PROCESS-VERIFIED REINFORCEMENT LEARNING FOR THEOREM PROVING VIA LEAN

Anonymous authors
Paper under double-blind review

ABSTRACT

While reinforcement learning from verifiable rewards (RLVR) typically has relied on a single binary verification signal, symbolic proof assistants in formal reasoning offer rich, fine-grained structured feedback. This gap between structured processes and unstructured rewards highlights the importance of feedback that is both dense and sound. In this work, we demonstrate that the Lean proof assistant itself can serve as a symbolic process oracle, supplying both outcome-level and fine-grained tactic-level verified feedback during training. Proof attempts are parsed into tactic sequences, and Lean's elaboration marks both locally sound steps and the earliest failing step, yielding dense, verifier-grounded credit signals rooted in type theory. We incorporate these structured rewards into a GRPO-style reinforcement learning objective with first-error propagation and first-token credit methods that balances outcome- and process-level advantages. Experiments with STP-Lean and DeepSeek-Prover-V1.5 show that tactic-level supervision outperforms outcome-only baselines in most settings, delivering improvements on benchmarks such as MiniF2F and ProofNet. Beyond empirical gains, our study highlights a broader perspective: symbolic proof assistants are not only verifiers at evaluation time, but can also act as process-level reward oracles during training. This opens a path toward reinforcement learning frameworks that combine the scalability of language models with the reliability of symbolic verification for formal reasoning.

1 INTRODUCTION

Automated theorem proving (ATP) is one of the long-term goals of AI (Newell et al., 1957). Compared to reasoning in natural language (NL), which often contains vague or ambiguous symbols, formal theorem proving based on formal logic and type theory provides technical and precise language for proving mathematical theorem (Church, 1940; Fitting, 1996). Currently, interactive theorem provers (ITP) such as Lean (de Moura et al., 2015; Moura & Ullrich, 2021), Isabelle (Nipkow et al., 2002) and Coq (Barras et al., 1997), serve as reliable and powerful tools for verifying mathematical proofs. Lean proofs are sequences of tactics, with automation handling routine arithmetic/logic and verification-so ITPs provide a middle ground between full automation and human guidance

By contrast, LLMs model next-token probabilities from large corpora via pre- and post-training, learning lexical correlations rather than rule-based symbolic manipulation (Brown et al., 2020). With further techniques such as instruction tuning and Reinforcement Learning from Human Feedback (RLHF), LLMs have evolved to handle a wide range of tasks, including question answering, summarization, dialogue (Ouyang et al., 2022; Bai et al., 2022). In particular, reinforcement learning (RL) approaches for reasoning tasks aim to enhance the model's reasoning ability by encouraging the generation of long chains of thought rationale (DeepSeek-AI et al., 2025; OpenAI et al., 2024).

Compared to other reasoning tasks which often verify or reward LLMs' response according to its final answer (Cobbe et al., 2021), the theorem prover can verify the correctness of entire proof when LLMs respond with formal language. In this context, given the human-in-the-loop nature of ITPs, there have been growing attempts to use LLMs for formal theorem proving tasks (AlphaProof and AlphaGeometry teams, 2024; Trinh et al., 2024). LLMs act as prover agents while theorem provers serve as verifiers, being used either at inference time-to search and validate tactics and premises-or for augmenting formal reasoning datasets with verified samples (Lample et al., 2022; Wang et al.,

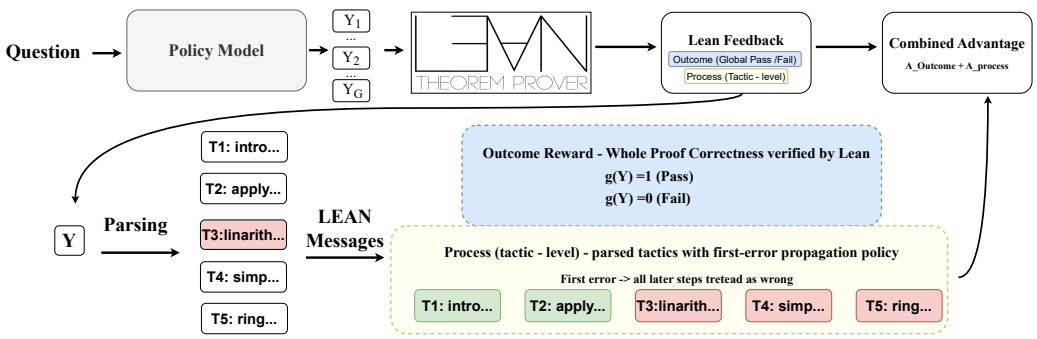

Figure 1: Overall framework for combining outcome and tactic level rewards via Lean: the proof Y is parsed into tactics $T_1, \ldots, T_5$, with Lean providing outcome feedback $g(Y)$ and step-level errors (e.g., failure at T_3 invalidates later tactics). Rewards are then assigned to the first token of each tactic.

2023; Ying et al., 2024a; Zhu et al., 2025). Furthermore, some recent studies incorporated binary feedback from the Lean theorem prover into its online RL framework (Xin et al., 2024b).

The tactic-based proof structure in Lean contains information relevant for reasoning tasks such as the positions of tactics or the nature of proof errors or failures. This structured information captures not just the outcome of a proof, but also the underlying reasoning process. However, despite its potential, only a few works have explored incorporating this kind of fine-grained supervision into the training of LLMs (Ji et al., 2025). At the same time, recent RL approaches for reasoning have increasingly emphasized the use of process-based reward models (PRMs) to guide model behavior. While these models show promising performance, there is still a lack of clarity around how PRMs are constructed, how the reasoning step or step reward should be defined, what training signals or datasets they should depend on (Yuan et al., 2024; Luo et al., 2024; Cui et al., 2025).

Unlike recent approaches that rely on PRMs or long NL CoT (Lin et al., 2025a;b), we directly leverage the Lean proof assistant as a *symbolic process oracle* during RL training, without any natural-language guidance. For each generated proof, Lean provides (i) a global outcome signal and (ii) fine-grained tactic-level feedback via info trees and error logs. We transform this symbolic supervision into structured process rewards and, by mapping tactics to tokens, integrate these heterogeneous signals into a Group Relative Policy Optimization (GRPO)-style objective combining outcome- and process-level advantages. This enables precise, type-theoretic credit assignment grounded in verifier feedback without the need for an auxiliary PRM. Empirically, we found that incorporating symbolic verifier feedback into the RL objective consistently improves performance on MiniF2F and ProofNet, demonstrating the value of fine-grained verifier signals for reliable credit assignment in reasoning tasks. Our key contributions are as follows:

- **Symbolic verifier as process oracle reward.** We formalized the use of the Lean proof assistant as a *symbolic verifier* of reasoning processes, parsing proofs into tactic-level reward signals.
- **Symbolic verifier-guided RL.** We integrate outcome- and tactic-level rewards derived from Lean into an RL framework, providing dense and verifiable credit assignment.
- **Stable improvements on benchmarks.** On MiniF2F and ProofNet, our approach consistently outperforms both outcome-only RL and vanilla baselines, yielding more stable and robust gains without NL notation or external PRM.

2 Related Work

Automatic Theorem Proving An automated theorem prover typically consists of two stages. The first is autoformalization, i.e., translating natural language mathematical statements into formal ones. LLMs have been used for this task (Wu et al., 2022), producing datasets such as MiniF2F, ProofNet, Deepseek-Prover, and LeanWorkbook (Zheng et al., 2022; Azerbayev et al., 2023; Xin et al., 2024a; Ying et al., 2024a). The second stage is proof generation, which can be performed step-by-step via tree search (Polu & Sutskever, 2020; Azerbayev et al., 2024; Wu et al., 2024; Xin et al., 2024b) or

by generating entire proofs at once (Xin et al., 2024a; Lin et al., 2025b). Existing approaches such as Lean-STaR and RMaxTS use Lean only as a step-checker during inference (Lin et al., 2025a; Xin et al., 2024b), whereas recent work has employed Lean as a whole-proof verifier during training (Wang et al., 2025a; Zhang et al., 2025; Ren et al., 2025). In this paper, we go further by leveraging Lean's parsing and elaboration to validate each tactic step and integrate step-level correctness as process-based rewards (Lightman et al., 2023).

Reinforcement Learning in Language Models Beyond algorithmic advances such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), reward shaping and credit assignment remain core challenges in RL. Outcome-based rewards (Cobbe et al., 2021), though widely used in RLHF, suffer from sparsity (Chan et al., 2024; Zheng et al., 2023). Process-based reward models (PRMs) address this by assigning step-level rewards (Lightman et al., 2023; Setlur et al., 2024; Kazemnejad et al., 2024; Yuan et al., 2024; Cui et al., 2025). Rewards can be defined implicitly (Cui et al., 2025) or explicitly via correctness annotations (Lightman et al., 2023) or Monte Carlo rollout success rates (Wang et al., 2024), but existing methods require large annotated datasets of step-level correctness. This motivates our approach of leveraging the Lean prover itself as a process oracle, automatically verifying each step without human labels or sampling. Additional discussion is in Appendix A.

3 PRELIMINARIES

3.1 LEAN4

In Lean theorem proving, a statement to be established is represented as an initial goal and incrementally reduced into subgoals through a sequence of tactics. Each tactic is parsed and elaborated by unifying it with lemmas or theorems in the library, generating new subgoals, and verifying their validity. The elaboration stage produces structured info trees that record proof states and error messages. Finally, the kernel ensures that the elaborated proof is type-theoretically consistent and constitutes a valid proof for the original theorem.

Formally, let x denote a theorem statement provided to an LLM, and let Y be the response, a proof expressed in the Lean language. Write $\mathcal{Y}$ for the set of Lean proofs and $\mathcal{T}$ for the set of tactics. For $Y \in \mathcal{Y}$, the Lean compiler parses Y into a set of tactics $\mathsf{TacSet}(Y) \subseteq \mathcal{T}$, where $\mathsf{TacSet}(Y) = \{\, T \mid T \text{ is a tactic parsed from } Y \,\}$. We then obtain a sequential representation by sorting $\mathsf{TacSet}(Y)$ in ascending order of each tactics starting position in Y:$(T_1, T_2, \ldots, T_{N(Y)})$ where $N(Y)$ is the number of tactic in Y, which aligns with the LLM's autoregressive generation process. Each tactic T_i comprises corresponding tokens y_t in Y. Lean represents tactics as Abstract Syntax Tree (AST) nodes; each node encodes the tactics syntactic structure and binding context, and may carry metadata such as error messages, proof states, and an index through which users (or training frameworks) can interact with Lean. If a tactic does not appear in the error log, then it has been elaborated successfully and passed Lean's internal rule-based verification, which guarantees that the step is locally sound under dependent type theory. Thus, any tactic not marked as an error constitutes a verified reasoning step-even if it does not contribute to closing the proof because some subgoals remain or later tactics fail. In other words, Lean ensures tactic-level soundness, while proof-level completeness depends on whether the entire sequence resolves all goals. Leveraging this parsing and validation feedback, we define the parsing function $f : \mathcal{Y} \to \mathcal{T}^*$ to be the sequence obtained by sorting $\mathsf{TacSet}(Y)$: $f(Y) = (T_1, \ldots, T_{N(Y)})$. We also define the global scoring function $g : \mathcal{Y} \to [0, 1]$, where $g(Y) = 1$ if Y passes the Lean verifier and 0 otherwise, and the per-tactic scoring function $\varphi : \{(Y, T) \mid Y \in \mathcal{Y},\, T \in \mathrm{TacSet}(Y)\} \longrightarrow \{1, d_1, d_2\}$. Specifically,

$$\varphi(Y,T) = \begin{cases} 1, & \text{if } g(Y) = 1, \\ d_1, & \text{else if } g(Y) = 0 \text{ and } T \text{ has no errors in Lean}, \\ d_2, & \text{else if } g(Y) = 0 \text{ and } T \text{ contains errors}. \end{cases}$$

Combining these components, we represent Lean's role via f, g, φ as

$$\begin{aligned} \mathrm{Lean}: \ & \mathcal{Y} \to \{0,1\} \times (\mathcal{T} \times \{1, d_1, d_2\})^*, \\ \mathrm{Lean}(Y) = \ & \big(g(Y), [(T_1, \varphi(Y, T_1)), \ldots, (T_{N(Y)}, \varphi(Y, T_{N(Y)}))]\big)_{f(Y)=(T_1,\ldots,T_{N(Y)})}. \end{aligned}$$

3.2 TACTIC-LEVEL MDP

We define a tactic-level Markov Decision Process (MDP) as the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, F, m)$. The state space $\mathcal{S}$ contains partial formal proofs; each $s \in \mathcal{S}$ is the proof prefix produced so far. The action space $\mathcal{A}$ coincides with the tactic space $\mathcal{T}$; each action $a \in \mathcal{A}$ is a single Lean tactic. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns a tactic-level reward $r(s, a)$. The transition function $F : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is deterministic: $s_{j+1} = F(s_j, a_j) = s_j \oplus a_j$, where $\oplus$ denotes concatenation of the tactic a_j to the proof s_j at time step j. Transitions are pure concatenations; Lean feedback affects r, not F. Let $\mathcal{S}_{\text{term}} \subseteq \mathcal{S}$ be EOS absorbing states. Let $m \in \mathcal{S}$ be the initial state. In Section 4, we extend this formulation with outcome- and tactic-level rewards derived from the Lean theorem prover to obtain the final training signal.

3.3 CREDIT ASSIGNMENT IN REINFORCEMENT LEARNING

PPO assigns a sparse end-of-sequence reward and propagates credit with a value model with Generalized Advantage Estimate (GAE), reducing variance at the cost of extra learning complexity; full details are deferred to Appendix E.

In contrast, REINFORCE style GRPO optimizes directly from verifiable whole-trajectory rewards without a value model. For a prompt q, we sample G responses $\{y_i\}_{i=1}^{G}$ from π_{old} and obtain rewards r_i. A normalized, response-level advantage is applied uniformly to all tokens of y_i:

$$\hat{A}_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}.$$

The objective is

$$L_{\text{GRPO}}(\theta) = \mathbb{E}\Big[\frac{1}{G}\sum_{i=1}^{G}\Big\{\min\Big(\frac{\pi_\theta(y_i \mid q)}{\pi_{\theta_{\text{old}}}(y_i \mid q)}\,\hat{A}_i,\ \text{clip}\Big(\frac{\pi_\theta(y_i \mid q)}{\pi_{\theta_{\text{old}}}(y_i \mid q)}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_i\Big) - \beta\, D_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}]\Big\}\Big].$$

We make this dense and sound by injecting *Lean-derived tactic advantages* into GRPO: the outcome signal remains at response level, while tactic-level signals are mapped to tokens at the first token of each tactic (Sec. 4). This preserves GRPO's simplicity while addressing sparse credit.

4 METHOD

4.1 DEFINE TACTIC-LEVEL REWARDS

In the previous section, we modeled the correctness of proofs Y generated by the Lean proof assistant and parsed and verified each tactic within Y. We now introduce a reward mechanism that integrates both outcome-based and process-based signals explicitly into the RL framework. Specifically, we employ an outcome-based reward defined through a function $g(Y)$, similar to approaches used by (DeepSeek-AI et al., 2025), as a global reward evaluating the entire proof. Additionally, we define a process-based reward $\varphi(Y, T)$, assessing the correctness or validity at the level of individual tactics $T \in Y$. Unlike implicit rewards or Monte Carlo estimations typically interpreted as process rewards, our method explicitly assigns correctness-based rewards at each tactic step.

Assume that, analogous to the GRPO training rollout framework, given a question q, an LLM generates a group of responses $\{Y_1, Y_2, \ldots, Y_G\}$. Lean produces an outcome-based rewards:

$$r_{\text{outcome}}(Y_i) \ = g(Y_i)$$

We define the outcome-based advantage for any token $y_{i,t}$ in response Y_i as:

$$A_{\text{outcome},\,i,\,t} \ = \ \frac{g\big(Y_i\big) \ - \ \text{mean}\big(g(Y_1), \ldots, g(Y_G)\big)}{\text{std}\big(g(Y_1), \ldots, g(Y_G)\big)}\,.$$

Beyond binary outcome verification signals, we further design elaborate rewards based on the AST feedback produced by the Lean parser as in section 3.1. We leverage this AST feedback to distinguish between different kinds of tactics: for example, whether a tactic is elaborated successfully (i.e., type-correct and locally sound), but may still leave unresolved subgoals that prevent the proof from being completed, or whether it has type errors or parser-level mismatches. This structured feedback allows us to assign more fine-grained process-based rewards. Since, we sorted the tree node containing proof state by increasing order, we apply a First Error Propagation rule when mapping Lean's feedback into tactic-level rewards as (Lu et al., 2024; Lightman et al., 2023). Given a sequence of

tactics $(T_1, \ldots, T_N)$, once an error is observed at T_j, we propagate this failure to all subsequent tactics, i.e., every T_k with $k \geq j$ is treated as erroneous for the purpose of reward assignment.

$$\text{Let } j = \min\{i : T_i \text{ contains an error}\}. \quad \varphi(Y, T_k) = \begin{cases} d_2, & g(Y) = 0 \text{ and } k \geq j, \\ d_1, & g(Y) = 0 \text{ and } k < j \text{ and no error}, \\ 1, & g(Y) = 1. \end{cases}$$

Unlike Lean, which parses proofs into a tree structure, the LLM generates proofs in an autoregressive, causal manner. Once the first erroneous tactic T_j occurs, the continuation $T_{j+1}, \ldots, T_N$ is conditioned on an invalid prefix, and therefore cannot constitute a valid reasoning process. First-error propagation enforces this principle by assigning error signals to all subsequent tactics, ensuring causal and type-theoretic credit assignment.

For any arbitrary response Y_i, composed of tactics $Y_i = \{T_{i,1}, T_{i,2}, \ldots\}$, if we set s_j, a_j as the state and tactic $T_{i,j}$ at step j in response Y_i, the process-based reward for tactic $T_{i,j}$ is:

$$r_{\text{process}}(s_j, a_j) = r_{\text{process},\, i,\, j} = \varphi(Y_i,\, T_{i,j}).$$

The corresponding process-based advantage is

$$A_{\text{process},\, i,\, j} = r_{\text{process},\, i,\, j} - \text{mean}\big(g(Y_1), \ldots, g(Y_G)\big).$$

Here, the subtraction of the mean outcome reward serves as a dynamic baseline reflecting the difficulty of the problem q as GRPO algorithm. If the problem is easier, the mean outcome reward becomes higher, thus penalizing incorrect proofs and their tactics more heavily. Conversely, for more challenging problems, the lower baseline imposes less severe penalties.

4.2 Integrating Lean into Tactic-based Reinforcement Learning

We then integrate these two types of advantages into the standard GRPO objective as follows.

$$A_{i,t} = A_{\text{outcome},\, i,\, t} + \mathbf{1}\{t = \text{first}(T_{i,s(i,t)})\} \cdot A_{\text{process},\, i,\, s(i,t)},$$

where $s(i,t) \in \{1, \ldots, N\}$ is the index of the tactic containing the token t in Y_i, $\text{first}(T_{i,j})$ indicates the first token of the tactic. i.e., we assign the tactic advantage only to the first token of each tactic. We applied the advantage $A_{i,t}$ into GRPO objective function:

$$L(\theta) = \mathbb{E}_{q \sim P(Q),\, \{Y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(Y|q)} \left[\frac{1}{G} \sum_{i=1}^{G} \Big\{ \frac{1}{|Y_i|} \sum_{t=1}^{|Y_i|} \min\Big(\rho_{i,t}\, A_{i,t},\ \text{clip}\big(\rho_{i,t},\, 1-\epsilon,\, 1+\epsilon\big)\, A_{i,t} \Big) - \beta\, D_{\text{KL}}\big[\pi_\theta \,\|\, \pi_{\text{ref}}\big] \Big\} \right]. \tag{1}$$

where $\rho_{i,t} = \dfrac{\pi_\theta\big(y_{i,t} | q, Y_{i,<t}\big)}{\pi_{\theta_{\text{old}}}\big(y_{i,t} | q, Y_{i,<t}\big)}$. This formulation explicitly leverages both the global correctness signal $A_{\text{outcome},\, i,\, t}$ from proof outcomes and the detailed, tactic level correctness assessment $A_{\text{process},\, i,\, s(i,t)}$. By combining them into a single advantage $A_{i,t}$, we enrich the learning signal provided to the LLM based proof generator under the GRPO framework.

Rather than propagating cumulative rewards across an entire proof trajectory, we collapse credit assignment to Lean-verified, tactic-level signals. In general RL, a suboptimal step may still obtain positive return if later rewards are high, but in mathematical proof, this could be unsound: once a tactic fails, all subsequent steps are invalid under first-error propagation. Empirically, return-based credit led to unstable optimization, as it requires a value function or auxiliary estimator to normalize scale and reduce variance. Hence, we adopt a simpler formulation that combines normalized outcome-level signals with tactic-level rewards, without computing returns (See Appendix G).

5 Experiments

5.1 Experimental Setup

We trained on 10k samples randomly drawn from the STP dataset (3.26M proofs). Proofs were verified via Lean through a REPL interface, with a 15s timeout per attempt. Baselines included STP-Lean and DeepSeek-Prover-V1.5-SFT, the latter additionally fine-tuned on 500k STP samples before RL. We used non-CoT prompt, response styles as in (Xin et al., 2024b). Full hyperparameters and training details are provided in Appendix B.

[1] **Budgets are not directly comparable:** tree-search budgets count expansions/verifier calls at inference, whereas our budgets count whole-proof samples. Our aim is to improve single-shot generation under a different compute regime.

Method	Model size	Budget [1]	MiniF2F-Test	ProofNet-Test
Whole-Proof Generation Methods				
DeepSeek-Prover-V1.5-SFT (Xin et al., 2024a)	7B	32	$46.2\% \pm 0.2$	$14.3\% \pm 0.3$
		64	$47.5\% \pm 0.1$	$15.05\% \pm 1$
DeepSeek-Prover-V1.5-RL (Xin et al., 2024a)	7B	32	$48\% \pm 0$	$16\% \pm 1$
		64	$48.8\% \pm 0.4$	$17.4\% \pm 0.6$
Goedel-Prover-SFT (Lin et al., 2025c)	7B	32	$56.9\% \pm 0.4$	$15.6\% \pm 0.5$
		64	$57.9\% \pm 0.5$	$16.7\% \pm 0$
STP-Lean (Dong & Ma, 2025)	7B	32	$55.9\% \pm 0.2$	$17.2\% \pm 0$
		64	$56.7\% \pm 0.2$	**19.1%** ± 0.4
STP-Lean + Ours	7B	32	**57.1%** ± 0.8	**18.6%** ± 0.3
		64	**59.2%** ± 0.5	$19\% \pm 0.3$
DeepSeek-Prover-V1.5 + STP	7B	32	$54.9\% \pm 0.7$	$16.8\% \pm 0.3$
		64	$57.2\% \pm 0.2$	$17.7\% \pm 0$
DeepSeek-Prover-V1.5 + STP + Ours	7B	32	**56.3%** ± 0.6	**17.6%** ± 0.8
		64	**57.8%** ± 0.4	**18.5%** ± 0.3
Tree Search Methods				
Lean-STaR	7B	$64 \times 1 \times 50$	46.3%	–
InternLM2-Math-Plus-7B (Ying et al., 2024b)	7B	$1 \times 32 \times 100$	48.8%	–
InternLM2.5-StepProver	7B	$4 \times 32 \times 600$	$58.5\% \pm 0.9$	–
DeepSeek-Prover-V1.5-RL + RMaxTS (Xin et al., 2024a)	7B	3,200	$55.0\% \pm 0.7$	$21.5\% \pm 0.8$

Table 1: Budgets for whole-proof methods denote the *sample budget* (N) per problem; for tree-search methods, budgets denote the authors reported *search expansions counts*. We compare with InternLM family and DeepSeek-Prover based tree search methods for fair comparison with our method. Bold indicates the best number within the whole-proof block. All our GRPO-style runs use the same STP subset, generations per query, and a 15s Lean timeout.The notation $\mu \pm \sigma$indicates the mean and the standard deviation each.

5.2 Main Results

In Table 1, the results on both the MiniF2F and ProofNet datasets demonstrate that training with tactic-based advantage via Lean consistently enhances model performance across most evaluation settings. For the STP-Lean model, our method improves MiniF2F performance up to $+2.5\%$p (pass@64), and ProofNet performance by $+1.4\%$p (pass@32), while showing a negligible decrease of -0.1%p on pass@64. Similarly, for DeepSeek-Prover-V1.5, our approach achieves marginal yet consistent increases across all benchmarks.

Across both MiniF2F and ProofNet, leveraging Lean as a *process-level oracle* yields consistent, stable gains over outcome-only reinforcement learning, without increasing training cost. In particular, in Table 2, when applied to DeepSeek-Prover models, GRPO fails to yield any gains on the ProofNet-Test set, and in some cases even underperforms relative to the supervised baseline. This highlights a key limitation of purely outcome-based credit assignment: it often lacks stability and fails to provide consistent guidance for proof search.

By comparison, tactic-level credit assignment yields more reliable improvements. While minor drops appear in some settings, it generally provides stable gains over outcome-only GRPO. For example, on MiniF2F (pass@64), STP-Lean + Ours improves by $+2.5\%$p over the baseline, compared to $+1.2\%$p with GRPO. As shown in Table 1 and 2, tactic-based training consistently matches or surpasses both the supervised baseline and GRPO. Importantly, this stability comes with almost no extra cost: since both methods already use REPL interactions with Lean, the additional sorting and scoring overhead is negligible.

Compared to strong search-based baselines (e.g., InternLM families, DeepSeek-Prover-RL+RMaxTS), our single-shot, whole-proof training approaches their reported accuracy (e.g., 59.2% vs. 58.5% pass@64 on MiniF2F) while avoiding large search-time compute.

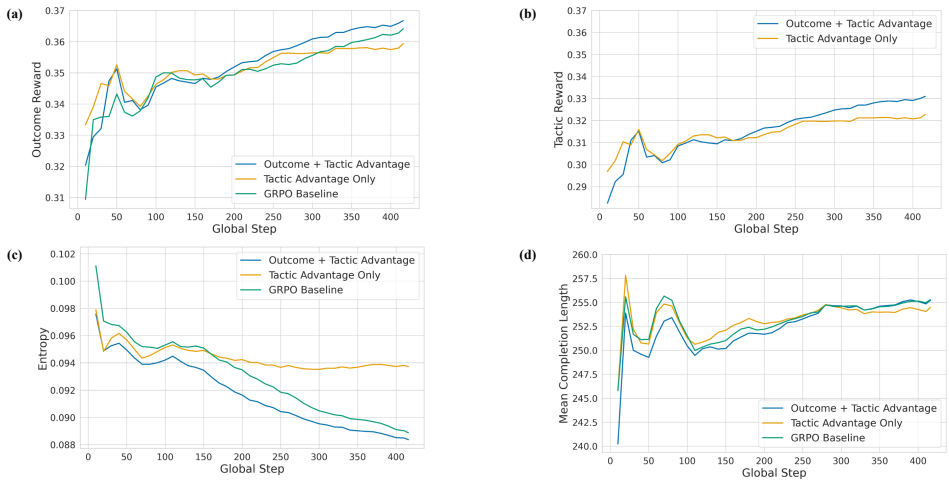

Figure 2: Training dynamics showing (a) outcome reward, (b) tactic reward, (c) entropy, and (d) mean of response length during reinforcement learning.

Model	Model Size	Sample Budget	MiniF2F - Test	ProofNet - Test
STP + Outcome only (GRPO)	7B	32	$55.7\% \pm 1$	$17.4\% \pm 0.6$
		64	$57.9\% \pm 0.5$	**19%** ± 0.3
STP + Tactic only	7B	32	$55.6\% \pm 0.6$	$18.3\% \pm 0$
		64	$56.8\% \pm 0.6$	$17.9\% \pm 0.8$
STP + Outcome+Tactic RL (ours)	7B	32	**57.1%** ± 0.8	**18.6%** ± 0.3
		64	**59.2%** ± 0.5	**19%** ± 0.3
DeepSeek-Prover-V1.5 + Outcome only (GRPO)	7B	32	$55.3\% \pm 0.4$	$16.8\% \pm 0.8$
		64	$57.4\% \pm 0.4$	$17.6\% \pm 0.8$
DeepSeek-Tactic only	7B	32	$54.9\% \pm 0.7$	$16.8\% \pm 0.8$
		64	$57.8\% \pm 1$	$17.6\% \pm 0.3$
DeepSeek-Prover-V1.5 + Outcome+Tactic RL (ours)	7B	32	**56.3%** ± 0.6	**17.6%** ± 0.8
		64	**57.8%** ± 0.4	**18.5%** ± 0.3

Table 2: Ablation study of STP-Lean with various verifier methods on MiniF2F-Test and ProofNet-Test benchmarks.

5.3 Analysis

The Role of Outcome and Tactic Rewards. Integrating both outcome-level and tactic-level signals yields more effective learning than employing either signal in isolation. Outcome-only RL, as in GRPO, is constrained by the sparsity of binary feedback: improvements are gradual and the final performance plateaus at a relatively low level (Figure 2(a)). In contrast, tactic-only training provides dense feedback but lacks a global objective, resulting in premature convergence. When combined, outcome rewards serve as a global objective function, while tactic rewards provide local credit assignment, enabling both rapid progress and higher performance. This complementary relationship is further reflected in Figure 2(b), where tactic-only supervision's tactic reward plateaus, but outcome-tactic combined rewards continue to increase steadily. The results in Table 2 supports this finding: outcome signals enforce proof-level correctness, while tactic signals supply verifiable intermediate feedback; only their integration consistently improves performance across benchmarks.

Entropy and Proof Length. The use of fine-grained rewards influences exploration not by indiscriminately broadening the search space but by focusing learning on more informative decision points. As shown in Figure 2(c), outcome+tactic training converges to lower entropy than tactic-only and outcome-only settings, indicating that the policy becomes more decisive as training progresses. This does not correspond to mode collapse: Figure 2(d) shows that the average proof length remains stable across all methods, suggesting that the performance gains are not attributable to trivial lengthening of outputs. Instead, denser intermediate rewards appear to reduce the need for broad stochastic exploration, guiding the model toward more efficient proof strategies.

Model	Model Size	Sample Budget	MiniF2F - Test	ProofNet - Test
All tokens	7B	32	$56.3\% \pm 0.6$	$18.1\% \pm 0.8$
		64	$57.8\% \pm 0.7$	$18.1\% \pm 0.8$
Entropy-based	7B	32	$56.4\% \pm 0.2$	$17.9\% \pm 0.8$
		64	$57.1\% \pm 0.5$	$18.5\% \pm 0.3$
Last token	7B	32	$56.7\% \pm 0.9$	$17.2\% \pm 0$
		64	$57.5\% \pm 0.6$	$17.7\% \pm 0.5$
First token	7B	32	**57.1%** ± 0.8	**18.6%** ± 0.3
		64	**59.2%** ± 0.5	**19%** ± 0.3

Table 3: Ablation study of STP-Lean on how to distribute tactic-level advantages across tokens.

Model	Model Size	Sample Budget	MiniF2F - Test	ProofNet - Test
No First Error	7B	32	$56.4\% \pm 0.9$	$17.4\% \pm 0.3$
		64	$58.2\% \pm 0.7$	$18.3\% \pm 0.3$
No Baseline	7B	32	$56.7\% \pm 0.2$	$17.9\% \pm 0.3$
		64	$57.4\% \pm 0.7$	$18.3\% \pm 0.5$
Same tactic reward	7B	32	**57.7%** ± 0.2	$17.6\% \pm 0.6$
		64	$58.7\% \pm 0.8$	$18.1\% \pm 0.6$
Outcome+Tactic RL (ours)	7B	32	$57.1\% \pm 0.8$	**18.6%** ± 0.3
		64	**59.2%** ± 0.5	**19%** ± 0.3

Table 4: Ablation study on reward strategies for tactic-level feedback in STP-Lean. Additional experiments include removing the first-error propagation policy (No First Error), removing the baseline extraction (No Baseline). and using equal penalties for all tactics (Same tactic reward).

Tactic to Token Level Credit Assignment. After defining tactic-level rewards, next step is how to distribute them across tokens. In our main method, the tactic advantage is assigned only to the first token of the tactic. For comparison, we conducted ablations where the tactic advantage was instead (i) distributed to all tokens of a tactic, (ii) assigned only to the last token, (iii) keep first token reward distribution, but additionally choose 10% tokens within the tactic with respect to high entropy. As Wang et al. (2025d) showed that high entropy tokens could be reasoning drive tokens, we speculated that this method can automatically select the tokens for serving as fork in formal reasoning. Assigning credit to the first token of each tactic achieves the most stable and consistent improvements, as evidenced by Table 3. Alternative strategies do not yield comparable gains and in some cases even degrade performance. This outcome aligns with the semantics of Lean proofs: the first token corresponds to the tactic keyword (e.g., `intro`, `apply`, `have`), determining the subsequent proof strategy and constrains the structure of subgoals. Concentrating credit on this decision point enhances the models ability to select tactics appropriately, resulting in more reliable downstream reasoning. This finding is also aligned with (Fang et al., 2025), showing that focusing on key tokens during training improves performance on long-context tasks.

Reward Strategy for Tactic-level Feedback. For tactic-level feedback to be effective, it must reflect the sequential dependency of proof construction, account for task difficulty, and distinguish between partially correct and erroneous steps. The first-error propagation rule ensures that once an error occurs, subsequent tactics are treated as invalid; removing this rule significantly reduces performance (Table 4), because once the first error occurs, the remaining tactics are evaluated in an invalid context and cannot salvage correctness. Incorporating a difficulty-normalized baseline further stabilizes training, while its absence leads to degraded results. Finally, differentiating penalties between partially correct tactics and outright erroneous ones proves essential: collapsing these into a single penalty $d_1 = d_2$ yields inconsistent outcomes- improvements on MiniF2F but declines on ProofNet. These results indicate that an effective tactic-level reward scheme must combine sequential error propagation, difficulty-aware normalization, and differentiated penalties in order to provide stable and semantically faithful learning signals. In the sensitivity analysis of Appendix C, assigning different values to d_1 and d_2 leads to robust performance, tending to outperform the GRPO baseline and yielding the strongest improvements on MiniF2F.

Effect of Verification Timeouts When using Lean as a verifier, long proofs can lead to excessive verification time, so we introduced timeout thresholds of 5, 10, 15, and 30s (Figure 3). A 5s

limit gave the worst results, since even relatively simple proofs often exceeded this window and produced too few valid reward signals. In contrast, 10-30s yielded much stronger performance, with 15s giving the best overall balance. Interestingly, 10-15s sometimes outperformed 30s despite the shorter allowance. We attribute this to the fact that discarding overly complex proofs biases training toward shorter and more efficient proof strategies. This effect is amplified in our setting because we evaluate non-CoT responses purely by Lean verification (without NL commentary): longer outputs are not only slower to check but also more error-prone. As a result, shorter verification limits encourage the model to generate concise, canonical proofs, which we hypothesize leads to better generalization at test time.

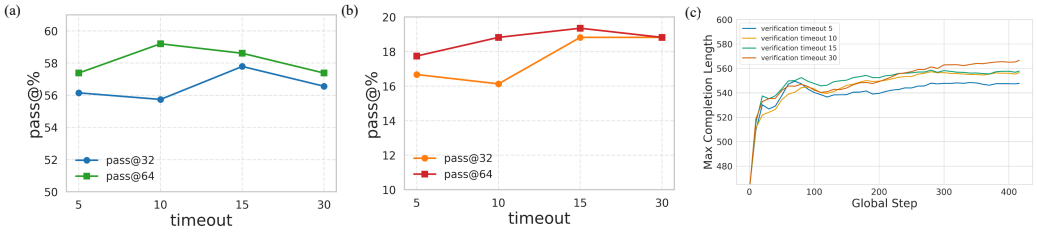

Figure 3: Ablation study of STP-Lean on different Lean verification timeouts (5, 10, 15, and 30 seconds) during outcome+tactic based training. We report evaluation performance on the MiniF2F and ProofNet benchmarks (a),(b), and the maximum response length observed during training (c).

Qualitative Analysis We conduct a qualitative analysis to better understand the differences between our tactic-reward-based approach and the baseline STP model. Specifically, we examine proofs from two benchmark problems: Imo_1960_p2 in the MiniF2F benchmark and 1_14 from ProofNet (See Appendix F). Table 8 presents the proof generated by our tactic-reward model, while Table 9 shows the corresponding proof from the STP model.The key difference in the first example lies in how the upper bound $x < 45/8$ is established. The STP model attempts to use the nonlinear inequality tactic nlinarith, which results in an error. By contrast, our tactic-reward model learns to penalize such invalid tactic choices. Instead, it carefully applies previously proven assumptions and intermediate lemmas before invoking nlinarith, thereby producing a correct and more robust proof. The second example comes from the ProofNet benchmark (1_14 exercise). As shown in Table 10, the tactic-reward model begins by normalizing the problem using a rewrite tactic. In contrast, the baseline model in Table 10 skips this normalization step and directly attempts inequality manipulations, which ultimately causes the proof to fail. Analysis for Failure case is in Appendix H These anecdotal examples illustrate plausible mechanisms; for definitive evidence, see Table 1.

6 Conclusion

We introduced a reinforcement learning framework that uses the Lean proof assistant as a process-level reward oracle. Unlike prior outcome-only methods, our approach leverages Lean's parsing and validation to provide both global outcome signals and fine-grained tactic rewards, integrated into a GRPO objective. This enables denser, verifiable credit assignment: outcome rewards enforce proof-level success, while tactic rewards guide step-level reasoning. Experiments on STP-Lean and DeepSeek-Prover-V1.5 show consistent improvements on MiniF2F and ProofNet, with stable gains achieved by assigning tactic rewards to the first token of each tactic and first error propagation method. Overall, proof assistants can serve not only as checkers at inference but also as structured feedback sources during training, pointing toward more stable and effective RL for reasoning.

Limitations

We did not compare against learned PRMs, as they rely on natural-language CoT supervision and large annotated datasets that are not yet available for Lean. Our models also generate pure Lean proofs without long CoT, leaving open how to design fine-grained rewards for long-form reasoning. In addition, tactic rewards in our method were fixed scores (d_1, d_2), which proved effective but somewhat sensitive across different models and datasets. Developing general advantage estimators and large-scale tactic-level datasets remains important future work.

Ethics Statement

This research does not involve human subjects, personal data, or sensitive information that could raise ethical concerns. All experiments were conducted on publicly available formal mathematical datasets, and the proposed models are solely trained and evaluated for automated theorem proving tasks.

Reproducibility statement

To ensure the reproducibility of our work, we provide a comprehensive description of our and training process in Section 5.1. For further details such as hyperparameter, version of Lean, we introduced it in Appendix B. We utilized Huggingface and trl library for our experiments, and we plan to release the source codes to facilitate future research.

References

AlphaProof and AlphaGeometry teams. Ai achieves silver-medal standard solving international mathematical olympiad problems. Google DeepMind Blog, July 2024. URL https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/. Accessed: 17 April 2025.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. URL https://arxiv.org/abs/2302.12433.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics, 2024. URL https://arxiv.org/abs/2310.10631.

Kaito Baba, Chaoran Liu, Shuhei Kurita, and Akiyoshi Sannai. Prover agent: An agent-based framework for formal mathematical proofs, 2025. URL https://arxiv.org/abs/2506.19923.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova Dassarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, John Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Christopher Olah, Benjamin Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *ArXiv*, abs/2204.05862, 2022. URL https://api.semanticscholar.org/CorpusID:248118878.

Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, Gérard Huet, César Muñoz, Chetan Murthy, Catherine Parent-vigouroux, Christine Paulin-Mohring, Amokrane Saïbi, and Benjamin Werner. The coq proof assistant reference manual : Version 6.1. 06 1997.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Meng Cao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Scar: Shapley credit assignment for more efficient rlhf, 2025. URL https://arxiv.org/abs/2505.20417.

Alex J. Chan, Hao Sun, Samuel Holt, and Mihaela van der Schaar. Dense reward for free in reinforcement learning from human feedback, 2024. URL https://arxiv.org/abs/2402.00782.

Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(3): 114–115, 1940. doi: 10.2307/2266866.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards, 2025. URL https://arxiv.org/abs/2502.01456.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *CADE*, 2015. URL https://api.semanticscholar.org/CorpusID:232990.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL https://arxiv.org/abs/2502.00212.

Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is wrong with perplexity for long-context language modeling?, 2025. URL https://arxiv.org/abs/2410.23771.

Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387945938.

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

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023. URL https://arxiv.org/abs/2210.12283.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment, 2024. URL https://arxiv.org/abs/2410.01679.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving, 2022. URL https://arxiv.org/abs/2205.11491.

Chengpeng Li, Zhengyang Tang, Ziniu Li, Mingfeng Xue, Keqin Bao, Tian Ding, Ruoyu Sun, Benyou Wang, Xiang Wang, Junyang Lin, and Dayiheng Liu. Cort: Code-integrated reasoning within thinking, 2025. URL https://arxiv.org/abs/2506.09820.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-star: Learning to interleave thinking and proving, 2025a. URL https://arxiv.org/abs/2407.10040.

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, 2025b. URL https://arxiv.org/abs/2502.07640.

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, 2025c. URL https://arxiv.org/abs/2508.03613.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, Zhicheng Yang, Jing Tang, and Zhijiang Guo. Process-driven autoformalization in lean 4, 2024. URL https://arxiv.org/abs/2406.01940.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Leonardo Moura and Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*, pp. 625–635. 07 2021. ISBN 978-3-030-79875-8. doi: 10.1007/978-3-030-79876-5_37.

A. Newell, J. C. Shaw, and H. A. Simon. Empirical explorations of the logic theory machine: a case study in heuristic. In *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*, IRE-AIEE-ACM '57 (Western), pp. 218230, New York, NY, USA, 1957. Association for Computing Machinery. ISBN 9781450378611. doi: 10.1145/1455567.1455605. URL https://doi.org/10.1145/1455567.1455605.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606122.

Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3540433767.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. Apollo: Automated llm and lean collaboration for advanced formal reasoning, 2025. URL https://arxiv.org/abs/2505.05758.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020. URL https://arxiv.org/abs/2009.03393.

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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning, 2024. URL https://arxiv.org/abs/2410.08146.

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.

Trieu Trinh, Yuhuai Tony Wu, Quoc Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. URL https://www.nature.com/articles/s41586-023-06747-5.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, Jian Yin, Zhenguo Li, and Xiaodan Liang. DT-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12632–12646, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.706. URL https://aclanthology.org/2023.acl-long.706/.

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.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Ruida Wang, Yuxin Li, Yi R. Fung, and Tong Zhang. Let's reason formally: Natural-formal hybrid reasoning enhances llm's math capability, 2025b. URL https://arxiv.org/abs/2505.23703.

Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. Ma-lot: Multi-agent lean-based long chain-of-thought reasoning enhances formal theorem proving, 2025c. URL https://arxiv.org/abs/2503.03205.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning, 2025d. URL https://arxiv.org/abs/2506.01939.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models, 2022. URL https://arxiv.org/abs/2205.12615.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems, 2024. URL https://arxiv.org/abs/2410.15700.

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, 2024a. URL https://arxiv.org/abs/2405.14333.

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. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2024b. URL https://arxiv.org/abs/2408.08152.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems, 2024a. URL https://arxiv.org/abs/2406.03847.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan 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, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels, 2024. URL https://arxiv.org/abs/2412.01981.

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, 2025. URL https://arxiv.org/abs/2504.06122.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2022. URL https://arxiv.org/abs/2109.00110.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo, 2023. URL https://arxiv.org/abs/2307.04964.

Thomas Zhu, Joshua Clune, Jeremy Avigad, Albert Qiaochu Jiang, and Sean Welleck. Premise selection for a lean hammer, 2025. URL https://arxiv.org/abs/2506.07477.

APPENDIX

A ADDITIONAL RELATED WORKS

Automatic Theorem Proving An automated theorem prover typically consists of two stages. The first is the process of translating mathematical statements written in natural language into formal statements. Wu et al. (2022) utilized large language models to translate mathematical questions into formal languages such as Isabelle and HOL. This process, known as autoformalization, is primarily used for constructing datasets intended for formal reasoning. Benchmarks or training datasets such as MiniF2F, LeanWorkbook, ProofNet, Deepseek-Prover have employed LLMs to translate natural language mathematical statements into formal expressions, contributing to the creation of high-quality formal reasoning datasets (Zheng et al., 2022; Azerbayev et al., 2023; Xin et al., 2024a; Ying et al., 2024a).

The second stage involves generating a formal proof from the translated formal statement. This proof generation process is typically divided into two approaches: one involves step-by-step inference, such as tree search during inference time (Polu & Sutskever, 2020; Azerbayev et al., 2024; Wu et al., 2024; Xin et al., 2024b), and the other generates the entire proof at once (Xin et al., 2024a; Lin et al., 2025b). Ospanov et al. (2025); Wang et al. (2025c); Baba et al. (2025) use Lean compiler as agent for complementing formal reasoning ability of LLMs, while (Dong & Ma, 2025) enhances formal reasoning by augmenting problems via conjecture. (Jiang et al., 2023) presents a unified framework that combines both autoformalization and proof generation in a single pipeline.

Existing methods such as Lean-STaR and RMaxTS (Lin et al., 2025a; Xin et al., 2024b) utilize Lean as a step-checker during inference, generating steps sequentially and searching optimally via tree search to find valid proofs. In contrast, in this paper, similar to (Wang et al., 2025a; Zhang et al., 2025; Ren et al., 2025; Ji et al., 2025), we utilize Lean as a whole-proof verifier during the training stage. Additionally, beyond merely providing correctness checks for the entire proof, we leverage Lean's parsing and elaboration capabilities to validate each individual tactic step, integrating this step-level validation into the training process. In other words, we employ the Lean proof assistant as a process-based reward model for validating the correctness of each reasoning steps. (Lightman et al., 2023).

Unlike prior work that leverages dense feedback from proof assistants, our research takes a different perspective: we rely solely on the rule-based signals of the symbolic engine, without introducing any natural language. Approaches such as Lin et al. (2025a;c); Wang et al. (2025b); Li et al. (2025),exploit natural language reasoning as a form of annotation to enhance LLMs formal reasoning abilities. In contrast, our method improves performance exclusively through reward signals provided by Lean, without any reliance on natural language.

Reinforcement Learning in Language Models While developing or applying algorithms such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024) plays a significant role in reinforcement learning, reward shaping and credit assignment are central challenges in reinforcement learning. (Cobbe et al., 2021) introduced a reward model based on the outcome of a response. However, similar to other areas of RLHF, this approach suffers from the limitation of sparse rewards (Chan et al., 2024; Zheng et al., 2023).

To address this, process-based reward model (PRM) assigns step-level rewards during inference to guide rationale generation (Lightman et al., 2023), and can also be used to reward responses during training (Setlur et al., 2024; Kazemnejad et al., 2024). (Yuan et al., 2024; Cui et al., 2025) derived an implicit PRM from the ORM without any data annotation or additional training. When assigning scores to reasoning steps, (Lightman et al., 2023) defined the reward as the correctness of each step, which required substantial human annotation effort. (Wang et al., 2024) instead adopted a Monte Carlo approach, defining the score of a step as the proportion of successful rollouts originating from that step. While recent PRM approaches show promise in natural language reasoning, they require large annotated datasets of step-level correctness. To the best of our knowledge, no such dataset exists for Lean or formal theorem proving, making a direct comparison with a learned PRM baseline infeasible. This further motivates our approach of leveraging the Lean verifier itself as a process oracle. In contrast, our process-based reward leverages the Lean theorem prover to automatically

verify the correctness of each step, thereby eliminating the need for human annotators or sampling many proofs steps.

Our work can also be interpreted through the lens of reward shaping (Ng et al., 1999). Prior approaches have explored different mechanisms for distributing reward signals: Chan et al. (2024) leverages the internal attention patterns of LLMs to assign higher weights to important tokens, Cao et al. (2025) employs Shapley values to allocate credit across actions, and Kazemnejad et al. (2024) uses Monte Carlo rollouts to estimate and distribute rewards over intermediate steps. In contrast, our method relies on an external parser-the Lean theorem prover-to parse tactics and assign reward to the first token, thereby implementing a form of credit assignment.

B Experimental Detail

Data. We randomly sampled 10k instances from the STP dataset (3.26M total) for RL training. For DeepSeek-Prover-V1.5-SFT, we applied an additional supervised fine-tuning step on 500k STP samples before RL, since the vanilla model produced low-quality proofs during RL training.

Verification. We use Lean 4.9.0-rc1 for all experiments in the paper. During training, we used a REPL (read-eval-print loop) interface with Lean to verify proofs and assign outcome- and tactic-level rewards. Each proof attempt was given a maximum of 15 seconds for verification; longer runs were treated as failures (both outcomes, tactic rewards are zero).

RL configuration. For GRPO training, we used $G = 4$ generations per prompt, sampling temperature 0.9, KL coefficient 0.04, clipping $\epsilon = 0.2$, and the DAPO upper bound 0.28 (Yu et al., 2025). Tactic-level rewards were fixed at $d_1 = -0.05$ and $d_2 = -0.1$ for partially valid and erroneous tactics in the main experiments, respectively. All experiments used non-CoT prompts, following Xin et al. (2024b).

Training details. We fine-tuned the models with LoRA (rank 64, $\alpha = 64$) using bf16 precision. The AdamW optimizer was used with a learning rate of 1.0×10^{-5}. Maximum response length was set to 1024 tokens during both training and evaluation.

Evaluation. For decoding we used temperature 1.0 and top-p 0.95. We re-evaluated all baselines under the same non-CoT and budget settings (32/64 samples). All reported results are from the final checkpoint.

Compute. Training was conducted on 4 $\times$ NVIDIA A6000 GPUs, requiring approximately 21-23 hours.

C Hyperparameter ablations on d_i

Setting	Model Size	Sample Budget	MiniF2F - Test	ProofNet - Test
STP-baseline	7B	32	$55.9\% \pm 0.2$	$17.2\% \pm 0$
		64	$56.7\% \pm 0.2$	$19.1\% \pm 0.4$
GRPO baseline	7B	32	$55.7\% \pm 1$	$17.4\% \pm 0.6$
		64	$57.9\% \pm 0.5$	$19\% \pm 0.3$
$d_1 = -0.05$, $d_2 = -0.10$	7B	32	$57.1\% \pm 0.8$	$18.6\% \pm 0.3$
		64	$59.2\% \pm 0.5$	$19\% \pm 0.3$
$d_1 = d_2 = -0.10$	7B	32	$57.7\% \pm 0.2$	$17.6\% \pm 0.6$
		64	$58.7\% \pm 0.8$	$18.1\% \pm 0.6$
$d_1 = -0.05$, $d_2 = -0.50$	7B	32	$57\% \pm 0.4$	$17.6\% \pm 0.3$
		64	$59.2\% \pm 0.5$	$18.6\% \pm 0.8$

Table 5: Ablation study on tactic-level penalties d_1, d_2. We compare outcome-only GRPO baseline with three variants of (d_1, d_2) settings. Results are reported as pass@32 and pass@64 (%) on MiniF2F and ProofNet test sets. The experiment is couducted with STP-Lean model.

This ablation shows that introducing a gap between d_1 and d_2 makes the method more robust: performance remains consistently above the GRPO baseline, with stable gains across different penalty scales and especially clear improvements on MiniF2F.

D PROMPTS

For training and evaluation, we used non-COT evaluation followed by (Dong & Ma, 2025) and (Xin et al., 2024b). The examples are introduced in Table 6, 7.

Prompt Template

```
Complete the following Lean 4 code:\n\n
```lean4\n{header}{formal_statement}
```

Table 6: Prompt template used in training and evaluation. We selected non-COT generation which is appropriate with our MDP setting

Prompt Example

```
Complete the following Lean 4 code:

```lean4
import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

theorem theorem_exercise_2011_2_257 (G : Type*) [Group G] [Fintype
↪  G]
(h : Fintype.card G | 2) (x : G) : x ^ 2 = 1
  ( x y : G, x * y = y * x)  ( a : G, a = aź)   a : G, a^2 = 1
  let p x : G  G := by
```

Table 7: A training sample used in training and evaluation. We selected non-COT generation which is appropriate with our MDP setting.

E CREDIT ASSIGNMENT IN REINFORCEMENT LEARNING

Let y_t be the t-th token of y, R denote the reward model, π_θ represent the policy model, and π_{ref} be the reference model. L denote the response length and B be a coefficient controlling the distance between the policy and the reference policy. In PPO, the token-level reward at position t is defined as: $r_t(x, y_t) = R(x, y)\mathbf{1}(y_t = L) - B \log\left(\frac{\pi_\theta(y_t|x)}{\pi_{\text{ref}}(y_t|x)}\right)$, where the non-zero reward $R(x, y)$ is assigned only to the last token. For all other tokens, only a KL divergence penalty is applied via a log ratio $\log\left(\frac{\pi_\theta(y_t|x)}{\pi_{\text{ref}}(y_t|x)}\right)$. Direct usage of rewards can lead to high variance; therefore, PPO reduces variance by utilizing a learned value model V.This value network assigns a value to each token y_t, from which the Temporal-Difference (TD) error is computed as: $\delta_t = r_t + \gamma V(y_{t+1}) - V(y_t)$ where γ is discounted factor. Then, the advantage for each token is recursively calculated as follows: $A_L = \delta_L$, $A_t = \delta_t + \gamma\lambda A_{t+1}$, for $t = L-1, L-2, \ldots, 1$. Subsequently, because the computed advantages A_t can exhibit high variance during exploration, normalization or similar techniques are applied, resulting in the final adjusted advantage A_t. This adjusted advantage is then utilized in the PPO loss defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\frac{\pi_\theta(y_t \mid x)}{\pi_{\theta_{\text{old}}}(y_t \mid x)}A_t,\ \text{clip}\left(\frac{\pi_\theta(y_t \mid x)}{\pi_{\theta_{\text{old}}}(y_t \mid x)},\ 1-\epsilon,\ 1+\epsilon\right)A_t\right)\right] \tag{2}$$

In contrast, REINFORCE-based methods such as GRPO and RLOO have proposed algorithms that optimize policies directly from verifiable rewards without requiring a value model, due to concerns about the computational cost and estimation capability associated with training value networks.

GRPO generates multiple response groups $\{ y_{(i)} \}_{i=1}^{G}$ for a given question q from an old policy π_{old}. Subsequently, a reward function outputs reward $r = \{ r_{(i)} \}_{i=1}^{G}$ for each response group. If we set $y_{i,t}$ as $t-$th token index of response y_i The advantage for $y_{i,t}$, $A_{i,t}$ is then computed by normalizing these rewards as follows: $\hat{A}_{i,t} = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$.

This advantage is uniformly assigned to each token $y_{i,t}$ constituting the response y_i. Subsequently, this identical token-level advantage is utilized in calculating the following loss:

$$L_{\text{GRPO}}(\theta) = \mathbb{E}_{q \sim P(Q),\, \{ y_i \}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(Y|q)} \left[\frac{1}{G} \sum_{i=1}^{G} \left\{ \min\left(\frac{\pi_\theta(y_{i,t} \mid q)}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid q)} \hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(y_{i,t} \mid q)}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid q)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_{i,t} \right) - \beta \, D_{\text{KL}}\left[\pi_\theta \,\|\, \pi_{\text{ref}} \right] \right\} \right]. \quad (3)$$

As our goal is to integrate Lean-based fine-grained rewards to GRPO algorithm, our method keep balance between value-based PPO's fine-grained credit assignmethod and GRPO's simple unifrom credit assignment approach.

F GENERATED PROOFS EXAMPLES

imo_1960_p2 in minif2f from tactic-reward-trained model

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem imo_1960_p2 (x : \mathbb{R}) (h_0 : 0  1 + 2 * x)
(h_1 : (1 - Real.sqrt (1 + 2 * x)) ^ 2  0)
    (h_2 : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9) :
    -(1 / 2)  x  x < 45 / 8 := by
  have h_3 : 0  (1 + 2 * x) := by linarith
  have h_4 : 0  Real.sqrt (1 + 2 * x):= Real.sqrt_nonneg(1 + 2 * x)
  have h_5 : 0  (1 + 2 * x) * Real.sqrt (1 + 2 * x) := by positivity
  have h_6 : 0 < (1 - Real.sqrt (1 + 2 * x)) ^ 2 := by
    apply sq_pos_of_ne_zero
    intro h
    apply h_1
    nlinarith
  constructor
  nlinarith [sq_sqrt (show 0  1 + 2 * x from by linarith), h_2]
  have h_7 : x < 45 / 8 := by
    apply lt_of_not_ge
    intro hge
    have h_8 : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2  2*x+9
    := by
      rw [ge_iff_le]
      rw [le_div_iff h_6]
      nlinarith [sq_sqrt (show 0  1 + 2 * x from by linarith), hge]
    linarith
  linarith
```

Table 8: Proof of `imo_1960_p2` from MiniF2F generated by our tactic-reward-trained model.

imo_1960_p2 in minif2f from baseline model

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem imo_1960_p2 (x : \mathbb{R})
(h_0 : 0  1 + 2 * x) (h_1 : (1 - Real.sqrt (1 + 2 * x)) ^ 2  0)
    (h_2 : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9) :
    -(1 / 2)  x  x < 45 / 8 := by
  have h_3 : 0 < (1 - Real.sqrt (1 + 2 * x)) ^ 2 := by positivity
  have h_4 : 0  Real.sqrt (1 + 2 * x) := Real.sqrt_nonneg _
  have h_5 : 0  1 + 2 * x := by linarith
  have h_6 : Real.sqrt (1 + 2 * x)  1 + x := by
    rw [Real.sqrt_le_iff]
    constructor
    ů linarith
    ů nlinarith
  have h_7 : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9
  :=
  by assumption
  have h_8 : -(1 / 2)  x := by
    rw [div_lt_iff h_3] at h_7
    nlinarith
  have h_9 : x < 45 / 8 := by
    rw [div_lt_iff h_3] at h_7
    nlinarith
  exact h_8, h_9
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
  <;> assumption
```

Table 9: Proof of `imo_1960_p2` from MiniF2F generated by the baseline STP model.

exercise_1_14 in ProofNet from tactic-reward-trained model.

```
import Mathlib

open Topology Filter Real Complex TopologicalSpace Finset
open scoped BigOperators

theorem exercise_1_14
  (z : \mathbb{C}) (h : abs z = 1)
  : (abs (1 + z)) ^ 2 + (abs (1 - z)) ^ 2 = 4 := by
  rw [Complex.normSq_eq_abs, Complex.normSq_eq_abs]
  simp [Complex.normSq_add, Complex.normSq_sub, h]
  ring_nf
  <;> simp [h, Complex.normSq_eq_abs]
  <;> ring_nf
```

Table 10: Proof of `exercise_1_14` from ProofNet generated by our tactic-reward-trained model.

exercise_1_14 in ProofNet from baseline

```
import Mathlib

open Topology Filter Real Complex TopologicalSpace Finset
open scoped BigOperators

theorem exercise_1_14
  (z : \mathbb{C}) (h : abs z = 1)
  : (abs (1 + z)) ^ 2 + (abs (1 - z)) ^ 2 = 4 := by
  have h_0 : abs z ^ 2 = 1 := by simp [h]
  simp [pow_two, abs_add, abs_sub, h_0]
  ring_nf
  simp_all [sq_abs]
```

Table 11: Proof of `exercise_1_14` from ProofNet generated by the baseline STP model.

G RESULTS ON RETURN-BASED ADVANTAGE

Model	Model Size	Sample Budget	MiniF2F - Test	ProofNet - Test
Return	7B	32	$55.3\% \pm 0.4$	$18.1\% \pm 0.3$
		64	$57.5\% \pm 0.2$	$18.6\% \pm 0.3$
Outcome+Tactic RL (ours)	7B	32	**57.1%** ± 0.8	**18.6%** ± 0.3
		64	**59.2%** ± 0.5	**19%** ± 0.3

Table 12: Ablation study on reward strategies for return-based advantage.

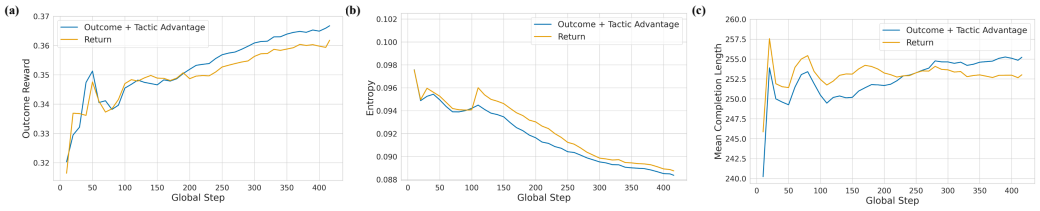

Figure 4: Training dynamics showing (a) outcome reward,(b) entropy, and (c) mean of response length during reinforcement learning in our method and return-based method.

H FAILURE CASE AND LIMITATION OF OUR METHOD

imo_2019_p1 in MiniF2f from tactic-reward-trained model.

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem imo_2019_p1 (f : \Z  \Z) :
    (\forall a b, f (2 * a) + 2 * f b = f (f (a + b)))
    \n \iff (\forall z, f z = 0)
    \n (\exists c, \forall z, f z = 2 * z + c) := by
  constructor
  intro h
  intro z
  have h_1 := h 0 0
  have h_2 := h 0 1
  have h_3 := h 1 0
  have h_4 := h 1 1
  have h_5 := h 2 0
  have h_6 := h 2 1
  have h_7 := h 0 2
  have h_8 := h 1 2
  have h_9 := h 2 2
  simp at h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
  norm_num at h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
  omega
  intro h
  intro a b
  have h_1 := h (2 * a)
  have h_2 := h b
  have h_3 := h (a + b)
  cases' h_1 with h_1 h_1 <;> cases' h_2 with h_2 h_2
  \n <;> cases' h_3 with h_3 h_3 <;> simp_all
  <;> omega
```

Table 13: Proof of `imo_2019_p1` in MiniF2f generated by our tactic-reward-trained model.

Consider a function $f : \mathbb{Z} \to \mathbb{Z}$ satisfying

$$\forall a, b \in \mathbb{Z}, \qquad f(2a) + 2\,f(b) \;=\; f\big(f(a+b)\big).$$

The task is to prove that necessarily one of the following holds:

(i) $\forall z \in \mathbb{Z},\ f(z) = 0$, or
(ii) $\exists c \in \mathbb{Z},\ \forall z \in \mathbb{Z},\ f(z) = 2z + c$.

Our model first introduced the assumption

$$h : \ \forall a, b \in \mathbb{Z},\ f(2a) + 2\,f(b) = f\big(f(a+b)\big),$$

and then instantiated it at several concrete pairs to create hypotheses h_i (e.g., $h_1 := h(0,0)$, $h_2 := h(0,1)$, ...). After some local simplification steps (e.g., `simp`, `norm_num`), it attempted to close the goal using the `omega` tactic, a decision procedure for Presburger arithmetic (linear integer arithmetic).

However, the `omega` call produced the first Lean error. While our method correctly assigns the d_2 penalty to this failing `omega` tactic under first-error propagation, it does not penalize the preceding tactics (`intro`, `have`, `simp`) because they elaborate successfully and thus appear locally valid. In other words, although introducing h and instantiating h_i is not logically incorrect, this route is strategically unproductive for this problem: the remaining goal still involves quantifiers, disjunction, and

an uninterpreted function f, which lie outside `omega`'s theory. Consequently, our current scheme only punishes the terminal failing step and fails to capture that the earlier (locally successful) steps did not make meaningful progress toward solving the global goal.

I LARGE LANGUAGE MODEL USAGE

In preparing this manuscript, we made limited use of large language models strictly for writing assistance. Specifically, we used ChatGPT-5 and Gemini-2.5 to improve grammar, enhance clarity of expression, and polish the overall presentation.