ProofFrontier: Turning Verified Proofs into Boundary-hard Tasks for Formal Theorem Proving

Anonymous ACL submission

Abstract

Formal theorem proving provides executable verification signals from proof assistants, making it a key domain for improving the reliable reasoning ability of large language models through reinforcement learning (RL). However, RL-based prover training is increasingly bottlenecked by the scarcity of high-quality hard tasks: many existing Lean instances are either already solvable by current provers or too uninformative to drive further improvement. We propose **ProofFrontier**, a proof-structure-guided method for generating statement-preserving boundary-hard training tasks from verified Lean proofs. Instead of synthesizing open-ended problems that may be ill-posed or unverifiable, ProofFrontier starts from existing theorems with Lean-certified proofs, preserving task completeness and formal verifiability. It selectively withholds proof-structural guidance around important local nodes, encouraging the prover to rediscover nontrivial intermediate reasoning while keeping the original theorem statement unchanged. We further introduce capability-aware filtering to retain tasks that are solvable in principle but not easily solved by the base prover, yielding hard yet rewardable RL data. Training with GRPO on ProofFrontier-generated tasks consistently improves DeepSeek-Prover-V2 and Goedel-Prover-V2 on miniF2F, ProofNet, and PutnamBench. Notably, these gains are achieved using only hundreds of synthesized tasks, showing that complete boundary-hard tasks can be more data-efficient than simply scaling weak or already-solved proof data for theorem-proving RL.

1 Introduction

Formal theorem proving provides a rare environment for reinforcement learning (RL) with exact and executable feedback. In proof assistants such as Lean, a proof trajectory is automatically checked by a verifier and either accepted or rejected, yielding an unambiguous reward signal. This makes formal theorem proving a natural testbed for studying reliable reasoning in large language models, especially in settings that require long-horizon reasoning, tool interaction, and verifiable trial-and-error learning. Recent progress in neural theorem proving has been driven by stronger pretrained models and larger formal corpora. Recent prover systems such as DeepSeek-Prover-V2 (Ren et al., 2025) and Goedel-Prover-V2 (Lin et al., 2025) further demonstrate the effectiveness of combining large-scale formal data, proof assistant feedback, synthetic data, and RL. These advances suggest that future progress depends not only on model scale or search, but also on the quality of training tasks used to expose useful verifier-grounded learning signals.

Despite this progress, RL for theorem proving remains constrained by the cost of obtaining high-quality training tasks that are both challenging and rewardable under proof-assistant verification. Verifier rewards in theorem proving are sparse, and their usefulness depends strongly on the task distribution: tasks already solved by the base prover provide little learning pressure, whereas tasks far beyond its reach yield mostly failed rollouts. Effective RL therefore requires **complete hard tasks**: tasks that are guaranteed to admit a Lean-verifiable solution, yet remain difficult for the current prover to solve. Existing formal corpora contain many certified theorem–proof demonstrations, but they are not organized as RL tasks near a prover's solvability frontier. They certify that proofs exist, but do not specify which proof information should be exposed or hidden to create challenging and verifiable training tasks. This creates a data bottleneck for verifier-grounded RL.

In this work, we propose **ProofFrontier**, a proof-structure-guided data synthesis method for constructing complete hard theorem-proving tasks from verified Lean proofs. Our key observation is

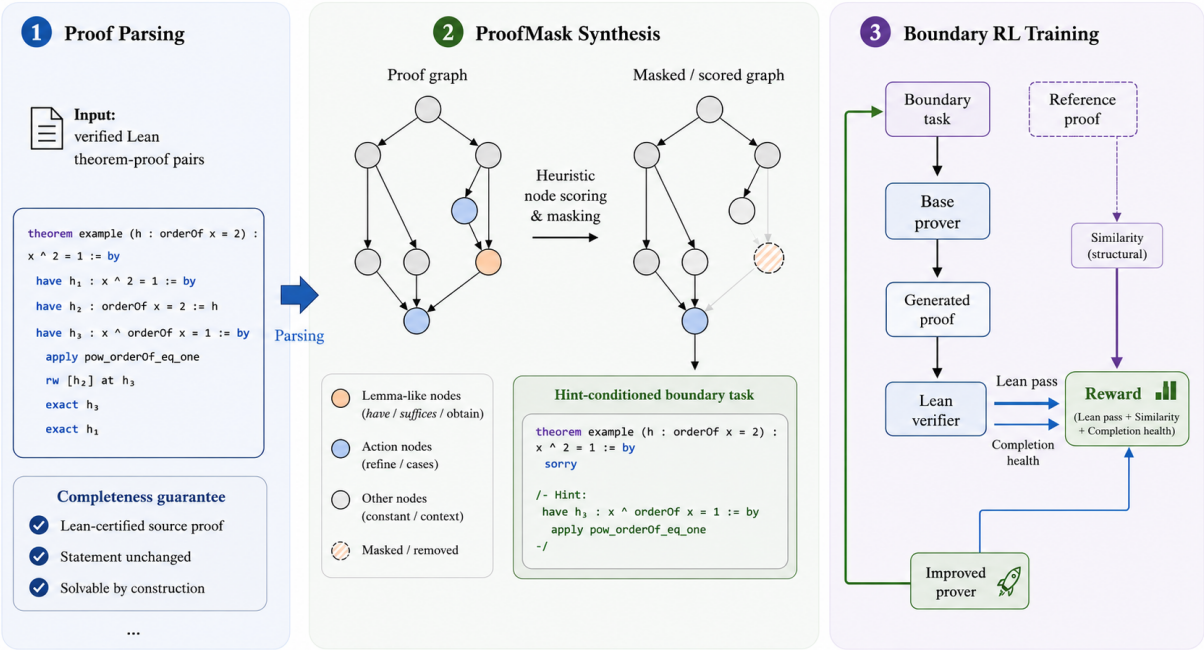

Figure 1: Overview of ProofFrontier. Starting from verified Lean theorem–proof pairs, we parse each proof into a local proof graph and score structurally important nodes. ProofFrontier builds node-centered mask families that control how much proof-structural information is visible, and converts the visible metadata into hints while preserving the original theorem statement. Base-prover pass rates are used to select boundary-hard tasks near the capability boundary, which are then used for verifier-grounded RL training.

that a formal proof is not merely a demonstration for imitation learning, but also a structured source from which challenging reconstruction problems can be derived. Starting from a verified theorem–proof pair, ProofFrontier parses the proof into a local proof graph and identifies structurally high-impact components, such as intermediate claims, dependency-bearing nodes, and key proof actions. It then constructs a family of candidate tasks by selectively hiding proof-structural information while optionally exposing controlled proof hints. Each task keeps the original theorem statement and requires the prover to produce a complete Lean proof, so solvability and formal verifiability are inherited from the source proof.

To turn these candidates into useful RL data, ProofFrontier applies capability-aware filtering. For each task family, we estimate the base prover's success rate under different masking levels and retain tasks that are solvable in principle but not easily solved in practice. These selected tasks lie near the prover's empirical solvability frontier: they are difficult enough to induce learning, yet still admit executable verification through Lean. During RL training, we use verifier feedback as the primary reward. Since each task is derived from a known verified proof, we also include a weak structure-aware auxiliary reward that measures similarity to the original proof trajectory, providing additional shaping signal without replacing verifier-based correctness.

We evaluate ProofFrontier on miniF2F (Zheng et al., 2021), ProofNet (Azerbayev et al., 2023), and PutnamBench (Tsoukalas et al., 2024). Across multiple prover backbones, training on ProofFrontier data consistently improves theorem-proving performance over base models and RL baselines. Notably, these gains are obtained with only a few hundred synthesized boundary-hard samples, suggesting that proof-structured hard tasks can be more data-efficient than simply scaling generic or already-solved training data.

Our contributions are summarized as follows:

1) We introduce **ProofFrontier**, a proof-structure-guided framework that transforms verified Lean theorem–proof pairs into complete hard theorem-proving tasks. Each generated task inherits a Lean-verifiable solution while increasing difficulty through controlled proof-structure mask.

2) We use ProofFrontier data for downstream verifier-grounded RL, and design a proof-similarity reward that leverages the availability of the original

verified proof to provide additional structure-aware training signal.

3) We demonstrate strong empirical performance across miniF2F, ProofNet, and PutnamBench, with particularly notable data efficiency: only a few hundred synthesized samples are sufficient to yield consistent improvements.

2 Related Work

2.1 Neural Theorem Proving in Lean

Neural theorem proving in Lean has advanced rapidly with open proof environments, retrieval-augmented proving, and increasingly strong LLM-based provers. Formal Mathematics Statement Curriculum Learning (Polu et al., 2022) introduced expert iteration for formal mathematics by interleaving proof search, verification, and training on discovered proofs. LeanDojo and ReProver (Yang et al., 2023) provide an open Lean interaction framework and show that retrieval over formal libraries improves proof generation. Lean Copilot (Song et al., 2024) integrates LLM inference into the Lean workflow for tactic suggestion, proof search, and premise selection. Draft, Sketch, and Prove (Jiang et al., 2023) uses informal proof sketches to guide formal proof search, illustrating the value of structured proof guidance.

Recent prover systems further scale model training, search, and verifier feedback. DeepSeek-Prover-V1.5 (Xin et al., 2024) combines supervised fine-tuning, proof-assistant feedback, RL, and Monte-Carlo tree search. DeepSeek-Prover-V2 (Ren et al., 2025) improves formal reasoning through synthetic cold-start data and RL for subgoal decomposition. Kimina-Prover (Wang et al., 2025a) studies large-scale RL for formal reasoning and shows strong scaling with sampling budget. Goedel-Prover-V2 (Lin et al., 2025) combines scaffolded data synthesis, self-correction, and model averaging for open-source Lean proving. Leanabell-Prover (Ji et al., 2025b) studies post-training scaling, while Leanabell-Prover-V2 (Ji et al., 2025a) incorporates verifier-integrated long reasoning with multi-turn Lean feedback. HunyuanProver (Li et al., 2024) uses iterative tactic-level data generation and guided tree search, and EvolProver (Tian et al., 2025) improves prover performance by evolving formal problem variants.

Instead of proposing a new prover architecture or search procedure, ProofFrontier focuses on the training task distribution: it starts from verified theorem-proof pairs, preserves the original theorem statement, and uses proof-structure masking with capability-aware filtering to construct complete boundary-hard tasks for RL.

2.2 Reinforcement Learning, Expert Iteration, and Synthetic Data

Reinforcement learning and expert iteration have become central tools for improving formal provers. Polu et al. (Polu et al., 2022) established expert iteration as a verifier-grounded loop of proof generation, checking, and retraining. DeepSeek-Prover-V1.5 (Xin et al., 2024) uses proof-assistant feedback as the reward signal, and DeepSeek-Prover-V2 (Ren et al., 2025) further uses RL to encourage subgoal decomposition. Lean-STaR (Lin et al., 2024) generates synthetic informal thoughts before proof steps and then applies expert iteration on Lean-verified proofs. LeanAgent (Kumarappan et al., 2024) studies lifelong learning for theorem proving across evolving mathematical domains, and LeanProgress (Huang et al., 2025) uses proof-progress prediction to guide search.

A related direction is to synthesize new training tasks or intermediate proof structures. MUSTARD (Huang et al., 2024) synthesizes theorem-and-proof data and filters it with a proof assistant. QDTSynth (Wang et al., 2025b) performs quality-driven synthesis of formal Lean theorems. ProD (Dong et al., 2024) rewards models for producing useful hierarchical decompositions and intermediate lemmas. STP (Dong and Ma, 2025) trains a self-play theorem prover in which a conjecturer generates barely provable conjectures for a prover. Seed-Prover and Seed-Prover 1.5 (Chen et al., 2025b,a) demonstrate the potential of large-scale agentic RL with Lean feedback. GAR (Wang et al., 2025d) jointly trains a problem composer and prover in an adversarial loop, creating an implicit curriculum over generated formal statements.

These works show that verifier feedback, generated tasks, and curriculum construction are crucial for theorem proving RL. ProofFrontier differs in that it does not synthesize new theorem statements or optimize a conjecture generator; instead, it derives complete hard tasks from existing Lean-certified proofs by controlling how much proof-structural guidance is exposed. This makes the resulting tasks verifiable by construction while allowing their difficulty to be calibrated against the current prover's capability boundary.

3 Method

3.1 Problem Setting

We study reinforcement learning for formal theorem proving in Lean. Let

$$\mathcal{D} = \{(x_i, p_i)\}_{i=1}^{N}$$

be a collection of verified theorem–proof pairs, where x_i is a formal theorem statement and p_i is a Lean-verified proof. Our goal is to turn these verified proofs into training tasks that are harder for a current prover, while still being complete theorem proving problems verifiable by Lean.

Specifically, for each theorem–proof pair (x, p), we construct a family of derived tasks that all preserve the original statement x but expose different amounts of proof-structural guidance. Thus, the prover is always asked to prove the same well-posed theorem, while the effective difficulty is controlled by how much information from the original proof is revealed.

Given a base prover π_{base}, we estimate the empirical solvability of a synthesized task $\tilde{x}$ by repeated sampled attempts $\hat{p}(\tilde{x})$, which is the pass rate over $\tilde{x}$ among all attempts. Success is determined by Lean verification. These pass rates are later used to select boundary-hard tasks for RL training.

3.2 Proof-Structure-Guided Masking

ProofFrontier constructs boundary-hard tasks by masking proof-structural information in verified Lean proofs. The masked proof is not used as an infilling target. Instead, masking serves as an internal mechanism for controlling how much proof guidance is exposed to the prover when it is asked to solve the original theorem. The whole pipeline is demonstrated in Figure 1.

Algorithm 1: ProofFrontier Pipeline

Input: Verified theorem–proof pairs $\mathcal{D}$; base prover π_{base}; target prover π_θ; mask levels $\mathcal{L}$; task budget B

Output: Trained prover π_θ

```
Initialize candidate pool P ← ∅;
foreach (x, p) ∈ D do
    Build source-level proof graph G(p) = (V, E);
    Score candidate nodes v ∈ V by s(v) = α s_bridge(v) + β s_usage(v) + γ s_change(v);
    Select structurally important nodes V_sel ⊆ V according to s(v);
    foreach v ∈ V_sel do
        foreach M_ℓ ∈ L do
            Extract visible metadata and convert to hint h_ℓ(v) = Φ_ℓ(z(v));
            Build statement-preserving task x̃_ℓ = concat(x, h_ℓ(v));
            Estimate pass rate p̂_ℓ using π_base and Lean verification;
            Add (x̃_ℓ, p̂_ℓ, s(v)) to P;
Select B rewardable boundary-hard tasks from P according to empirical pass rates;
Train π_θ on selected tasks with GRPO using Lean pass reward and auxiliary shaping;
return π_θ;
```

Source-level proof graph. Given a verified proof p, we first construct a syntax-level local proof graph $G(p) = (V, E)$ from the Lean source. This graph is a lightweight approximation of local proof flow rather than a complete semantic dependency graph. In particular, our main pipeline does not require querying Lean tactic states for every proof step.

Each node corresponds to a contiguous proof span and its local metadata. We consider three types of nodes. **Lemma-like nodes** include local assertions introduced by `have`, `suffices`, and `obtain`. **Action-like nodes** include proof actions such as `refine`, `constructor`, `cases`, `rcases`, `simp`, and `nlinarith`. **Context nodes** include local hypotheses, introduced symbols, and nearby supporting proof spans.

The edges combine three source-level relations: textual adjacency, local containment, and symbol use. Sequential edges connect adjacent proof nodes; containment edges connect a compound proof construct to its internal tactic block; and symbol-use edges connect a node that introduces a local symbol to later nodes that reference it. These edges are used to propose structurally meaningful mask targets, while the final difficulty of generated tasks is determined empirically by base-prover pass rates.

Node scoring. We score candidate nodes according to three heuristic signals. **Bridge score** favors nodes likely to mediate important proof transitions, such as named intermediate claims, `suffices` steps, and goal-restructuring tactics. **Usage score**

measures whether symbols introduced by a node are referenced later in the proof. **Change score** is a source-level proxy for proof-state change based on observable features such as node type, span length, newly introduced local claims, goal-restructuring tactics, and local symbol flow.

Specifically, we use a weighted sum to combine the components, where the components are normalized within each proof. These scores are not intended to recover exact semantic proof dependencies. They only rank candidate mask targets; the actual difficulty of the resulting tasks is later calibrated by Lean-verified base-prover performance.

Node-centered mask families. For each selected node, ProofFrontier constructs a family of mask levels. Each level corresponds to a visibility profile over the node metadata, including the local lemma name, lemma statement or header, entry hint, exit hint, local symbols, nearby support context, and proof body. Easier levels hide only the proof body while preserving richer structural guidance. Harder levels progressively remove the lemma name, statement, entry and exit hints, local symbols, and support context. The strongest levels may additionally suppress adjacent support spans. Our settings are illustrated in Appendix B.

The mask levels are ordered from weak to strong masking, but their indices should be interpreted only as level identifiers rather than calibrated numerical difficulties. We use a relatively dense set of levels to search for useful difficulty variations around the prover's boundary. The true difficulty of each level is not assumed in advance; it is estimated by running the base prover and checking the generated proofs with Lean.

3.3 From Mask Families to Boundary Tasks

Each mask level is converted into a statement-preserving theorem proving task. Given a selected node v, the mask level determines which metadata remain visible. These visible fields are then converted into a short natural-language or Lean-style hint $h_\ell(v)$. In our implementation, we construct the hint as is a deterministic template-based procedure. It renders the visible metadata into a concise hint comment, preserving only the fields allowed by the current visibility profile.

The final task $\tilde{x}_\ell$ is built by combining the original theorem statement, a placeholder proof body such as `sorry`, the synthesized hint, and the official prover prompt template. The theorem statement remains unchanged. The prover is still required to solve the full theorem, and masking affects only the amount of proof-structural guidance exposed through the hint.

For each synthesized task, we estimate its empirical pass rate with the base prover. We then keep tasks on the hard side of the solvability boundary: the hint should be weak enough that the task is no longer trivial, but not so weak that verifier feedback becomes uninformative. Aggregating selected tasks over the corpus yields the boundary-hard training set $\widetilde{\mathcal{D}}_{\mathrm{hard}}$.

This calibration step is central to ProofFrontier. The proof graph and mask levels only generate candidate difficulty axes. The final training examples are selected according to empirical prover behavior under Lean verification. In this way, lightweight source-level heuristics are separated from verifier-grounded difficulty calibration.

3.4 Boundary GRPO Training

We train the target prover on $\widetilde{\mathcal{D}}_{\mathrm{hard}}$ with GRPO. For each synthesized task, the model generates a proof plan followed by a Lean proof. The generated proof is checked by Lean, and the resulting verification signal is used as the primary reward. The algorithmic training procedure is summarized in Algorithm 1.

The scalar reward combines Lean verification with lightweight shaping:

$$r(\tilde{x}, y) = \lambda_{\mathrm{p}} r_{\mathrm{p}}(\tilde{x}, y) + \lambda_{\mathrm{sim}} r_{\mathrm{sim}}(\tilde{x}, y) + \lambda_{\mathrm{h}} r_{\mathrm{h}}(\tilde{x}, y).$$

Here $r_{\mathrm{p}} \in \{0, 1\}$ is the Lean verifier pass signal. The similarity reward r_{sim} is a normalized token-overlap score between the generated proof and the reference proof trajectory. The completion-health reward r_{h} checks lightweight format constraints, such as whether the output contains a complete Lean proof block, avoids empty or truncated completions, and follows the required prover template. Both auxiliary rewards are normalized to $[0, 1]$.

The verifier pass reward is the dominant signal, while the similarity and health terms serve as auxiliary shaping signals to reduce reward sparsity and discourage malformed outputs. Training on $\widetilde{\mathcal{D}}_{\mathrm{hard}}$ concentrates rollouts near the prover's capability boundary, where tasks are challenging but still rewardable. This makes the training distribution more informative than generic theorem proving data, rather than relying on scale alone, ProofFrontier improves the density of useful RL

Table 1: Main results on miniF2F, ProofNet, and PutnamBench. We compare each base prover against the same backbone further trained on ProofFrontier boundary-hard tasks. Some baseline results are taken from GAR (Wang et al., 2025d) or its official release. All results are evaluated by Lean verification.

Method	**Sample budget**	**miniF2F**	**ProofNet**	**PutnamBench**
Lean-STaR (Lin et al., 2024)	$64 \times 1 \times 50$	46.31	-	-
InternLM-2.5-StepProver (Wu et al., 2025)	$4 \times 32 \times 600$	50.70	18.80	-
DeepSeek-Prover-V1.5-RL (Xin et al., 2024)	128	50.00	18.20	-
STP-Lean (Dong and Ma, 2025)	128	56.15	19.50	-
MA-LoT (Wang et al., 2025c)	32	61.07	15.47	-
Kimina-Prover-Distill-7B (Wang et al., 2025a)	32	63.11	-	-
DeepSeek-Prover-V2-7B (Ren et al., 2025)	32	70.49	22.58	5.43
Goedel-Prover-V2-8B (Lin et al., 2025)	32	77.87	26.08	8.85
DeepSeek-Prover-V2-7B w/ GAR (Wang et al., 2025d)	32	74.18	25.81	7.45
Goedel-Prover-V2-8B w/ GAR	32	80.33	29.30	10.87
Our models				
DeepSeek-Prover-V2-7B **w/ ProofFrontier**	32	**75.41**	**26.34**	**10.09**
Goedel-Prover-V2-8B **w/ ProofFrontier**	32	**81.15**	**31.45**	**14.29**

signals by aligning task difficulty with the current prover's competence.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate ProofFrontier on three Lean theorem proving benchmarks: miniF2F, ProofNet, and PutnamBench. These benchmarks cover different theorem styles and difficulty levels, allowing us to test whether boundary training improves prover performance beyond a single dataset or theorem domain. Unless otherwise specified, all reported scores are computed by Lean verification of generated proofs. Note that at evaluation time, no ProofFrontier hints, reference proofs, masked metadata, or source-proof-derived comments are provided. The model is evaluated on the original benchmark theorem statement using the standard prover prompt. Moreover, the placeholder `sorry` appears only in the input template and is not accepted in model outputs.

Prover backbones. We study two strong Lean prover backbones: DeepSeek-Prover-V2-7B and Goedel-Prover-V2-8B. For both models, we use the official prover-style prompt format and preserve the standard generation protocol, where the model first produces a proof plan and then outputs the final Lean proof. This allows us to evaluate whether the gains from ProofFrontier transfer across prover families rather than depending on a single model implementation.

Training data construction. Our training tasks are constructed from the open-source dataset `Goedel-LM/SFT_dataset_v2`, which contains verified Lean theorem–proof pairs. Starting from these proofs, ProofFrontier builds boundary-hard theorem proving tasks by masking proof-structural information and converting the visible metadata into hints. Throughout the synthesis pipeline, the theorem statement is unchanged, and the final task remains a full theorem proving problem.

For each synthesized task family, we estimate the empirical solvability of different masking levels using a base prover and Lean verification. We then select boundary-hard levels near the estimated capability boundary with a pass rate less than 5 and more than 3 over 16 budgets. In these cases, tasks that are no longer trivial for the base prover, but still yield informative verifier feedback. In our main experiments, we train on 500 selected boundary-hard tasks unless otherwise specified.

Training and rewards. We train the target prover with GRPO on the synthesized boundary-hard task set. For each input task, the model generates a proof plan followed by a Lean proof, which is then checked by Lean. The reward is defined as

$$r(\tilde{x}, y) = r_{\mathrm{p}}(\tilde{x}, y) + 0.1 r_{\mathrm{sim}}(\tilde{x}, y) + 0.1 r_{\mathrm{h}}(\tilde{x}, y),$$

where r_{p} is the Lean verifier pass signal, r_{sim} is a token-level similarity reward with respect to the target proof trajectory, and r_{h} measures completion quality. The verifier pass reward is the dominant signal, while the similarity and health terms are used as auxiliary shaping rewards.

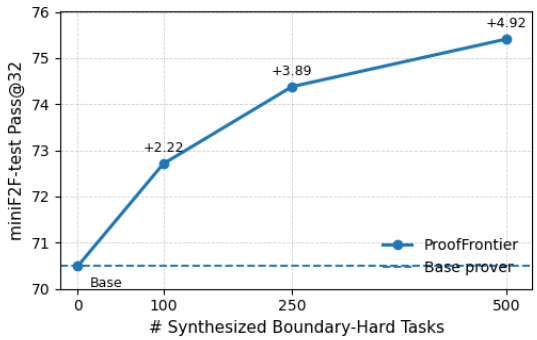

Figure 2: Data efficiency of ProofFrontier on miniF2F. We vary the number of synthesized boundary-hard tasks used for GRPO training. Even 100 tasks improve the base prover by 2.22 points, while 500 tasks yield a 4.92-point improvement. This suggests that boundary-hard task construction improves the information density of theorem proving RL data.

Evaluation protocol and baselines. For each benchmark and prover backbone, we compare the original base prover against the same backbone further trained with ProofFrontier. We also compare with previously reported strong theorem proving systems. In particular, we include GAR (Wang et al., 2025d) as a recent RL-based baseline. GAR jointly trains a problem composer and a prover in an adversarial loop, dynamically adapting theorem difficulty to the evolving capability of the prover. This makes it closely related to our goal of capability-boundary training, but differs in task construction: GAR generates new formal statements through statement fusion, whereas ProofFrontier preserves the original theorem statement and controls only the amount of proof-structural guidance exposed to the prover, thus preserving the well-posedness of the generated tasks.

Published baseline results are taken from the GAR experimental summary. Rows marked "w/ GAR" for miniF2F and ProofNet are taken from GAR or its official release. Our ProofFrontier models are evaluated with a sample budget of 32 generated proof attempts per theorem. We further report ablations on task construction, reward design, and training data size to isolate the contribution of boundary-hard data synthesis.

4.2 Main Results

Table 1 reports the main results on miniF2F, ProofNet and PutnamBench. We compare ProofFrontier against the original prover backbones, previously reported theorem proving systems, and GAR-trained variants under the same sample budget when available. Overall, ProofFrontier consistently improves both prover backbones across all evaluated benchmarks, and the benefit is not tied to a single prover backbone.

Table 2: Ablation on masking and boundary-hard task construction. We compare ProofFrontier against other heuristic task-synthesis strategies. All methods use the same RL framework and prover backbone. Only the task construction pipeline changes.

Task Construction	miniF2F	ProofNet
Raw theorem proving data	71.72	22.04
Coarse masking	71.72	-
Random masking	72.95	24.46
ProofFrontier	**75.41**	**26.34**

Notably, these gains are obtained using only 500 synthesized boundary-hard tasks. This suggests that ProofFrontier improves not only final theorem proving performance, but also the data efficiency of verifier-grounded RL. We study this effect more directly in the next section.

4.3 Data Efficiency

A central motivation of ProofFrontier is to improve the data efficiency of theorem proving RL. To evaluate this, we vary the number of synthesized boundary-hard tasks used for GRPO training and evaluate the resulting prover on miniF2F. We use the same prover backbone and training setup as in the main experiments, changing only the number of ProofFrontier-generated training tasks.

Figure 2 shows that performance improves steadily as more boundary-hard tasks are used. Starting from the base prover score of 70.49, training with only 100 synthesized tasks improves miniF2F performance to 72.71. Increasing the training set to 250 tasks further improves performance to 74.38, and using 500 tasks reaches 75.41, yielding a 4.92-point gain over the base prover. Note that GAR is based on a repository of 793,243 natural-language statements and dynamically generates fused formal statements during training.

These results indicate that ProofFrontier provides high-density RL supervision. Rather than relying on large-scale synthetic data, a small number of carefully constructed boundary-hard tasks is already sufficient to produce measurable improvements. This supports our hypothesis that aligning training examples with the prover's capability boundary can substantially improve the data efficiency of verifier-grounded theorem proving RL.

4.4 Ablation Study

Impact of data construction strategy. We first ablate the task-construction pipeline to understand whether our boundary masking are responsible for the gains of ProofFrontier. We compare the full method with three heuristic masking strategies: (i) Raw theorem proving data without masking, (ii) coarse masking (masking a whole node at a time without structural details), (iii) random masking (masking without node score guidance). All variants use the same prover backbone and RL setup. Only the construction of training tasks differs.

Table 2 shows that ProofFrontier outperforms all other heuristic construction strategies. Training directly on raw theorem proving data performs worst. Other masking strategies improve over raw data, but remain substantially below ProofFrontier. This indicates that the gains do not come from arbitrary guidance. Instead, effective training requires a detailed proof-structure-guided masking strategy.

Impact of reward components. We next ablate the reward design used in boundary-hard GRPO training. The full reward combines the Lean verifier pass reward r_{p}, token-level similarity reward r_{sim}, and completion-health reward r_{h}. This ablation separates the contribution of executable verifier feedback from the auxiliary shaping terms.

As shown in Table 3, the Lean verifier pass reward is the primary supervision signal. Using r_{p} alone already yields strong performance, reaching 73.36 on miniF2F-test with a sample budget of 32. Adding token-level similarity further improves the score to 74.59, indicating that similarity provides useful denser feedback when binary verifier rewards are sparse. Completion-health reward alone provides a smaller gain, but helps discourage malformed or degenerate outputs. The full reward achieves the best performance under both budget 1 and budget 32. These results support that executable Lean verification remains the core reward, while similarity and health terms serve as lightweight shaping signals.

Table 3: Ablation on reward design. We compare different reward combinations used in boundary-hard GRPO training. r_{p} denotes the Lean verifier pass reward, r_{sim} denotes token-level similarity reward, and r_{h} denotes completion-health reward. Results are reported on miniF2F.

Reward Design	budget	miniF2F-test
r_{p} only	1	55.33
r_{p} only	32	73.36
$r_{\mathrm{p}} + r_{\mathrm{sim}}$	1	56.97
$r_{\mathrm{p}} + r_{\mathrm{sim}}$	32	74.59
$r_{\mathrm{p}} + r_{\mathrm{h}}$	1	55.74
$r_{\mathrm{p}} + r_{\mathrm{h}}$	32	73.36
$r_{\mathrm{p}} + r_{\mathrm{sim}} + r_{\mathrm{h}}$	1	**58.20**
$r_{\mathrm{p}} + r_{\mathrm{sim}} + r_{\mathrm{h}}$	32	**75.41**

4.5 Discussions

Our results suggest that the effectiveness of theorem proving RL depends strongly on the quality of the training task distribution. Instead of treating verified proofs only as supervised demonstrations, ProofFrontier uses them as structured templates for constructing complete hard tasks. This perspective is useful because a verified Lean proof already contains both a formally valid solution and rich proof-internal structure, such as intermediate lemmas, bridge steps, local symbols, and supporting contexts. By selectively controlling how much of this structure is exposed, we can create tasks that remain formally well-posed while becoming substantially harder for the current prover.

The data-efficiency results further indicate that boundary alignment can be more important than simple data scale. Training on small amount of boundary-hard tasks yields consistent gains across benchmarks. This suggests that verifier-grounded RL benefits most from tasks that are neither already solved nor completely unrewardable. In this sense, ProofFrontier improves the information density of RL training data by concentrating rollouts near the prover's capability boundary.

5 Conclusion

We introduced ProofFrontier, a proof-structure-guided task synthesis method for data-efficient reinforcement learning in formal theorem proving. Starting from verified Lean proofs, ProofFrontier constructs complete boundary-hard tasks by masking proof-structural guidance. Base-prover pass rates are then used to select tasks that are challenging but still rewardable under Lean verification.

Experiments show that training on ProofFrontier-generated tasks consistently improves strong prover backbones. Notably, these gains are achieved with only small amount of synthesized tasks, demonstrating that carefully constructed boundary-hard data can provide high-density RL supervision.

Limitations

ProofFrontier relies on verified Lean theorem–proof pairs. This guarantees that synthesized tasks remain solvable and formally verifiable, but also limits the method to domains where sufficiently many certified proofs are available. Its proof-structure masking mechanism is based on source-level heuristics, so structurally important proof nodes may not always correspond to the reasoning steps that are actually difficult for a given prover. The capability-aware filtering stage is also model-dependent: different base provers, decoding budgets, or verifier settings may require recalibrating the hard-side task selection criteria.

Another limitation is the use of auxiliary proof-similarity reward. Although verifier correctness remains the primary reward, similarity-based shaping may bias the model toward the reference proof trajectory and under-reward alternative valid proofs. Finally, our experiments focus on Lean and standard theorem proving benchmarks. Extending ProofFrontier to larger Lean libraries, other proof assistants, and broader formal-reasoning workflows remains an important direction for future work.

Ethical Considerations

This work aims to improve the reliability and data efficiency of formal theorem proving systems. Since generated proofs are checked by Lean, the method provides an explicit correctness filter and reduces the risk of accepting unsupported mathematical claims. The work does not use private or personal data, and all synthesized tasks are derived from existing verified theorem–proof pairs.

The main risks are indirect. More capable theorem provers may benefit formal mathematics, software verification, and hardware verification, but they may also be applied in high-stakes or dual-use technical domains. We therefore recommend human oversight when deploying theorem-proving agents in safety-critical settings. In addition, ProofFrontier inherits the domain coverage of its source Lean corpora, and auxiliary similarity rewards may bias models toward reference proof styles rather than diverse valid proofs. Future work should further examine robustness, proof diversity, and safe deployment of verifier-guided theorem proving systems.

References

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*.

Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, and 1 others. 2025a. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *arXiv preprint arXiv:2512.17260*.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, and 1 others. 2025b. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*.

Kefan Dong and Tengyu Ma. 2025. Stp: Self-play llm theorem provers with iterative conjecturing and proving. In *Proceedings of the 42nd International Conference on Machine Learning*.

Kefan Dong, Arvind Mahankali, and Tengyu Ma. 2024. Formal theorem proving by rewarding llms to decompose proofs hierarchically. *arXiv preprint arXiv:2411.01829*.

Suozhi Huang, Peiyang Song, Robert Joseph George, and Anima Anandkumar. 2025. LeanProgress: Guiding search for neural theorem proving via proof progress prediction. *arXiv preprint arXiv:2502.17925*.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. 2024. MUSTARD: Mastering uniform synthesis of theorem and proof data. *arXiv preprint arXiv:2402.08957*.

Xingguang Ji, Yahui Liu, Qi Wang, Jingyuan Zhang, Yang Yue, Rui Shi, Chenxi Sun, Fuzheng Zhang, Guorui Zhou, and Kun Gai. 2025a. Leanabell-prover-v2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning. *arXiv preprint arXiv:2507.08649*.

Xingguang Ji and 1 others. 2025b. Leanabell-prover: Posttraining scaling in formal reasoning. *arXiv preprint arXiv:2504.06122*.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*.

Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. 2024. LeanAgent: Lifelong learning for formal theorem proving. *arXiv preprint arXiv:2410.06209*.

Y. Li and 1 others. 2024. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. 2024. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, and 1 others. 2025. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2022. Formal mathematics statement curriculum learning. In *Proceedings of the 39th International Conference on Machine Learning*.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, and 1 others. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Lean copilot: Large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*.

Yuchen Tian, Ruiyuan Huang, Xuanwu Wang, Jing Ma, Zengfeng Huang, Ziyang Luo, Hongzhan Lin, Da Zheng, and Lun Du. 2025. Evolprover: Advancing automated theorem proving by evolving formalized problems via symmetry and difficulty. *arXiv preprint arXiv:2510.00732*.

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

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, and 1 others. 2025a. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*.

Lei Wang, Ruobing Zuo, Gaolei He, Jianlin Wang, and Zhengfeng Yang. 2025b. QDTSynth: Quality-driven formal theorem synthesis for enhancing proving performance of LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14683–14698. Association for Computational Linguistics.

Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. 2025c. Ma-lot: Model-collaboration lean-based long chain-of-thought reasoning enhances formal theorem proving. *Preprint*, arXiv:2503.03205.

Ruida Wang, Jiarui Yao, Rui Pan, Shizhe Diao, and Tong Zhang. 2025d. Gar: Generative adversarial reinforcement learning for formal theorem proving. *arXiv preprint arXiv:2510.11769*.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Zheng Yuan, Wenwei Zhang, Dahua Lin, and Kai Chen. 2025. Internlm2.5-stepprover: Advancing automated theorem proving via critic-guided search. *Preprint*, arXiv:2410.15700.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2024. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. Minif2f: A cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*.

A GRPO Objective

Given the synthesized boundary-hard set $\widetilde{\mathcal{D}}_{\text{hard}}$, we train a target prover π_θ with GRPO. Each task $\tilde{x}$ is a full theorem proving problem with the original statement preserved and proof-structural hints calibrated to the hard side of the base prover's capability boundary. Note that the base prover is a frozen copy of the corresponding pretrained prover before ProofFrontier training. For each backbone, boundary-hard tasks are calibrated with that backbone as π_{base}, and the target prover π_θ is initialized from the same backbone for RL training.

For each task $\tilde{x}$, GRPO samples a group of K completions from the old policy:

$$\{y_i\}_{i=1}^{K} \sim \pi_{\theta_{\text{old}}}(\cdot \mid \tilde{x}), \tag{1}$$

where each completion contains a proof plan followed by a Lean proof. Each completion is checked by Lean and assigned a scalar reward

$$r_i = r(\tilde{x}, y_i). \tag{2}$$

GRPO estimates the advantage by normalizing rewards within the sampled group:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^{K})}{\text{std}(\{r_j\}_{j=1}^{K}) + \epsilon}. \tag{3}$$

This group-relative advantage avoids learning an explicit value function and makes the update depend on relative proof quality among completions sampled for the same task.

For token t in completion y_i, define the policy ratio

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid \tilde{x}, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid \tilde{x}, y_{i,<t})}. \tag{4}$$

We optimize the standard clipped GRPO objective.

$$\mathcal{L}(\theta) = -\frac{1}{K}\sum_{i=1}^{K}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_i\Big). \tag{5}$$

B Masking Details

Table 4 summarizes the visibility profiles used by ProofFrontier. Each masking level controls which proof-structural metadata are exposed to the prover through the synthesized hint. The levels are ordered from weak masking to strong masking, but they should be interpreted as discrete visibility profiles rather than calibrated numerical difficulty scores. In all levels, the original proof body of the selected node is never exposed. The actual difficulty of each synthesized task is determined empirically by base-prover pass rates under Lean verification.

Here, *lemma name* refers to the local identifier introduced by a proof node, such as `h` in `have h : P := ...`. *Lemma statement* denotes the local claim or tactic header associated with the node. *Entry hint* describes why the node may be useful, while *exit hint* describes how the resulting fact can help close later proof goals. *Local symbols* are previously introduced hypotheses or local facts referenced near the selected node.

Weak masking levels such as M_0 and M_1 provide rich structural guidance and are often useful for constructing easier variants within a mask family. Stronger levels such as M_3 and M_4 remove most node-specific information and produce harder variants. Since the same visibility profile may be easy for one theorem but difficult for another, ProofFrontier does not rely on the mask level alone to determine difficulty. Instead, all generated tasks are calibrated with a base prover, and only rewardable hard-side tasks are retained for RL training.

When converting unmasked information into hints, we adopt a template series. An example of lemma statement template is

```
Hint:
Introduce an intermediate claim:
`{lemma_statement}`.
This claim can be proved using {
 local_symbols}.
Afterwards, use it to {exit_hint}.
```

C Case study

In this section, we provide a example demonstrating the reasoning capabilities of our RL compared to the base prover on a challenging problem from the miniF2F benchmark. Evaluation on base prover yields a timeout without reasonable outputs.

Problem statement

```
/--
A sequence (a_1, b_1), (a_2, b_2), (a_3, b_3), ... of
    points in the coordinate plane
    satisfies
(a_{n+1}, b_{n+1}) = (√3 a_n − b_n, √3 b_n + a_n),   n = 1, 2, 3, ....

Suppose that (a_100, b_100) = (2, 4). What is
    a_1 + b_1?
-/
```

Table 4: Visibility profiles of ProofFrontier masking levels. The levels are ordered from weak to strong masking. They are not calibrated numerical scores. Actual task difficulty is determined by base-prover pass rates. The proof body of the selected node is never exposed.

Mask level	Lemma name	Lemma statement	Entry hint	Exit hint	Local symbols
M_0	visible	visible	visible	visible	full
M_1	visible	visible	visible	visible	partial
M_2	hidden	visible	visible	hidden	partial
M_3	hidden	hidden	visible	hidden	partial
M_4	hidden	hidden	hidden	hidden	hidden

```
theorem amc12a_2008_p25 (a b : ℕ → ℝ) (h₀
      : ∀ n, a (n + 1) = Real.sqrt 3 *
    a n - b n)
    (h₁ : ∀ n, b (n + 1) = Real.sqrt 3 * b
      n + a n) (h₂ : a 100 = 2) (h₃ :
      b 100 = 4) :
    a 1 + b 1 = 1 / 2 ^ 98 := by
  sorry
```

Proof plan. To find $a_1 + b_1$ given a_{100} and b_{100}, evaluating the sequence step-by-step for 99 iterations is computationally intractable for the Lean simplifier. We must find a structural invariant. By applying the recurrence relation, we can find a 3-step periodic relation:

$$a_{n+3} = -8b_n, \qquad b_{n+3} = 8a_n.$$

Applying it twice gives a 6-step relation:

$$a_{n+6} = -64a_n, \qquad b_{n+6} = -64b_n.$$

Since

$$100 = 1 + 6 \times 16 + 3,$$

we can bridge the sequence using induction over the 6-step relation 16 times, followed by one 3-step relation, to efficiently solve the linear system for a_1 and b_1.

Lean proof.

```
import Mathlib

theorem amc12a_2008_p25 (a b : ℕ → ℝ) (h₀
      : ∀ n, a (n + 1) = Real.sqrt 3 *
    a n - b n)
    (h₁ : ∀ n, b (n + 1) = Real.sqrt 3 * b
      n + a n) (h₂ : a 100 = 2) (h₃ :
      b 100 = 4) :
    a 1 + b 1 = 1 / 2 ^ 98 := by
  have h_step3_a : ∀ n, a (n + 3) = -8 * b
      n := by
    intro n
    rw [h₀ (n + 2), h₁ (n + 1), h₀ (n + 1),
      h₁ n, h₀ n]
    ring
  have h_step3_b : ∀ n, b (n + 3) = 8 * a
      n := by
    intro n
    rw [h₁ (n + 2), h₁ (n + 1), h₀ (n + 1),
      h₁ n, h₀ n]
    ring
  have h_step6_a : ∀ n, a (n + 6) = -64 *
      a n := by
    intro n
    rw [h_step3_a (n + 3), h_step3_b n]
    ring
  have h_step6_b : ∀ n, b (n + 6) = -64 *
      b n := by
    intro n
    rw [h_step3_b (n + 3), h_step3_a n]
    ring
  have h_pow : ∀ k : ℕ, a (1 + 6 * k) =
      (-64) ^ k * a 1 ∧ b (1 + 6 * k) =
        (-64) ^ k * b 1 := by
    intro k
    induction' k with k ih
    · simp
    · rcases ih with ⟨iha, ihb⟩
      have eq_a : a (1 + 6 * (k + 1)) = a
      (1 + 6 * k + 6) := by ring_nf
      have eq_b : b (1 + 6 * (k + 1)) = b
      (1 + 6 * k + 6) := by ring_nf
      rw [eq_a, eq_b, h_step6_a, h_step6_b,
      iha, ihb]
      constructor <;> ring
  have h_100 : 100 = 1 + 6 * 16 + 3 := by
      norm_num
  rw [h_100] at h₂ h₃
  have eq_a2 : a (1 + 6 * 16 + 3) = -8 * b
      (1 + 6 * 16) := h_step3_a _
  have eq_b2 : b (1 + 6 * 16 + 3) = 8 * a
      (1 + 6 * 16) := h_step3_b _
  rw [eq_a2, eq_b2] at h₂ h₃
  have hk := h_pow 16
  rcases hk with ⟨hka, hkb⟩
  rw [hka, hkb] at h₂ h₃
  linarith
```

D Statements

Artifact license We use publicly available Lean theorem proving datasets and benchmarks under their respective licenses. The synthesized ProofFrontier tasks and code will be released for research use under an open-source license upon publication. Users should comply with the licenses of the origi-

nal source datasets and Lean libraries from which the derived tasks are constructed.

Artifact use and intended use. We use existing Lean theorem proving datasets, benchmarks, and prover checkpoints for research on formal theorem proving. Our use of these artifacts is consistent with their intended research and evaluation purposes. The synthesized ProofFrontier tasks are derived from verified Lean theorem–proof pairs and are intended for research use in theorem-proving data synthesis, prover training, and reproducibility studies. Any released derived data or code should be used in accordance with the licenses and access conditions of the original datasets, benchmarks, Lean libraries, and model checkpoints.

Personally identifying info and offensive content. The data used in this work consist of formal Lean theorem statements, Lean proofs, benchmark problems, and synthesized theorem-proving tasks derived from verified formal proofs. These artifacts are mathematical and programmatic in nature, and are not intended to contain personally identifying information or offensive content. We did not collect data from human subjects or private user sources. We also manually inspected samples from the synthesized tasks and found no personally identifying information or offensive content. If future releases include generated artifacts, we will preserve this data type restriction and release only formal theorem-proving data derived from public research datasets and Lean libraries.

Information about AI assistants. AI assistants are used for manuscript polishing.

E Implementation details

We implement GRPO training using TRL with PyTorch, HuggingFace Transformers, Accelerate, and vLLM for rollout acceleration. Unless otherwise specified, we use the official tokenizer and prompt format of each prover backbone. We set `max_prompt_length=8192`. Generated Lean proofs are evaluated using the Lean verifier under the same environment across all experiments. For reward computation, r_{p} is the binary Lean verification signal, while r_{sim} and r_{h} are normalized to $[0, 1]$. All reported pass rates are computed with the specified sample budget in each table. We use Nvidia-A100 for all the experiments. For PutnumBench, we adopt a 660 sample version, aligned with our reference paper (Tsoukalas et al., 2024).