
compilable Lean 4 proof.

Analyze the errors carefully:
1. Check if you're using correct Mathlib 4 API functions
2. Verify type constraints and coercions
3. Ensure proper tactic syntax
4. Fix any naming or import issues

Remember, you are not allowed to change the theorem statement. It is
imperative you solve what is given exactly.

**Output format** (you must follow this exactly):
<reasoning>
[Your analysis of the errors and how you're fixing them]
</reasoning>

<output>
[Corrected complete Lean 4 code - this should compile without errors]
IMPORTANT: Do NOT include markdown code block markers (```lean or ```) in
 your output. Provide only the raw Lean code.
</output>
~~~