

Learning to Explore the Crux in Formal Theorem Proving

Linhan Li^{1,2}, Wenyao Cui¹, Gongduo Zhang², Lihong Gu²,
Lin Zhou², Jianyun Shang¹, Huaping Zhang^{1,*}

¹Beijing Institute of Technology, ²Ant Group

Correspondence Author

Abstract

Recent advancements in the application of Large Language Models (LLMs) to formal theorem proving in Lean have unveiled new prospects in complex logical reasoning scenarios. Acknowledging that the difficulty across reasoning steps in a formal theorem proof is not uniform, prior proof search algorithms have not systematically investigated which proof steps are more crucial to explore. This paper introduces **Entropy-Based Best-First Search (EBFS)**, an entropy-inspired heuristic tree search algorithm that enhances the efficiency of proof generation models to explore proof trees, thereby increasing the probability of finding the correct proof. We also introduce a reinforcement fine-tuning approach specifically for proof tree search. Using EBFS during the generation stage further boosts the model’s performance in proof generation. Evaluations demonstrate that our fine-tuned EBFS model achieves consistent improvements over the baseline models. Under the same sample budget, it outperforms whole proof generation by 3.2% on the ProofNet testset, with further gains of 1.9% on MiniF2F and 1.5% on Prover-Bench.

1 Introduction

A proof is a series of logical reasoning processes based on existing axioms, theorems, definitions, and conditions, aimed at demonstrating the correctness of a proposition. It serves as foundational work in mathematical research. Formal proofs can be generated and verified by computers, ensuring completeness and unambiguous correctness of the proof. Recent advances in large language models have introduced new research directions in tree-search algorithms for formal proofs (Yang et al., 2024). By replacing components of traditional formal proof systems with large language models (LLMs), more flexible proof-assistance func-

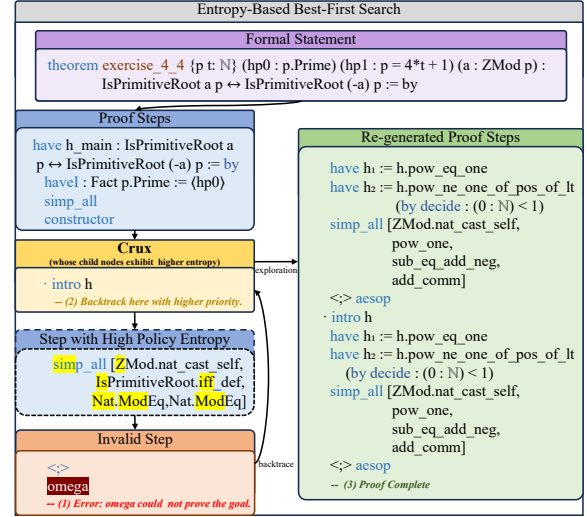


Figure 1: An example of the EBFS search process, where (1) the LLM fails in some proof attempts, splitting the existing valid steps; (2) calculating the entropy for all valid steps, and then sorting and selecting the best node to traceback and expand; (3) backtracking the node and making another attempt to complete the proof. The high-entropy tokens are marked in yellow.

tionalties can be achieved (Song et al., 2023), increasing the potential to solve more complex and challenging problems. Meanwhile, proof assistants such as Isabelle (Paulson, 1994), HOL Light (Harrison, 1996), Coq (Coq, 1996), and Lean (Moura and Ullrich, 2021) also provide feedback signals for LLM-generated proofs, helping LLMs continuously refine their proof strategies.

Previous research has primarily investigated whole proof generation methods and proof tree search algorithms separately. Whole proof generation aligns more closely with the output format of current reasoning language models and can yield favorable results under limited sampling budgets. However, it faces challenges in scalability when larger sampling budgets are involved, as restarting a proof from the beginning due to minor errors

is inherently inefficient. In contrast, proof tree search methods allow more fine-grained optimization of proof steps, yet require verification at each step, thereby demanding substantially larger sampling budgets. To address these limitations, we aim to integrate the strengths of both approaches by emulating the proof workflow of human experts, thereby enhancing the efficiency of formal proof generation.

We introduce Entropy-based Best-First Search, a novel approach that leverages the policy entropy over tokens generated by LLMs to prioritize exploration and expansion of proof steps, as shown in Figure 1. EBFS is designed to mirror the way human experts reason. When experts encounter a proof error, they usually re-evaluate the current step or backtrack to the point where they felt the most uncertain. In the context of large language models, this uncertainty is measured through policy entropy.

A natural question arises: *does the root cause of a proof failure actually occur earlier than the point where Lean reports the error, and can token-level entropy reliably signal these critical steps?* To answer this, we conduct a large-scale empirical study (Appendix A.1) in which failed proofs are annotated by three independent LLM annotators. Under their majority-vote consensus, we find that **77.3%** of root causes occur before the Lean error position, and **63.5%** of root cause steps rank in the top 25% of their own proof by entropy peak count (Figure 2), providing strong evidence that entropy peaks are a highly effective heuristic for identifying critical steps.

As for proof generation in Lean, EBFS first truncates the proof by the tactics, calculates the policy entropy of the probability distribution of each token, and then obtains a sequence consisting of the entropy values of all tokens in each proof step. By analyzing the token entropy sequences during proof generation, EBFS offers a straightforward yet effective method for assessing the relative influence of intermediate steps in the reasoning process. Nodes are identified as cruxes if their children contain a high proportion of high-entropy tokens or fail verification in the first attempt. These crucial steps mark points where the model’s uncertainty signals a high potential for discovering successful new proof branches.

In order to make EBFS more compatible with reinforcement fine-tuning, we design a trajectory sampling method for the tree search algorithm, en-

suring that the rollout samples have a more reasonable granularity when performing reinforcement fine-tuning, avoiding rewarding or penalizing any controversial steps. Finally, we integrate EBFS with reinforcement fine-tuning, enabling the LLM to systematically improve its performance in overcoming bottlenecks encountered in critical proof steps. Our experimental results on MiniF2F (Zheng et al., 2022), ProofNet (Vishwakarma and Monticone, 2024) and ProverBench (Ren et al., 2025) demonstrate that EBFS improves proof pass rates over whole proof generation and MCTS-like method.

In summary, our contributions are as follows:

1. We introduce Entropy-based Best-First Search, a method that uses the policy entropy from LLMs to identify and prioritize the exploration of high-uncertainty proof steps, mimicking human backtracking.
2. We integrate EBFS with reinforcement learning and design a trajectory sampling technique that enables effective reinforcement fine-tuning on EBFS rollout data, allowing the LLM to learn systematically from its mistakes at crucial steps.
3. Experiments demonstrate that the combined EBFS and fine-tuning approach achieves higher proof pass rates on standard benchmarks than both whole-proof generation and Monte Carlo Tree Search-based methods.

2 Related Work

2.1 Proof Generation

Large language models have enabled proof generation as a language modeling task. Models such as Minerva (Lewkowycz et al., 2022), Llemma (Azerbayev et al., 2024), InternLM-Math (Ying et al., 2024), and DeepSeek-Math (Shao et al., 2024) are trained on mathematical corpora for natural-language proof generation. For formal proofs, GPT-f (Polu and Sutskever, 2020) pioneered step-wise tactic generation, followed by InternLM2.5-StepProver (Wu et al., 2024) and BFS-Prover (Xin et al., 2025b). Whole-proof generation methods include Baldur (First et al., 2023), DeepSeek-Prover-V1.5 (Xin et al., 2025a), and GoedelProver (Lin et al., 2025). LeanDojo (Yang et al., 2023) provides retrieval-augmented proving; LEGO-Prover (Wang et al., 2024) augments LLMs with a growing lemma

library; LeanAgent (Kumarappan et al., 2025) introduces lifelong learning for theorem proving. PACT (Han et al., 2022) decomposes proof generation into sub-tasks including next-lemma and masked-proof prediction. Recent work leverages CoT reasoning (Wei et al., 2022): DeepSeek-Prover-V2 (Ren et al., 2025) and Kimina-Prover (Wang et al., 2025b) generate longer reasoning chains, Seed-Prover (Chen et al., 2025b) iteratively refines proofs via compiler feedback, and multi-agent systems such as DSP+ (Cao et al., 2025), Seed-Prover 1.5 (Chen et al., 2025a), and MA-LoT (Wang et al., 2025c) collaborate to convert natural-language sub-goals into formal proofs.

2.2 Proof Tree Search

Proof tree search systematically explores proof paths via heuristic algorithms. Classical approaches include Aesop (Limperg and From, 2023), lazyCoP (Rawson and Reger, 2021), HTTPS (Lample et al., 2022), and ABEL (Gloeckle et al., 2024). DeepSeek-Prover-V1.5 (Xin et al., 2025a) proposed RMaxTS, which uses intrinsic rewards based on novel node discovery to address reward sparsity. HunyuanProver (Li et al., 2024) assigns importance-weighted sampling budgets via η MCTS. InternLM2.5-StepProver (Wu et al., 2024) combines BFS with a learned value model, while BFS-Prover (Xin et al., 2025b) uses length-normalized log-likelihood scoring. LeanProgress (George et al., 2025) predicts proof progress to guide search, and COPRA (Thakur et al., 2024) uses in-context learning with execution feedback for backtracking search.

2.3 Reinforcement Fine-Tuning

RFT research has focused on mitigating entropy collapse: Clip-Cov (Cui et al., 2025) suppresses logit changes from high covariances, AEPO (Dong et al., 2025) applies stop-gradient on high-entropy terms, AER (Zhang et al., 2025) modulates regularization dynamically, and EPO (Xu et al., 2025) enforces temporal entropy smoothing. Orthogonally, Agarwal et al. (2025) show that entropy minimization alone substantially improves LLM reasoning, suggesting token-level entropy is a fundamental signal for reasoning quality.

3 Method

In this section, we will describe the motivation and principles of the EBFS algorithm, as well as the

usage of EBFS for proof search in reinforcement fine-tuning.

3.1 Entropy-Based Best-First Search

When using LLM to generate proofs and construct proof trees, the EBFS algorithm can locate critical steps and allocate more search budget to resolve those steps, which closely resembles the process of human developers working with Lean, as shown in Figure 1.

A key insight motivating EBFS is that proof failures are often caused by early strategic decisions rather than local errors at the point where Lean reports the failure. Our empirical analysis confirms this: under the consensus annotation, 77.3% of root causes occur before the Lean error position, and root cause steps carry significantly more entropy peaks than other steps across every annotator (Figure 2), indicating that problematic steps tend to introduce uncertainty well before the failure surfaces. These findings underscore the need for a search algorithm that can identify and prioritize such critical steps *before* the error manifests, which is precisely what EBFS achieves through entropy-based scoring.

In terms of implementation, we adopted the format of whole proof generation to construct the proof tree. Specifically, the proof generated by the LLM is truncated at the first error step after verification by the Lean compiler, and the validated steps are segmented by tactics to build the nodes of the proof tree; the tactic-level segmentation reuses the open-source method from DeepSeek-Prover-V1.5 (Xin et al., 2025a). The EBFS algorithm then assigns higher scores to critical steps based on entropy, treating them as prioritized search targets.

Cruxes. The EBFS uses the policy entropy generated by the LLM during inference to determine whether the current proof step is critical. Specifically, for the i -th step of the proof, EBFS first calculates the entropy of each token to obtain a sequence of entropy values for the step:

$$e_{i,t} = - \sum_{v \in \mathcal{V}} \pi(v|q^i, o_{<t}^i) \log(\pi(v|q^i, o_{<t}^i)) \quad (1)$$

We then calculate the number of peaks M_i of the sequence $\mathbf{e}_i = [e_{i,t}]_{t=1}^n$.

$$M_i = \sum_{t=1}^n \mathbb{1}(e_{i,t} \geq \lambda) \quad (2)$$

Here, λ is a hyperparameter.

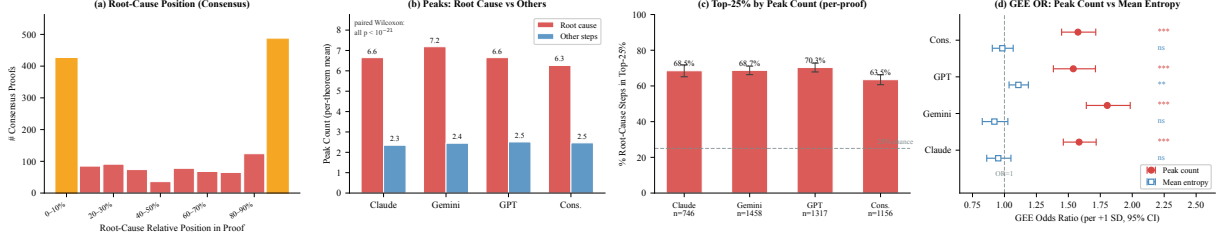


Figure 2: Empirical analysis of failed proofs from ProverBench, annotated by three independent LLM annotators and their majority-vote consensus. (a) Under the consensus labels, root-cause positions follow a bimodal distribution, with 77.3% occurring before the Lean error and 46.3% in the first half of the proof. (b) Across all annotators, root-cause steps contain far more entropy peaks than other steps, with a paired Wilcoxon test at $p < 10^{-21}$ for every annotator. (c) Under each annotator, 63.5–70.3% of root-cause steps rank in the top 25% of their own proof by peak count, far exceeding the 25% chance rate. (d) A generalized estimating equations analysis clustered on theorem shows that peak count is a strong predictor of root-cause steps under every annotator, whereas mean entropy is not significant. These findings motivate EBFS’s use of entropy peaks as the primary heuristic for identifying critical proof steps, and show the conclusion does not depend on any single annotator.

Why Peak Counting. We choose to count the number of high-entropy tokens (peaks) rather than using mean entropy as the uncertainty measure for a proof step. This design is motivated by two observations from our empirical analysis in Appendix A.1:

1. **Sparse high-entropy tokens dominate uncertainty.** Within a proof step, the vast majority of tokens exhibit low entropy (e.g., deterministic tactic names, fixed Lean syntax), while only a few tokens carry genuine decision uncertainty. Taking the mean entropy dilutes these critical signals. In contrast, peak counting directly captures the number of uncertain decision points.
2. **Peak count is a stronger root cause predictor than mean entropy.** Our empirical study shows that peak count is a robust predictor of root cause steps under cluster-aware statistics, whereas mean entropy is not significant under the consensus labels once proof/theorem clustering is respected, as shown by the generalized estimating equations analysis in Figure 2d. Peak count also achieves strong discriminative power: under the consensus annotation, 63.5% of root cause steps rank in the top 25% of their own proof by peak count, far above the 25% chance rate (Figure 2c). This demonstrates that peak counting provides a more reliable signal for identifying critical proof steps.

Additionally, a small number of tokens with high entropy can significantly influence the generation

of subsequent text, as the model’s uncertainty at these decision points propagates through the rest of the proof. Therefore, counting peaks provides a more faithful characterization of step-level criticality for guiding proof tree search.

Score Function. The scoring function of EBFS is shown in Equation (3), which primarily achieves three objectives: (1) prioritizing the search for newly expanded leaf nodes first, (2) the higher the uncertainty at the current step, the higher the score, and (3) penalizing nodes that have failed to expand multiple times.

$$V_s = \frac{1 + \log(1 + \sum_{c \in C_s} M_c)}{|C_s| + N + \epsilon} \quad (3)$$

C_s represents the child nodes corresponding to the node at step s , N is the number of times the current step s has attempted expansion but failed, and ϵ is a constant used to avoid division by zero. When ϵ is very small, newly generated leaf nodes, due to $|C_s| = N = 0$, receive a higher score, causing EBFS to prioritize their expansion in the next search. Equation (3) also demonstrates that the node scoring in EBFS is dynamically updated, where the score of each node depends on the search results of its child nodes.

Backtracking. EBFS maintains all nodes with dynamically updated scores. Before each selection, the node with the highest V_s is chosen for expansion, naturally enabling backtracking: as failed expansions accumulate on a branch, ancestor nodes with high entropy are re-prioritized. Unlike MCTS, EBFS does not backpropagate rewards to distant ancestors since failure at a node more likely reflects

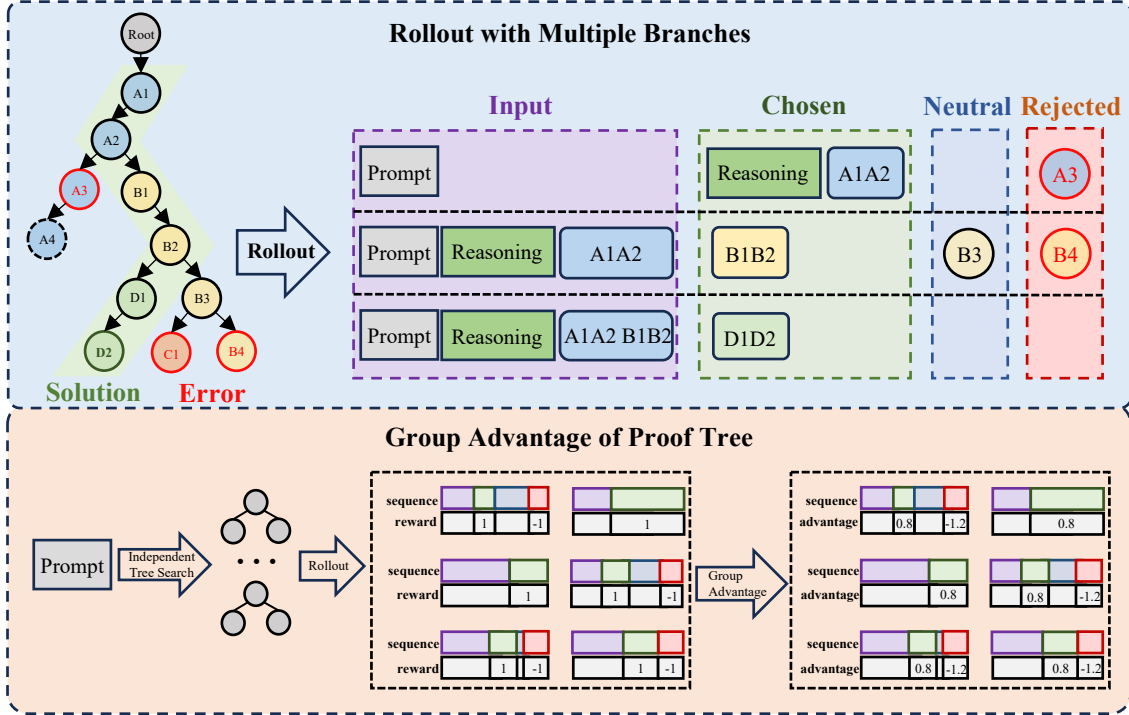


Figure 3: Our rollout method of proof trees in RFT. Each rollout sample is based on a whole proof generation. Nodes that serve as the ultimate complete proof are marked as **chosen**, nodes that failed to compile are marked as **rejected**. Nodes between these two categories are marked as **neutral**. Rewards are allocated for chosen and rejected outputs, with advantage values computed over a batch of multiple proof-tree rollouts.

a local code generation error than a flawed distant ancestor.

3.2 Rollout and Credit Assignment

In the crux step of proof, LLM often encounters multiple tactics. To improve the likelihood of LLMs selecting correct tactics, we adopt a reinforcement fine-tuning approach for proof generation, with a focus on crux steps, as inspired by prior research (Wang et al., 2025d). Specifically, the policy model π conducts proof search using EBFS over a batch of data $\mathcal{B}$. For those search trees that have found at least one complete proof, we employ the method illustrated in Figure 3 for rollout and credit assignment. Specifically, the proof search tree of EBFS is initially formed by generating a whole proof as the first branch, corresponding to A1-A4 in Figure 3. Subsequently, some nodes are backtracked to other proof steps, thereby generating additional branches, corresponding to B1-B4 in Figure 3. The final correct proof may result from a combination of nodes from multiple branches.

Rollout is conducted at the granularity of each whole proof generation. Within this process, child nodes that constitute valid steps toward the final

correct proof are labeled as chosen, the first node where a Lean compiler error occurs is designated as rejected, and nodes falling between these two categories are marked as neutral. Nodes not involved in the current whole proof generation but present in prior steps are treated as prompts. Subsequently, we assign a reward of $R_i = -1$ to rejected nodes, while chosen nodes receive a reward of $R_i = 1$. Neutral nodes are neither assigned rewards nor contribute to the computation of policy loss. For each problem, we generate multiple independent proof trees, with their rollouts forming a group for advantage calculation.

DAPO Filtering. Following DAPO (Yu et al., 2025), we filter out three categories of problems before advantage computation: (1) problems where all proof attempts failed to produce a complete proof, (2) problems where all attempts succeeded, and (3) among the remaining problems, those whose rollout samples contain no rejected nodes. This filtering ensures that each training group contains meaningful contrast between successful and failed proof steps, enabling effective advantage estimation. A concrete rollout example is provided

in Appendix A.13.

3.3 Entropy Weighted Policy Loss

We propose Entropy Weighted Policy Loss to address two challenges of proof generation in reinforcement fine-tuning stage: (1) mitigating entropy collapse during positive advantage updates and (2) preventing excessive punishment of generating the term spans of Lean code during non-positive advantage updates.

Trapezoidal weights of positive advantage.

When the advantage estimate $\hat{A}$ is positive, the model should be reinforced toward a higher likelihood of these actions. However, aggressively optimizing for all tokens can lead to premature entropy collapse, reducing exploration and making our EBFS algorithm degenerate. To mitigate this, we introduce a linearly-decaying weighting scheme based on the token-level entropy $e_{i,t}$. Specifically, we define the positive advantage entropy weight $E_{i,t}^p$ as:

$$E_{i,t}^p = \begin{cases} 1, & \text{if } e_{i,t} > \beta_p, \\ \frac{e_{i,t} - \alpha_p}{\beta_p - \alpha_p}, & \text{if } \alpha_p \leq e_{i,t} \leq \beta_p, \\ 0, & \text{if } e_{i,t} < \alpha_p. \end{cases} \quad (4)$$

which progressively down-weights the loss contribution of tokens with entropy below β_p , reaching zero when entropy falls below α_p . By reducing the gradient influence of low-entropy tokens under positive advantage, the model maintains healthier exploration dynamics while still reinforcing high-value actions.

Mask strategy of non-positive advantage. In Lean code generation, many semantic units (e.g., tactic names, theorem and lemma identifiers) are tokenized into multiple sub-tokens. While the initial token in such spans may exhibit high entropy due to variability, subsequent tokens often form fixed or highly predictable sequences, as shown in Figure 1, where we marked the high-entropy tokens in yellow. Standard policy optimization penalizes all tokens under negative advantage, potentially damaging the model’s ability to generate these critical multi-token constructs. To address this, for $\hat{A}_{i,t} \leq 0$, we employ a masking strategy based on

entropy thresholds:

$$E_{i,t}^n = \begin{cases} 1 - \frac{e_{i,t} - \alpha_n}{\beta_n - \alpha_n}, & \text{if } \alpha_n \leq e_{i,t} \leq \beta_n, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

This implementation excludes tokens with entropy below α_n , which might indicate fixed-span tokens, from contributing to the loss, preserving their generation patterns. Simultaneously, tokens with entropy exceeding β_n are also masked, as the model already maintains sufficient uncertainty and requires no further optimization incentive. Only tokens with moderate entropy are subject to loss penalties, with a linear decay in weighting.

Integrated Policy Loss The final entropy-guided weight $E_{i,t}$ is selected based on the advantage sign:

$$E_{i,t} = \begin{cases} E_{i,t}^p, & \text{if } \hat{A}_{i,t} > 0, \\ E_{i,t}^n, & \text{if } \hat{A}_{i,t} \leq 0. \end{cases} \quad (6)$$

This weight is incorporated into the objective function of DAPO (Yu et al., 2025), which employs the average reward of multiple sampled outputs generated for the same question as a baseline and uses this to calculate the relative advantage in a group. We also removed groups of proof trees with reward values of all zero or all one.

$$\mathcal{J}(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} E_{i,t} \cdot \min(r_{i,t} \hat{A}_{i,t}, \text{clip}(r_{i,t}, 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) \hat{A}_{i,t}), \quad (7)$$

where the token-level importance ratio is $r_{i,t} = \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}$, and the advantage is estimated at the proof (outcome) level and shared across all tokens of the same output, $\hat{A}_{i,t} = \hat{A}_i = \frac{R_i - \text{mean}(R)}{\text{std}(R)}$.

By dynamically adjusting the loss contribution per token based on advantage and entropy, our approach enhances training stability for Lean code generation while respecting the structural constraints of tokenized term spans of Lean.

4 Experiments

4.1 Setup

We compare against DeepSeek-Prover-V1.5-RL (Xin et al., 2025a), STP (Dong and Ma, 2025), and DeepSeek-Prover-V2-7B (Ren et al., 2025) (see Appendix A.3 for details). All models have 7B parameters. We further fine-tune DeepSeek-Prover-V2-7B using a combined SFT dataset of

Model	Budget	MiniF2F-test	ProofNet-test
<i>Baseline</i>			
DeepSeek-Prover-V1.5-RL (Xin et al., 2025a)	3200	54.9% $\pm$ 0.7	22.0% $\pm$ 0.5
DeepSeek-Prover-V1.5-RL + RMaxTS (Xin et al., 2025a)	1 $\times$ 3200	55.0% $\pm$ 0.7	21.5% $\pm$ 0.8
STP (Dong and Ma, 2025)	3200	65.0%	23.9% $\pm$ 0.6
Kimina-Prover-Preview-Distill-7B (Wang et al., 2025b)	1024	70.8%	–
DeepSeek-Prover-V2-7B (non-CoT) (Ren et al., 2025)	1024	73.2% $\pm$ 0.5	24.7%
DeepSeek-Prover-V2-7B (CoT) (Ren et al., 2025)	1024	79.9% $\pm$ 0.3	29.6%
DeepSeek-Prover-V2-7B (CoT) [†]	1024	79.1%	29.0%
<i>Inference with EBFS</i>			
DeepSeek-Prover-V1.5-RL	1 $\times$ 3200	57.8%	23.1%
STP	100 $\times$ 32	66.4%	25.2%
DeepSeek-Prover-V2-7B	32 $\times$ 32	77.4%	30.6%
<i>EBFS + Reinforcement Fine-Tuning</i>			
DeepSeek-Prover-V2-7B	32 $\times$ 32	81.8% $\pm$ 0.4	32.8% $\pm$ 0.5

Table 1: Experimental results on MiniF2F-test and ProofNet-test. The sampling budget for whole-proof generation denotes the number of independent proof attempts; for tree search methods, it is formatted as proof_budget $\times$ step_budget. The notation $\mu\% \pm \sigma$ denotes the mean and standard deviation over 3 independent runs. All models have 7B parameters. Baseline scores are sourced from their original papers; [†] denotes our reproduction in the same evaluation environment as EBFS.

40k samples and an RFT dataset of 8k samples curated through iterative difficulty-based filtering (Appendix A.4). We evaluate on MiniF2F-test (Zheng et al., 2022) (244 problems), ProofNet-test (Vishwakarma and Monticone, 2024) (186 problems), and ProverBench (Ren et al., 2025) (325 problems including 15 AIME 24&25 problems); benchmark descriptions are in Appendix A.5.

The composition of the tree search sampling budget in our experiment is: proof_budget $\times$ step_budget. The proof budget refers to the maximum number of times the proof tree can be constructed from scratch, and the step budget refers to the maximum number of branches that can be expanded. The total sampling budget in our evaluation is kept consistent with the baseline models. Under the “EBFS” setting in Table 1 and Table 2, we maintain the model and prompts consistent with the corresponding baselines, only altering the search or proof generation method to EBFS.

4.2 Implementation Details

We implemented our reinforcement fine-tuning algorithm based on OpenRLHF (Hu et al., 2024), with DeepSpeed Zero (Rasley et al., 2020; Rajbhandari et al., 2020) as the distributed framework and vLLM (Kwon et al., 2023) for inference. All

experiments were conducted on 128 NVIDIA A100 80GB GPUs with Lean v4.9. Full implementation details and hyperparameters are provided in Appendix A.7.

4.3 Benchmarks and Results

We evaluated our algorithm and models on MiniF2F (Zheng et al., 2022), ProofNet (Vishwakarma and Monticone, 2024), and ProverBench (Ren et al., 2025). The evaluation methods included running proof search using EBFS directly on the baseline models, as well as running proof search on our fine-tuned model. The evaluation environment is consistent with DeepSeek-Prover-V1.5 (Xin et al., 2025a).

The experimental results presented in Table 1 indicate that as a drop-in replacement for whole-proof generation or RMaxTS, EBFS improves the weaker base provers: it improves DeepSeek-Prover-V1.5-RL by 2.8% on MiniF2F and STP by 1.4%, and it improves DeepSeek-Prover-V2-7B on ProofNet. On the strongest base model, however, inference-only EBFS slightly regresses on MiniF2F; this is a CoT-sampling-breadth effect rather than a tree-search defect, which we isolate with a controlled experiment in Appendix A.2. Consistent improvement over the baselines is re-

Model	Budget	ProverBench (All)	AIME 24&25
<i>Baseline</i>			
STP (Dong and Ma, 2025)	128	31.4% $\pm$ 1.1	1/15
	512	36.3%	1/15
DeepSeek-Prover-V2-7B (non-CoT) (Ren et al., 2025)	128	48.8% $\pm$ 0.2	1/15
	512	49.5%	1/15
DeepSeek-Prover-V2-7B (CoT) (Ren et al., 2025)	128	50.8% $\pm$ 0.5	1/15
	512	51.7%	1/15
DeepSeek-Prover-V2-7B (CoT) [†]	512	51.7%	1/15
<i>Inference with EBFS</i>			
STP	8 $\times$ 16	30.8%	1/15
	32 $\times$ 16	36.5%	1/15
DeepSeek-Prover-V2-7B	8 $\times$ 16	50.0%	2/15
	32 $\times$ 16	52.3%	2/15
<i>EBFS + Reinforcement Fine-Tuning</i>			
DeepSeek-Prover-V2-7B	8 $\times$ 16	50.8% $\pm$ 0.3	2/15
	32 $\times$ 16	53.2% $\pm$ 0.5	2/15

Table 2: Experimental results on ProverBench. The “All” category represents the complete evaluation set of 325 problems; AIME 24&25 is a subset of 15 formalized competition problems. The notation $\mu\% \pm \sigma$ denotes the mean and standard deviation over 3 independent runs. All models have 7B parameters. Baseline scores are sourced from DeepSeek-Prover-V2 (Ren et al., 2025); [†] denotes our reproduction.

alized in the EBFS + reinforcement fine-tuning setting. After reinforcement fine-tuning, the model achieves 81.8% on MiniF2F and 32.8% on ProofNet, representing absolute gains of 1.9% and 3.2% over the strongest CoT baseline, respectively. Experimental results shown in Table 2 demonstrate that the fine-tuned model achieves the best overall performance, reaching 53.2% on ProverBench, surpassing both the CoT baseline and the EBFS-only configuration. Concrete examples of EBFS proof search with backtracking behavior on all three benchmarks are provided in Appendix A.12. However, the effectiveness of our method will be compromised under a small sampling budget, such as 8×16 . This is fundamentally because tree search-based approaches are significantly more limited in terms of the number of tokens generated and the range of tactics that can be explored compared to whole proof generation. As the sampling budget increases, this limitation diminishes, and the method continues to outperform whole proof generation. This can be attributed to the efficiency of tree search, which, to some extent, avoids redundant generation of straightforward proof steps. Figure 9 shows the pass rate scaling curves as proof budget increases from 1 to 32 on all three bench-

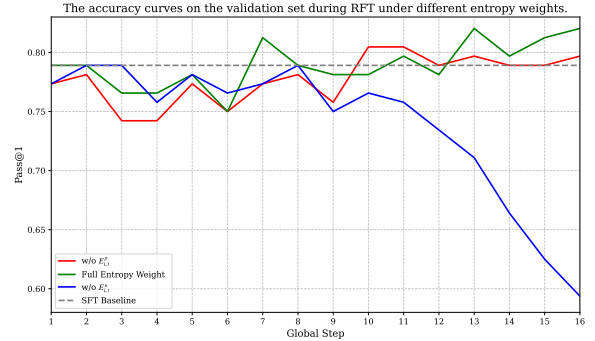


Figure 4: Validation results in RFT process. Here, w/o $E_{i,t}^X$ means that we set $E_{i,t}^X \equiv 1$.

marks, confirming rapid early gains followed by diminishing returns.

4.4 Ablation and Analysis

We further conducted an ablation study on the two entropy weights $E_{i,t}^p, E_{i,t}^n$ introduced in Section 3.3, evaluating the pass rate curves on the validation set under different weight settings during the RFT process. The validation set consists of 128 problems selected from the miniF2F validation split.

Specifically, we fixed each of the two entropy weights to 1 independently and conducted respec-

tive RFT runs based on these settings. The performance on the validation set is shown in Figure 4. For comparison, we used the performance of the SFT model on the validation set as a baseline.

1. Without $E_{i,t}^n$, the model collapses in the later stages of training. We believe that this is caused by an excessive penalty on the semantic units of lean, which leads the model to catastrophically forget how to utilize those high-frequency and deterministic tactics.
2. Without $E_{i,t}^p$, the model exhibits faster initial convergence but shows a slight performance degradation at the end of training, ultimately failing to achieve superior results compared to the SFT model.

Additionally, we conducted sensitivity analyses for the entropy peak threshold λ and the entropy weight parameters $\alpha_p, \beta_p, \alpha_n, \beta_n$. The results shown in Appendix A.8 confirm that λ is robust across a broad range (0.1–0.6) and that setting $\alpha_n=0$ causes significant degradation, further validating our design in Section 3.3.

Scoring-Function Ablation. To verify that the gain comes from the entropy heuristic rather than the tree structure alone, we compare node-scoring criteria under an identical search/verify harness on ProverBench at 8×16 , varying only the score used to select the next node for expansion, as shown in Table 3. Our peak-count score outperforms mean entropy and the length-normalized log-likelihood criterion of BFS-Prover (Xin et al., 2025b), and removing the entropy signal entirely is worst. Since the BFS-Prover criterion was designed for stepwise tactic-generation provers while ours is a whole-proof generator truncated into tactic nodes, we report this as a controlled test of the selection criterion, not a reproduction of the BFS-Prover system.

Scoring criterion	Solved / 325	AIME 24&25
EBFS peak-count	162	2/15
Log-likelihood	158	1/15
Mean entropy	155	1/15
Uniform	152	1/15

Table 3: Scoring-function ablation on ProverBench at 8×16 under an identical search/verify harness. Peak-count beats mean entropy, and removing the entropy signal is worst, showing the gain comes from the entropy heuristic rather than the tree structure alone.

Component Attribution. Taken together, the ablations isolate the contribution of each component. First, EBFS contributes independently of any fine-tuning, since applied to frozen base models it improves DeepSeek-Prover-V1.5-RL by +2.8% and STP by +1.4% with no training compute, as reported in Table 1. Second, given EBFS rollout data, the entropy-weighted loss is necessary, since removing either the E^p or the E^n weight degrades or collapses validation performance relative to the full method in Figure 4, and the full method outperforms the SFT baseline. Third, the peak-count scoring criterion itself drives the search gain, as shown in Table 3.

5 Conclusion

In this paper, we introduced Entropy-Based Best-First Search, a novel proof tree search algorithm that leverages the policy entropy to identify and prioritize critical proof steps for expansion. Our empirical study on consensus-annotated failed proofs validates the core hypothesis that entropy peaks reliably signal root cause steps, providing a principled foundation for the search heuristic. Experiments across MiniF2F, ProofNet, and ProverBench demonstrate that EBFS improves the weaker base provers as a drop-in replacement, and that our full EBFS + entropy-weighted reinforcement fine-tuning approach consistently improves over the baselines, achieving up to 3.2% gains on ProofNet and 1.9% on MiniF2F, boosting performance to 81.8% on MiniF2F and 53.2% on ProverBench and establishing new results for 7B-parameter provers under comparable sampling budgets.

6 Limitations

This paper investigates EBFS, a proof tree search algorithm designed to more efficiently explore diverse branches of proofs. However, EBFS is optimized exclusively for a single proof generator, whereas recent research (Cao et al., 2025; Chen et al., 2025a) suggests that multi-agent systems possess higher potential performance in proof generation. Effectively extending the proof tree search approach to multi-agent settings and enhancing their efficiency in exploring proof trees remains an open and pressing research challenge.

References

Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. 2025. [The unreasonable effectiveness](#)

- of entropy minimization in LLM reasoning. *CoRR*, abs/2505.15134.
- Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir R. Radev, and Jeremy Avigad. 2023. [Proofnet: Autoformalizing and formally proving undergraduate-level mathematics](#). *ArXiv*, abs/2302.12433.
- Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. [Llemma: An open language model for mathematics](#). In *The Twelfth International Conference on Learning Representations*.
- Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, HUI XUE, and Fan Yang. 2025. [Reviving DSP for advanced theorem proving in the era of reasoning models](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, and 1 others. 2025a. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *arXiv preprint arXiv:2512.17260*.
- Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, and 17 others. 2025b. [Seed-prover: Deep and broad reasoning for automated theorem proving](#). *CoRR*, abs/2507.23726.
- Projet Coq. 1996. The coq proof assistant-reference manual. *INRIA Rocquencourt and ENS Lyon, version*, 5.
- Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. 2025. [The entropy mechanism of reinforcement learning for reasoning language models](#). *CoRR*, abs/2505.22617.
- Tri Dao. 2024. [Flashattention-2: Faster attention with better parallelism and work partitioning](#). In *The Twelfth International Conference on Learning Representations*.
- Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. 2025. [Agentic entropy-balanced policy optimization](#). *CoRR*, abs/2510.14545.
- Kefan Dong and Tengyu Ma. 2025. [Stp: Self-play llm theorem provers with iterative conjecturing and proving](#). *ArXiv*, abs/2502.00212.
- Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023. [Baldur: Whole-proof generation and repair with large language models](#). In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023*, page 1229–1241, New York, NY, USA. Association for Computing Machinery.
- Robert Joseph George, Suozhi Huang, Peiyang Song, and Anima Anandkumar. 2025. [Leanprogress: Guiding search for neural theorem proving via proof progress prediction](#). *Trans. Mach. Learn. Res.*, 2025.
- 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*.
- Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward Ayers, and Stanislas Polu. 2022. [Proof artifact co-training for theorem proving with language models](#). In *International Conference on Learning Representations*.
- John Harrison. 1996. Hol light: A tutorial introduction. In *International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer.
- Jian Hu, Xibin Wu, Weixun Wang, Dehao Zhang, Yu Cao, OpenLLMAI Team, Netease Fuxi, AI Lab, and Alibaba Group. 2024. [Openrlhf: An easy-to-use, scalable and high-performance rlhf framework](#). *ArXiv*, abs/2405.11143.
- Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. 2025. [Leanagent: Lifelong learning for formal theorem proving](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23*, page 611–626, New York, NY, USA. Association for Computing Machinery.
- Guillaume Lample, Timothee Lacroix, Marie anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. 2022. [Hypertree proof search for neural theorem proving](#). In *Advances in Neural Information Processing Systems*.
- Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. [Solving quantitative reasoning problems with](#)

- language models. In *Advances in Neural Information Processing Systems*.
- Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. 2024. [Hunyuan-prover: A scalable data synthesis framework and guided tree search for automated theorem proving](#). *ArXiv*, abs/2412.20735.
- Jannis Limperg and Asta Halkjær From. 2023. [Aesop: White-box best-first proof search for lean](#). In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023*, page 253–266, New York, NY, USA. Association for Computing Machinery.
- Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025. [Goedel-prover: A frontier model for open-source automated theorem proving](#). *ArXiv*, abs/2502.07640.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction—CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 625–635. Springer.
- Lawrence C Paulson. 1994. *Isabelle: A generic theorem prover*. Springer.
- Stanislas Polu and Ilya Sutskever. 2020. [Generative language modeling for automated theorem proving](#). *ArXiv*, abs/2009.03393.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '20*. IEEE Press.
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. [Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters](#). In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20*, pages 3505–3506, New York, NY, USA. Association for Computing Machinery.
- Michael Rawson and Giles Reger. 2021. [lazycop: Lazy paramodulation meets neurally guided search](#). In *Automated Reasoning with Analytic Tableaux and Related Methods: 30th International Conference, TABLEUX 2021, Birmingham, UK, September 6–9, 2021, Proceedings*, page 187–199, Berlin, Heidelberg. Springer-Verlag.
- Z. Z. Ren, Zhihong Shao, Jun-Mei Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liye Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shiron 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](#). *ArXiv*, abs/2504.21801.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *ArXiv*, abs/2402.03300.
- Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2023. [Towards large language models as copilots for theorem proving in lean](#). In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.
- Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. 2024. [An in-context learning agent for formal theorem-proving](#). In *First Conference on Language Modeling*.
- Rahul Vishwakarma and Pietro Monticone. 2024. [Proofnet-lean4](#). <https://github.com/rahul3613/ProofNet-lean4>. Accessed: 2024-12-01.
- 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. 2025a. [Kimina-prover preview: Towards large formal reasoning models with reinforcement learning](#). *CoRR*, abs/2504.11354.
- 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. 2025b. [Kimina-prover preview: Towards large formal reasoning models with reinforcement learning](#).
- Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2024. [Lego-prover: Neural theorem proving with growing libraries](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. 2025c. [Ma-lot: Multi-agent lean-based long chain-of-thought reasoning enhances formal theorem proving](#). *CoRR*, 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. 2025d. [Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning](#). *ArXiv*, abs/2506.01939.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. [Chain of thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*.
- 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](#). *ArXiv*, abs/2410.15700.
- Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanbiao Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. 2025a. [Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search](#). In *The Thirteenth International Conference on Learning Representations*.
- Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. 2025b. [BFS-prover: Scalable best-first tree search for LLM-based automatic theorem proving](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32588–32599, Vienna, Austria.
- Wujiang Xu, Wentian Zhao, Zhenting Wang, Yu-Jhe Li, Can Jin, Mingyu Jin, Kai Mei, Kun Wang, and Dimitris N. Metaxas. 2025. [EPO: entropy-regularized policy optimization for LLM agents reinforcement learning](#). *CoRR*, abs/2509.22576.
- Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2024. [Formal mathematical reasoning: A new frontier in ai](#). *ArXiv*, abs/2412.16075.
- Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. [Leandojo: Theorem proving with retrieval-augmented language models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 21573–21612. Curran Associates, Inc.
- 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, and 3 others. 2024. [Internlm-math: Open math large language models toward verifiable reasoning](#). *ArXiv*, abs/2402.06332.
- Qiyang 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](#). *ArXiv*, abs/2503.14476.
- Xiaoyun Zhang, Xiaojian Yuan, Di Huang, Wang You, Chen Hu, Jingqing Ruan, Kejiang Chen, and Xing Hu. 2025. [Rediscovering entropy regularization: Adaptive coefficient unlocks its potential for LLM reinforcement learning](#). *CoRR*, abs/2510.10959.
- Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. [minif2f: a cross-system benchmark for formal olympiad-level mathematics](#). In *International Conference on Learning Representations*.

A Appendix

A.1 Empirical Analysis of Root Cause Positions

To validate the hypothesis that proof failures often originate far earlier than the Lean compiler error position, and that token-level entropy can reliably signal these critical steps, we conducted a large-scale empirical study on the ProverBench test set (Ren et al., 2025).

A.1.1 Data Generation

We used DeepSeek-Prover-V2-7B as the proof generation model. For each of the 325 theorems in ProverBench, we sampled 32 independent proof attempts with temperature 0.6 and a maximum generation length of 10,240 tokens. During generation, we collected the top-20 logprobs for each token to enable subsequent entropy analysis. The generation was conducted using 64 parallel vLLM engines on a Ray cluster.

Each generated proof was verified by the Lean 4 compiler and segmented at the tactic level. For failed proofs, we recorded: (1) the sequence of verified tactic segments that pass the Lean type checker individually, (2) the first Lean error message and its position, and (3) the remaining unverified code. For each verified segment, we computed the per-token entropy sequence and the number of entropy peaks, defined as tokens with entropy ≥ 0.3 , from the proof state after that step.

A.1.2 Root Cause Annotation

Candidate Selection. From the 10,400 total proof attempts, we selected failed proofs with at least 3 verified tactic segments, ensuring sufficient context for meaningful root cause analysis. For

System Prompt

You are an expert in Lean 4 theorem proving and formal verification. Your task is to analyze a failed proof attempt generated by an LLM-based prover and determine the most likely cause of the failure.

A proof failure can originate from different locations:

- It could be at or near the point where Lean reports the error (e.g., a wrong tactic application, a missing lemma invocation, or a type mismatch).
- It could be at an earlier step where a strategic decision (e.g., choice of induction variable, case split, or intermediate lemma) set the proof on a path that eventually becomes unrecoverable.
- It could even be that the overall proof approach is reasonable but the final steps simply contain a local mistake.

Your job is to objectively assess each step of the proof and determine where the failure most likely originates, without any prior assumption about whether the cause is early or late in the proof. Base your judgment solely on the mathematical content and proof state evolution.

Figure 5: System prompt used for the root cause annotation agent. The prompt explicitly avoids positional bias by instructing the model to assess without prior assumptions about failure location.

Round 1: THINK

Here is a Lean 4 theorem that an LLM-based prover attempted to prove but failed.

```
**Theorem:**  
``lean4  
{formal_statement}  
``
```

Before seeing the prover's attempt, think about how you would approach proving this theorem. What proof strategies seem most natural? What are the key difficulties or potential pitfalls?

Provide a brief proof sketch (2-5 key steps).

Figure 6: Round 1 (THINK) prompt template. The annotator reasons about the theorem independently before seeing the failed proof, establishing an unbiased mathematical judgment. Placeholder {formal_statement} is filled with the Lean 4 theorem statement.

each theorem, we retained the longer failed samples ranked by number of verified steps, preferring longer proofs for richer analysis. Each retained proof was independently annotated by all three annotators, yielding 1,532 proofs with a majority-vote consensus root cause.

Annotation Model. Our primary annotator is Claude Opus 4.6, accessed via the Anthropic Messages API with extended thinking enabled. The model was configured with temperature 0.6 and a maximum output of 65,536 tokens per response to allow thorough reasoning. To assess the validity of the annotations rather than rely on self-reported confidence, we additionally run the identical protocol with Gemini-3.1-pro-preview and GPT-5.5; the resulting cross-annotator analysis is reported in

Section A.1.5.

Multi-Turn Annotation Protocol. Each annotation consists of a structured 3-round conversation:

- **Round 1 (THINK):** The model is shown only the theorem statement and asked to reason about natural proof strategies, key difficulties, and potential pitfalls, forming an independent mathematical judgment before seeing the prover's attempt.
- **Round 2 (ANALYZE):** The model is presented with the full segmented proof: all verified steps (with proof states shown at every 20th step, plus the first and last steps), the Lean error information, and the remaining unverified code. The model assesses each step

Round 2: ANALYZE

Now here is the prover's failed proof attempt. The proof has been segmented at the tactic level. For each step that passed Lean's type checker, I show the tactic code. The full proof state (goal + hypotheses) is shown at every 20th step, plus the first and last verified steps.

****Verified steps**** (these steps individually pass Lean's checker):
{verified_steps}

****First Lean error:****
{error_info}

****Remaining code after the error point**** (did NOT pass verification):
``lean4
{remaining_code}
``

Analyze this proof attempt and determine where the failure most likely originates. The cause could be:

- At the error point itself (the failing tactic is simply wrong)
- At a nearby preceding step (a small mistake that propagates)
- At an earlier strategic decision (makes the goal unrecoverable)

For each verified step, briefly assess whether it is:

- (a) Strategically sound and making progress toward the goal
- (b) Valid but potentially problematic (explain why)
- (c) A consequence of an earlier decision that may be questionable

Format your answer as:

ROOT_CAUSE_CANDIDATES: [list of step indices, most likely first]

PRIMARY_ROOT_CAUSE: <step index>

REASONING: <explanation>

Figure 7: Round 2 (ANALYZE) prompt template. The annotator receives the full segmented proof with verified steps, error information, and remaining code, then produces a structured root cause assessment. Placeholders: {verified_steps} contains tactic code and periodic proof states; {error_info} contains the Lean error type, position, and message; {remaining_code} is the unverified suffix.

as (a) strategically sound, (b) valid but potentially problematic, or (c) a consequence of an earlier questionable decision. It then outputs structured fields: ROOT_CAUSE_CANDIDATES (ranked list), PRIMARY_ROOT_CAUSE (step index), and REASONING.

- **Round 3 (CONFIRM):** If multiple candidates were identified in Round 2, the model is asked to select the single most likely root cause with justification, outputting FINAL_ROOT_CAUSE, CONFIDENCE (high/medium/low), and JUSTIFICATION.

System Prompt Design. The system prompt instructs the annotator to act as “an expert in Lean 4 theorem proving and formal verification” and explicitly states that it should determine where the failure originates “without any prior assumption about whether the cause is early or late in the proof,” basing judgment “solely on the mathematical con-

tent and proof state evolution.” This design prevents systematic bias toward early or late positions. The full prompts used in each round are shown below.

A.1.3 Annotation Quality

As a secondary check, the primary Claude annotator reported *high* or *medium* confidence on essentially all of its annotations, with none rated *low*. We treat this self-reported confidence only as a weak signal, and take the primary validity evidence to be the cross-annotator robustness reported in Section A.1.5, which shows the entropy-peak finding is stable across three independent annotators and their consensus.

A.1.4 Statistical Analysis

We joined the consensus annotation data with the per-segment entropy data from the generation pipeline. Multiple steps are nested within each proof and multiple proofs within each theorem, so

Round 3: CONFIRM

You identified multiple candidates for the cause of failure: {candidates}.

Please reconsider carefully and select the SINGLE most likely cause.

Think about:

1. Which step, if changed, would most likely lead to a successful proof?
2. Are some candidates merely consequences of an earlier decision, or are they independent issues?
3. Is the primary cause a local mistake at a specific step, or a strategic decision that affects multiple later steps?

Do not bias toward any particular position in the proof – choose based on which candidate best explains the failure.

Provide your final answer:

FINAL_ROOT_CAUSE: <step index>

CONFIDENCE: <high/medium/low>

JUSTIFICATION: <brief explanation>

Figure 8: Round 3 (CONFIRM) prompt template. This round is only triggered when Round 2 yields multiple candidates. The annotator disambiguates and provides a final answer with confidence rating. Placeholder {candidates} is filled with the candidate list from Round 2.

we use cluster-aware statistics rather than pooling all steps as independent observations. The key findings are:

- **Gap Analysis.** Under the consensus labels, 77.3% of root causes occur before the Lean error position, and 46.3% are in the first half of the proof. The distribution is bimodal, as shown in Figure 2a.
- **Entropy Signal from peak count.** Per proof, 63.5% of consensus root-cause steps fall in the top 25% of their own proof by peak count, far above the 0.25 chance rate. Per theorem, root-cause steps have more peaks than other steps in 74.9% of theorems, with a paired Wilcoxon test at $p = 5 \times 10^{-22}$ and all per-annotator tests at $p < 10^{-21}$. A generalized estimating equations logistic regression clustered on theorem gives an odds ratio of 1.58 per +1 standard deviation of peak count at $p = 8 \times 10^{-26}$. These results are illustrated in Figure 2b and Figure 2c.
- **Mean entropy is not robust.** Under the same cluster-aware treatment, mean entropy is *not* significant at the theorem level under the consensus labels, even though it appears significant when steps are pooled. Figure 2d contrasts the two: the peak-count odds ratio is well above 1 for every annotator, whereas the mean-entropy odds ratio is not significant under the consensus and for most annotators.

This reinforces our design choice to count entropy peaks rather than use mean entropy, as discussed in Section 3.3.

A.1.5 Cross-Annotator Robustness

Because the primary analysis relies on a single LLM annotator, Claude Opus 4.6, and because self-reported confidence is not a validity measure, we re-ran the entire 3-round protocol with two additional proprietary annotators from different providers, namely Gemini-3.1-pro-preview and GPT-5.5, forming a three-model panel that also controls for single-vendor bias.

Inter-annotator agreement. The annotators agree only moderately, with a Fleiss κ of 0.42, which is consistent with the view that a single “root cause” is partly ambiguous. Table 4 reports pairwise agreement.

Annotator pair	exact step	within-3	Spearman ρ
Claude–Gemini	71.2%	83.5%	0.74
Claude–GPT	34.6%	59.7%	0.39
Gemini–GPT	35.1%	60.2%	0.38

Table 4: Pairwise inter-annotator agreement on the identified root-cause step, measured by exact match, within-3-step match, and Spearman rank correlation. The Fleiss κ across the three annotators is 0.42.

The entropy-peak finding is invariant to the annotator. Re-deriving the cluster-aware statistics from each annotator’s labels, and from their

majority-vote consensus, gives essentially the same answer throughout, as summarized in Table 5. Peak count is a strong predictor under every annotator, so the paper’s conclusion does not depend on any single model. We therefore foreground this cross-annotator robustness as the validity evidence and treat the self-confidence figure discussed above as secondary.

Labels	rc steps	top-25%	OR (GEE)
Claude	746	68.5%	1.58 ($p=10^{-28}$)
Gemini	1458	68.7%	1.80 ($p=5\times 10^{-34}$)
GPT	1317	70.3%	1.54 ($p=3\times 10^{-15}$)
Consensus	1156	63.5%	1.58 ($p=8\times 10^{-26}$)

Table 5: Cross-annotator robustness of the peak-count finding. For each annotator’s labels, together with their majority-vote consensus, we report the number of root-cause (rc) steps, the fraction of rc steps that fall in the top 25% within their own proof by peak count under the per-proof cluster-aware criterion, and the generalized estimating equations (GEE) logistic-regression odds ratio per +1 standard deviation (SD) of peak count clustered on theorem. The signal is stable across all annotators, and the per-proof top-25% fractions here match those plotted in Figure 2c.

A.2 CoT Sampling Breadth vs. Search Depth

Inference-only EBFS slightly regresses on the strongest base model on MiniF2F, scoring 77.4% versus 79.1% in Table 1. We show this is a chain-of-thought (CoT) sampling-breadth effect at fixed budget, not a defect of tactic-level truncation or backtracking. EBFS regenerates the global CoT only once per proof attempt and backtracks only the Lean code, so at `proof_budget= 32` it samples 32 distinct CoTs, whereas `whole-proof@1024` samples 1024.

We isolate the cause on the 6 MiniF2F theorems that `whole-proof@1024` solved but `EBFS@32 × 32` failed. These are hard cases, each solved only once in the 1024 baseline samples, for a usable-CoT rate of about 1/1024. We run the identical EBFS tree search under the three conditions listed in Table 6.

Conditions 1 and 2 together isolate the cause: the self-sampled CoTs fail both when searched as a tree and when generated as plain whole proofs at 1024 generations per theorem, so the CoTs themselves are unusable and neither truncation nor backtracking is responsible. Condition 3 shows the same search solves all six almost immediately once given a good CoT. The injected CoTs come from our reproduced baseline@1024 successes for these

Condition	Result
(1) Tree search on 32 self-generated CoTs	0/6
(2) Same 32 CoTs, plain whole-proof, 32× each	0/6
(3) Tree search with one injected known-good CoT	6/6

Table 6: Isolating the normal-budget MiniF2F regression on the 6 theorems `whole-proof@1024` solved but `EBFS@32 × 32` failed. Conditions 1 and 2 share the model’s self-sampled CoTs, while condition 3 injects a known-good CoT drawn from the reproduced baseline@1024 successes at the same model, prompt, and temperature. Condition 3 solves all six with about 1.5 LLM calls per theorem.

theorems, that is, from the same model at the same prompt and temperature, and are exactly the CoTs EBFS could produce but did not sample within 32 attempts, whereas 1024 whole-proof samples eventually hit one. EBFS thus trades CoT breadth for code-search depth at a fixed budget, which is why its benefit on the strongest model is realized through reinforcement fine-tuning rather than drop-in inference.

A.3 Baseline Models

DeepSeek-Prover-V1.5-RL (Xin et al., 2025a) is fine-tuned on a large synthetic dataset and trained with online reinforcement learning. Its search algorithm employs an improved Monte Carlo Tree Search (RMaxTS), where intrinsic rewards are obtained when expanding new nodes.

STP (Dong and Ma, 2025) iteratively generates synthetic data for training using a conjecture model and a proof model. The conjecture model generates problems with increasing difficulty through iterative training, thereby enhancing the capability of the proof model via expert iteration.

DeepSeek-Prover-V2-7B (Ren et al., 2025) combines global deep reasoning with recursive decomposition of proof objectives, distilled from a 671B model, achieving stable performance on harder problems.

A.4 Training Dataset

We incorporate formal statements from two open-source datasets (Dong and Ma, 2025; Wang et al., 2025a) and employ a data annotation pipeline. Initially, DeepSeek-Prover-V2-7B samples 32 proofs per question. Problems with pass rate > 0.25

are filtered out; those with pass rate in $(0, 0.25]$ form Dataset $\mathcal{D}_1$. For zero-pass-rate questions, DeepSeek-Prover-V2-671B is used for annotation: proofs with pass rate > 0.25 are assigned to $\mathcal{D}_2$, and those in $(0, 0.25]$ to $\mathcal{D}_3$. This procedure is iterated until $\mathcal{D}_3$ reaches sufficient size. The final SFT dataset ($\mathcal{D}_1 \cup \mathcal{D}_2$) contains 40k samples, and the RFT dataset ($\mathcal{D}_3$) contains 8k samples.

A.5 Benchmark Descriptions

MiniF2F (Zheng et al., 2022) is a formal mathematics benchmark consisting of 488 problem statements curated from high school and undergraduate math, as well as competitions (AMC, AIME, IMO). Difficulty ranges from basic algebra to Olympiad-level challenges. The dataset is divided equally into 244 validation and 244 test problems; we use the test split.

ProofNet (Azerbayev et al., 2023) is a formalized proof dataset with undergraduate-level difficulty. We use the Lean 4 version (Vishwakarma and Monticone, 2024) with the test split (186 problems) partitioned by DeepSeek-Prover-V1.5 (Xin et al., 2025a).

ProverBench (Ren et al., 2025) comprises 325 problem descriptions including 15 AIME 2024&2025 questions, with difficulty covering high school competitions and undergraduate mathematics.

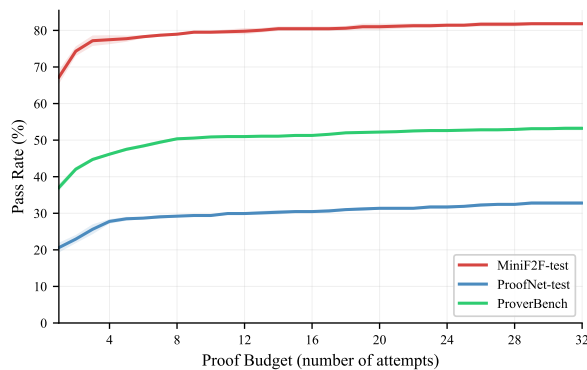


Figure 9: Pass rate as a function of proof budget (number of proof tree construction attempts) on three benchmarks, with the step budget fixed at 32. Solid lines denote the mean over 3 independent runs; shaded regions indicate ± 1 standard deviation. Performance improves rapidly in the first few attempts and exhibits diminishing returns beyond budget 10–15, demonstrating the sample efficiency of EBFS.

A.6 Scaling and Ablation

Algorithm 1 presents the pseudo code of EBFS. At each iteration, the model generates a proof continuation from the selected node, the Lean compiler verifies it, and the validated portion is segmented into tactic-level nodes. Each node’s entropy score is computed and used to update the priority queue. The node with the highest score is selected for the next expansion.

Figure 9 shows how the pass rate scales with proof budget across all three benchmarks. Performance improves rapidly during the first 5–8 attempts and plateaus after budget 15, demonstrating the sample efficiency of EBFS. The ablation of the entropy weights $E_{i,t}^p$ and $E_{i,t}^n$ during RFT training is presented in the main text (Section 3.3, Figure 4).

Algorithm 1 EBFS

Input: theorem statement q , proof generate model π

Parameter: step budget b

Output: root

```

Let  $t = 0$ 
Let root = init_root( $q$ )
node = root
while  $t < b$  do
   $t = t + 1$ 
   $o, e = \text{generate}(\text{node}, \pi)$ 
   $r = \text{verification}(\text{node.code}, o)$ 
   $S, E = \text{segmentation}(\text{node.code}, o, r, e)$ 
  for  $s \in S, e_s \in E$  do
    node  $\leftarrow$  node.expand( $s, e_s$ )
  if node.valid then
    root.update_candidate(node)
  end if
end for
root.sort_all_nodes()
node = root.select()
end while
return root

```

A.7 Implementation Details and Hyperparameters

We implemented our reinforcement fine-tuning algorithm based on OpenRLHF (Hu et al., 2024). Additionally, we employed DeepSpeed’s Zero (Rasley et al., 2020; Rajbhandari et al., 2020) as the distributed framework and utilized FlashAttention-2 (Dao, 2024) to accelerate training. For the optimizer, we adopted AdamW (Loshchilov and Hutter,

2019). During the inference phase, we deployed the generation service using vLLM (Kwon et al., 2023). All experiments were conducted on a cluster with 128 NVIDIA A100 80GB GPUs. The Lean compiler version is v4.9. The evaluation environment is consistent with DeepSeek-Prover-V1.5 (Xin et al., 2025a).

Table 7 summarizes all hyperparameters used in our experiments.

Hyperparameter	SFT	RFT
<i>Training</i>		
Learning rate	1×10^{-5}	5×10^{-6}
Batch size	128	256
Rollout batch size	–	512
Sampling budget (per problem)	–	16×32
Sampling temperature	–	1.0
<i>EBFS Algorithm</i>		
Entropy peak threshold λ		0.3
Score function ϵ		0.01
<i>Entropy Weighted Loss</i>		
Positive weight α_p		0.2
Positive weight β_p		1.0
Negative weight α_n		0.1
Negative weight β_n		1.2
<i>Evaluation</i>		
Sampling temperature		0.8
Max generation length		16384 tokens

Table 7: Hyperparameters for supervised fine-tuning (SFT) and reinforcement fine-tuning (RFT). EBFS algorithm and entropy-weighted loss parameters are shared across both stages.

A.8 Parameter Sensitivity Analysis

A.8.1 Sensitivity of Entropy Weight Parameters

Due to the substantial computational cost of full RFT, we conducted parameter exploration on a lightweight RFT setting with a 4k data subset and 4×16 rollout budget. We varied one parameter at a time while keeping the others at their default values ($\alpha_p=0.2, \beta_p=1.0, \alpha_n=0.1, \beta_n=1.2$). Pass rates are evaluated on the MiniF2F validation set at the final RFT step.

A.8.2 Sensitivity of λ (Entropy Peak Threshold)

We conducted a parameter sweep for the entropy peak threshold λ in Equation (2) using the fine-tuned model on ProofNet-test with a budget of 32×32 . The results are shown in Table 9.

α_p	0.0	0.2	0.4	0.6
Pass rate (%)	80.4	80.4	79.6	77.3
β_p	0.4	0.7	1.0	1.3
Pass rate (%)	79.6	81.3	80.4	81.3
α_n	0.0	0.1	0.3	0.5
Pass rate (%)	68.7	80.4	79.7	80.4
β_n	0.6	0.9	1.2	1.5
Pass rate (%)	78.1	80.4	80.4	80.4

Table 8: Sensitivity of entropy weight parameters on validation set. β_p and β_n are stable across a broad range. Setting $\alpha_n=0.0$ causes significant degradation (68.7%), confirming the necessity of protecting low-entropy semantic units from excessive penalization (Section 3.3).

This insensitivity can be attributed to the intrinsic characteristics of Lean tactic code generation: over 80% of tokens exhibit very low entropy, as shown in Table 10.

A.8.3 Top- k Entropy Approximation Error

Our entropy is computed from the top-20 logprobs collected during generation rather than the full vocabulary. To quantify the resulting approximation error, we recomputed full-vocabulary entropy on 100 test proofs. The top-20 entropy matches the full-vocabulary entropy to a mean absolute error of 5×10^{-5} , and at $\lambda = 0.3$ all 4,403 detected entropy peaks are preserved. This is a direct consequence of the concentration shown in Table 10: token distributions over Lean code are peaked enough that the top-20 truncation captures essentially the entire distribution, even at the more uncertain steps. The heuristic is also robust by construction, since the EBFS score depends only on peak *counts*: any change in k can be absorbed by re-choosing λ .

A.9 Systematic Statistics and Computational Cost

Table 11 summarizes the computational characteristics of EBFS proof search across the three evaluation benchmarks.

The complete RFT procedure consumed approximately 172 hours on 128 NVIDIA A100 80GB GPUs. The rollout phase consists of $8000/512 = 16$ rollout steps, with each step generating approximately 2k training instances (varying due to proof tree branching and DAPO filtering). The total training steps during RFT can be approximated as $2000/256 \times 16 \approx 128$.

λ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Pass rate (%)	26.8	33.3	32.3	33.3	33.3	32.3	33.3	30.1	29.6

Table 9: Parameter sensitivity of λ on ProofNet-test. The method is insensitive across a broad range (0.1–0.6). At $\lambda = 0.0$, EBFS degenerates into prioritizing steps by token count rather than entropy, causing degradation.

Entropy range	0–0.1	0.1–0.2	0.2–0.3	0.3–0.4	0.4–0.5	0.5–0.6	0.6–0.7	0.7–0.8	>0.8
Ratio (%)	80.14	3.26	1.98	1.89	1.54	1.80	4.62	1.83	2.94

Table 10: Distribution of token-level entropy in Lean proof steps on the training dataset. Over 80% of tokens have entropy below 0.1, confirming that high-entropy tokens are sparse and that the peak threshold λ is robust across a wide range.

We note that both RFT and evaluation experiments are executable on a single node with $8 \times \text{A100 80GB GPUs}$. The larger cluster was employed solely to enhance parallelism and reduce wall-clock time.

Token-level budget accounting. To clarify that the “same sample budget” comparison against whole-proof generation is also fair at the token level: under the same `proof_budget` $\times$ `step_budget`, each EBFS expansion resumes from a verified prefix rather than restarting the proof. It therefore decodes *fewer* output tokens, because the shared prefix is reused rather than regenerated, but consumes *more* input/prefill tokens, because that prefix is re-fed as prompt on each expansion. These two effects offset, so the total input+output token count is approximately equal to whole-proof generation at the same `proof` $\times$ `step` budget. The Lean REPL call and wall-clock statistics in Table 11 characterize the remaining verification and latency cost.

A.10 Analysis of AIME 24&25 Gains

The gains on the 15 AIME 24&25 problems are modest, as shown in Table 2. Analyzing the base model’s behavior on these problems reveals two failure clusters. In about half, the base model produces no viable proof prefix, with a median of 0 verified steps, while in the remainder it reaches 5–27 verified steps before an error. EBFS can only assist the latter, where a backtrackable prefix exists. This explains why the gain is modest: EBFS improves the localization of recoverable errors but cannot supply the missing mathematical insight that the hardest problems require, consistent with the limitations stated in the main text.

A.11 Relationship to MCTS

EBFS and Monte Carlo Tree Search (MCTS) are complementary. EBFS uses a local, single-pass entropy signal and deliberately does not backpropagate rewards to distant ancestors, because in our setting failures localize around high-uncertainty steps rather than distant ancestors, as discussed in the Backtracking part of Section 3.3. MCTS, in contrast, relies on value backup through the tree. A natural hybrid would treat the entropy peak count as an intrinsic-reward prior inside a UCT selection rule, combining EBFS’s uncertainty-based localization with MCTS’s value backpropagation. We leave this hybrid to future work.

A.12 EBFS Proof Search Examples

We present concrete examples of the EBFS proof search process on problems from each benchmark. Each figure shows multiple consecutive attempts: the first attempt fails at a certain point, EBFS identifies the highest-entropy step as the backtrack target, and the next attempt reuses the shared prefix while generating an alternative branch from the backtrack point.

A.13 Rollout Sample Example

We provide a concrete example below. Each rollout sample is a dictionary with four fields:

- **input:** The prompt context, consisting of the theorem statement, the model’s chain-of-thought reasoning, and the proof code generated in prior steps.
- **chosen:** The generated code segment that constitutes a valid step toward the final correct proof.
- **neutral:** Code segments that are valid but lie between the chosen path and the rejected point

	MiniF2F	ProofNet	ProverBench
Avg. Lean REPL calls	110	118	126
Avg. branches per proof	1.8	1.7	2.0
Avg. tokens of Lean code	2935	1516	3117
Wall-clock time	20h	16h	27h

Table 11: Systematic statistics of EBFS proof search on three benchmarks. Metrics are computed on the final proof trees that successfully complete a proof. The number of backtracks equals the number of branches minus 1.

since they neither contribute to nor detract from the proof.

- **rejected:** The first code segment where the Lean compiler reports an error.

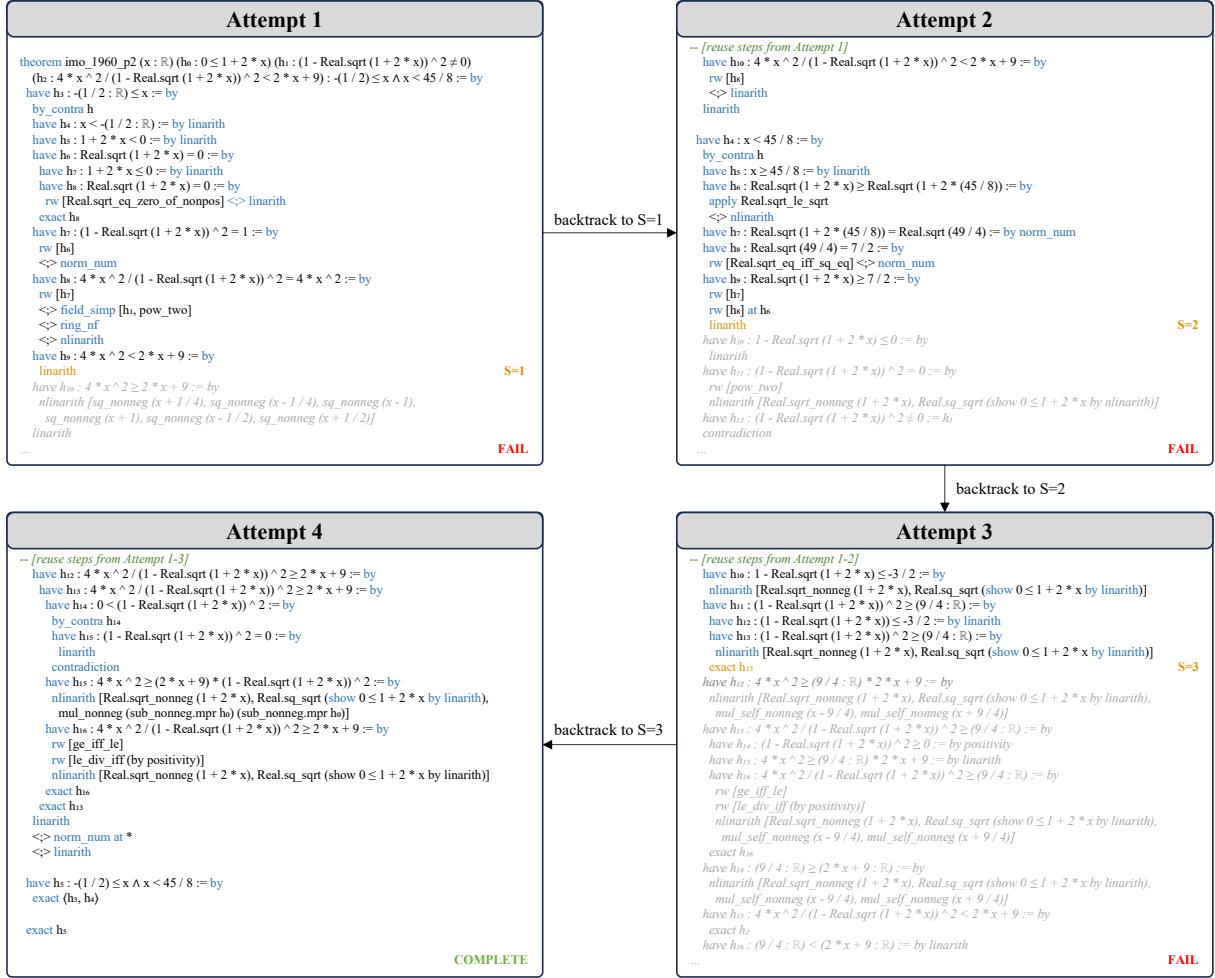


Figure 10: EBFS proof search example on MiniF2F.

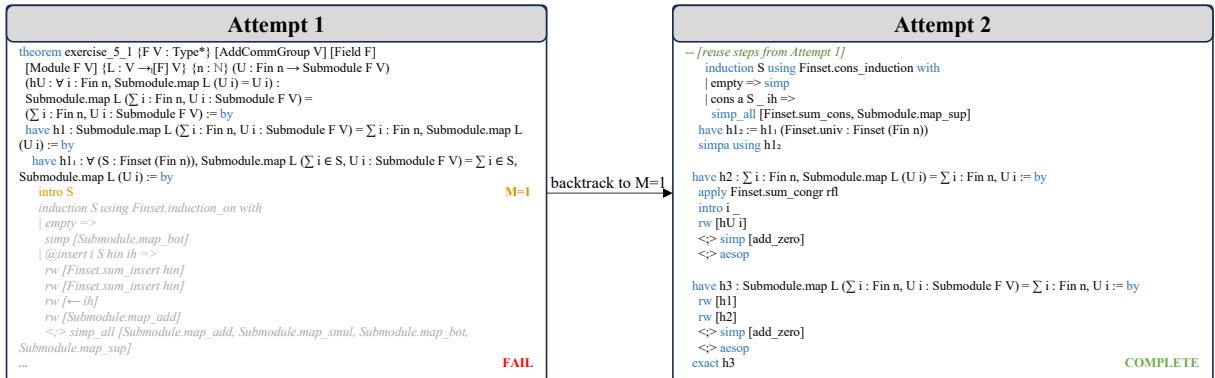


Figure 11: EBFS proof search example on ProofNet.

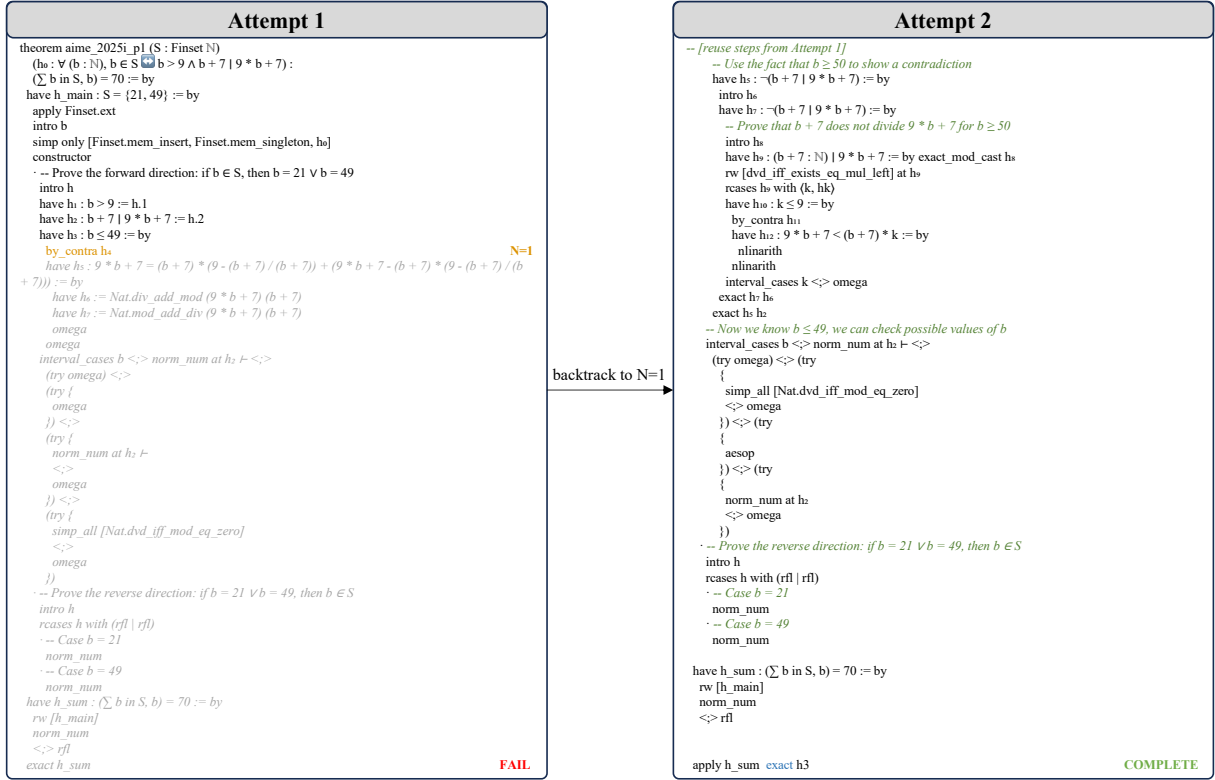


Figure 12: EBFS proof search example on ProverBench.

Rollout Sample (abridged)

```

{
  "input":
    "Complete the following Lean 4 code:..."
    (theorem statement + CoT reasoning + prior proof steps)
    have h2 : \forall k \in Finset.Icc 0 (n - 1), (x k)^2 \leq (n : R) := by
      intro k hk
      rw [h1 k]
      have h3 : (k : R) \geq 0 := by
        exact Nat.cast_nonneg k"

  "chosen":
    "have h4 : (n:R) - (k:R) >= 0 := by
      have h5 : k <= n - 1 := by simp [Finset.mem_Icc] at hk; omega
      have h6 : (k:R) <= (n:R) - 1 := by norm_num at hk | - ...",

  "neutral":
    "<> (try omega) <> (try linarith) <> (try norm_num)
    <> (try ring_nf at * <> linarith)",

  "rejected":
    "linarith"
}

```

Figure 13: An abridged rollout sample from the RFT training data. The input field contains the theorem and prior proof context. The chosen segment advances the proof correctly. The neutral segment is syntactically valid but inconclusive. The rejected segment (linarith) fails Lean verification at this proof state. Only chosen and rejected receive rewards; neutral is excluded from loss computation.