Rewarding the Unlikely: Lifting GRPO Beyond Distribution Sharpening

Anonymous ACL submission

Abstract

Reinforcement learning is emerging as a primary driver for improving language model reasoning capabilities. A fundamental question is whether current reinforcement learning algorithms—such as Group Relative Policy Optimization (GRPO), the *de facto* standard algorithm used to improve language model reasoning—merely sharpen the base model's distribution around problems it can already solve. We investigate this question in the context of formal theorem proving, which has access to a perfect verifier. We identify a degenerate *rank bias* in GRPO in which highly probable trajectories are reinforced and rare ones are neglected. This results in distribution sharpening: the model can solve some problems with fewer samples, but underperforms simply sampling more solutions from the original model. To overcome GRPO's rank bias we introduce *unlikeliness reward*, a simple method for explicitly up-weighting rare but correct solutions. We show that unlikeliness reward mitigates rank bias and improves pass@N across a large range of N in both synthetic and real theorem proving settings. We also uncover an unexpected link between rank bias and a seemingly mundane hyperparameter—the number of updates per batch—that leads to a second, complementary mitigation. We combine our insights into a revised GRPO training recipe for formal theorem proving, yielding an open pipeline that achieves competitive performance to DeepSeek-Prover-V1.5-RL on the miniF2F-test benchmark.

1 Introduction

Reinforcement learning (RL) has recently emerged as a powerful framework for enhancing the reasoning capabilities of large language models (LLMs). In domains such as mathematics and code generation, RL has been applied at scale to elicit complex reasoning behaviors using only problem instances and their corresponding outcome rewards (DeepSeek-AI et al., 2025; Yu et al., 2025).

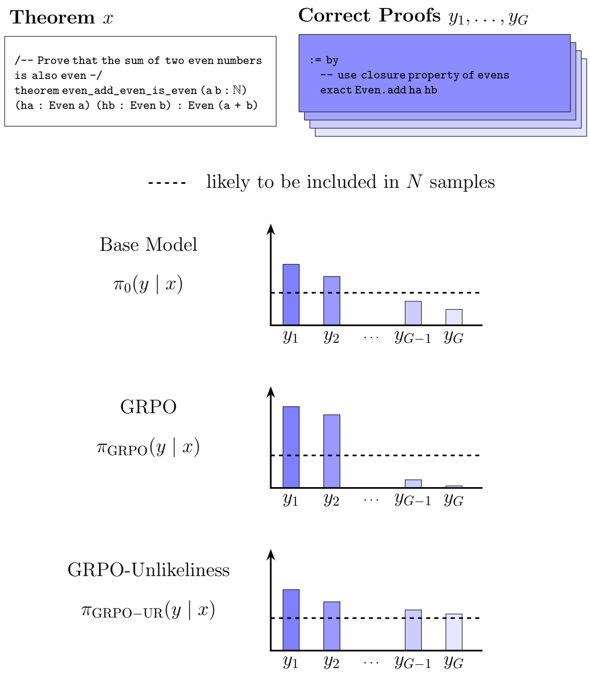


Figure 1: We identify a *rank bias* in GRPO in which model updates only reinforce already probable solutions and fail to surface new ones. This sharpens the distribution and impairs pass@N performance for large N. Our *unlikeliness reward* addresses rank bias by explicitly encouraging uplifting low-probability correct solutions.

Formal theorem proving is a particularly attractive domain for studying LLM reasoning. Formal systems such as Lean and Isabelle (de Moura et al., 2015; Paulson, 1994) can verify mathematical proofs step-by-step, ensuring that models are only rewarded for fully correct solutions. Since verification is fully automated and immune to spurious solutions, formal mathematics serves as an ideal testbed for reinforcement learning algorithms.

An important open challenge is designing reinforcement learning algorithms that do more than "sharpen the distribution"—that is, we want the RL-trained model to solve problems that cannot be solved by simply sampling more from the original model. Consistent with the findings of Yue et al. (2025), our initial experiments identify this as a key limitation of existing RL recipes based on Group Relative Policy Optimization (GRPO) (Shao et al., 2024), the de facto standard algorithm for improving LLM reasoning. While GRPO improves single-sample accuracy, it often fails to improve and can even impair pass@N metrics at larger N in our theorem proving setting (Figure 2). This is a significant limitation in domains with a perfect verifier, such as formal mathematics, since these domains naturally lend themselves to sampling and verifying many candidates at test time.

We argue that improving pass@N performance requires specifically increasing the probability of *low probability correct responses* under the model. We construct a toy model to demonstrate this phenomenon, and reveal empirically that GRPO suffers from *rank bias*: a tendency to reinforce already high-likelihood responses while neglecting the long tail of rare but correct ones. This reduces sample diversity and degrades multi-sample performance over time. To address this, we introduce **Unlikeliness Reward**, which up-weights correct outputs that are less likely than others. Doing so dramatically changes how GRPO learns from less likely trajectories, translating to more output diversity and higher pass@N across a range of N values.

Furthermore, we uncover an unexpected link between GRPO's distribution sharpening and a seemingly mundane hyperparameter: the number of PPO epochs per batch. Increasing the number of epochs adds extra gradient steps on low-likelihood sequences after the high-likelihood ones saturate, amplifying training signal for unlikely solutions. Tuning this often-ignored hyperparameter is a complementary approach to the unlikeliness reward, and offers insight into the optimization dynamics that can lead to distribution sharpening.

We demonstrate that our revised training recipe substantially improves pass@N metrics across a range of values for N, while also substantially outperforming standard expert iteration. We combine unlikeliness reward and our insights into PPO epochs into a full recipe for reinforcement learning in formal theorem proving. We apply our recipe to theorem proving in Lean, resulting in a fully open pipeline that achieves competitive performance with DeepSeek-Prover-V1.5-RL on the miniF2F-test benchmark.

2 Problem Setup

We study the problem of training a language model for formal theorem proving, where the goal is to generate valid proofs of theorems in a proof assistant. We use Lean (de Moura et al., 2015), a proof assistant based on dependent type theory that supports the construction and verification of mathematical proofs. Lean has recently attracted interest in the AI and mathematics communities (e.g., Yang et al. (2024); Tao (2025)).

Let $\mathcal{D} = \{x_i\}_{i=1}^M$ be a dataset of theorem statements. Each statement consists of a natural language description and a formal statement expressing the theorem in Lean. Let R denote the verifier, which also functions as the reward function. Given a theorem statement x and a candidate proof y, the Lean verifier returns a binary reward indicating whether y constitutes a successful proof of x:

$$R(x, y) = \mathbb{1}\{y \text{ proves } x\}.$$

We assume access to an initial prover model $\pi_{\text{base}}(y \mid x)$, a large language model (LLM) with some basic capability to generate proofs. Given a theorem statement x, the model samples a completion y that attempts to prove the statement. Our goal is to fine-tune this model to improve its proof success rate, using problem instances from $\mathcal{D}$ and the reward signal provided by R.

2.1 Evaluation Metric

To evaluate the prover's performance, we use the **pass@N** metric, which measures the probability that at least one of N independently sampled proof attempts succeeds. This metric is widely adopted in prior work due to its simplicity and close alignment with the practical use case of generating and verifying many proof attempts per theorem to find at least one that succeeds.

Let $x \in \mathcal{D}_{\text{test}}$ be a theorem, and let $\{y_j\}_{j=1}^N \sim \pi_\theta(\cdot \mid x)$ denote N independent samples drawn from the model. The empirical pass@N metric for a single theorem is defined as:

$$\text{pass@}N(x; \pi_\theta) = \mathbb{1}\left\{\max_{1 \le j \le N} R(x, y_j) = 1\right\}$$

The average pass@N score on a test set $\mathcal{D}_{\text{test}} = \{x_i\}_{i=1}^M$ is the average over individual theorems:

$$\text{pass@}N(\pi_\theta) = \frac{1}{M} \sum_{i=1}^{M} \text{pass@}N(x_i; \pi_\theta)$$

In the context of reinforcement learning, a high pass@N also indicates that we are likely to receive a positive reward signal when sampling N completions per problem.

2.2 Reinforcement Learning

We use Group Relative Policy Optimization (GRPO) as the foundation of our reinforcement learning experiments. GRPO was introduced by (Shao et al., 2024) and has been successfully applied to train models such as DeepSeek-R1 and DeepSeek-Prover-V1.5-RL (DeepSeek-AI et al., 2025; Xin et al., 2024), showing strong performance in both informal and formal settings.

GRPO is an extension of Proximal Policy Optimization (PPO) (Schulman et al., 2017) that omits the critic model. For each question x, GRPO samples a group of outputs $\{y_1, \ldots, y_G\} \sim \pi_{\theta_{old}}(y \mid x)$ from the current policy and maximizes the following objective:

$$
\begin{aligned}
&\mathcal{J}_{GRPO}(\theta) \\
&= \frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(y_i \mid x)}{\pi_{\theta_{old}}(y_i \mid x)} A_i,\right. \\
&\qquad \left.\text{clip}\left(\frac{\pi_\theta(y_i \mid x)}{\pi_{\theta_{old}}(y_i \mid x)}, 1-\epsilon, 1+\epsilon\right) A_i\right) \\
&\quad - \beta_{\text{KL}} \mathcal{D}_{\text{KL}}[\pi_\theta \parallel \pi_{\text{ref}}]
\end{aligned}
$$

GRPO differs from PPO in how it computes the advantages A_i. Instead of subtracting a baseline predicted by the critic model, GRPO normalizes rewards within the group of samples. Let $r_i = R(x, y_i)$, then the advantages are computed as:

$$
A_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\})}
$$

Note that when all or none of the samples solve the problem, $A_i = 0$ for all i and there is no gradient with respect to model parameters θ (except for the KL term). To be more efficient with model updates, we implement a trick similar to Dynamic Sampling (Yu et al., 2025). We maintain a buffer of recent samples that have nonzero advantage and only perform model updates once the buffer reaches the target batch size.

3 Does GRPO Improve Pass@N?

We begin by investigating how GRPO behaves when applied to formal theorem proving. Our setup closely follows Xin et al. (2024) in terms of model choice and hyperparameter settings, though we curate our own dataset, as theirs has not been released.

3.1 Dataset

The Lean Workbook dataset is a large-scale collection of approximately 140K Lean 4 theorem statements that were auto-formalized from natural language math problems (Ying et al., 2024). Since unsolvable problems do not provide useful gradients during RL, we select a 10K subset of problems that were found to be solvable in Wu et al. (2024). These statements are still moderately challenging, as the solutions were discovered through an extremely compute-intensive search process. In addition, we also include the 244 problems from miniF2F-valid (Zheng et al., 2021).

From this combined dataset, we hold-out 200 theorems for validation, leaving 9.6K for training. Although miniF2F-test (Zheng et al., 2021) is a standard benchmark for theorem proving, we found high variance and inconsistent results on it when training at our scale, likely due to distribution shift and large difficulty gaps between problems. Thus, we primarily evaluate on our I.I.D. held-out set ($\mathcal{D}_{\text{val}}$) and only use miniF2F-test for our final large-scale experiments. We will refer to our training and validation sets as $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$, respectively.

3.2 Training

Our implementation of GRPO is built on the verl framework (Sheng et al., 2024), with modifications to support reward feedback from the Lean REPL. We use the Python wrapper for the Lean REPL released by Xin et al. (2024), which we found to be more robust than previous open-source alternatives. The base model is DeepSeek-Prover-V1.5-SFT, which has moderate theorem-proving capabilities (Xin et al., 2024). We adopt the hyperparameters reported in Xin et al. (2024) where available:

- Learning rate = 5e-6
- KL loss coefficient = 0.02
- Number of samples per problem = 32.

However, we found the original learning rate to be unstable and use a reduced value of 1e-6. Due to compute constraints, we only train for one epoch on $\mathcal{D}_{\text{train}}$ and truncate the response length to 512 tokens, which suffices for over 99.5% of samples.

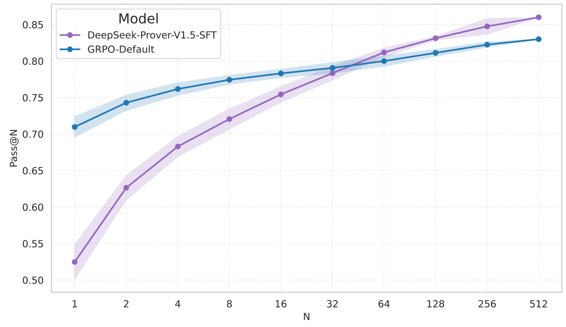


Figure 2: Finetuning DeepSeek-Prover-V1.5-SFT with GRPO, evaluated on $\mathcal{D}_{\text{val}}$. GRPO improves pass@N significantly for small N, but performs worse than the base model for large N. We aim to understand this behavior and develop methods to overcome it.

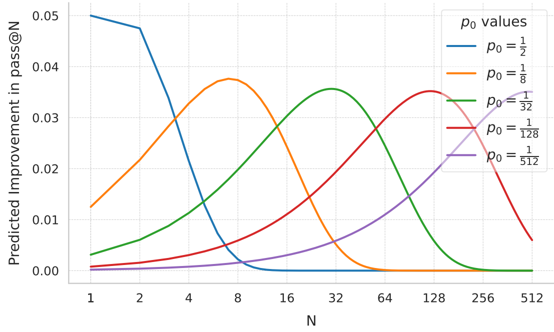


Figure 3: Improvement in expected pass@N assuming RL increases correct solution probabilities by a factor of $1+\epsilon$ with $\epsilon = 0.2$. Each curve corresponds to an initial $p_0 \in 1/2, 1/8, 1/32, 1/128, 1/512$.

3.3 GRPO Fails to Improve Pass@N

Figure 2 presents model performance on $\mathcal{D}_{\text{val}}$, evaluated up to pass@512. GRPO substantially boosts pass@1 to pass@16, but the improvement diminishes for larger N. This pattern suggests that GRPO is effective at increasing the likelihood of already probable correct solutions but fails to surface new ones into the high-probability set, which is consistent with the findings of Yue et al. (2025) and Shao et al. (2024). Note that this is not an inherent failure of RL—boosting single-sample accuracy increases expected reward, but the benefit for formal theorem proving is limited. Next, we consider if and how RL can improve pass@N at large N.

3.4 Can RL Optimize Pass@N?

In this section, we argue that improving pass@N for large N specifically requires RL to increase the probability of *low-probability correct solutions* under the model.

Suppose that the initial model π_0 has a probability p_0 to solve a problem x, i.e.,

$$\sum_{y \text{ s.t. } R(x,y)=1} \pi_0(y \mid x) = p_0.$$

The expected pass@N can then be expressed as:

$$\mathbb{E}[\text{pass@}N(\pi_0)] = 1 - (1 - p_0)^N.$$

Now, we consider how RL training affects p_0. The exact outcome of taking gradient steps against the GRPO objective is impossible to predict analytically, but we can make estimates by assuming that we maximize the objective. For simplicity, we only consider early training steps, so that $\pi_{\theta_{old}} \approx \pi_0$, and disregard the KL term. The simplified GRPO objective is:

$$\begin{aligned}&\mathcal{J}_{\text{GRPO}}(\theta)\\ &= \frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(y_i \mid x)}{\pi_0(y_i \mid x)} A_i,\right.\\ &\left.\text{clip}\left(\frac{\pi_\theta(y_i \mid x)}{\pi_0(y_i \mid x)}, 1-\epsilon, 1+\epsilon\right) A_i\right).\end{aligned}$$

We make the simplifying assumption that the probability of each positive sample y_+ with $A_i > 0$ can be optimized independently. In the GRPO objective, each sample stops contributing gradient once $\pi_\theta(y_+ \mid x)/\pi_0(y_+ \mid x) \geq 1+\epsilon$, thus we expect that the final ratio is close to the clipping bound:

$$\frac{\pi_{\text{RL}}(y_+ \mid x)}{\pi_0(y_+ \mid x)} \approx 1 + \epsilon.$$

We can then predict the accuracy of the trained model:

$$p_{\text{RL}} \approx (1+\epsilon)p_0$$

$$\mathbb{E}[\text{pass@}N(\pi_{\text{RL}})] \approx 1 - (1 - (1+\epsilon)p_0)^N.$$

Figure 3 plots the expected improvement in pass@N for different initial p_0. When p_0 is large, the marginal gain in pass@512 is small. Conversely, when p_0 is small, gains are negligible for pass@1. In general, we see that increasing pass@N requires the training algorithm to increase the probability of solutions with $p_0 \approx 1/N$. Thus, RL must specifically uplift the probability of *low-probability correct solutions* to achieve improvements in pass@N for large N.

3.5 Does GRPO Reinforce Unlikely Solutions?

The analysis above, and our empirical observation that GRPO is not increasing pass@N, together

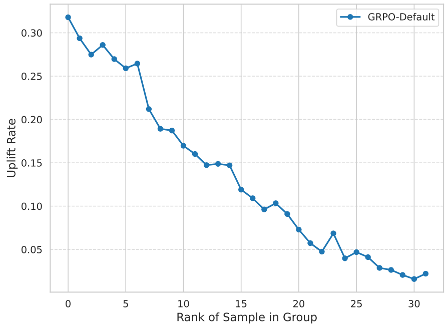


Figure 4: Uplift rate u_j as a function of rank j among positive samples. GRPO rarely increases the probability of lowest-ranked (i.e. rarest) correct samples.

suggest that GRPO may not be effectively uplifting low-probability correct solutions. To verify this, we examine training samples for the first 800 problems, computing their probabilities under the initial model and final GRPO-trained model.

Let x_i be the i-th training problem and $y_{i,j}$ be the j-th corresponding solution. We compute $\pi_0(y_{i,j} \mid x_i)$ and $\pi_{\text{GRPO}}(y_{i,j} \mid x_i)$ for all pairs. We are interested in whether $\pi_{\text{GRPO}}(y_{i,j} \mid x_i)/\pi_0(y_{i,j} \mid x_i) \approx 1+\epsilon$, especially when $\pi_0(y_{i,j} \mid x_i)$ is small.

We find that the raw probability ratios are highly variable, containing extreme outliers, and the scale of $\pi_0(y_{i,j} \mid x_i)$ also differs widely across problems. This makes it difficult to analyze the raw model probabilities directly. Instead, we use the rank of a sample within its group as a proxy for its probability and consider the simpler, binary metric of whether $\pi_{\text{GRPO}}(y_{i,j} \mid x_i)$ is greater than $\pi_0(y_{i,j} \mid x_i)$.

Formally, for each problem x_i, we sort the solutions $\{y_{i,1}, \ldots, y_{i,G}\}$ in descending order of $\pi_0(y_{i,j} \mid x_i)$ to obtain $\{\tilde{y}_{i,1}, \ldots, \tilde{y}_{i,G}\}$. We are interested in the relationship between the rank of a solution and how likely it is to be uplifted by GRPO. For each rank $j \in \{1, \ldots, G\}$, we compute the "uplift rate", averaging over positive samples:

$$u_j = \underset{i:\, R(x_i, \tilde{y}_{i,j})=1}{\text{mean}} \Big(\mathbb{1}\{\pi_{\text{GRPO}}(\tilde{y}_{i,j} \mid x_i) > \pi_0(\tilde{y}_{i,j} \mid x_i)\}\Big)$$

Figure 4 shows a clear positive correlation: GRPO is more likely to increase the probability of already high-probability correct solutions. In contrast, the low-probability positive samples – those most critical for improving pass@N at large N – are almost never uplifted. We confirm this behavior in a controlled toy environment (see Appendix A) and refer to this phenomenon as *rank bias*.

4 Improving GRPO for Multi-Sample Performance

While the GRPO objective itself does not inherently favor high-probability solutions, our earlier analysis revealed a clear empirical bias: low-probability correct solutions are rarely reinforced. This behavior is counterintuitive – when $\pi_0(y \mid x)$ is small, increasing the ratio $\pi_{\text{RL}}(y \mid x)/\pi_0(y \mid x)$ requires less absolute probability mass and contributes equally to the GRPO objective. In principle, this should make low-probability solutions more attractive to optimize. The observed rank bias is therefore not a feature of the GRPO loss but likely a consequence of the optimizer's biases.

In this section, we introduce the *unlikeliness reward* to directly counteract this implicit bias, with the goal of improving pass@N performance at large N. We also provide complementary analysis on the effect of certain hyperparameters on rank bias, which we later incorporate into our overall training recipe.

4.1 Unlikeliness Reward

To explicitly correct for rank bias, we propose the **unlikeliness reward** – a simple modification to the reward function that discourages reinforcing already high-probability solutions. For a group of samples $y_1, \ldots, y_G$, let $\text{rank}(y_i) \in \{1, 2, \ldots, G\}$ denote the rank of y_i under the current policy $\pi_{\theta_{old}}(y_i \mid x)$, with rank 0 corresponding to the highest-probability sample. We modify the reward to be

$$r_i = R(x, y_i)\left(1 - \beta_{\text{rank}}\frac{G - \text{rank}(y_i)}{G}\right).$$

A multiplicative penalty is applied to higher-probability solutions, increasing the relative advantage of rarer positive samples. Incorrect solutions remain unaffected, receiving $r_i = 0$ regardless of rank. The coefficient β_{rank} controls the strength of this perturbation; we fix $\beta_{\text{rank}} = 0.25$ in our experiments.

Moreover, we continue to skip all samples that have zero advantage *before* the perturbation. This ensures that no batch is dominated solely by the unlikeliness reward, and $R(x, y_i)$ still determines the direction of optimization for each sample.

4.2 Effects of PPO Epochs

In addition to perturbing rewards, we find that increasing the number of optimization steps per sample (**ppo-epochs**) also mitigates rank bias. Standard implementations of PPO and GRPO typically use a single optimization step per batch (Sun, 2024; Sheng et al., 2024; Yu et al., 2025), which we found to produce biased updates. When taking multiple gradient steps, the initial steps may push high-rank solutions beyond the clipping threshold, so that subsequent steps are forced to focus on low-rank samples that are still unclipped. In this way, increasing ppo-epochs indirectly amplifies learning signal for low-rank samples.

However, increasing ppo-epochs makes training substantially slower (Appendix B.1) and potentially unstable. Thus, we prefer the unlikeliness reward as the more direct and efficient solution to address rank bias.

5 Experiments

For our main experiments, we use $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ for training and evaluation. We compare several GRPO variants with different hyperparameter settings, summarized in Table 1. We increase the KL penalty because we found that it helps prevent deteriorating pass@N, but this change alone was not enough to improve pass@N substantially (discussed in Appendix D). All unlisted hyperparameters are kept the same.

Model	K	β_{KL}	β_{rank}
GRPO-Default	1	0.02	–
GRPO-Unlikeliness-1	1	0.10	0.25
GRPO-Unlikeliness-2	2	0.10	0.25
GRPO-Epochs-2	2	0.10	–
GRPO-Epochs-3	3	0.10	–

Table 1: Hyperparameter settings for GRPO variants in our experiments. K is the number of PPO epochs.

5.1 Results: Pass@N

Figure 5 shows the performance of GRPO variants evaluated on $\mathcal{D}_{\text{val}}$. Introducing the unlikeliness reward leads to substantial improvements in pass@N at large N, with a minor tradeoff in pass@1 and pass@2. Interestingly, increasing PPO epochs also leads to improvements, consistent with our analysis in Section 4.2. However, increasing PPO epochs leads to a significant increase in training time (Appendix B.1).

We also track the cumulative accuracy of the 32 samples generated per problem during training, including the baseline performance of a static model with no updates. Table 2 reports the number of problems solved by each variant. All GRPO variants outperform the static model, with GRPO-Unlikeliness-2 solving the most problems. Since training runs for only one epoch, each example is effectively unseen at the time of sampling, indicating generalization within the epoch.

Model	Solved	Δ Static
Static (V1.5-SFT)	7707 / 9600	–
GRPO-Default	7860 / 9600	+153
GRPO-Epochs-2	8008 / 9600	+301
GRPO-Epochs-3	8006 / 9600	+299
GRPO-Unlikeliness-1	8023 / 9600	+316
GRPO-Unlikeliness-2	**8065 / 9600**	**+358**

Table 2: Number of training problems solved during one epoch on $\mathcal{D}_{\text{train}}$. GRPO variants improve over the static model, with GRPO-Unlikeliness-2 achieving the largest gain.

5.2 Analysis: Rank Bias

To assess whether the proposed methods mitigate rank bias, we repeat the analysis from Section 3.5 by computing the u_j metrics over the training samples for each GRPO variant. The results, shown in Figure 6, indicate substantial changes in GRPO's behavior. GRPO-Unlikeliness-2 reverses the original pattern and is more likely to reinforce low-probability solutions. We also show that unlikeliness reward mitigates rank bias in our controlled environment (see Appendix A.4).

In GRPO-Epochs-2 and GRPO-Epochs-3, the bias remains, but the overall strength of reinforcement is increased so that low-probability solutions are also sufficiently uplifted.

5.3 Analysis: Sample Diversity

Throughout training, we track the number of unique proofs generated per step, shown in Figure 7. GRPO-Unlikeliness-2 exhibits unique dynamics where diversity initially drops but later recovers, unlike other variants where diversity declines monotonically. This may reflect a self-correcting mechanism: initially dominant solutions are penalized, allowing low-probability correct solutions to resurface. This continuous rebalancing helps preserve a broad distribution of strategies throughout training.

We also observe that higher PPO epochs consistently increases sample diversity, up to ppo-epochs $= 4$ where training becomes unstable. While this may seem counterintuitive – since more optimiza-

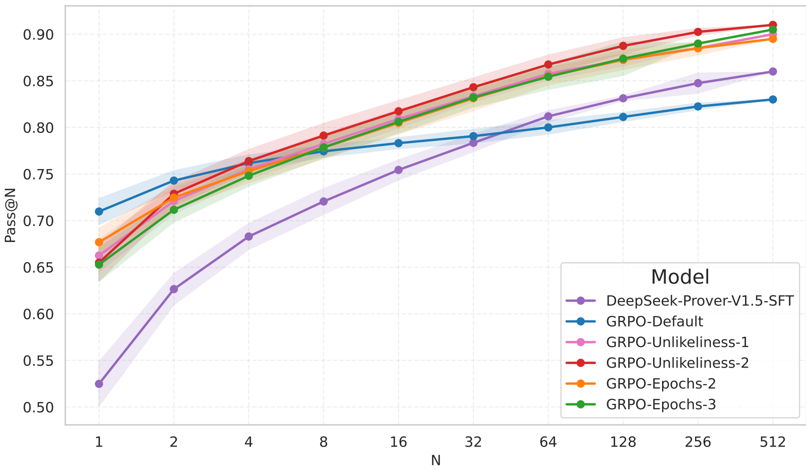


Figure 5: Performance of GRPO variants on $\mathcal{D}_{\text{val}}$. Both the unlikeliness reward and additional PPO epochs improve pass@N. Appendix C details how we compute these metrics.

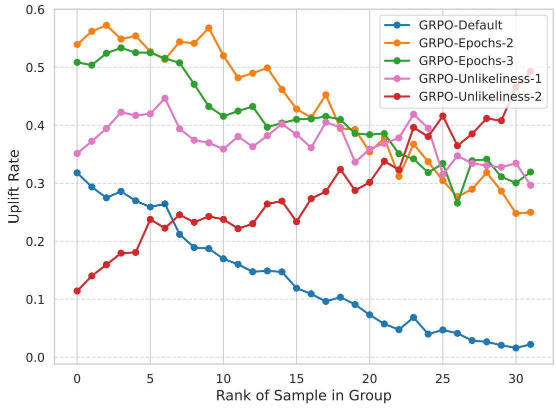


Figure 6: Uplift rate u_j as a function of rank j for GRPO variants. The proposed methods improve the rate of reinforcing low-probability correct solutions.

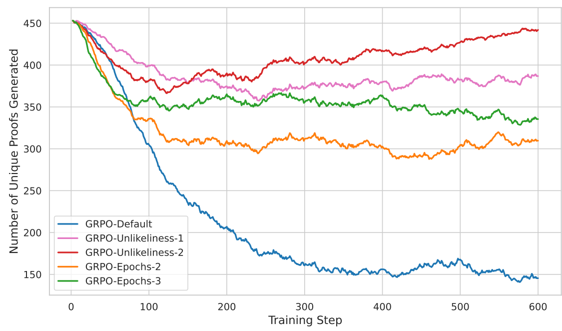


Figure 7: Number of unique proofs generated at each training step (smoothed with EMA). Unlikeliness reward significantly improves sample diversity during training.

tion steps deviate the model further from its initial distribution – it aligns with our earlier analysis. Higher PPO epochs indirectly amplifies rare solutions, thereby mitigating the sharpening effect typically caused by GRPO updates.

5.4 Putting It All Together

Finally, we evaluate **GRPO-Unlikeliness-2** in a large-scale experiment. We train the model on a dataset of 11k theorems, a larger and more challenging subset of Lean-Workbook that was solved and released by Lin et al. (2025b), making sure to exclude theorems in $\mathcal{D}_{\text{val}}$. We evaluate the resulting model on MiniF2F-test (Zheng et al., 2021), a widely recognized benchmark for neural theorem proving, as well as $\mathcal{D}_{\text{val}}$. As reported in Table 3, **GRPO-Unlikeliness-2** achieves competitive results compared to DeepSeek-Prover-V1.5-RL (Xin et al., 2024) on both datasets.

6 Related Work

Automated Theorem Proving: Polu and Sutskever (2020) pioneered transformer-based theorem provers that interact with proof assistants like Lean or Isabelle (de Moura et al., 2015; Paulson, 1994). Subsequent work has developed state-tactic models (Polu et al., 2022; Wu et al., 2024; Xin et al., 2025) that generate one proof step at a time and full-proof models (Xin et al., 2024; Lin et al., 2025b) that produce complete proofs autoregressively, reducing interaction overhead.

Recent work has explored various directions in LLM-based theorem proving. Lample et al. (2022), Xin et al. (2024), and Xin et al. (2025) explore the application of inference-time algorithms for proof discovery. Jiang et al. (2023) and Lin et al. (2025a) use informal reasoning to guide formal proofs by integrating LLMs capable of reasoning in natural

Model	pass@32	pass@128
	MiniF2F-test	
V1.5-SFT	$47.1 \pm 0.6\%$	$49.2 \pm 0.6\%$
V1.5-RL	$49.2 \pm 0.6\%$	$51.2 \pm 0.3\%$
Ours	$48.8 \pm 0.7\%$	$50.6 \pm 0.5\%$
	$\mathcal{D}_{\text{val}}$	
V1.5-SFT	$78.3 \pm 0.9\%$	$83.1 \pm 0.2\%$
V1.5-RL	$84.8 \pm 0.9\%$	$87.5 \pm 0.7\%$
Ours	$84.3 \pm 0.9\%$	$88.8 \pm 0.9\%$

Table 3: pass@N performance of our model compared to DeepSeek-Prover-V1.5-SFT and -RL from Xin et al. (2024) on MiniF2F-test and $\mathcal{D}_{\text{val}}$. Our model achieves competitive performance with DeepSeek-Prover-V1.5-RL while being fully open.

language. Hu et al. (2024) investigates training models that can incorporate novel context at test time. Our work is mainly focused on the post-training of theorem provers using reinforcement learning, which we detail next.

Expert Iteration for Theorem Proving: Expert iteration alternates between search and learning (Anthony et al., 2017), and was first applied to theorem proving by Polu et al. (2022). It has since become the dominant paradigm, appearing in recent work like Wu et al. (2024), Xin et al. (2025), and Lin et al. (2025b). Xin et al. (2025) explores the viability of best-first search for data collection, while Wu et al. (2024) and Lin et al. (2025b) achieve state-of-the-art performance at the time by performing large-scale expert iteration on autoformalized theorem statements.

RL for Theorem Proving: Compared to expert iteration, the use of more general RL algorithms is relatively underexplored. A notable exception is Xin et al. (2024), which showed GRPO can enhance a SFT model using only additional theorem statements and the verifier reward. In the low-data setting, Gloeckle et al. (2024) successfully trained a strong theorem prover by adapting the AlphaZero algorithm (Silver et al., 2017) to proof trees. Xin et al. (2025) used direct preference optimization (Rafailov et al., 2023) in their pipeline, but only for the minor role of training against proof steps that cause immediate errors.

More recent work has begun adapting techniques from OpenAI o1 (OpenAI et al., 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025) to train reasoning models for theorem proving (Wang et al., 2025; Ren et al., 2025; Zhang et al., 2025). These works have achieved state-of-the-art performance by building models that can engage in long chain-of-thought style reasoning, either calling formal proof models as subroutines (Ren et al., 2025) or devising hierarchical strategies to break down the problem (Wang et al., 2025).

RL for Multi-Sample Performance: Several existing works specifically investigate the issue of RL's pass@N performance. Yue et al. (2025) argues that instead of learning novel capabilities, RL with verifier reward mainly concentrates the model's outputs around correct answers already present in the base model's samples. Their experiments also show an improvement in pass@ small N and deterioration at large N. Chow et al. (2024) and Tang et al. (2025) consider novel RL formulations that explicitly optimize for best-of-N performance. They derive BoN-aware RL algorithms and demonstrate improved performance, but still consider a smaller range of N (pass@32) than is typical in formal theorem proving. In the expert iteration setting, Dang et al. (2025) identifies that pass@N deteriorates due to diversity collapse and shows that interpolating model weights with an early checkpoint mitigates this issue.

Compared to these previous works, we are the first to attribute RL's poor multi-sample performance to an inability to reinforce low-probability samples. We also provide a simple and direct solution to address this issue and improve pass@N performance.

7 Conclusion

We investigated GRPO's poor multi-sample performance in the setting of formal theorem proving, theorizing a connection between degraded pass@N at large N and the failure to reinforce low-probability solutions. Our analysis revealed an implicit bias in GRPO: it preferentially reinforces already high-probability sequences while largely ignoring rare but correct ones. To address this, we introduced the *unlikeliness reward*, a simple yet effective modification that directly shifts reinforcement toward rare samples. Our experiments confirm that the unlikeliness reward enables GRPO to make significant gains in pass@N at large N and drastically improves sample diversity compared to existing methods. Using our revised recipe, we train a model that is competitive with DeepSeek-Prover-V1.5-RL and release our implementation publicly.

Limitations

While we offer a lightweight solution for improving GRPO's multi-sample performance, future work could explore other strategies for uniformly reinforcing correct samples or for directly optimizing performance under specific inference-time algorithms. In particular, developing inference-aware reinforcement learning algorithms that are efficient to train remains an open direction.

Moreover, recent applications of RL have shifted toward the reasoning paradigm, where models generate long reasoning paths often involving behaviors such as planning, backtracking, and self-critique. In these settings, the behavior of algorithms like GRPO may differ qualitatively due to the increased diversity and complexity of possible reasoning paths. We leave as future work to determine whether methods that amplify rare but correct solutions can similarly enhance exploration and generalization in reasoning models.

References

Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. *Preprint*, arXiv:1705.08439.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Sridhar Thiagarajan, Craig Boutilier, Rishabh Agarwal, Aviral Kumar, and Aleksandra Faust. 2024. Inference-aware fine-tuning for best-of-n sampling in large language models. *Preprint*, arXiv:2412.15287.

Xingyu Dang, Christina Baek, Kaiyue Wen, Zico Kolter, and Aditi Raghunathan. 2025. Weight ensembling improves reasoning in language models. *Preprint*, arXiv:2504.10478.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The lean theorem prover (system description). In *CADE*.

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, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Fabian Gloeckle, Jannis Limperg, Gabriel Synnaeve, and Amaury Hayat. 2024. ABEL: Sample efficient online reinforcement learning for neural theorem proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Jiewen Hu, Thomas Zhu, and Sean Welleck. 2024. minictx: Neural theorem proving with (long-) contexts. *arXiv preprint arXiv:2408.03350*.

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. *Preprint*, arXiv:2210.12283.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. 2022. Hypertree proof search for neural theorem proving. *Preprint*, arXiv:2205.11491.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. 2025a. Lean-star: Learning to interleave thinking and proving. *Preprint*, arXiv: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. 2025b. Goedel-prover: A frontier model for open-source automated theorem proving. *Preprint*, arXiv:2502.07640.

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, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Lawrence C. Paulson. 1994. *Isabelle: A Generic Theorem Prover*. Springer Verlag.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2022. Formal mathematics statement curriculum learning. *Preprint*, arXiv:2202.01344.

Stanislas Polu and Ilya Sutskever. 2020. Generative language modeling for automated theorem proving. *Preprint*, arXiv:2009.03393.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Preprint*, arXiv:1712.01815.

Zhiqing Sun. 2024. Gpt-accelera: Simple and efficient pytorch-native transformer training and inference (batched). `https://github.com/Edward-Sun/gpt-accelera`.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. 2025. Optimizing language models for inference time objectives using reinforcement learning. *Preprint*, arXiv:2503.19595.

Terence Tao. 2025. Machine-assisted proof. *Notices of the American Mathematical Society*, 72(1):6–15.

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, and 21 others. 2025. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *Preprint*, arXiv:2504.11354.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *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. *Preprint*, arXiv:2408.08152.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. 2025. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *Preprint*, arXiv:2502.03438.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2024. Formal mathematical reasoning: A new frontier in ai. *Preprint*, arXiv:2412.16075.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Preprint*, arXiv:2406.03847.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, 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, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Preprint*, arXiv:2504.13837.

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. 2025. Leanabell-prover: Post-training scaling in formal reasoning. *Preprint*, arXiv:2504.06122.

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 Toy Environment

After observing that GRPO failed to improve pass@N metrics, we constructed a simplified toy environment to isolate the issue and efficiently test potential solutions. This appendix details the design of the environment and presents our experimental results within it.

A.1 Environment Design

We design a minimalistic toy environment for rapid experimentation. The environment is fully observable, with state space $\mathcal{S} = \mathbb{R}^{10}$ and discrete action space $\mathcal{A} = \{1, \ldots, 128\}$. Each action $a \in \mathcal{A}$ is associated with a fixed, randomly initialized but hidden vector $v_a \in \mathbb{R}^{10}$.

The binary reward function $R_\tau : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$ is defined as:

$$R_\tau(s, a) = \mathbb{1}\{s^\top v_a \geq \tau\}$$

Here, τ is a threshold controlling environment difficulty. Higher τ values restrict the reward to fewer actions, thus increasing difficulty. We fix $\tau =$

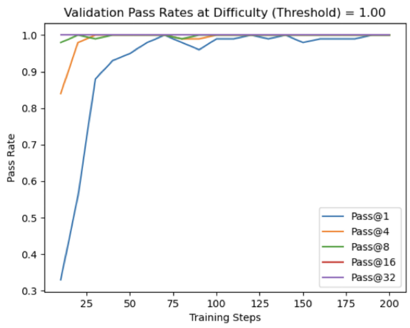
1.0 during training but vary τ during evaluation to simulate different difficulty levels.

A.2 Policy Model

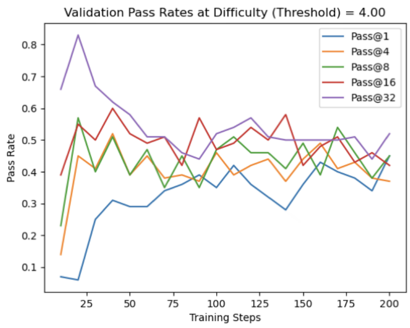
The policy model $\pi_\theta(a \mid s)$ is a simple two-layer multilayer perceptron (MLP) mapping state s to a probability distribution over actions in $\mathcal{A}$.

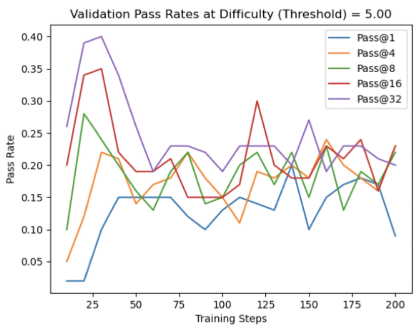
A.3 GRPO Training and Diagnosis

We train the model using GRPO for 200 steps and evaluate pass@N metrics at $N \in \{1, 4, 8, 16, 32\}$. Initial evaluations at training difficulty $\tau = 1.0$ suggest GRPO improves pass rates across all N:

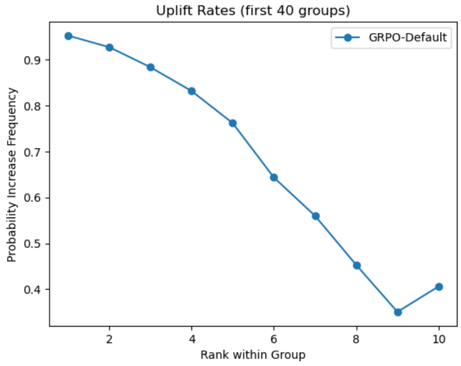


However, evaluations at increased difficulties ($\tau = 4.0$ and $\tau = 5.0$) reveal pass@32 deteriorates over training, aligning with observations in the original setting:



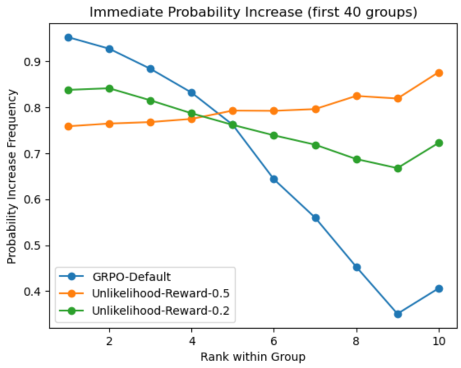


Analyzing uplift rate metrics (Section 3.5), we identify a rank bias in GRPO, showing preferential reinforcement of already high-probability solutions:



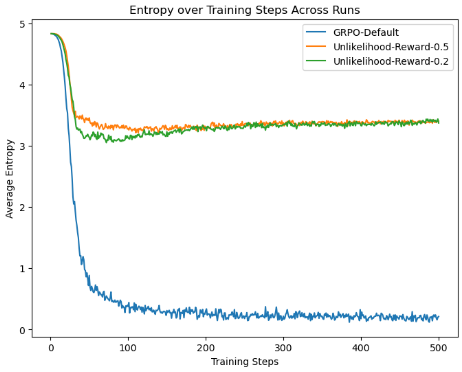
A.4 Unlikeliness Reward

We investigate the impact of unlikeliness reward within this toy environment. It effectively neutralizes the rank bias, making the uplift rates notably more uniform:



Consequently, the unlikeliness reward significantly improves pass@32 performance in the difficult setting $\tau = 5.0$, contrasting sharply with default GRPO, whose pass@32 performance declines to near chance levels:

Additionally, incorporating the unlikeliness reward substantially increases the entropy of the predicted action distribution:



B Training Setup

The main experiments in Section 5 are conducted on 4 NVIDIA L40S GPUs, with 500GB of RAM and 48–64 CPUs allocated for running parallel instances of the Lean REPL.

B.1 Training Time

All training runs in the main experiment complete within 36 hours. Each training step primarily consists of three stages: sequence generation, proof verification, and policy model updates. The generation and verification stages are shared across all methods and take approximately 120 seconds per batch (16 problems × 32 attempts). The duration of the policy update step depends on the number of PPO epochs, as shown below:

PPO Epochs	Policy Update Time (s)
1	≈ 70
2	≈ 140
3	≈ 210

C Evaluation Metrics

We begin by selecting a maximum sample size $N_{\max}$ (512 in our experiments) and generate $N_{\max}$ responses for each problem. To compute pass@n, we divide the responses for each problem into $N_{\max}/n$ chunks and assign each chunk a binary reward indicating whether any proof within it is valid. The i-th trial of pass@n is then computed by averaging the binary rewards across the i-th chunk of all problems. We report the mean and standard deviation across trials. Note that for pass@512, there is only a single trial, so we omit the standard deviation in our plots.

D Effects of KL Penalty

Recent results have shown that the pass rates of theorem prover models can continue to improve with increased sampling, up to hundreds of thousands of passes (Lin et al., 2025b). This suggests that the distribution of the base model is highly diverse and crucial to preserve during fine-tuning. Prior work addressed this in the SFT setting by ensembling fine-tuned model weights with the original (Dang et al., 2025). Since GRPO already has a regularization mechanism through the KL penalty, we simply increase the KL loss coefficient to 0.1 to better preserve the original distribution.

We find that this change by itself did not improve pass@N, likely because the updates still fail to uplift low-rank samples. Thus, we treat KL regularization as a supporting modification rather than a solution in itself.