# PYTHAGORAS-PROVER: ADVANCING EFFICIENT FORMAL PROVING VIA AUGMENTED LEAN FORMALISATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

LLM-based theorem provers have achieved strong performance on various benchmarks, but still require substantial training and inference compute due to scarce verified proof data and generating long reasoning traces. We introduce Pythagoras-Prover, a compute-efficient, open-source family of Lean theorem provers comprising 4B and 32B autoregressive models and a first diffusion-based proof of concept that iteratively refines proofs at inference time. To improve training efficiency, we apply curriculum supervised fine-tuning to a Lean-verified corpus stratified into easy, medium, and hard problems, progressing from simpler proofs to harder ones. Dynamic proof-reasoning filtering further preserves informative traces within an 8K-token context budget. We then introduce Augmented Lean Formalisation (ALF), which generates structured variants of verified statements and populates them through self-distillation, providing additional training signal without requiring every variant to be formally verified and reducing reliance on statement surface forms. Pythagoras-Prover-4B surpasses DeepSeek-Prover-V2-671B at pass@32 on MiniF2F-Test (86.1% versus 82.4%) with roughly 167× fewer parameters. Pythagoras-Prover-32B achieves state-of-the-art performance among open-source neural theorem provers, reaching 93.0% on MiniF2F-Test and solving 93 of 672 PutnamBench problems. We also release MiniF2F-ALF, a contamination-sensitive benchmark of ALF-generated perturbations. Together, these results demonstrate that strong Lean theorem proving is achievable under practical compute budgets without relying exclusively on frontier-scale models.

## 1 INTRODUCTION

Recent progress in mathematical reasoning has made large language models (LLMs) considerably more capable, yet not reliably correct. Although modern LLMs can solve complex problems end-to-end, their reasoning remains susceptible to hallucinations and subtle logical errors that natural-language inspection cannot reliably detect (Leang et al., 2025a; Lyu et al., 2023). Automated Theorem Proving (ATP) addresses this limitation by grounding model outputs in interactive proof assistants such as Lean (de Moura et al., 2015), Isabelle (Paulson, 1994) and Rocq (Coquand & Huet, 1988), whose deterministic type-checkers reject any argument that is not mechanically verifiable. This formal grounding eliminates an entire class of hallucinations and yields reasoning whose correctness can be audited step by step.

In recent years, frontier systems such as DeepMind's AlphaProof and AlphaGeometry (Chervonyi et al., 2025; Hubert et al., 2025) and ByteDance's Seed-Prover (Chen et al., 2025) have demonstrated that AI systems can attain International Mathematical Olympiad (IMO) medal-level performance. In the open-source community, DeepSeek-Prover-V2 (Ren et al., 2025), Kimina-Prover (Wang et al., 2025a), and Goedel-Prover-V2 (Lin et al., 2026) have reported strong results on standard benchmarks including MiniF2F (Zheng et al., 2022) and PutnamBench (Tsoukalas et al., 2024). These successes, however, are typically achieved either through very large models with hundreds of billions of parameters, or through computationally intensive inference that relies on elaborate search procedures, self-correction, or large sampling budgets. State-of-the-art ATP therefore remains largely inaccessible to researchers and practitioners without substantial compute, and a sizeable gap persists between small open-source provers and their largest counterparts.



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.

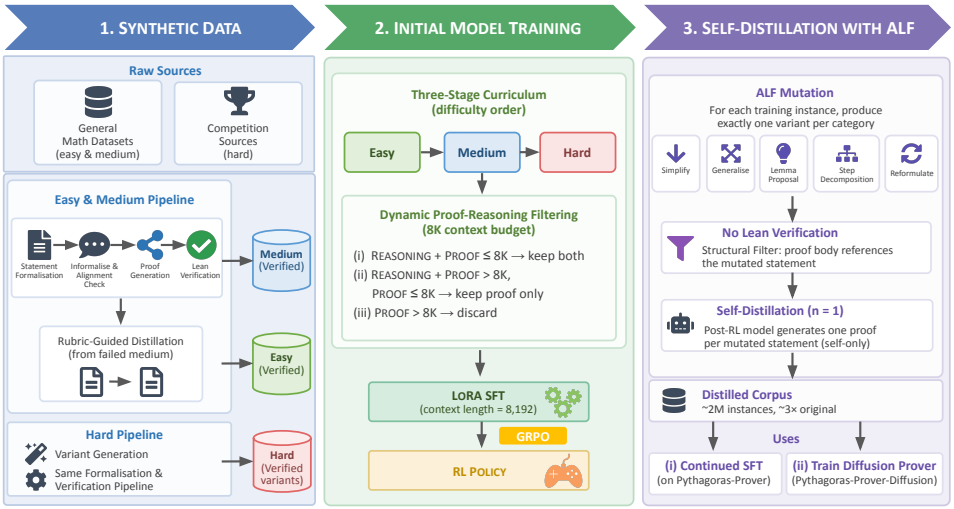



Figure 1: Overview of Pythagoras-Prover pipeline.

In this work, we release Pythagoras-Prover, a compute-efficient family of open-source theorem provers for Lean 4 that challenges the assumption that stronger formal reasoning requires frontier-scale models. At 4B parameters, Pythagoras-Prover-4B surpasses DeepSeek-Prover-V2-671B (Ren et al., 2025) at pass@32 on MiniF2F-Test ($82.4\% \rightarrow 86.1\%$), despite being roughly $167\times$ smaller. At 32B parameters, Pythagoras-Prover-32B achieves the strongest reported performance among open-weight neural Lean provers evaluated without self-correction, obtaining the highest pass rate on MiniF2F-Test while relying on standard restart sampling rather than inference-time corrective procedures. Beyond autoregressive proving, the family includes Pythagoras-Prover-Diffusion, to our knowledge the first proof-of-concept diffusion-based theorem-proving model for Lean. Under a matched 4B setting on the same corpus and evaluation harness, Pythagoras-Prover-Diffusion trails Pythagoras-Prover-4B in raw pass@32 accuracy but generates proofs $2.58\times$ faster on the same hardware, placing it on the side of an accuracy-efficiency frontier for formal theorem proving.

All our releases share a compute-frugal data pipeline for Lean theorem proving. We first build a Lean-verified seed corpus of roughly 800K instances using predominantly small open models, with a single 235B model as the only larger component. The corpus is then stratified into easy, medium, and hard tiers for curriculum fine-tuning. We then introduce *Augmented Lean Formalisation* (ALF), which generates structured variants of each verified instance through five mutation types: simplification, generalisation, lemma proposal, proof-step decomposition, and reformulation. ALF uses lightweight sanity checks rather than full Lean verification, expanding the seed corpus by roughly $2.5\times$. These variants serve as self-distillation inputs for post-training both autoregressive and diffusion provers. Applying the same mutation operator to MiniF2F-Test yields *MiniF2F-ALF*, a contamination-sensitive perturbation benchmark on which contemporary provers degrade substantially.

In summary, our contributions are as follows:

- **A compute-efficient family of Lean theorem provers.** We release Pythagoras-Prover, a family of open-source Lean 4 theorem provers spanning two generation paradigms: autoregressive provers at 4B and 32B parameters, and Pythagoras-Prover-Diffusion-4B, to our knowledge the first proof-of-concept diffusion-based theorem-proving model.
- **A compute-frugal Lean data pipeline.** We construct a Lean-verified corpus of roughly 800K instances stratified into easy, medium, and hard tiers. We then introduce ALF, a structured mutation scheme that expands this corpus by roughly $2.5\times$. Applying the same mutation operator to MiniF2F-Test yields *MiniF2F-ALF*, a companion benchmark for probing transfer across nearby formal variants of benchmark statements.
- **An efficient adaptation recipe for Lean-proving.** We combine parameter-efficient supervised fine-tuning, curriculum learning, self-distillation, and reinforcement learning from Lean verification. During supervised fine-tuning, a dynamic proof–reasoning filter preserves informative reasoning traces while keeping each instance within an 8K-token context budget.

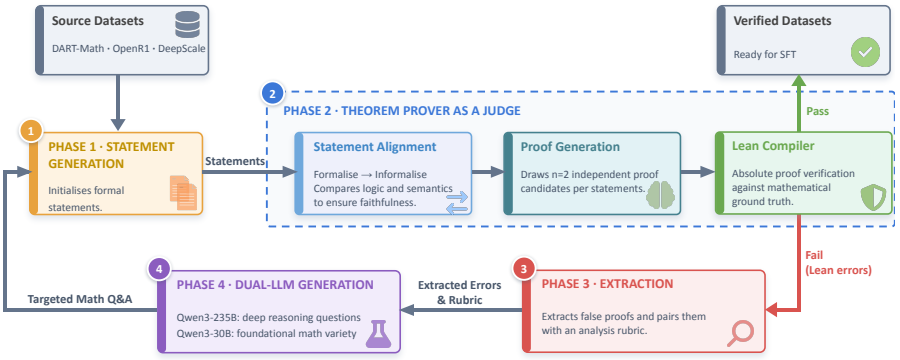



Figure 2: The seed synthetic data generation pipeline for *easy* and *medium tier* problems.

## 2 METHODOLOGY

The Pythagoras-Prover pipeline starts by constructing a Lean-verified *seed* corpus (§2.1) and trains base autoregressive provers (§2.2). *Augmented Lean Formalisation* (ALF) then expands the corpus through structured mutations without per-instance Lean verification, followed by self-distillation (§2.3). Lastly, we apply the same self-distillation corpus to train the diffusion prover (§2.4).

### 2.1 SYNTHETIC DATA CREATION

All seed instances pass the Lean type-checker and belong to three difficulty tiers ordered monotonically by construction. *Easy* and *medium* instances are autoformalised from general mathematical-reasoning datasets (§2.1.1): medium problems require routine multi-step reasoning, while easy instances are rubric-driven simplifications of medium ones (§2.1.1). *Hard* instances are competition proofs requiring non-routine techniques absent from general maths corpora (§2.1.2). Appendix A.7 gives representative examples; §2.3 scales this modest-sized, high-quality seed through ALF.

#### 2.1.1 THE EASY AND MEDIUM TIERS

Figure 2 shows the three-stage easy/medium pipeline: formal statement synthesis, formal proof verification, and rubric-guided distillation.

**Formal statements and formal proofs.** Inspired by the theorem-prover-as-a-judge pipeline (Leang et al., 2025b), we filter data through deterministic proof-assistant verification. While Leang et al. (2025b) formalise only GSM8K and MATH500 with GPT-4 (OpenAI et al., 2024), we broaden difficulty and domain coverage using natural-language problems from *DART-Math-Hard* (Tong et al., 2024), *DeepScaleR-Preview* (Luo et al., 2025), and *OpenR1-Math* (Hugging Face, 2025). Goedel-Autoformaliser-v2 (Lin et al., 2026) produces a Lean statement for each problem. To mitigate logical errors introduced by autoformalisation, we apply *auto-informalisation* (Leang et al., 2025b): each statement is translated back into natural language, compared with its source, and discarded on disagreement. For each retained statement, Goedel-Prover-v2-32B (Lin et al., 2026), an open model far smaller than the GPT-4 prover of Leang et al. (2025b), generates $n{=}2$ independent proof attempts. We accept a (statement, proof) pair if at least one attempt passes the Lean type-checker, so every training example is Lean-verified.

**Rubric-guided distillation.** Existing autoformalisers and provers still fail on complex mathematical statements (Lin et al., 2026; Ren et al., 2025; Wang et al., 2025a). We parse the Lean compiler errors of rejected instances and manually annotate a representative subset, identifying seven categories: (i) *invalid projection or field errors*, referencing non-existent fields; (ii) *unsolved goals*, remaining after proof execution; (iii) *tactic failures* (e.g. `linarith`, `simp`, `rewrite`) on expressions that resist automated simplification; (iv) *type mismatches*, such as integers versus natural numbers; (v) *synthesis failures*, from unresolved type-class properties such as finiteness or decidability; (vi) *unknown constant or identifier errors*, typically hallucinated theorem names; and (vii) *other errors*.

Appendix A.1 gives explanations, examples, and the resulting rubric, which prompts an LLM to simplify each failed problem by targeting the responsible algebraic manipulation or logical step. Unlike free-form synthesis of unconstrained simpler or harder variants (Lin et al., 2026), ours conditions generation on a fine-grained Lean error taxonomy to address concrete failure causes. We generate two variants per instance using Qwen3-235B and Qwen3-30B (Qwen Team et al., 2025), pairing a high-capacity generator with a smaller model that tends to produce more diverse and effective task-specific outputs (Kim et al., 2025). The rubric yields a $30\%$ relative improvement in autoformalisation success (Appendix F.1).

#### 2.1.2 THE HARD TIER

For the hard tier we draw on Big-Math-RL-Verified (Albalak et al., 2025), restricting selection to its Olympiads, AIME, AMC, `aops_forum`, and Level 4/5 MATH subsets, which emphasise multi-step, proof-oriented reasoning in advanced number theory, combinatorics, inequalities and geometry. This addresses the routine-problem bias of the easy/medium sources: DART-Math-Hard is rejection-sampled from GSM8K and MATH; DeepScaleR combines AIME and AMC with Omni-MATH (Gao et al., 2025) and routine contest preparation; and OpenR1-Math is distilled from NuminaMath-CoT (Li et al., 2024), dominated by lower-difficulty sources such as `cn_k12`. Exact or near-duplicates across medium and hard sources are assigned to hard and removed from medium, preventing cross-tier contamination.

Using Qwen3.6-27B (Qwen Team, 2026), we generate $n = 4$ question variants per retained problem ($n = 2$ for easy/medium), so that enough survive Lean filtering, following the pipeline of Leang et al. (2025b) (step 2 in Figure 2). Only variants enter SFT; the original Big-Math hard problems are reserved for RL, preserving both a challenging on-policy reward signal and an unleaked probe of generalisation. We do *not* apply rubric-guided distillation to the hard tier, as simplifying hard-source failures into local subproblems would leak hard-problem structure into the supervised corpus.

### 2.2 TRAINING ALGORITHM

We train Pythagoras-Prover in three stages: supervised fine-tuning on the seed corpus under a difficulty-ordered curriculum, reinforcement learning on held-out hard problems, and a final continued-SFT stage on the ALF self-distillation corpus of §2.3.

**Supervised fine-tuning and a three-stage curriculum.** We perform supervised fine-tuning with LoRA (Hu et al., 2022) at a context length of $8{,}192$ tokens. To our knowledge, Pythagoras-Prover is the first prover to reach state-of-the-art performance without full-parameter SFT (Lin et al., 2026; Ren et al., 2025). Training proceeds as a three-stage difficulty curriculum over the tiers of §2.1: easy, then medium, then hard. An 8K budget is restrictive for theorem proving, where reasoning chains are long and discarding them outright degrades performance, so we adopt a *dynamic proof-reasoning filtering* scheme, where we (i) train on the full sequence when reasoning and proof both fit, (ii) keep the proof alone when they do not, and (iii) discard the instance when the proof alone exceeds the budget. This beats naive truncation and naive instance filtering, matching full-context fine-tuning at a fraction of the cost (Appendix C.5). More training details are provided in Appendix B.1.

**Reinforcement learning.** We apply full-parameter GRPO (Shao et al., 2024) for one epoch with $8$ rollouts per problem, rewarding proofs that compile under Lean and align with their formal statements. We use DAPO-style dynamic filtering (Yu et al., 2025), retaining only problems whose number of successful rollouts lies in $\{1, \ldots, 5\}$, and remove the KL-divergence penalty to encourage exploration. RL yields modest gains, consistent with limited headroom after the policy acquires strong proof-search behaviour from the Lean-verified, difficulty-ordered corpus. Appendix A.6 provides further training details, while Appendix C.3 quantifies each stage's contribution.

### 2.3 AUGMENTED LEAN FORMALISATION AND SELF-DISTILLATION

The seed corpus of §2.1 is bounded in size by Lean verification throughput, since every retained instance passes the type-checker. To scale beyond this bound, we introduce *Augmented Lean Formalisation* (ALF), a structured mutation operator producing exactly one formal variant per instance in each of five categories: *simplification*, *generalisation*, *lemma proposal*, *proof-step decomposition*,

and *reformulation*. ALF replaces per-instance verification with a lightweight consistency check, in three steps. *(i) Statement mutation:* Qwen3.6-27B generates statement variants without proofs. *(ii) Proof self-distillation:* the post-RL Pythagoras-Prover of §2.2 generates one candidate proof ($n$=1) per mutated statement, with both the 4B and 32B models each proving their own mutations. *(iii) Statement-alignment filtering:* we retain a pair only when the proof references its target statement. This check does not invoke the type-checker or certify correctness, so ALF data is not Lean-verified; dropping per-instance verification removes the pipeline's dominant computational cost. This choice is motivated by evidence that unverified self-distilled samples remain beneficial in domains with executable correctness signals, such as coding (Zhang et al., 2026); we examine whether this extends to formal proving in Appendix C.3. Lean verification of a random subset of 2,000 ALF instances yields an $87.8\%$ pass rate, indicating high quality despite the absence of exhaustive verification.

The filtered generations form ∼2M instances, roughly $2.5\times$ the seed corpus. We use this corpus to complete Pythagoras-Prover training through continued SFT, again using LoRA (§2.2), and to train the diffusion prover (§2.4), so both branches share one self-distillation recipe. Cross-model distillation (a 32B generator, 4B trainee) proved unstable, so we use no larger teacher.

### 2.4 PYTHAGORAS-PROVER-DIFFUSION

As an alternative use of the self-distillation corpus, we train Pythagoras-Prover-Diffusion, to our knowledge the first diffusion-based theorem prover, adopting block diffusion (Arriola et al., 2025) via `dllm` (Zhou et al., 2026). Autoregressive-to-diffusion conversion destabilises training as target lengths increase (Zhou et al., 2026); we find that the *dynamic proof-reasoning filtering scheme* of §2.2 maintains stability during training. Our central modification is *tactic-based masking* within blocks, which aligns denoising with the complete Lean proof steps committed at inference time (Figure 8).

**Tactic-based masking.** We use masked diffusion rather than autoregressive decoding because diffusion models fill sequence positions in any order, can revisit earlier commitments once later context disambiguates them, and commit several positions per forward pass (Sahoo et al., 2024; Nie et al., 2025). We adopt the discrete masked diffusion formulation of Nie et al. (2025) with the linear schedule $\alpha_t = 1 - t$, making diffusion time $t \in [0, 1]$ the per-unit mask probability. Let $\mathbf{x}_0 = (x_0^1, \ldots, x_0^L)$ be a clean Lean proof of length $L$, $\mathbf{x}_t = (x_t^1, \ldots, x_t^L)$ its corrupted counterpart containing the mask symbol $[\mathrm{M}]$, and $p_\theta(\cdot \mid \mathbf{x}_t)$ the denoiser predicting clean tokens at masked positions. The theorem statement (header, goal, `:=by`) remains a fixed conditioning block; only the proof body is corrupted, with $t$ resampled independently for each training sequence.

Per-token masking treats positions as exchangeable and allows masked tokens to be inferred from visible siblings within a tactic. Thus, we change only the corruption unit to complete tactics, requiring reconstruction of whole proof steps from surrounding context. Let $\mathcal{T}(\mathbf{x}_0) = (\tau_1, \ldots, \tau_K)$ denote tactic spans extracted from $\mathbf{x}_0$ by a Lean parser, such as `intro x y z` or **`have`** `hnum : x * y * z * (x + y + z) = 36 :=` **`by`**, where each $\tau_k \subseteq \{1, \ldots, L\}$ is a contiguous token range. Residual structural positions $S = \{1, \ldots, L\} \setminus \bigcup_k \tau_k$ (the `by` keyword, indentation, and theorem header) remain unmasked: $x_t^i = x_0^i$ for all $i \in S$. Tactic spans are corrupted independently:[1]

$$\mathbf{x}_t^{\tau_k} \sim \begin{cases} [\mathrm{M}]^{|\tau_k|} & \text{with probability } t, \\ \mathbf{x}_0^{\tau_k} & \text{with probability } 1 - t, \end{cases} \qquad k = 1, \ldots, K, \tag{1}$$

so each tactic is either fully masked or retained intact. The denoiser remains a per-token model, while the objective groups the standard MDLM cross-entropy into per-span sums of token log-probabilities:

$$\mathcal{L}_{\mathrm{tac}}(\theta) = -\mathbb{E}_{t \sim \mathcal{U}[0,1],\, \mathbf{x}_0,\, \mathbf{x}_t} \left[ \frac{1}{t} \sum_{k=1}^{K} \mathbf{1}\left[\mathbf{x}_t^{\tau_k} = [\mathrm{M}]^{|\tau_k|}\right] \sum_{i \in \tau_k} \log p_\theta\left(x_0^i \,\middle|\, \mathbf{x}_t\right) \right], \tag{2}$$

where the expected number of masked positions per example is $t(L - |S|)$. The $1/t$ factor therefore normalises the expected mask count, giving each eligible token constant expected masking weight across noise levels. Setting $\tau_k = \{k\}$ for every $k$ ($K = L$) reduces equation 2 to the per-token MDLM objective of Sahoo et al. (2024), making tactic-level masking a strict generalisation of token-level masking.

[1] For a symbol $a$, a sequence $\mathbf{x}$ and a set of integers $b$, we denote by $a^{|b|}$ the symbol repeated $|b|$ times and by $\mathbf{x}^b$ the subsequence indexed by elements of $b$.

Table 1: Pass@$N$ (%) on MiniF2F-Test. "Best ($N$)" represents each method's highest reported pass rate, with the corresponding sampling budget in parentheses. † marks Kimina-Prover variants, and the highest overall result is highlighted in bold. Results in the upper section are copied from the respective papers. A detailed comparison is provided in Table 3.

| Method | #Params | Pass@32 | Pass@1024 | Best ($N$) |
|---|---|---|---|---|
| Goedel-Prover-SFT (Lin et al., 2025b) | 7B | 57.6 | – | 62.7 (3200) |
| STP (Dong & Ma, 2025) | 7B | – | – | 67.6 (25600) |
| Kimina-Prover-Preview-72B (Wang et al., 2025a) | 72B | 68.85 | – | 80.74 (8192) |
| DeepSeek-Prover-V2-7B (Ren et al., 2025) | 7B | 75.6 | – | 82.0 (8192) |
| DeepSeek-Prover-V2-671B | 671B | 82.4 | – | 88.9 (8192) |
| Kimina-Prover-8B-Distill† (Wang et al., 2025a) | 8B | 77.86 | – | – |
| Kimina-Prover-70B† | 70B | 84.0 | 87.7 | 92.2 (TTRL)* |
| Goedel-Prover-V2-8B (Lin et al., 2026) | 8B | 84.6 | 87.9 | 90.2 (8192) |
| *(+Self Correction)* | 8B | 86.7 | 89.3 | – |
| Goedel-Prover-V2-32B | 32B | 88.1 | 91.8 | 92.2 (8192) |
| *(+Self Correction)* | 32B | 90.4 | 92.6 | – |
| **Pythagoras-Prover-4B** | 4B | 86.1 | 88.1 | 89.8 (2048) |
| **Pythagoras-Prover-32B** | 32B | 89.8 | 92.6 | **93.0** (2048) |

## 3 Experimental Setup

We evaluate Pythagoras-Prover under the benchmarks and protocol described below, keeping all design choices consistent across runs unless stated otherwise.

**Dataset.** In our experiments, we use: (i) MiniF2F (Zheng et al., 2022) comprising 488 problem statements in Lean (244 validation and 244 test) drawn from high-school level competitions; (ii) PutnamBench (Tsoukalas et al., 2024) targeting college-level mathematics through 672 problems sourced from the William Lowell Putnam Mathematical Competition (1962–2023); and (iii) MiniF2F-ALF, containing 488 Lean-checked statements derived from the 244 MiniF2F-Test problems using ALF (§2.3) and controlled statement perturbations. We detail these benchmarks in Appendix B.2.

**Evaluation.** All experimental results are evaluated with pass@$N$ on Lean 4.9.0-rc1, using a similar evaluation environment to previous work (Lin et al., 2026; Ren et al., 2025). A proof attempt is judged correct only if (i) it compiles under Lean with no errors and contains no `sorry`, `admit`, or otherwise unproved goals, and (ii) the target formal statement appears verbatim in the generated output, ensuring that the model proves the stated goal rather than a rewritten or weaker one.

## 4 Experimental Results

The evaluation results on MiniF2F-Test, PutnamBench, and MiniF2F-ALF are shown in Tables 1–2 and Figure 3, respectively. Below we summarise and discuss the results: state-of-the-art accuracy at modest scale, the strongest open-source result on PutnamBench under matched effective compute, robustness to statement-level perturbation under MiniF2F-ALF, and a first viable diffusion-based theorem prover trained on the same corpus.

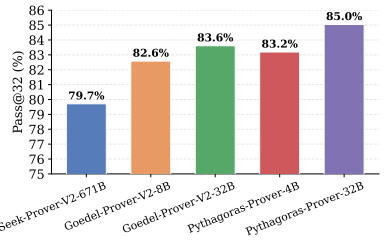



Figure 3: Performance of state-of-the-art provers on MiniF2F-ALF. As MiniF2F-ALF is introduced in this work, all results are evaluated by us under a unified setup.

**State-of-the-art performance at modest scale.** Based on Table 1, Pythagoras-Prover-32B achieves the highest reported MiniF2F-Test pass rate, $\mathbf{93.03}\%$ at pass@2048, surpassing the prior open-source best (Goedel-Prover-V2-32B, $92.2\%$ at pass@8192) with a quarter of the sampling budget and no inference-time self-correction or test-time reinforcement learning. Among methods without self-correction, it leads at every evaluated budget: $89.8\%$ at pass@32 versus Goedel-Prover-V2-32B ($88.1\%$), DeepSeek-Prover-V2-671B ($82.4\%$, roughly $20\times$ larger), and Kimina-Prover-70B ($84.0\%$). It also matches self-corrected Goedel-Prover-V2-32B at pass@1024 ($92.6\%$ vs. $92.6\%$) and surpasses

Table 2: PutnamBench leaderboard (problems solved out of 672). Pythagoras-Prover solves 93 problems at pass@2048, the best open-source result, surpassing the previous best (Goedel-Prover-V2, 86 at pass@184 with self-correction).*

| # | Model | Num-solved | Open-source | Compute |
|---|---|---|---|---|
| 1 | **Pythagoras-Prover** | 93 | ✓ | pass@2048 |
| 1 | **Pythagoras-Prover** | 59 | ✓ | pass@64 |
| 1 | **Pythagoras-Prover** | 48 | ✓ | pass@32 |
| 2 | Goedel-Prover-V2 (self-correction mode) | 86 | ✓ | pass@184 |
| 2 | Goedel-Prover-V2 (self-correction mode) | 57 | ✓ | pass@32 |
| 2 | Goedel-Prover-V2 | 43 | ✓ | pass@32 |
| 3 | DeepSeek-Prover-V2 | 47 | ✓ | pass@1024 |
| 3 | DeepSeek-Prover-V2 | 22 | ✓ | pass@32 |
| 4 | DSP+ | 23 | ✓ | pass@128 |
| 5 | Bourbaki | 14 | ✓ | pass@512 |
| 6 | Kimina-Prover-7B-Distill | 10 | ✓ | pass@192 |
| 7 | Self-play Theorem Prover | 8 | ✓ | pass@3200 |
| 8 | Goedel-Prover-SFT | 7 | ✓ | pass@512 |
| 9 | ABEL | 7 | ✗ | pass@596 |

*Seed Prover (Chen et al., 2025), successfully solved 331 problems on PutnamBench. However, the prover is closed-source, and the computational budget at test time remains unclear.

it at pass@2048, indicating that learned proof search largely obviates multi-round repair. On PutnamBench (Table 2), it achieves the strongest open-source result: 93 problems solved at pass@2048 versus Goedel-Prover-V2's 86 at pass@184 with self-correction. All evaluated models lose accuracy on MiniF2F-ALF relative to MiniF2F-Test, but Pythagoras-Prover-32B remains strongest (85.0%; Figure 3). Pythagoras-Prover-4B (83.2%) nearly matches the 8× larger Goedel-Prover-V2-32B (83.6%), with a smaller drop (2.9 vs. 4.5 percentage points). Appendix C details independent restart sampling, comparisons across sampling budgets, and the full perturbation analysis; Appendix F analyses effective decoding compute.

**Pythagoras-Prover-4B outperforms substantially larger baselines.** Based on Table 1, the efficiency of our recipe is clearest at the 4B scale. Under a matched pass@32 budget, Pythagoras-Prover-4B attains 86.1%, exceeding DeepSeek-Prover-V2-671B (82.4%) by 3.6 points despite being roughly 167× smaller, and outperforming Kimina-Prover-70B (84.0%), Goedel-Prover-V2-8B (84.6%), and DeepSeek-Prover-V2-7B (75.6%). The advantage persists with budget: at pass@2048 Pythagoras-Prover-4B reaches 89.8%, exceeding DeepSeek-Prover-V2-671B's pass@8192 result (88.9%) at a quarter of the budget. Against the 8B Goedel-Prover-V2, Pythagoras-Prover-4B leads at every shared budget without self-correction (88.1% vs. 87.9% at pass@1024) and stays within 1.2 points even of its self-corrected variant, at half the parameters. The training recipe thus transfers cleanly to the small-model regime, where hundred-billion-scale baselines are impractical.

**Pythagoras-Prover-Diffusion: the corpus transfers across decoding regimes.** By training on the same ALF self-distillation corpus (§2.3), Pythagoras-Prover-Diffusion-4B reaches 63.25% on MiniF2F-Test at pass@32 with block size 32, roughly 23 points below Pythagoras-Prover-4B (86.1%; Figure 4). We introduce Tactic-based masking (§2.4) to denoise with complete proof steps, while ALF supplies additional Lean training data. These interventions address limitations of random masking and data scarcity, but stable training context remains a bottleneck (Ye et al., 2025; Leang et al., 2026a; Nie et al., 2025). At 8,192 tokens, autoregressive training converges while diffusion exhibits sustained high gradient norms beyond warm-up; reducing the context to 4,096 restores stability (Figure 8). Retraining Pythagoras-Prover at this shorter context also substantially reduces its accuracy, indicating that restricted training context contributes materially to the gap. This matters for theorem proving, which relies on test-time scaling of reasoning chains (Lin et al., 2026; Varambally et al., 2026), even when evaluation permits outputs longer than the training context. These results establish an initial feasibility

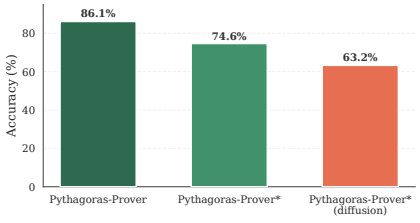



Figure 4: MiniF2F comparison. * denotes a 4096-token training limit and evaluation solely at 8192 tokens.

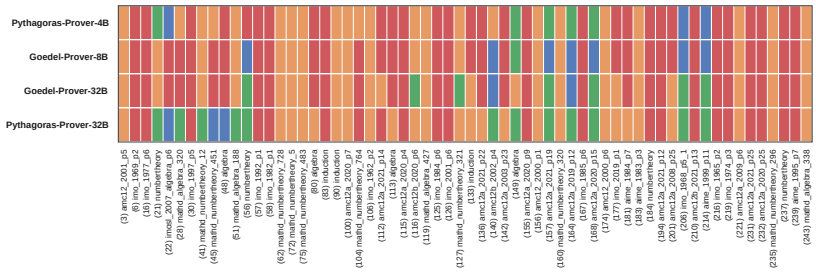



Figure 6: Per-pair outcomes on MiniF2F-Test and MiniF2F-ALF. : both solved; : both failed; : MiniF2F-Test solved, MiniF2F-ALF failed; : MiniF2F-ALF solved, MiniF2F-Test failed.

baseline for diffusion-based Lean proving. Section 5.3 quantifies the context-controlled accuracy comparison and throughput trade-off; Appendix C.6 provides extended analysis.

# 5 Analysis

In this section, we present a scaling study on MiniF2F (§5.1), a per-instance analysis on MiniF2F and MiniF2F-ALF (§5.2), and a comparison of diffusion and autoregressive provers (§5.3).

## 5.1 Scaling Analysis

Figure 5 compares Pythagoras-Prover (4B/32B), Goedel-Prover-V2 (8B/32B, with and without self-correction), and DeepSeek-Prover-V2 (7B/671B) over $N \in \{32, \ldots, 2048\}$. Among methods without self-correction, Pythagoras-Prover-32B leads at every budget, reaching $93.03\%$ at pass@2048, suggesting that training internalises proof search that can substitute for iterative verifier-guided correction. Despite $167\times$ fewer parameters, Pythagoras-Prover-4B outperforms DeepSeek-Prover-V2-671B at every shared budget; its $89.8\%$ at pass@2048 exceeds the latter's $88.9\%$ at pass@8192. Both curves combine high accuracy at small $N$ with shallow gains, indicating competitive performance with fewer samples than the baselines. The recipe thus transfers to small models suited to compute-, memory-, or latency-constrained deployment, while scaling unchanged to Pythagoras-Prover-32B on the upper Pareto frontier.

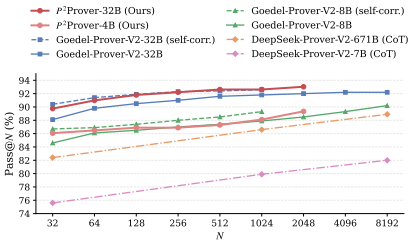



Figure 5: Scaling behaviour on MiniF2F test split. We compare Pythagoras-Prover with Goedel-Prover-V2, with and without self-correction, and DeepSeek-Prover-v2. Pythagoras-Prover is evaluated up to $N = 2048$ due to computational constraints.

## 5.2 Analysis towards MiniF2F and MiniF2F-ALF

Based on Section 4, we observe that across a representative panel of Pythagoras-Prover-4B, Pythagoras-Prover-32B, Goedel-Prover-V2-8B, and Goedel-Prover-V2-32B, all four solve $83.20\%$ of problems; at least one fails on $41/244$ ($16.80\%$), and all fail on $22/244$ ($9.02\%$). Failures are domain-skewed (Figure 10, Appendix D): AMC and IMO contribute $34.15\%$ and $31.71\%$ of at-least-one-wrong cases, respectively; among all-model failures, IMO contributes $50.00\%$ and AMC $27.27\%$. Most problems offer little model separation, leaving a narrow, familiar residual that limits assessment of broad reasoning robustness. *MiniF2F-ALF diversifies failures and exposes model differences.* Specifically, MiniF2F-ALF preserves problem families under controlled statement mutations: all-model failures rise from $9.02\%$ to $13.93\%$, with at least one model failing on $18.24\%$ of instances. We see that the failures diversify, with MathD accounting for $20.22\%$ alongside AMC, IMO, and other categories. Pairwise comparisons expose complementary failures, although Pythagoras-Prover retains more exclusive solves than Goedel-Prover-V2 at both scales. The orange region in Figure 6 marks instances solved originally but failed after mutation, exposing brittleness hidden by high MiniF2F pass rates. As ALF also drives training augmentation (§2.3), these transitions probe structural generalisation limits. Appendix D.1 details failure distributions and pairwise comparisons.

### 5.3 Compute Efficiency: Pythagoras-Prover vs Pythagoras-Prover-Diffusion

We compare Pythagoras-Prover-4B and Pythagoras-Prover-Diffusion-4B on MiniF2F at pass@32 using the same ALF self-distillation corpus (§2.3), evaluation harness, and single $8\times$ H100 node with batch size $4$. Pythagoras-Prover accuracy is higher ($86.1\%$ versus $63.3\%$). Retraining Pythagoras-Prover at diffusion's $4{,}096$-token context and evaluating both at $8{,}192$ tokens narrows the gap from $22.8$ to $11.4$ points ($74.6\%$ versus $63.2\%$; Figure 4). Training context thus accounts for roughly half the gap; the residual advantage may also reflect Qwen's more extensive autoregressive pretraining and post-training. Pythagoras-Prover-Diffusion achieves $10.56$ versus $4.10$ output tokens per second per GPU worker, a $2.58\times$ speedup. Its fixed denoising budget and block-parallel generation contrast with sequential autoregressive decoding, whose cost grows with outputs permitted by `max_new_tokens`$= 30{,}000$. Defining $S =$ accuracy $\times$ TPS with pass@32 accuracy gives $S_{\text{AR}} = 0.8607 \times 4.10 = 3.53$ and $S_{\text{Diff}} = 0.6325 \times 10.56 = 6.68$, a $1.89\times$ diffusion advantage. This coarse, throughput-dominated proxy is proportional to solved problems per GPU-second at a fixed sampling budget only when average tokens per attempt are comparable. These proof-of-concept results favour autoregressive proving for accuracy at fixed sampling budgets, while diffusion offers higher throughput and potentially better efficiency when generation cost or latency dominates. Longer stable training contexts and proof generations (§2.4) may improve this trade-off.

## 6 Related Work

**LLM reasoning.** LLM reasoning abilities have advanced significantly on complex mathematical reasoning tasks (Llama Team, 2024), particularly through chain-of-thought (CoT) reasoning (Wei et al., 2022). Large reasoning models (LRMs) (Guo et al., 2025; Qwen Team et al., 2025; Gemma Team, 2025) instead tackle complex problems by generating long CoT traces, and are further enhanced through techniques such as confidence selection (Leang et al., 2026b), iterative refinement (Snell et al., 2025), multi-path exploration (Guan et al., 2025) and others. It is also well established that test-time scaling improves performance in both general reasoning (Muennighoff et al., 2025) and formal reasoning (Varambally et al., 2026). More recently, diffusion models have proven highly capable on mathematical and coding tasks (Bie et al., 2026; Li et al., 2026; Leang et al., 2026a); given this strength, extending diffusion models to formal proving is a natural fit.

**Formal theorem proving.** Formal theorem proving grounds LLM reasoning in verifiable proofs (Leang et al., 2025b; Zhang et al., 2025b;c). Existing approaches explore multiple proof trajectories through Monte Carlo Tree Search or breadth-first search to construct valid proofs (Xin et al., 2024; Ji et al., 2025; Zhang et al., 2025a; Lin et al., 2025b; Xin et al., 2025), with applications extending beyond mathematics (Meadows et al., 2026; Xu et al., 2026). Seed-Prover attains IMO medal-level performance through extensive test-time search and refinement, although such systems rely on substantial computation (Chen et al., 2025). Other approaches explore bootstrapping (Lin et al., 2025a), self-training (Dong & Ma, 2025) and others (Hubert et al., 2025). Our work is also concurrent with Leanstral-1.5 (Kabra et al., 2026), which relies more heavily on agentic reasoning.

## 7 Conclusions

We introduced Pythagoras-Prover, a compute-efficient family of open-source Lean 4 provers comprising 4B and 32B autoregressive models and Pythagoras-Prover-Diffusion-4B, the first proof-of-concept diffusion prover for Lean. Pythagoras-Prover-4B surpasses a 671B-parameter prover on MiniF2F-Test with roughly $167\times$ fewer parameters, and Pythagoras-Prover-32B achieves state-of-the-art MiniF2F-Test performance and the strongest open-source PutnamBench result. The diffusion model demonstrates feasible Lean proof generation, motivating comparisons of throughput and proofs per GPU-second alongside accuracy. Behind all three is one compute-frugal pipeline: a Lean-verified corpus stratified into difficulty tiers for curriculum learning, expanded by *Augmented Lean Formalisation* (ALF) into structured variants screened by lightweight checks rather than per-instance Lean verification. The same mutation scheme yields *MiniF2F-ALF*, on which contemporary provers degrade, suggesting that high MiniF2F-Test accuracy does not guarantee transfer to nearby formal variants. With parameter-efficient fine-tuning, dynamic proof-reasoning filtering and reinforcement learning from Lean verification, the pipeline performs strongly under practical compute budgets, suggesting that careful data construction and efficient training can partly substitute for model scale.

## REPRODUCIBILITY STATEMENT

Pythagoras-Prover will be released as an open-source model, and the training data will be made publicly available for model training and use. We describe the training algorithm in Section 2 and provide the hyperparameters for both training and evaluation in Appendix B. We believe these materials, together with the released models, are sufficient to reproduce our results.

## AI USE STATEMENT

LLMs are part of our research pipeline. Goedel-Autoformaliser-v2 and Goedel-Prover-V2-32B generate Lean statements and proofs for the seed corpus, and every retained instance is verified by Lean (§ 2.1.1). Qwen3-235B, Qwen3-30B and Qwen3.6-27B generate rubric-guided problem variants, hard-tier variants and ALF mutations (§ 2.1, § 2.3). For MiniF2F-ALF, Codex (GPT-5.5) proposes candidate mutations and Claude Opus 4.6 judges well-formedness, and all retained statements are checked by the Lean compiler (Appendix B.3). LLMs were also used to polish text and check grammar. The authors reviewed all LLM-generated data and text and take full responsibility for the paper's content.

## REFERENCES

Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and Nick Haber. Big-Math: A Large-Scale, High-Quality Math Dataset for Reinforcement Learning in Language Models, 2025. URL https://arxiv.org/abs/2502.17387.

Anthropic. System Card: Claude Opus 4.6. https://anthropic.com/claude-opus-4-6-system-card, 2026.

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding Up Text Diffusion via Token Editing, 2026. URL https://arxiv.org/abs/2602.08676.

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.

Yuri Chervonyi, Trieu H. Trinh, Miroslav Olsák, Xiaomeng Yang, Hoang H. Nguyen, Marcelo Menegali, Junehyuk Jung, Junsu Kim, Vikas Verma, Quoc V. Le, and Thang Luong. Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2. *J. Mach. Learn. Res.*, 26: 241:1–241:39, 2025. URL https://jmlr.org/papers/v26/25-1654.html.

Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988. URL https://doi.org/10.1016/0890-5401(88)90005-3.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp

(eds.), *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, Lecture Notes in Computer Science, pp. 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6_26. URL https://doi.org/10.1007/978-3-319-21401-6_26.

Kefan Dong and Tengyu Ma. STP: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/dong25h.html.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss Library. *IEEE Transactions on Big Data*, 12(2):346–361, 2026. doi: 10.1109/TBDATA.2025.3618474. URL https://doi.org/10.1109/TBDATA.2025.3618474.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-MATH: A Universal Olympiad Level Mathematic Benchmark for Large Language Models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yaqPf0KAlN.

Gemma Team. Gemma 3 Technical Report, 2025. URL https://arxiv.org/abs/2503.19786.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/guan25f.html.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-Level Formal Mathematical Reasoning with Reinforcement Learning. *Nature*, 2025. URL https://www.nature.com/articles/s41586-025-09833-y.

Hugging Face. Open R1: A fully open reproduction of DeepSeek-R1, January 2025. URL https://github.com/huggingface/open-r1.

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.

Aditi Kabra, Albert Q. Jiang, Andrew Zhao, Dhia Garbaya, Indraneel Mukherjee, Jason Rute, Mert Unsal, Roman Soletskyi, and Simon Sorg. Leanstral, 2026. URL https://www.alphaxiv.org/abs/2608.leanstral.

Seungone Kim, Juyoung Suk, Xiang Yue, Vijay Viswanathan, Seongyun Lee, Yizhong Wang, Kiril Gashteovski, Carolin Lawrence, Sean Welleck, and Graham Neubig. Evaluating Language Models as Synthetic Data Generators. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6385–6403, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.320. URL https://aclanthology.org/2025.acl-long.320/.

Joshua Ong Jun Leang, Aryo Pradipta Gema, and Shay B Cohen. CoMAT: Chain of Mathematically Annotated Thought Improves Mathematical Reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20245–20274, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1024. URL https://aclanthology.org/2025.emnlp-main.1024/.

Joshua Ong Jun Leang, Giwon Hong, Wenda Li, and Shay B Cohen. Theorem Prover as a Judge for Synthetic Data Generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29941–29977, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1448. URL https://aclanthology.org/2025.acl-long.1448/.

Joshua Ong Jun Leang, Yu Zhao, Mihaela Cătălina Stoian, Wenda Li, Shay B. Cohen, and Eleonora Giunchiglia. Can I Have Your Order? Monte-Carlo Tree Search for Slot Filling Ordering in Diffusion Language Models, 2026a. URL https://arxiv.org/abs/2602.12586.

Joshua Ong Jun Leang, Zheng Zhao, Aryo Pradipta Gema, Sohee Yang, Wai-Chung Kwan, Xuanli He, Wenda Li, Pasquale Minervini, Eleonora Giunchiglia, and Shay B. Cohen. PiCSAR: Probabilistic Confidence Selection And Ranking for Reasoning Chains, 2026b. URL https://arxiv.org/abs/2508.21787.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions, 2024. URL https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf.

Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. ReFusion: A Diffusion Large Language Model with Parallel Autoregressive Decoding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=LBtWaUc7FE.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards General Text Embeddings with Multi-stage Contrastive Learning, 2023. URL https://arxiv.org/abs/2308.03281.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-STaR: Learning to Interleave Thinking and Proving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=SOWZ59UyNc.

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. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=x2y9i2HDjD.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-Prover-V2: Scaling Formal Theorem Proving with Scaffolded Data Synthesis and Self-Correction. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j4C0nALrgK.

Llama Team. The Llama 3 Herd of Models, 2024. URL https://arxiv.org/abs/2407.21783.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B Model by Scaling RL. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2, 2025.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful Chain-of-Thought Reasoning. In Jong C. Park, Yuki Arase, Baotian Hu, Wei Lu, Derry Wijaya, Ayu Purwarianti, and Adila Alfa Krisnadhi (eds.), *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 305–329, Nusa Dua, Bali, November 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.ijcnlp-main.20. URL https://aclanthology.org/2023.ijcnlp-main.20/.

Jordan Meadows, Lan Zhang, and Andre Freitas. FormalScience: Scalable Human-in-the-Loop Autoformalisation of Science with Agentic Code Generation in Lean, 2026. URL https://arxiv.org/abs/2604.23002.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan LI. Large Language Diffusion Models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 50608–50646. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

OpenAI. OpenAI GPT-5 System Card, 2026. URL https://arxiv.org/abs/2601.03267.

OpenAI, Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o System Card, 2024. URL https://arxiv.org/abs/2410.21276.

Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994. ISBN 3-540-58244-4. doi: 10.1007/BFB0030541. URL https://doi.org/10.1007/BFb0030541.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the Training-Inference Mismatch via FP16, 2025. URL https://arxiv.org/abs/2510.26788.

Qwen Team. Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team, An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2504.21801.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A Flexible and Efficient RLHF Framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. DART-Math: Difficulty-Aware Rejection Tuning for Mathematical Problem-Solving. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 7821–7846. Curran Associates, Inc., 2024. doi: 10.52202/079017-0251. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0ef1afa0daa888d695dcd5e9513bafa3-Paper-Conference.pdf.

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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 11545–11569. Curran Associates, Inc., 2024. doi: 10.52202/079017-0368. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1582eaf9e0cf349e1e5a6ee453100aa1-Paper-Datasets_and_Benchmarks_Track.pdf.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively Building Formal Proofs with Informal Reasoning, 2026. 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, 2025a. URL https://arxiv.org/abs/2504.11354.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-Prover Preview: Towards Large Formal Reasoning Models with Reinforcement Learning, 2025b. URL https://arxiv.org/abs/2504.11354.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data, 2024. URL https://arxiv.org/abs/2405.14333.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. BFS-Prover: Scalable Best-First Tree Search for LLM-based Automatic Theorem Proving. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 32588–32599. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1565/.

Qiyuan Xu, Xiaokun Luan, Renxi Wang, Joshua Ong Jun Leang, Peixin Wang, Haonan Li, Wenda Li, and Conrad Watt. Neural Theorem Proving for Verification Conditions: A Real-World Benchmark. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=MfDyickxQA.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models, 2025. URL https://arxiv.org/abs/2508.15487.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, juncai liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a4277440d50f1f15d2cb4c14f7e0c0d2-Paper-Conference.pdf.

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. Leanabell-Prover: Posttraining Scaling in Formal Reasoning, 2025a. URL https://arxiv.org/abs/2504.06122.

Lan Zhang, Marco Valentino, and André Freitas. Autoformalization in the Wild: Assessing LLMs on Real-World Mathematical Definitions. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 1720–1738. Association for Computational Linguistics, 2025b. doi: 10.18653/V1/2025.EMNLP-MAIN.90. URL https://doi.org/10.18653/v1/2025.emnlp-main.90.

Lan Zhang, Marco Valentino, and André Freitas. MASA: LLM-Driven Multi-Agent Systems for Autoformalization. In Ivan Habernal, Peter Schulam, and Jörg Tiedemann (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - System Demonstrations, Suzhou, China, November 4-9, 2025*, pp. 615–624. Association for Computational Linguistics, 2025c. doi: 10.18653/V1/2025.EMNLP-DEMOS.44. URL https://doi.org/10.18653/v1/2025.emnlp-demos.44.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly Simple Self-Distillation Improves Code Generation, 2026. URL https://arxiv.org/abs/2604.01193.

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.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dLLM: Simple Diffusion Language Modeling, 2026. URL https://arxiv.org/abs/2602.22661.

# A Additional Details on Synthetic Data Generation

## A.1 Rubric Distillation Prompt

For our rubric-based distillation stage, we use the prompt shown below to convert each failed Lean proof attempt into a self-contained synthetic mathematics problem with a step-by-step solution. The prompt instructs the generator to identify the root-cause error, classify it under our extraction rubric, and emit a strictly formatted JSON object containing the synthetic question and its solution.

**Prompt: ALF Failure-to-Synthetic-Problem Generation**

```
You are an expert mathematician and dataset creator. Your task is to
analyze a failed Lean 4 proof attempt, extract the exact mathematical
context where it failed, generate a WHOLE NEW SIMPLER SYNTHETIC, FULLY
SOLVABLE MATH PROBLEM, and provide its complete step-by-step
mathematical solution.

### CORE OBJECTIVE
Analyze a failed Lean 4 proof to identify the exact mathematical step
where the logic or formalization breaks down. Extract the specific
mathematical parameters, formulas, and premises from that step and the
original theorem statement. Use these extracted details to generate a
SIMPLER, SYNTHETIC, FULLY SOLVABLE MATH PROBLEM in pure natural
language, along with its step-by-step mathematical solution. This must
be a standalone mathematical question that contains all the necessary
numbers and context required to calculate the answer from scratch,
without needing to reference the original Lean code.

### ROOT CAUSE
When multiple errors are present, identify all the error by line
position as the root cause. However, ignore downstream errors that
would resolve once the root cause is fixed.
**Determine Failure Level:**
   - **Formalization error:** Malformed theorem statement (wrong
     types/fields). Target the underlying mathematical
     conceptualization of the claim.
   - **Proof error:** Bug inside the tactic block. Target mathematical
     strategy or logic.

### EXTRACTION RUBRIC (Map Error Category to Conceptual Math)
* **Invalid projection/field:** The proof calls a field that doesn't
  exist for that mathematical object. Extract the actual mathematical
  object and generate a problem that requires calculating or properly
  interacting with its true mathematical properties.
* **Unsolved goals:** Extract the exact remaining goal state and the
  available hypotheses. Turn this exact state into a self-contained
  "Prove that..." or "Calculate..." math problem.
* **Tactic failures (linarith, simp, rewrite, etc.):** Extract the
  specific algebraic equation or inequality that failed to simplify.
  Create a synthetic math problem asking the student to manually
  perform that specific algebraic manipulation or simplification.
* **Type mismatch:** Identify expected vs. actual domains (e.g.,
  integers vs. natural numbers, sets vs. finite collections). Create a
  problem that requires converting or mapping between these specific
  domains using the numbers from the theorem.
* **Failed to synthesize:** Missing mathematical property (e.g.,
  finiteness, decidability). Create a problem asking to prove that the
  specific object in the theorem possesses this required property.
* **Unknown constant/identifier:** The LLM hallucinated a theorem.
  Extract the logical step it was trying to make, and create a math
  problem that requires proving that specific logical step from first
  principles.
* **Other errors:** Extract the technical failure and translate it
  into a logic or calculation problem based on the active variables.

### CONSTRAINTS FOR THE SYNTHETIC QUESTION & SOLUTION
1. **COMPLETELY SELF-CONTAINED & SOLVABLE:** The generated question
   MUST include the actual mathematical values, coordinates, formulas,
   or premises involved in the failed step. (e.g., If the error
   involves Point F, you MUST state the exact coordinates of Point F
   in the question). It must be solvable without looking at the Lean
   code.
2. **PURE MATHEMATICS (NO CODE):** The question and solution MUST NOT
   contain Lean 4 code, tactics (`simp`, `rw`), Lean keywords (`let`,
   `have`), or backticks (` `). Frame it purely as a standard math
   competition or textbook problem.
3. **NO DIAGNOSTIC OR "WHY" QUESTIONS:** Do not ask "Why did this
```

```
   fail?" or "What is the correct logical approach?". Ask direct
   mathematical questions like "Calculate X", "Find Y", or "Prove that
   Z is true given [Premises]."
4. **EXPLICIT STEP-BY-STEP SOLUTION:** You must provide a clear,
   logically sound, step-by-step mathematical solution to the
   synthetic question you just generated.

### EXAMPLES (Bad vs. Good)
- **BAD (Diagnostic/Code-heavy):** "Why does the `intro` tactic fail
  when applied to Point F?"
- **GOOD (Synthetic Math Problem):** "Point A is at (0, 0) and Point B
  is at (8/3, 4). If Point F is defined as the midpoint of segment AB,
  calculate the exact sum of the x and y coordinates of Point F."

- **BAD (Diagnostic/Vague):** "What missing logical step is required
  to prove that this sum is strictly positive?"
- **GOOD (Synthetic Math Problem):** "Given that x > 5 and y = -2,
  prove that the sum of x and y is strictly positive."

---
### BEGIN USER PROMPT
PREVIOUS QUERY
{{QUESTION}}

FORMAL STATEMENT:
```lean4
{{FORMAL_STATEMENT}}

FAILED PROOF:


{{MERGED_PROOF}}
ERROR MESSAGE:
{{ERROR_MESSAGE}}
```

### A.2 Similarity Filtering for Intra-corpus Deduplication

For the *easy* tier we deduplicate by semantic content rather than surface form, since paraphrase-level duplicates are common in our problem sources and surface methods (exact-match or normalised edit distance) miss variable-renamed or lightly reworded copies. Each statement is encoded with `Alibaba-NLP/gte-Qwen2-7B-instruct` (Li et al., 2023) into a $3{,}584$-dimensional dense embedding. Embeddings are $\ell_2$-normalised so that cosine similarity reduces to an inner product, and indexed with FAISS (Douze et al., 2026) using an inner-product index for efficient $k$-nearest-neighbour search.

**Chunked overlapping search.** Exhaustive pairwise comparison across the full easy-tier pool is memory-bound, so we partition the embedded instances into chunks of $5{,}000$ items with a $100$-instance overlap between consecutive chunks, and run top-$k$ nearest-neighbour retrieval within each chunk. The overlap guarantees that near-duplicates whose embeddings happen to straddle a chunk boundary are still compared in at least one chunk, preventing duplicates from leaking through partitioning.

**Filtering rule.** A pair $(x_i, x_j)$ with cosine similarity above $\tau$ is treated as a near-duplicate, and the lower-priority member is dropped by source-tier priority (curated problems over autoformalised paraphrases; older sources over newer reformulations). We set $\tau = 0.75$ via a small held-out sweep that balances false-positive removal (distinct problems flagged as duplicates) against false-negative retention (near-duplicates passing through).

### A.3 Dataset Decontamination Sweep

Beyond intra-corpus deduplication (Appendix A.2), we run a five-stage decontamination sweep against every benchmark we report on (MiniF2F-Valid, MiniF2F-Test, and PutnamBench); a training instance is discarded if any stage triggers a hit. (i) *Surface-form hashing:* we normalise the natural-language statement (lowercase, collapsed whitespace, stripped punctuation, canonicalised numerals) and the Lean statement ($\alpha$-renamed binders, comments and formatting removed) and compare MD5 hashes. (ii) *$n$-gram overlap:* we flag any instance sharing an $n$-gram with a benchmark item, with $n = 10$ on natural language and $n = 5$ on Lean (the shorter window reflects the lower lexical entropy of formal code). (iii) *Embedding similarity:* surviving candidates are encoded with
```

`gte-Qwen2-7B-instruct` and indexed in FAISS as in Appendix A.2, and removed if their cosine similarity to any benchmark statement exceeds $\tau_{\text{decon}}$. (iv) *Structural equivalence:* we compare $\alpha$-normalised Lean statement trees and remove any instance that elaborates to the same formal theorem as a benchmark item despite a different surface form. (v) *Proof-template overlap:* as a cheaper alternative to retraining-based checks, we strip variable names and constants from both training and benchmark proofs, normalise the residual tactic skeletons, and flag any training instance whose skeleton is near-identical to that of a benchmark proof, catching cases where a benchmark proof is reproduced up to renaming.

Because MiniF2F-ALF is obtained by mutating MiniF2F-Test, any MiniF2F-Test contamination is also an ALF-seed contamination; we additionally run the first three stages against the $488$ mutated ALF statements to catch the residual case where a training instance matches a mutated variant rather than the original. The resulting decontaminated set is used for all supervised fine-tuning, self-distillation, and reinforcement-learning runs reported in this paper.

### A.4 ALF Self-Distillation

For our ALF Self-Distillation generation, each mutated statement's formal proof is generated by Pythagoras-Prover-Post-RL teacher at temperature $T = 1.2$ and nucleus top-$p = 0.95$, and the completion is used as a training target. Given the scale (2M instances per model), we omit per-sample Lean verification; Zhang et al. (2026) report that this omission does not materially degrade downstream performance in the coding domain, and Appendix C.3 indicates the same holds for formal proving, with self-distillation contributing consistently on top of the SFT baseline at both 4B and 32B.

### A.5 Verification of ALF Synthetic Data

Verifying all 2M ALF self-distilled instances with the Lean compiler is computationally prohibitive. We therefore conduct a random audit: sampling $2{,}000$ retained ALF instances uniformly from the final self-distillation corpus and checking each complete generated theorem, including both the mutated formal statement and its proof, under the same Lean environment used for evaluation. Of these audited instances, $87.8\%$ compile successfully, indicating that the quality of the synthetic data remains high and that exhaustive verification is unnecessary for our purposes. We do, however, verify the formal *statement* alongside the formal proof rather than the proof alone: checking only the proof teaches the model to alter the statement to fit a flawed proof, an effect we observe in practice and find necessary to prevent. Full verification of every instance is left out only on grounds of compute. Inspecting the remaining $12.2\%$ ($244$ of $2{,}000$) non-compiling instances, we find that the failures are overwhelmingly syntax-level (e.g. minor tactic or bracket errors) rather than reasoning-level: the underlying proof strategy still targets the mutated statement correctly, and closely tracks the reasoning chain of the pre-mutation seed proof, which was itself Lean-verified. This is precisely the distinction between the statement-alignment check and full mathematical correctness: alignment certifies that a plausible, on-topic proof strategy is offered, not that every low-level tactic step type-checks. We therefore expect this residual noise to mostly perturb surface syntax rather than corrupt the reasoning signal used for self-distillation, consistent with self-distillation contributing positively on top of the SFT baseline at the full 2M-instance scale (Appendix C.3).

### A.6 Reinforcement-learning details

Our RL corpus combines three sources: Goedel instances generated without self-correction (Lin et al., 2026), the manually annotated subset of NuminaMath-Lean (Wang et al., 2025b), and the original Big-Math hard problems held out from §2.1.2.

We train with GRPO (Shao et al., 2024) for one epoch with $8$ rollouts per problem. A rollout receives a reward iff its proof compiles under Lean and aligns with the target formal statement, and $0$ otherwise. DAPO-style dynamic filtering (Yu et al., 2025) retains only problems whose number of successful rollouts lies in $\{1, \ldots, 5\}$. We remove the KL-divergence penalty to encourage exploration. Unlike the LoRA-based SFT stage, RL uses full-parameter fine-tuning because we found the LoRA-adapted policy unstable under GRPO updates (Qi et al., 2025).

### A.6.1 WHY SELF-DISTILLATION?

After supervised fine-tuning, Pythagoras-Prover already attains a strong pass rate on MiniF2F-Test (Table 4); the remaining headroom is not a question of base capability but of exposing the model to a wider distribution of proof trajectories. Preference-based post-training is poorly matched to this regime: Direct Preference Optimisation (Rafailov et al., 2023) expects a dense preference signal, but the Lean verifier returns only a sparse binary judgement, and the preference pairs we derived from it proved unstable in our runs. GRPO suffers the same scarcity, yielding only marginal gains because the binary success/fail reward leaves little gradient on the long tail of structurally distinct but equally valid proofs. Self-distillation via ALF sidesteps both issues. It supplies dense, full-trajectory supervision: the teacher emits a complete proof for each mutated statement, rather than a single bit of reward. It *expands* rather than refines the training distribution, drawing formally distinct theorem variants from each seed. Self-distillation is therefore the right tool for this regime: full proof trajectories are more informative than sparse preference signals, and ALF is the mechanism that turns one seed into many.

## A.7 EXAMPLES OF PYTHAGORAS-PROVER-DATASET FOR EACH DIFFICULTY

**Easy tier example.** Synthetic, rubric-simplified algebra problem (rectangle perimeter).

**Formal statement**

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Let `a` and `b` be positive real numbers
    such that `b = a + 9` and `a * b = 360`.
    Prove that `2 * (a + b) = 78`. -/
theorem my_algebra_theorem
    (a b : ℝ) (ha : a > 0) (hb : b > 0)
    (h1 : b = a + 9) (h2 : a * b = 360) :
    2 * (a + b) = 78 := by
  sorry
```

**Formal proof**

```
theorem my_algebra_theorem
    (a b : ℝ) (ha : a > 0) (hb : b > 0)
    (h1 : b = a + 9) (h2 : a * b = 360) :
    2 * (a + b) = 78 := by
  have h3 : a * (a + 9) = 360 := by
    rw [← h1]; exact h2
  have ha15 : a = 15 := by
    nlinarith [sq_nonneg (a - 15), sq_nonneg (a +
24), ha]
  have hb24 : b = 24 := by
    rw [h1, ha15]; norm_num
  rw [ha15, hb24]; norm_num
```

**Medium tier example.** Autoformalised general-reasoning word problem (camper marshmallow count).

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- At camp Wonka there are 96 campers.
    Two-thirds are boys, one-third are girls.
    50% of the boys and 75% of the girls want
    a marshmallow. How many marshmallows are
    needed in total? -/
theorem my_combinatorics_theorem :
    let total_campers       : ℕ := 96
    let boys                : ℕ := (2 *
total_campers) / 3
    let girls               : ℕ := total_campers /
3
    let boys_wanting_toast  : ℕ := boys / 2
    let girls_wanting_toast : ℕ := (3 * girls) / 4
    let total_marshmallows  : ℕ :=
      boys_wanting_toast + girls_wanting_toast
    total_marshmallows = 56 := by
  sorry
```

```
theorem my_combinatorics_theorem :
    let total_campers       : ℕ := 96
    let boys                : ℕ := (2 *
total_campers) / 3
    let girls               : ℕ := total_campers /
3
    let boys_wanting_toast  : ℕ := boys / 2
    let girls_wanting_toast : ℕ := (3 * girls) / 4
    let total_marshmallows  : ℕ :=
      boys_wanting_toast + girls_wanting_toast
    total_marshmallows = 56 := by
  intro total_campers boys girls
        boys_wanting_toast girls_wanting_toast
        total_marshmallows
  have h_boys : boys = 64 := by
    dsimp [boys, total_campers] <;> norm_num <;>
rfl
  have h_girls : girls = 32 := by
    dsimp [girls, total_campers] <;> norm_num <;>
rfl
  have h_boys_toast : boys_wanting_toast = 32 := by
    dsimp [boys_wanting_toast, boys, total_campers]
at *
    <;> norm_num at * <;> rfl
  have h_girls_toast : girls_wanting_toast = 24 :=
by
    dsimp [girls_wanting_toast, girls,
total_campers] at *
    <;> norm_num at * <;> rfl
  have h_total : total_marshmallows = 56 := by
    dsimp [total_marshmallows,
          boys_wanting_toast, girls_wanting_toast]
at *
    <;> norm_num [h_boys_toast, h_girls_toast] at *
    <;> rfl
  exact h_total
```

**Hard tier example.** Competition-style number theory: smallest prime $p$ such that $p^2 + 144$ is a perfect square.

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Let `p` be the smallest prime such that
    `p^2 + 144` is also a perfect square.
    Determine the value of `p`. -/
theorem my_favorite_prime :
    Nat.Prime 5 and
    (exists k : N, 5^2 + 144 = k^2) and
    (forall p : N, Nat.Prime p → p < 5 →
      ¬exists k : N, p^2 + 144 = k^2) := by
  sorry
```

```
theorem my_favorite_prime :
    Nat.Prime 5 and
    (exists k : N, 5^2 + 144 = k^2) and
    (forall p : N, Nat.Prime p → p < 5 →
      ¬exists k : N, p^2 + 144 = k^2) := by
  have h_prime_5 : Nat.Prime 5 := by decide
  have h_exists_k : exists k : N, 5^2 + 144 = k^2
:= by
    refine' ⟨13, _⟩
    norm_num
  have h_no_smaller_prime :
      forall p : N, Nat.Prime p → p < 5 →
        ¬exists k : N, p^2 + 144 = k^2 := by
    intro p hp hlt
    have h₁ : p = 2 or p = 3 := by
      have h₄ : p ≤ 4 := by linarith
      interval_cases p <;>
        norm_num [Nat.Prime] at hp ⊢ <;> aesop
    rcases h₁ with (rfl | rfl)
    · intro h
      rcases h with ⟨k, hk⟩
      have h₂ : 2 ^ 2 + 144 = k ^ 2 := by simpa
using hk
      have h₃ : k ^ 2 = 148 := by linarith
      have h₄ : k ≤ 12 := by nlinarith
      have h₅ : k >= 13 := by nlinarith
      linarith
    · intro h
      rcases h with ⟨k, hk⟩
      have h₂ : 3 ^ 2 + 144 = k ^ 2 := by simpa
using hk
      have h₃ : k ^ 2 = 153 := by linarith
      have h₄ : k ≤ 12 := by nlinarith
      have h₅ : k >= 13 := by nlinarith
      linarith
  exact ⟨h_prime_5, h_exists_k, h_no_smaller_prime⟩
```

# B Experimental Settings

## B.1 Training Details

All models are based on the Qwen3 series (Qwen Team et al., 2025), specifically Qwen3-4B and Qwen3-32B, and are fine-tuned for two epochs using LoRA of rank 64 applied to all linear layers. Our models are trained using LlamaFactory (Zheng et al., 2024) for SFT, and Volcano Engine Reinforcement Learning (verl) (Sheng et al., 2024) for RL. We use learning rates of $3e{-4}$ and $7e{-5}$ for the SFT stages. For self-distillation, we reduce the learning rate to $1e{-5}$, as larger values induced training instability. Lastly, for Pythagoras-Prover-Diffusion we perform full fine-tuning with a learning rate of $3e{-5}$.

## B.2 Benchmark Details

**MiniF2F** (Zheng et al., 2022) comprises 488 problem statements in Lean (244 validation and 244 test) drawn from high-school level competitions, including the AMC, AIME, and the International Mathematical Olympiad (IMO). We adopt the Kimina-revised release (Wang et al., 2025a) and report results on MiniF2F-Test, in which several erroneous statements from the original release have been corrected.

**PutnamBench.** PutnamBench (Tsoukalas et al., 2024) targets college-level mathematics through 672 problems sourced from the William Lowell Putnam Mathematical Competition (1962–2023), covering algebra, analysis, number theory, geometry, combinatorics, probability, and set theory.

**MiniF2F-ALF.** MiniF2F-ALF contains 488 Lean-checked statements derived from the 244 MiniF2F-Test problems using ALF (§2.3) and controlled statement perturbations. It serves as a *contamination-sensitive perturbation probe*, testing robustness beyond exact benchmark recall, and a *transfer probe*, assessing generalisation from ALF-augmented training data to ALF-mutated test statements. Appendix B.3 details construction, candidate selection, and verification.

Table 3: Pass@$N$ on MiniF2F-Test across inference budgets $N$. Results in the top part are copied as reported by the respective papers. Values are percentages; † Kimina-Prover variants; best in bold.

| **Method** | **Budget ($N$)** | **Performance Pass@$N$** |
|---|---|---|
| Goedel-Prover-SFT (Lin et al., 2025b) | 32 | 57.6 |
| | 3200 | 62.7 |
| STP (Dong & Ma, 2025) | 128 | 61.2 |
| | 3200 | 65.0 |
| | 25600 | 67.6 |
| Kimina-Prover-Preview-72B (Wang et al., 2025a) | 32 | 68.85 |
| | 8192 | 80.74 |
| DeepSeek-Prover-V2-7B (Ren et al., 2025) | 32 | 75.6 |
| | 8192 | 82.0 |
| DeepSeek-Prover-V2-671B | 32 | 82.4 |
| | 8192 | 88.9 |
| Kimina-Prover-8B† (Wang et al., 2025a) | 32 | 78.3 |
| Kimina-Prover-70B† | 32 | 84.0 |
| | 1024 | 87.7 |
| w/ Test-Time Reinforcement Learning | – | 92.2 |
| Goedel-Prover-V2-8B (Lin et al., 2026) | 32 | 84.6 |
| | 1024 | 87.9 |
| | 8192 | 90.2 |
| w/ Self-Correction | 32 | 86.7 |
| | 1024 | 89.3 |
| Goedel-Prover-V2-32B | 32 | 88.1 |
| | 1024 | 91.8 |
| | 8192 | 92.2 |
| w/ Self-Correction | 32 | 90.4 |
| | 1024 | 92.6 |
| **Pythagoras-Prover-4B** | 32 | 86.1 |
| | 1024 | 88.1 |
| | 2048 | 89.8 |
| **Pythagoras-Prover-32B** | 32 | 89.8 |
| | 1024 | 92.6 |
| | 2048 | **93.0** |

### B.3 MINIF2F-ALF CREATION

We apply ALF (§2.3) to all 244 MiniF2F-Test problems, together with numerical and variable perturbations. This follows GSM-Symbolic (Mirzadeh et al., 2025), which exposes memorisation effects on GSM8K by perturbing variables and numerical values while preserving problem structure and difficulty. In particular, renaming variables reduces reliance on memorised statement surface forms. Models dependent on exact benchmark recall are expected to degrade more than those with transferable proof behaviour.

Some of ALF's five mutation operators produce near-identical variants with little diagnostic value. We therefore generate five candidates per problem using Codex (GPT-5.5) (OpenAI, 2026) and rank their semantic divergence from the original statement by cosine distance under `Alibaba-NLP/gte-Qwen2-7B-instruct` (Li et al., 2023), the encoder described in Appendix A.2. Retaining the two most divergent candidates per problem yields 488 statements.

Unlike the ALF self-distillation corpus, where per-sample verification is omitted at scale (§2.3), every retained benchmark statement undergoes verification. Claude Opus 4.6 (Anthropic, 2026) judges whether it is a well-formed Lean theorem, and the Lean compiler checks it. All 488 retained statements pass both checks.

# C Extended Results and Ablations

## C.1 Extended Experimental Results

**PutnamBench: restart sampling reaches the strongest open-source result.** On PutnamBench (Table 2), the comparison is more nuanced. Indeed, self-correction is more sample-efficient at small budgets: Goedel-Prover-V2 with self-correction solves 57 problems at pass@32, compared with Pythagoras-Prover's 48. However, Pythagoras-Prover uses independent *restart* sampling, generating fresh complete proofs and verifying each with Lean. This strategy scales well with the sampling budget: Pythagoras-Prover solves 59 problems at pass@64 and 93 at pass@2048, achieving the *strongest open-source* result on the leaderboard. This exceeds Goedel-Prover-V2's 86 solved problems at pass@184 with self-correction by 7 problems, and nearly doubles DeepSeek-Prover-V2's 47 at pass@1024. This suggests that, although self-correction can be effective at low budgets, it may become brittle when the initial proof attempt contains an early logical error, since subsequent correction rounds often inherit the same mistake and fail to recover (Appendix E). Restart sampling avoids this failure mode by starting each attempt from scratch. As we show in Appendix F, it is also competitive in terms of *effective* decoding compute.

**Pythagoras-Prover is robust under statement-level perturbation.** To probe robustness, we re-evaluate DeepSeek-Prover-V2-671B, Goedel-Prover-V2-8B/32B, and Pythagoras-Prover-4B/32B on MiniF2F-ALF, applying our pass criterion uniformly. Figure 7 shows that every model loses accuracy relative to MiniF2F, confirming that ALF mutation induces a non-trivial distribution shift. Pythagoras-Prover-32B retains the highest absolute pass rate of any evaluated model (85.0%); Pythagoras-Prover-4B (83.2%) is almost on par with the 8× larger Goedel-Prover-V2-32B (83.6%) and degrades less under perturbation than that model does (2.9 vs. 4.5 points). Since our training corpus is itself ALF-augmented (§2.3), this is the empirical signature we would expect: models trained on ALF mutations should retain accuracy under ALF-mutated test data more readily than models that have not seen this kind of structural variation, and the robustness gap we observe relative to Goedel-Prover-V2 is consistent with this prediction. MiniF2F-ALF therefore plays two roles simultaneously: it acts as a sharper stress test than the saturated original split, preserving the same problem families while exposing failures under nearby formal variants; and it probes whether the augmentation strategy used during training transfers to the evaluation side. Under both readings, these results do not rule out high-level theorem-family overlap, but they suggest that Pythagoras-Prover is less sensitive than the evaluated baselines to surface-level changes in MiniF2F statements.

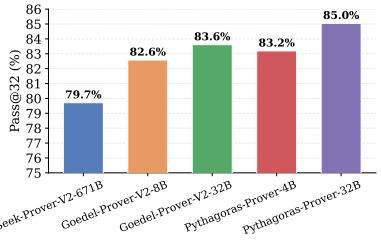



Figure 7: Performance of current state-of-the-art provers on MiniF2F-ALF. As MiniF2F-ALF is introduced in this work, all results are evaluated by us under a unified setup.

## C.2 Full Pass@N Results

Table 3 reports pass@$N$ on MiniF2F-Test across the full sweep of inference budgets we evaluate, covering every open-source baseline both with and without self-correction.

At the 4B scale, Pythagoras-Prover-4B reaches 86.1% at pass@32. This places it ahead of Goedel-Prover-V2-8B without self-correction (84.6%) and at near parity with that model's self-corrected variant (86.7%), despite using half the parameters and no inference-time correction loop. The advantage widens with budget: Pythagoras-Prover-4B attains 88.1% at pass@1024 and 89.8% at pass@2048, the latter already exceeding the pass@8192 result of DeepSeek-Prover-V2-671B (88.9%) at a quarter of the sampling budget and with roughly 167× fewer parameters.

At the 32B scale, Pythagoras-Prover-32B leads Goedel-Prover-V2-32B without self-correction by 1.7 points at pass@32 (89.8% vs. 88.1%), trails its self-corrected counterpart by only 0.6 points at the same budget (89.8% vs. 90.4%), and matches it at pass@1024 (92.6% vs. 92.6%). At pass@2048 it reaches **93.03**%, the strongest reported pass rate on MiniF2F-Test, surpassing the pass@8192 result of Goedel-Prover-V2-32B (92.2%) at a quarter of the budget and without any iterative repair step.

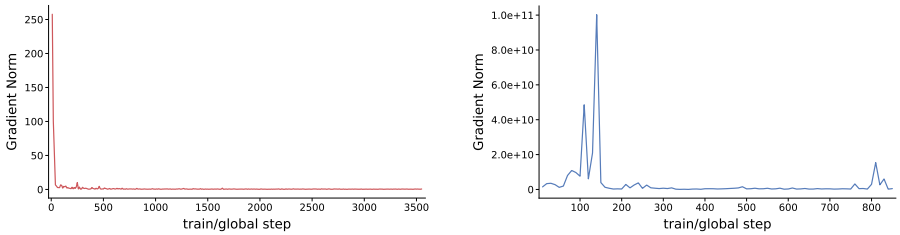
### C.3 Decomposition Performance

Table 4: Ablation of Pythagoras-Prover training components on MiniF2F-Test under a pass@32 inference budget. SFT-only rows isolate the contribution of the synthetic-data supervised fine-tuning stage prior to ALF self-distillation.

| Method | Budget | Pass@32 (%) |
|---|---|---|
| Pythagoras-Prover-4B (SFT only) | 32 | 79.10 |
| Pythagoras-Prover-4B (SFT + ALF Self-Distillation) | 32 | 86.07 |
| Pythagoras-Prover-32B (SFT only) | 32 | 84.02 |
| Pythagoras-Prover-32B (SFT + ALF Self-Distillation) | 32 | 89.75 |

Table 4 decomposes the contributions of supervised fine-tuning and ALF self-distillation on MiniF2F-Test at pass@32. Supervised fine-tuning on our synthetic-data corpus alone already yields a strong prover (79.10% at 4B and 84.02% at 32B) which we read as direct evidence that the corpus itself carries most of the signal required for competitive formal reasoning, prior to any further training stages. Adding ALF self-distillation lifts the two models to 86.1% and 89.8%, gains of +6.97 and +5.73 points respectively, showing that mutation-driven self-distillation delivers a consistent improvement on top of an already strong SFT baseline rather than substituting for it.

### C.4 Why Long-Context Diffusion Training Destabilises




(a) Stable training at 4,096-token context: gradient norm settles after warm-up and remains bounded throughout training.

(b) Unstable training at 8,192-token context: gradient norm fails to settle and exhibits sustained spikes well past warm-up.

Figure 8: Gradient-norm dynamics of Pythagoras-Prover-Diffusion-4B under the two context-length settings. The 8,192-token setting that converges cleanly under the autoregressive objective destabilises diffusion training, motivating the reduced 4,096-token budget used for Pythagoras-Prover-Diffusion (§2.4).

The instability illustrated in Figure 8 is not a tuning artefact but a consequence of the variance structure of the MDLM objective. We give the analysis here, and connect it to the design choices reported in prior masked-diffusion LM work.

**Variance of the per-sequence loss.** For a fixed sequence $\mathbf{x}_0$ of length $L$ and fixed noise level $t$, the forward process masks each position independently with probability $t$, so the number of masked positions is $|S| \sim \mathrm{Binomial}(L, t)$. The MDLM stochastic loss for this sample is

$$\widehat{\mathcal{L}}(\theta; \mathbf{x}_0, \mathbf{x}_t, t) \;=\; -\frac{1}{t} \sum_{i=1}^{L} \mathbf{1}\big[x_t^i = [\mathrm{M}]\big] \log p_\theta(x_0^i \mid \mathbf{x}_t). \tag{3}$$

Treating the per-position log-probabilities as bounded by an average magnitude $\bar{\ell}$, the conditional variance over the masking draw is

$$\mathrm{Var}\Big[\widehat{\mathcal{L}} \,\Big|\, t, \mathbf{x}_0\Big] \;\approx\; \bar{\ell}^{\,2} \cdot \frac{L\,(1-t)}{t}. \tag{4}$$

Here $\bar{\ell}$ denotes a representative per-position log-probability magnitude, $\bar{\ell} \approx |\log p_\theta(x_0^i \mid \mathbf{x}_t)|$, treated as roughly constant across positions. We stress that equation 4 is approximate: collapsing the

per-position losses to a single scale $\bar{\ell}$ treats them as independent and omits their cross-position covariance, whereas in practice the $\log p_\theta(x_0^i \mid \mathbf{x}_t)$ are correlated, since all masked positions are predicted from the same partially-masked context. We therefore read equation 4 as a behavioural scaling, $\mathrm{Var} \propto L(1-t)/t$, rather than an exact variance.

Although equation 4 is the variance of the loss, the gradient $\nabla_\theta \widehat{\mathcal{L}}$ shares the same $1/t$-reweighted sum-over-masked-positions structure, so its variance carries the same $L(1-t)/t$ scaling; this is the quantity reflected in the gradient-norm spikes of Figure 8b.

Two properties of equation 4 matter for stability: (i) the variance is *linear in the sequence length* $L$, so doubling the context from $4{,}096$ to $8{,}192$ doubles per-step gradient variance at every noise level; and (ii) the variance *diverges as* $t \to 0$, so rare low-$t$ realisations, which are unavoidable under the uniform schedule $t \sim \mathcal{U}[0,1]$ used by LLaDA (Nie et al., 2025) and by Diffusion-Pythagoras-Prover, produce heavy-tailed contributions whose magnitude also scales with $L$. The compound effect is that long-context training amplifies both the typical gradient variance and the tail mass of large-gradient steps, so even short bursts of low-$t$ samples are enough to push the optimiser past gradient clipping in sustained spikes, which is precisely the pattern visible in Figure 8b.

This effect compounds with the memory budget: doubling the context roughly halves the largest feasible batch size, so the per-sequence variance increase of equation 4 is averaged over fewer samples per step, further raising the gradient variance the optimiser actually sees.

**Extension to tactic-based masking.** The analysis above is stated for the token-level objective, whereas Pythagoras-Prover-Diffusion masks whole tactic spans (equation 2). The same argument transfers with the unit of masking changed from tokens to tactics. Writing $K$ for the number of tactic spans and $\ell_{\tau_k} \approx |\tau_k|\,\bar{\ell}$ for the summed per-token loss of span $\tau_k$, the per-sequence variance becomes

$$\mathrm{Var}\Big[\widehat{\mathcal{L}}_{\mathrm{tac}} \,\Big|\, t, \mathbf{x}_0\Big] \;\approx\; \frac{1-t}{t} \sum_{k=1}^{K} \ell_{\tau_k}^2 \;\approx\; \bar{\ell}^{\,2}\, \bar{s}\, \frac{L(1-t)}{t}, \tag{5}$$

where $\bar{s}$ is the mean tactic length (with equality when spans are equal-length; variable span lengths increase the bound further). Tactic-based masking therefore inherits the same $L$- and $t$-dependence and amplifies it by roughly $\bar{s}$, so the instability is, if anything, more pronounced for Pythagoras-Prover-Diffusion than for a token-level masked-diffusion model at the same context length.

**Contrast with autoregressive training.** This variance is specific to the diffusion objective. The autoregressive loss $-\sum_i \log p_\theta(x_0^i \mid \mathbf{x}_0^{<i})$ is a deterministic sum over all $L$ positions, with no masking draw and no $1/t$ reweighting, so it carries none of the $L(1-t)/t$ terms. This is why the same $8{,}192$-token setting that destabilises diffusion converges cleanly under the autoregressive objective (§2.4).

**Consistency with prior masked-diffusion LM work.** Every existing masked-diffusion LM that scales to large models trains at a relatively short, fixed context window and explicitly engineers around the variance term of equation 4. LLaDA (Nie et al., 2025) pretrains at a *fixed* $4{,}096$-token context and reserves $1\%$ of the data for random shorter lengths, with no recipe given for $8{,}192$. MDLM (Sahoo et al., 2024) reduces ELBO variance through a low-discrepancy $t$-sampler that draws correlated $t_i$ rather than i.i.d.. Block diffusion (Arriola et al., 2025) goes further still, introducing gradient-variance estimators and data-driven noise schedules and decomposing the sequence into smaller blocks so that no single forward pass has to absorb the full $L(1-t)/t$ variance term. The shared pattern across these works is that long-sequence stability in masked diffusion is treated as a *variance-reduction* problem rather than an optimiser-hyperparameter problem, which matches our empirical observation: the only intervention that consistently stabilised Pythagoras-Prover-Diffusion at $8{,}192$ tokens in our preliminary experiments was reducing the context budget, while increased warm-up, lower learning rate, and tighter gradient clipping each only postponed the spike pattern of Figure 8b. We therefore use the $4{,}096$-token budget throughout for Pythagoras-Prover-Diffusion, and leave variance-reduction techniques (low-discrepancy / antithetic $t$-sampling, block decomposition) to future work.

### C.5 Training Performance between full context and dynamic *proof-reasoning filtering*

In this section, we perform an ablation study between full-context training and our full-context dynamic training with proof-reasoning filtering. Both settings share the same model backbone, synthetic-data corpus, optimiser configuration, and SFT recipe; they differ only in whether reasoning traces are filtered on the fly during training. The full-context baseline trains on every reasoning–proof pair extracted from the synthetic corpus, whereas the dynamic-filtering variant prunes each minibatch in real time by discarding pairs whose reasoning trace is inconsistent with its closing tactic sequence, concentrating gradient updates on traces whose chain-of-thought actually supports the discharged theorem.

Table 5 reports pass@32 on MiniF2F-Test under both settings. The dynamic filter delivers a consistent improvement at both scales, $+3$ solved instances at 4B and $+2$ instances at 32B, without changing parameter count, training budget, or inference protocol. The gain is meaningful in the near-saturated regime of MiniF2F-Test: the 4B model picks up three additional problems on a benchmark where the four-prover panel of §5.2 leaves only $41/244$ at-least-one-wrong cases to contest. The smaller absolute gain at 32B is consistent with the diminishing-returns pattern observed elsewhere in the analysis: the 32B baseline already discharges most reasoning-noisy traces robustly, so filtering shifts mass on a narrower residual.

Table 5: Ablation of dynamic proof-reasoning filtering against full-context training on MiniF2F-Test at pass@32. Both rows share the SFT recipe and synthetic-data corpus; the only difference is whether reasoning traces are filtered on the fly during training. Values are pass rates (%) with absolute solved counts out of 244 in parentheses.

| **Training setting** | **Pythagoras-Prover-4B** | **Pythagoras-Prover-32B** |
|---|---|---|
| Full context (no filtering) | 79.10 | 83.20 |
| + Dynamic proof-reasoning filtering | 80.70 | 84.02 |
| $\Delta$ | $+1.60$ | $+0.82$ |

### C.6 Further Experimental Results for Pythagoras-Prover-Diffusion: Extended Results and Analysis

Pythagoras-Prover-Diffusion-4B achieves $63.25\%$ at pass@32 on MiniF2F-Test with block size 32, approximately 23 percentage points below autoregressive Pythagoras-Prover-4B ($86.1\%$). Both use the ALF self-distillation corpus (§2.3), demonstrating that this corpus supports both autoregressive generation and diffusion-style iterative refinement.

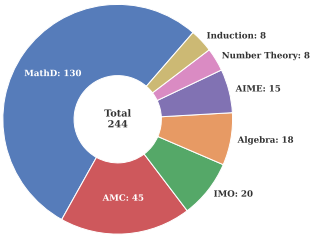
**Training constraints.** We attribute the gap to three constraints: (i) the random-masking objective provides sparser supervision than next-token prediction; (ii) current diffusion language models have limited stable context lengths (Ye et al., 2025; Leang et al., 2026a; Nie et al., 2025); and (iii) diffusion-suitable Lean data that remain informative within these context limits are scarce. Tactic-based masking (§2.4) addresses the first constraint and ALF self-distillation (§2.3) the third, while context length remains limiting.

**Context length and initialisation.** Autoregressive Pythagoras-Prover-4B converges stably with an $8{,}192$-token context, whereas diffusion training at the same length exhibits sustained high gradient norms beyond warm-up (Figure 8). Reducing the training context to $4{,}096$ tokens restores stable convergence; evaluation uses an $8{,}192$-token budget. Retraining autoregressive Pythagoras-Prover-4B at the matched $4{,}096$-token context substantially reduces its accuracy (Figure 4), indicating that restricted training context contributes materially to the gap. The autoregressive model nevertheless retains an advantage. We attribute this residual gap partly to the Qwen backbone's more extensive autoregressive pretraining and post-training, which provide a stronger initialisation for Lean proving.

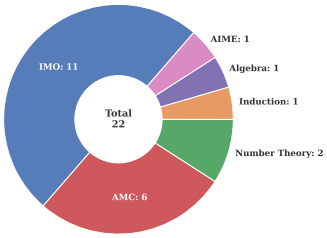
This context constraint is particularly relevant to competitive theorem proving, where test-time scaling (Chen et al., 2025; Lin et al., 2026; Varambally et al., 2026) relies on long reasoning chains routinely exceeding $4{,}096$ tokens. The shorter stable training context therefore remains a significant limitation even when evaluation permits longer outputs.

**Scope and outlook.** We present these results as an existence proof, rather than a final performance limit: a diffusion language model trained on our corpus can verifiably solve Lean theorems at non-trivial rates. Longer contexts may improve its ability to capture dependencies ranging from parenthetical agreement to logical relations between clauses. Diffusion's random access to token generation could also support constructing proofs section by section according to these dependencies, rather than strictly following token order.

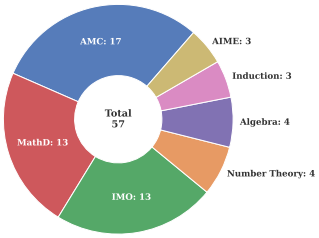
## D Detailed Analysis of MiniF2F




(a) Domain composition of the full MiniF2F-Test benchmark (all instances).




(b) Domain composition of the all-model-wrong set on original MiniF2F-Test.




(c) Domain composition of the at-least-one-model-wrong set on MiniF2F-ALF, after applying the ALF mutation scheme to each MiniF2F-Test statement.

Figure 9: Categories are taken directly from MiniF2F. The competition-derived slices comprise *IMO* (International Mathematical Olympiad), *AMC* (American Mathematics Competitions, AMC 8/10/12), and *AIME* (American Invitational Mathematics Examination); *MathD* groups problems imported from the MATH (Hendrycks et al., 2021). The remaining slices are topic-tagged hand-written instances grouped by area: *Algebra*, *Number Theory*, and *Induction*.

Extending the analysis of §5.2, we examine the domain composition of the residual failure set on original MiniF2F-Test under our four-prover panel (Pythagoras-Prover-4B, Pythagoras-Prover-32B, Goedel-Prover-V2-8B, Goedel-Prover-V2-32B). Figure 10 shows that the at-least-one-model-wrong subset is dominated by olympiad-level material: AMC and IMO together account for roughly two-thirds of all failures, with the remaining categories forming a thin long tail. The all-model-wrong subset (Figure 9b) is even more skewed: IMO alone accounts for half of all universal failures and AMC for the bulk of the remainder, so the residual difficulty on which contemporary provers can still be separated sits almost entirely at the upper end of the difficulty spectrum. Combined with the headline saturation statistic that $83.20\%$ of MiniF2F-Test is solved by all four systems, this leaves only a narrow band of olympiad instances on which the benchmark can meaningfully discriminate between strong provers. Figure 6 breaks this residual set down per problem, showing for each prover which instances it solves and which it misses, and confirming that disagreement is confined to a smaller, olympiad-heavy tail.

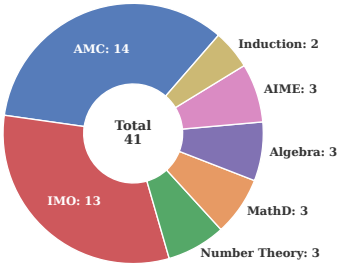



Figure 10: Domain composition of the at-least-one-model-wrong set on original MiniF2F.

Figure 9c shows the same decomposition after ALF mutation. The distribution shifts substantially: MathD, a small slice of the original failure set, rises to roughly $20\%$ of at-least-one-wrong instances on MiniF2F-ALF, and Algebra, Number Theory, Induction, and AIME each occupy a non-trivial share. The IMO/AMC concentration that characterises the original failure set is broken, with failures redistributing across categories that should not be intrinsically harder than instances the same systems solve on the original split. This redistribution is the empirical signature we look for: rather than amplifying the olympiad tail, ALF surfaces a broader set of structurally simple cases on which strong provers turn out to be brittle, providing discriminative signal across the full difficulty range rather than only at its top.

### D.1 MiniF2F-ALF Failure and Disagreement Analysis

**Failure rates and composition.** MiniF2F-ALF holds problem families fixed while applying controlled statement mutations. The all-model failure rate increases from $9.02\%$ on MiniF2F-Test to $13.93\%$ $(68/488)$, while the at-least-one-wrong set contains $18.24\%$ $(89/488)$ of mutated instances. Failures diversify across categories: AMC accounts for $30.34\%$, IMO for $24.72\%$, MathD for $20.22\%$, Algebra for $7.87\%$, Number Theory for $6.74\%$, Induction for $5.62\%$, and AIME for $4.49\%$. This redistribution exposes weaknesses beyond the dominant competition categories, showing where structural generalisation fails under the same ALF operator used for training augmentation (§2.3).

**Pairwise model comparisons.** On original MiniF2F-Test, Pythagoras-Prover-4B solves 7 problems missed by Goedel-Prover-V2-8B, with 0 exclusive solves in the reverse direction. The corresponding counts for Pythagoras-Prover-32B and Goedel-Prover-V2-32B are 8 and 1. Under ALF mutation, the smaller-model comparison yields 6 exclusive solves for Pythagoras-Prover-4B and 3 for Goedel-Prover-V2-8B, increasing total disagreements from 7 to 9. At 32B, the respective counts become 12 and 5, increasing disagreements from 9 to 17. Thus, Pythagoras-Prover retains more exclusive solves at both scales on both benchmarks, while mutation reveals additional complementary failures.

These increases concern absolute counts: MiniF2F-Test contains 244 statements, whereas MiniF2F-ALF contains 488. The disagreement rates therefore change from $2.87\%$ to $1.84\%$ for the smaller-model pair and from $3.69\%$ to $3.48\%$ for the 32B pair. Accordingly, the comparisons reveal different failure patterns but do not establish a higher disagreement rate. Figure 6 provides a complementary diagnostic: its orange region isolates instances solved on the original benchmark but failed after mutation.

## E A Failure Mode of Self-Correction

**Putnam 1966 A4.** Non-square positive integers indexed by closest-square distance.

**Formal statement**

```
theorem putnam_1966_a4
    (a : ℕ → ℤ)
    (ha1 : a 1 = 2)
    (hai : forall n >= 1, a (n + 1) =
        (if exists m : ℤ, m^2 = a n + 1 = True
         then a n + 2
         else a n + 1)) :
   forall n >= 1, a n = n + round (Real.sqrt n) := by
  sorry
```

**Putnam 1966 A4: Pythagoras-Prover successful proof.**

```
theorem putnam_1966_a4
(a : ℕ → ℤ)
(ha1 : a 1 = 2)
(hai : forall n >= 1, a (n + 1) =
    (if exists m : ℤ, m^2 = a n + 1 = True then a n + 2 else a n + 1))
: forall n >= 1, a n = n + round (Real.sqrt n) := by
  have h_round_of_block : forall (n k : ℕ),
    [TRUNCATE]

  have h_square_branch_of_block : forall (n k : ℕ),
      k * k - k + 1 ≤ n → n ≤ k * k + k →
      (((exists m : ℤ, m ^ 2 = (n : ℤ) + (k : ℤ) + 1) = True) ↔ n = k * k + k) := by
    [TRUNCATE]

  have h_block : forall n : ℕ, n >= 1 → exists k : ℕ,
      a n = (n : ℤ) + (k : ℤ) and
      k * k - k + 1 ≤ n and n ≤ k * k + k := by
    [TRUNCATE]

  intro n hn
  [TRUNCATE]
```

**Putnam 1966 A4: A self-correction failure mode.** A structural induction on $n$ that decomposes the non-square indices into blocks $[k^2 - k + 1,\ k^2 + k]$, with a rounding lemma identifying $\mathrm{round}(\sqrt{n})$ inside each block.

**Goedel-Prover-V2 (first round, failed)**

```
theorem putnam_1966_a4
(a : ℕ → ℤ)
(ha1 : a 1 = 2)
(hai : forall n >= 1, a (n + 1) = (if exists m : ℤ,
m^2 = a n + 1 then a n + 2 else a n + 1))
: forall n >= 1, a n = n + round (Real.sqrt n) :=
by
  have h_main : forall (n : ℕ), n >= 1 → a n = n +
round (Real.sqrt n) := by
    intro n hn
    have h₁ : forall n : ℕ, n >= 1 → a n = n +
round (Real.sqrt n) := by
      intro n hn
      induction' hn with n hn IH
      · norm_num [ha1]
        <;> simp [round_eq, Int.floor_eq_iff,
Real.le_sqrt, Real.sqrt_lt]
        <;> norm_num
        <;> linarith [Real.sqrt_nonneg 1,
              Real.sq_sqrt (show 0 ≤ (1 : ℝ) by
norm_num)]
      · have h₂ : a (n + 1) =
            (if exists m : ℤ, m ^ 2 = a n + 1 then
a n + 2 else a n + 1) := by
          apply hai
          <;> omega
        rw [h₂]
        have h₃ : a n = n + round (Real.sqrt n) :=
IH
        rw [h₃]
        have h₄ : (if exists m : ℤ, m ^ 2 = (n : ℤ)
+ round (Real.sqrt n) + 1
                    then (n : ℤ) + round (Real.sqrt
n) + 2
                    else (n : ℤ) + round (Real.sqrt
n) + 1)
                  = (n + 1 : ℤ) + round (Real.sqrt (n
+ 1)) := by
          split_ifs with h
          · have h₈ : round (Real.sqrt (n + 1 : ℝ))
                      = round (Real.sqrt (n : ℝ)) + 1
:= by
              sorry
            sorry
          · have h₈ : round (Real.sqrt (n + 1 : ℝ))
                      = round (Real.sqrt (n : ℝ)) :=
by
              sorry
            sorry
        rw [h₄]
    exact h₁ n hn
  intro n hn
  exact h_main n hn
```

**Self-correction round (failed)**

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat Filter

theorem putnam_1966_a4
(a : ℕ → ℤ)
(ha1 : a 1 = 2)
(hai : forall n >= 1, a (n + 1) = (if exists m : ℤ,
m^2 = a n + 1 then a n + 2 else a n + 1))
: forall n >= 1, a n = n + round (Real.sqrt n) :=
by
  have h_main : forall n >= 1, a n = (n : ℤ) +
round (Real.sqrt n) := by
    intro n hn
    have h₁ : n ≤ 16 or n > 16 := by omega
    cases h₁ with
    | inl h₁ =>
      interval_cases n
      <;> norm_num [ha1, hai, round_eq,
Int.floor_eq_iff,
                    Real.sqrt_le_iff, Real.lt_sqrt,
Real.sqrt_lt] at *
      <;> (try { nlinarith [Real.sqrt_nonneg 2,
              Real.sq_sqrt (show (0:ℝ) ≤ 2 by
norm_num)] })
      <;> (try { nlinarith [Real.sqrt_nonneg 3,
              Real.sq_sqrt (show (0:ℝ) ≤ 3 by
norm_num)] })
      <;> (try { nlinarith [Real.sqrt_nonneg 5,
              Real.sq_sqrt (show (0:ℝ) ≤ 5 by
norm_num)] })
      <;> (try { aesop }) <;> (try { linarith })
    | inr h₁ =>
      exfalso
      have h₃ : a 1 = 2 := ha1
      have h₄ : a 2 = 3 := by
        have h₅ := hai 1 (by norm_num); norm_num
[ha1] at h₅ ⊢
        <;> simp_all [ha1] <;> aesop
      have h₅ : a 3 = 5 := by
        have h₆ := hai 2 (by norm_num); norm_num
[h₄] at h₆ ⊢
        <;> simp_all [ha1, h₄] <;> aesop
      have h₆ : a 4 = 6 := by
        have h₇ := hai 3 (by norm_num); norm_num
[h₅] at h₇ ⊢
        <;> simp_all [ha1, h₄, h₅] <;> aesop
      have h₇ : a 5 = 7 := by
        have h₈ := hai 4 (by norm_num); norm_num
[h₆] at h₈ ⊢
        <;> simp_all [ha1, h₄, h₅, h₆] <;> aesop
      have h₁₈ : a 16 = 20 := by
        have h₁₉ := hai 15 (by norm_num); norm_num
[h₁₇] at h₁₉ ⊢
        <;> simp_all [ha1, h₄, h₅, h₆, h₇, h₈, h₉,
h₁₀,
                      h₁₁, h₁₂, h₁₃, h₁₄, h₁₅, h₁₆,
h₁₇]
        <;> aesop
      <;> norm_num <;> aesop
  intro n hn
  have h₁ : a n = (n : ℤ) + round (Real.sqrt n) :=
h_main n hn
  norm_cast at h₁ ⊢
  <;> simp_all [add_assoc] <;> linarith
```

## F Effective Token Complexity Calculation

Raw generated-token counts do not fully capture the decoding cost of self-correction. With KV caching, each newly generated token still attends to the entire prefix available at that step. Hence, under dense attention the work of generating $a$ tokens from a context of length $m$ grows super-linearly. To capture this effect, we define the *effective token complexity*, a worst-case proxy for cumulative attention work:

$$\mathrm{ETC}(m, a) \;=\; ma \;+\; \frac{a(a+1)}{2},$$

Table 6: Average input and generated token counts used for the effective token complexity (ETC) estimate. Goedel-Prover decodes three self-correction rounds per attempt; Pythagoras-Prover draws independent restarts from a short input context.

| Method | Pass@ | Input context (tokens) | Avg. generated tokens |
|---|---|---|---|
| Goedel-Prover, round 1 | 184 | 284.88 | 18,367.83 |
| Goedel-Prover, round 2 | 184 | 7,838.74 | 21,638.70 |
| Goedel-Prover, round 3 | 184 | 8,309.06 | 23,835.80 |
| Pythagoras-Prover restart | 1024 | 284.88 | 18,741.80 |

where $ma$ is the cost of attending to the fixed input at every step and $a(a{+}1)/2$ the cost of the growing generated prefix. ETC counts cumulative query–key interaction under an idealised dense-attention model; it is not a wall-clock measurement, and Appendix F states the assumptions and limitations in full. We note that ETC is size-agnostic: it counts query–key interactions without reference to model width, and so neither credits nor penalises Pythagoras-Prover for its smaller parameter count. We therefore do not read the matched-ETC comparison as a model-size efficiency result; it reflects only the difference in how much context each attempt conditions on: a short prompt for restart sampling versus long accumulated histories for self-correction. Self-correction and restarting differ sharply under this measure: later self-correction rounds condition on long accumulated contexts, whereas each restart conditions only on a short prompt. Table 6 reports the relevant token statistics. Goedel-Prover-V2 runs three self-correction rounds at pass@184, with generation lengths of $18.4$K, $21.3$K, and $23.0$K and roughly $15$K of additional proof context across rounds two and three, giving an estimated $\mathrm{ETC} \approx 1.95 \times 10^{11}$ (full derivation in Appendix F). Pythagoras-Prover with $1024$ independent restarts conditioned on a short ($\approx 285$-token) prompt with $\approx 18.7$K generated tokens each, for $\mathrm{ETC} \approx 1.85 \times 10^{11}$. At comparable ETC, Pythagoras-Prover solves $88$ problems versus Goedel-Prover-V2's $86$, so restart sampling matches or improves on self-correction without incurring the decoding cost of long correction histories.

**Intuition.** Wall-clock cost and pass@$N$ are both misleading proxies for compute when two systems decode under very different regimes: pass@$N$ ignores per-attempt length, and wall-clock conflates batching, hardware, and serving stack. We instead estimate the work the attention mechanism performs under an idealised model. For a single decoded block of $a$ tokens conditioned on an input of $m$ tokens, the generated token at within-block position $t$ attends to the $m$ prompt tokens together with the $t$ tokens produced up to and including itself ($m + t$ keys in total), so the cumulative query–key work over the block is

$$\sum_{t=1}^{a}(m+t) \;=\; \underbrace{m\,a}_{\text{prompt attention}} \;+\; \underbrace{\frac{a(a+1)}{2}}_{\text{self-attention}}. \tag{6}$$

We call the quantity in equation 6 the *effective token complexity* (ETC), denoted by $\mathrm{ETC}(m, a)$. The first term scales linearly with the input context (attending to a fixed prompt across the whole block); the quadratic term is the intra-block self-attention cost, and it dominates whenever $a$ is comparable to or larger than $m$. For pass@$N$ we multiply the per-attempt ETC by $N$, summing across rounds for multi-round systems.

**Assumptions and scope.** ETC is deliberately idealised and should be read as a worst-case, asymptotic proxy rather than an estimate of realised inference cost; three caveats apply. First, it assumes *dense global attention* in every layer, so each step pays a cost proportional to its full context length. This does not hold for models that replace global attention with sparse attention (e.g. the Qwen3 family) or linear or gated delta net attention (e.g. Qwen3.5-Omni) in most layers, for which the per-step context cost is sub-linear and ETC over-counts the linear term. Second, ETC ignores inference-engine optimisations such as KV- and prefix-caching, as well as the memory-bound versus compute-bound trade-off that depends jointly on context length and batch size. Third, it is not a FLOP count: converting it to FLOPs would require architecture-specific factors (model width, head count, attention type), not a single multiplicative constant. What survives these caveats is the *geometric asymmetry* the measure is designed to expose, which holds under any super-linear attention cost: generating many short independent blocks is asymptotically cheaper than conditioning on a long

accumulated context. Concretely, $k$ independent blocks of length $n$ cost on the order of $k \cdot n^2/2$ under the self-attention term, against $(kn)^2/2 = k^2 \cdot n^2/2$ for a single block of the same total length, a factor-$k$ saving. Restart sampling realises this saving; multi-round self-correction does not. We therefore use ETC only to characterise this scaling behaviour, and report the per-attempt token statistics in Table 6 so the estimate can be recomputed under any alternative cost model.

**Illustrative ETC under the idealised model.** We report per-attempt ETC values under the dense-attention idealisation to illustrate the scaling regime, not as an estimate of realised decoding cost; the caveats above (in particular the over-counted linear term under sparse attention) mean the absolute figures should be read as an upper-bound-style illustration rather than a measurement. For Goedel-Prover, summing its three self-correction rounds at pass@184 with the per-round token statistics in Table 6, we obtain

$$\begin{aligned}\mathrm{ETC}_{\text{Goedel-Prover}} = 184 \Big[ & \underbrace{\mathrm{ETC}(284.88,\ 18367.83)}_{\textit{round 1}} \\ & + \underbrace{\mathrm{ETC}(7838.74,\ 21638.70)}_{\textit{round 2}} \\ & + \underbrace{\mathrm{ETC}(8309.06,\ 23835.80)}_{\textit{round 3}} \Big] \\ \approx\ & 184 \left[ 1.7393{\times}10^8 + 4.0375{\times}10^8 + 4.8214{\times}10^8 \right] \\ \approx\ & 1.95 \times 10^{11}.\end{aligned}$$

For Pythagoras-Prover, with an input context of $284.88$ tokens and an average restart length of $18{,}741.80$ tokens at pass@1024,

$$\begin{aligned}\mathrm{ETC}_{\text{Pythagoras-Prover}} &= 1024 \cdot \mathrm{ETC}(284.88,\ 18741.80) \\ &\approx 1024 \cdot 1.8098{\times}10^8 \ \approx\ 1.85 \times 10^{11}.\end{aligned}$$

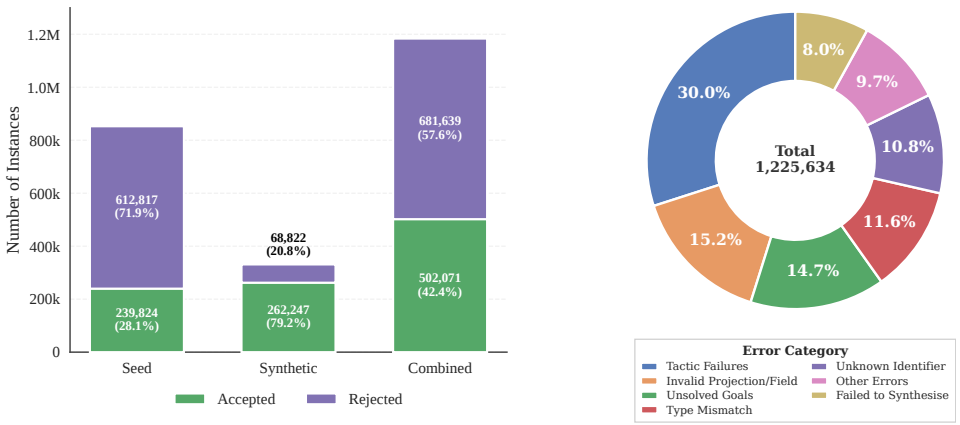
Under this idealisation the two estimates are within roughly $5\%$ of one another. We deliberately do *not* read this margin as evidence that either system has lower cost: a $5\%$ difference lies well inside the error introduced by assuming dense global attention, and because Goedel-Prover's later rounds condition on much larger input contexts $m$ than our restarts, the linear-term over-counting does not cancel between the two systems. Moving off dense attention, to the sparse or linear kernels used in practice, lowers the absolute cost of *both* regimes but does not reverse the direction of the asymmetry established above, since that asymmetry arises from how context grows across attempts rather than from the attention kernel. We therefore treat the two systems as operating at *comparable* effective token complexity, at which Pythagoras-Prover solves more problems ($88$ vs. $86$), and rely on the scaling behaviour rather than the point estimate.

### F.1 Dataset Decomposition

We present a detailed analysis of the Lean-verified seed corpus underlying our training pipeline. The seed is constructed in three difficulty tiers (§2.1), produced by two complementary mechanisms: an initial autoformalisation pass on natural-language problems from existing mathematical-reasoning datasets, which yields the medium and hard tiers (§2.1.1, §2.1.2); and a rubric-guided distillation stage (§2.1.1) that simplifies failed medium instances to produce the easy tier. The ALF augmentation of §2.3 is layered on top of this seed and is treated separately.

#### F.1.1 The Easy and Medium Tiers

**Seed verification (Medium).** The seed collection from DART-Math-Hard, OpenR1-Math, and DeepScaleR yields $852{,}641$ instances, of which $239{,}824$ ($28.1\%$) are verified by the Lean type-checker and $612{,}817$ ($71.9\%$) are rejected, as shown by the *Seed* bar in Figure 11a. DART-Math-Hard contributes the majority of verified proofs ($199{,}452$; $83.2\%$ of all verified seeds), reflecting both its larger scale and its emphasis on competition-style problems that tend to be more amenable to automated formalisation. OpenR1-Math and DeepScaleR contribute $28{,}530$ ($11.9\%$) and $11{,}842$




(a) Accepted and rejected instances across seed, synthetic, and combined data.

(b) Distribution of error categories for rejected pass@2 attempts.

Figure 11: Data filtering and error analysis. Left: accepted and rejected instances across data sources. Right: distribution of error categories among rejected attempts.

(4.9%) respectively. The substantial gap between DART-Math-Hard and the remaining sources motivates the rubric-guided distillation stage below, which targets the harder instances that initial proof synthesis could not resolve.

**Error analysis and rubric-guided synthesis.** We categorise every type-checker failure on the 612,817 rejected seeds into the seven error classes of §2.1.1; Figure 11b reports the distribution. The dominant failure modes are tactic failures (30.0%, concentrated in `linarith`, `simp`, `ring`, and `omega`), unsolved goals (14.7%), invalid projection/field errors (15.2%), and type mismatches (11.6%, typically between $\mathbb{N}$ and $\mathbb{Z}$ or related subtypes); unknown-identifier (10.8%), failed-to-synthesise (8.0%), and a long tail of miscellaneous errors (9.7%) account for the remainder. As these categories are not mutually exclusive at the instance level, our rubric addresses each class with targeted simplification rather than a single monolithic correction. From the rejected pool we retain 258,648 instances with actionable error signatures and have Qwen3-30B and Qwen3-235B each produce $n$=2 simplified question variants. After near-duplicate filtering, the two models retain $198,641$ and $132,428$ instances respectively, whose union of 331,069 forms the easy synthetic pool.

**Synthetic verification (Easy) and final composition.** As shown by the *Synthetic* bar in Figure 11a, of the 331,069 candidates entering Lean verification, 262,247 (79.2%) pass and 68,822 (20.8%) are rejected, a 51.1-point absolute gain over the seed verification rate (28.1%) that is consistent with the rubric's intent: each easy variant isolates the single manipulation responsible for an original failure and is therefore far more amenable to automated proof synthesis. Combined with the seed corpus, the easy/medium dataset comprises $2.04M$ instances with 502,071 verified proofs. Synthetic augmentation alone contributes 262,247 verified instances on top of the 239,824 seed-verified instances, approximately a 2x increase in verified training data, obtained without any additional human annotation.

### F.1.2 The Hard Tier

The hard tier comprises $328,966$ instances drawn from the hardest subset of our seed sources. Unlike the easy and medium tiers, we do *not* apply rubric-guided synthesis here, since the hardest problems admit no faithful simplification. We instead retain the standard generate-and-verify pipeline and raise the per-problem sampling budget to $n$=4 to compensate for the increased difficulty, yielding approximately 1.32M candidate instances before Lean filtering. After Lean verification with $n$=4 per instance, $338,683$ instances (25.74%) remain. We read this as evidence that the larger sampling budget offsets the difficulty gap, yielding a hard-tier corpus of comparable density without resorting to simplifications that would alter the underlying problems.

## G LIMITATIONS AND FUTURE WORK

While our work makes several contributions, a number of natural extensions remain. (1) *Diffusion-based formal proving.* Although Pythagoras-Prover-Diffusion currently trails its autoregressive counterpart, we believe diffusion models represent a promising research direction due to the mechanism of iterative refinement, and a meaningful architectural shift for LLM reasoning. Scaling the diffusion regime to longer effective contexts, and hybridising parallel diffusion refinement with autoregressive verification, are natural next steps. (2) *Richer mutation operators.* Our data pipeline uses five mutation operators, which already yields a substantial diagnostic uplift on MiniF2F-ALF. We believe mutation is an efficient route to generating synthetic formal data, and that a broader operator family, including type-driven, lemma-graph, and proof-trace mutations, could further widen the failure surface that future provers must handle. (3) *Beyond MiniF2F.* Our analysis indicates that MiniF2F is approaching saturation under contemporary Lean provers. Existing alternative benchmarks offer only partial substitutes: many are calibrated to a difficulty regime in which strong open-source language models register pass rates in the low single digits, while others have already approached saturation under current systems — in either case providing limited resolution for distinguishing among the strongest provers. We position MiniF2F-ALF as a stronger drop-in variant within the MiniF2F family, but a broader community effort is needed to construct evaluation suites with controlled difficulty stratification across theorem-proving regimes. (4) *Attribution of modest RL gains.* In §2.2 we attribute the modest gains from RL over the curriculum-SFT model to the policy already internalising strong proof-search behaviour before RL. We acknowledge this is not the only plausible explanation: the sparsity of the binary pass/fail reward signal under GRPO is a complementary factor, since a single compile-or-not outcome provides limited gradient signal on long formal proofs. Disentangling these two effects, for instance through denser, process-level reward signals, is a promising direction for future work.