PUSHING THE MATH REASONING FRONTIER BY MAKING BEYOND-CAPABILITY PROBLEMS LEARNABLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have made substantial progress in mathematical reasoning, which is largely driven by reinforcement learning (RL). However, further pushing the mathematical reasoning frontier of LLMs through RL remains challenging, as models rarely generate successful rollouts on problems beyond their current capability frontier, resulting in a scarcity of effective learning signals. To address this challenge, we propose **LeanTree**, which decomposes beyond-capability problems into learnable subproblems. The major challenges in developing LeanTree are twofold. (1) LLM-based decompositions rely on stronger models and still introduce logical hallucinations. (2) The resulting subproblems lack a difficulty hierarchy, leaving the training order unclear. LeanTree leverages the intermediate reasoning structure of existing formal proofs (machine-verifiable representations of mathematical reasoning) to generate *reliable* and *hierarchical* subproblems. By training on these hierarchical subproblems, the model progressively acquires the reasoning skills needed to solve the corresponding beyond-capability problems. Extensive experiments on mathematical reasoning benchmarks demonstrate that LeanTree provides more effective learning signals and improves overall reasoning accuracy.

1 INTRODUCTION

Large language models (LLMs) have made substantial progress in mathematical reasoning, achieving strong performance even on challenging competition problems (DeepSeek-AI et al., 2025; Luong & Lockhart, 2025). Reinforcement learning with verifiable rewards (RLVR) has been a key driver of this progress, allowing models to improve by sampling solutions to mathematical problems and receiving feedback on answer correctness (Shao et al., 2024; DeepSeek-AI et al., 2025). However, for problems beyond a model's current capabilities, a practical rollout budget may yield few or no successful reasoning trajectories (Zhang et al., 2026; Mahrooghi et al., 2026). When all sampled solutions fail, algorithms such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) receive no reward contrast to guide the model toward better solutions (Yu et al., 2025). These problems consequently provide little useful feedback, even though they may contain the reasoning skills the model needs to acquire. Obtaining effective learning signals from such problems is therefore a central challenge in further pushing the mathematical reasoning frontier.

A natural approach is to decompose a difficult problem into more accessible subproblems (Zhou et al., 2023; Khot et al., 2023). Existing methods commonly use LLMs to generate these intermediate questions or reasoning steps. Even when a model cannot solve the original problem, training on accessible intermediate tasks can provide useful reward feedback and help it acquire the skills needed for the complete solution (Xi et al., 2026). This motivates data-centric approaches that use model-generated reasoning explanations (Hsieh et al., 2023; Mukherjee et al., 2023) or targeted synthetic problems (Liang et al., 2025) to improve reasoning.

However, using generated subproblems for training raises two challenges. First, LLM-generated decompositions can introduce logical hallucinations. A reliable decomposition should preserve the assumptions and intermediate reasoning that connect each subproblem to the original problem. In practice, generated mathematical tasks can omit necessary conditions or contain logical inconsistencies (Shen et al., 2025), and a checkable answer alone does not establish that this connection is valid. One way to obtain such supervision is to use a stronger model to generate or label training tasks (Liang

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2025; Xi et al., 2026). However, for a target model already near the reasoning frontier, a reliably stronger teacher may not be available (Liang et al., 2025). Second, a valid decomposition does not by itself specify an effective training order. Some subproblems may remain beyond the learner's reach, while others may already be too easy to offer useful learning signals (Mahrooghi et al., 2026). A suitable training order can help the model build intermediate skills before returning to the original problem (Xi et al., 2026). These challenges motivate our central question: **How can we *construct reliable subproblems* and *organize them* to make beyond-capability problems learnable?**

To address this question, we propose **LeanTree**, which decomposes beyond-capability problems into learnable subproblems. The key idea of LeanTree is to leverage existing formal proofs as a source of *verifiable* intermediate goals and *hierarchical* reasoning structure. Lean is an interactive theorem prover that represents mathematical reasoning in a machine-checkable language (de Moura & Ullrich, 2021). In Lean, the intermediate goals of a proof can be organized into a tree, with the original problem at the root and each node specifying a conclusion under explicit assumptions. This structure provides a formal basis for decomposition without requiring a stronger LLM to discover the intermediate reasoning steps. LeanTree converts these goals into self-contained natural-language subproblems and organizes them into a curriculum using their proof-tree hierarchy. Using proof-tree depth as a practical proxy for difficulty, this curriculum progresses from deeper, local subproblems toward the original root problems. The model can thus acquire intermediate reasoning skills before learning to solve the complete problem.

Experiments on mathematical reasoning benchmarks demonstrate that LeanTree improves reasoning performance across different model backbones and RL algorithms. Beyond benchmark gains, the analyses support the intended learning process: accessible intermediate tasks provide more informative rewards, and the depth-based curriculum helps the model build prerequisite skills before returning to the original difficult problems. The improvements on these original problems highlight the value of using formal proof structure to turn beyond-capability tasks into opportunities for progressive learning.

Our contributions are threefold:

- We introduce LeanTree, an offline synthesis pipeline that reconstructs goal-state trees from existing Lean proofs and converts their intermediate goals into reliable training tasks for beyond-capability problems.
- We develop a within-proof curriculum that uses proof-tree depth to order these tasks from local intermediate goals toward the original difficult problems.
- We demonstrate improved mathematical reasoning across different backbones and provide evidence linking intermediate-task learning to more informative rewards and better performance on the original difficult problems.

2 Preliminaries

In this section, we introduce the formal foundations and algorithms that underpin our framework. We first introduce Lean 4 (de Moura & Ullrich, 2021), highlighting its mechanical verifiability and intrinsic tree structure. We then briefly review Group Relative Policy Optimization (GRPO) (Shao et al., 2024), the core RL algorithm utilized for our mathematical reasoning curriculum.

2.1 Formal Theorem Proving and Lean 4

Lean 4 is an interactive theorem prover (ITP) and functional programming language designed to formalize mathematics. Unlike natural language reasoning, which relies on implicit human understanding and is highly susceptible to logical gaps or hallucinations, Lean formalizes mathematical statements into strict types and proofs into verifiable programs. To illustrate its mechanics, consider the following formal proof snippet demonstrating that $x^2 + y^2 = 0$ implies both $x = 0$ and $y = 0$:

```
theorem sq_add_sq_eq_zero (x y : ℝ)
    (h : x^2 + y^2 = 0) : x = 0 ∧ y = 0 := by
  -- Split the main goal into two parallel sub-goals
```

```
constructor
· -- Sub-goal 1: Prove x = 0
  have h1 : x^2 ≤ 0 := by linarith [sq_nonneg y]
  -- ... (remaining steps to deduce x = 0 from h1)
· -- Sub-goal 2: Prove y = 0
  have h2 : y^2 ≤ 0 := by linarith [sq_nonneg x]
  -- ... (remaining steps to deduce y = 0 from h2)
```

Mechanical Verifiability. Every tactic step in a Lean proof (e.g., `constructor` or `linarith`) is strictly validated by the compiler via rigorous type checking. A proof either fully compiles, assuring absolute mathematical correctness, or fails. In the example above, if the `linarith` tactic cannot mathematically deduce the intermediate lemma `h1` from the premise `h`, the compilation halts immediately. This deterministic verification eliminates the need for external heuristic validation. Consequently, any intermediate goal state extracted from a compiled Lean proof is strictly valid and free of logical hallucinations, providing a pristine source of ground-truth data.

Tree Structure. While natural language proofs are typically presented as linear sequences of text, a formal proof in Lean naturally manifests as a tree. At any given point in a proof, the state is defined by a goal $\mathcal{G} = (\mathcal{H}, \mathcal{C})$, consisting of a set of local hypotheses (premises) $\mathcal{H}$ (e.g., `h : x^2 + y^2 = 0`) and a target conclusion $\mathcal{C}$ (e.g., `x = 0 ∧ y = 0`). The application of mathematical tactics recursively decomposes this root goal into new nodes, explicitly constructing a proof tree. As illustrated in the code snippet above, the initial premise `h` can be conceptualized as the root node of this tree, representing the overarching problem to be solved. To prove the conjunction ($\wedge$), the `constructor` tactic decomposes this root conclusion into two completely independent, parallel subproblems (denoted by the · bullets): proving $x = 0$ and proving $y = 0$. Consequently, the `have` tactic within each branch generates intermediate lemmas `h1` ($x^2 \leq 0$) and `h2` ($y^2 \leq 0$), which serve as two distinct child nodes branching out from the root node `h`. These child nodes represent self-contained, intermediate milestones required to conquer the parent problem.

Because of this rigid structure, each non-root node in the proof tree represents a self-contained, rigorously verified subproblem. The *depth* of a node naturally correlates with its mathematical difficulty: deeper nodes represent foundational subproblems, while shallower nodes (or the root itself) demand chaining multiple reasoning steps. This inherent topological property provides a systematic metric to gauge problem difficulty, making it uniquely suited for constructing curricula.

2.2 Group Relative Policy Optimization (GRPO)

To align LLMs with desired behaviors, recent research formulates language modeling as an RL problem, where the model acts as a policy π_θ that interacts with an environment by generating a reasoning path and final answer $\mathbf{y}$ given a query $\mathbf{x} \sim \mathcal{D}$. Each response receives a reward $r(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$ that reflects its quality.

GRPO circumvents the need for a standalone critic model by estimating the baseline directly from a group of sampled responses. Specifically, for each query $\mathbf{x}$, GRPO samples a group of G sampled responses $\{\mathbf{y}^{(i)}\}_{i=1}^{G}$ from the old policy $\pi_{\theta_{\text{old}}}(\cdot \mid \mathbf{x})$ and assigns each response a scalar score $r^{(i)} = r(\mathbf{x}, \mathbf{y}^{(i)})$ using a rule-based reward model. The advantage $A^{(i)}$ for the i-th response is then calculated by normalizing the rewards within the sampled group:

$$A^{(i)} = \frac{r^{(i)} - \text{mean}\left(\{r^{(j)}\}_{j=1}^{G}\right)}{\text{std}\left(\{r^{(j)}\}_{j=1}^{G}\right)}. \tag{1}$$

Defining $\text{clip}_a^b(x) = \max(a, \min(x, b))$ as the standard clipping function, the policy parameters θ are updated by maximizing the following surrogate objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\{\mathbf{y}^{(i)}\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|\mathbf{x})} \left[\frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(\mathbf{y}^{(i)} \mid \mathbf{x})}{\pi_{\theta_{\text{old}}}(\mathbf{y}^{(i)} \mid \mathbf{x})} A^{(i)}, \text{clip}_{1-\varepsilon}^{1+\varepsilon}\left(\frac{\pi_\theta(\mathbf{y}^{(i)} \mid \mathbf{x})}{\pi_{\theta_{\text{old}}}(\mathbf{y}^{(i)} \mid \mathbf{x})} \right) A^{(i)} \right) - \beta \mathbb{D}_{\text{KL}}^{(i)}(\pi_\theta \| \pi_{\text{ref}}) \right], \tag{2}$$

where ε is the clipping threshold determining the trust region, β is a hyperparameter, and $\mathbb{D}_{\mathrm{KL}}^{(i)}(\pi_\theta \| \pi_{\mathrm{ref}})$ represents the Kullback-Leibler divergence penalty between the current policy π_θ and a reference policy π_{ref} to prevent excessive deviation.

By evaluating multiple reasoning paths and leveraging relative advantages, GRPO provides a highly efficient and critic-free policy optimization mechanism. However, its efficacy is fundamentally contingent on the variance of rewards within the sampled group. When tasked with excessively complex problems, the model often fails to produce any correct reasoning paths across all G samples (i.e., $r^{(i)} = -1$ for all i). In such cases, because a small constant (e.g., 10^{-4}) is typically added to the denominator in practice to prevent division by zero, the relative advantages $A^{(i)}$ uniformly collapse to zero. This leads to vanishing policy gradients and severely stagnated learning. This critical vulnerability inherently underpins the exploration bottleneck on hard reasoning tasks, highlighting the necessity for a curriculum-driven approach.

3 METHOD

LeanTree uses the intermediate reasoning structure of existing formal proofs to construct reliable subproblems and organize them for progressive learning. Figure 1 summarizes its three stages. First, we reconstruct proof trees from the Lean goal states of problems that the target model fails to solve (Section 3.1). Second, we prune trivial goals and translate the retained goals into self-contained natural-language problems with checkable answers (Section 3.2). Finally, we use proof-tree depth to organize these problems into a curriculum that progresses from local intermediate goals to the original problems (Section 3.3).

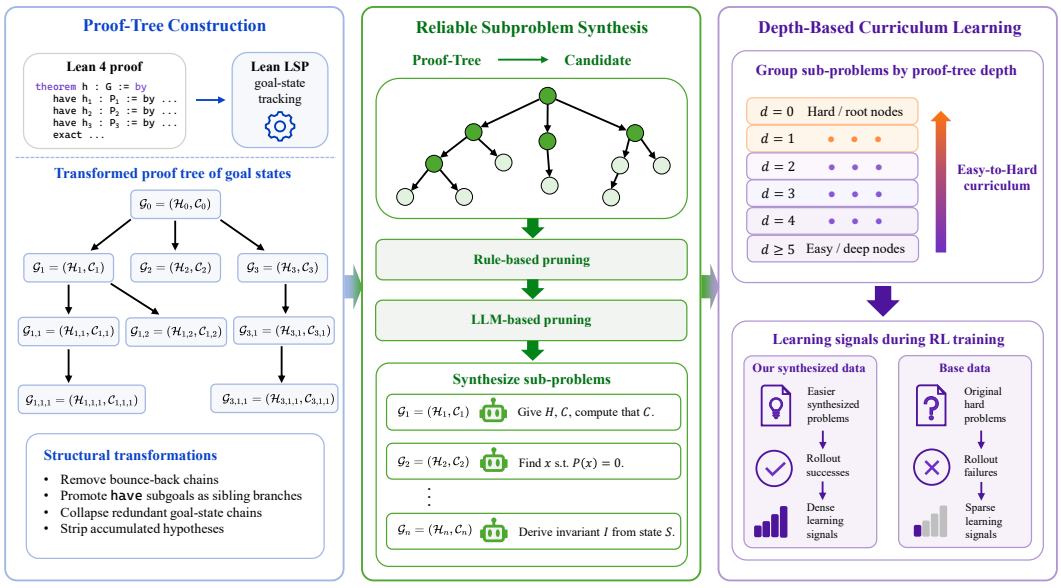


Figure 1: **Overview of LeanTree.** (1) Proof-tree construction extracts intermediate goals and their hierarchical structure from existing Lean proofs. (2) Reliable subproblem synthesis prunes trivial goals and translates retained goals into self-contained natural-language problems with checkable answers. (3) Depth-based curriculum learning orders these problems from deeper subproblems to the original root problems.

3.1 PROOF-TREE CONSTRUCTION

We first evaluate the target model on the original training set, denoted by $\mathcal{D}_{\mathrm{base}}$. We collect the problems it fails to solve into $\mathcal{D}_{\mathrm{hard}}$ and use these observed failures to identify candidate beyond-capability problems. For each problem in $\mathcal{D}_{\mathrm{hard}}$ with an available Lean proof, we extract its intermediate goals and reconstruct their hierarchical structure. This provides a decomposition

grounded in an existing formal derivation, without requiring an LLM to discover the intermediate reasoning steps.

Goal State Tracking. We use a Lean Language Server Protocol (LSP) client to query the `$/lean/plainGoal` endpoint at successive positions in the proof script. The returned goals record how tactic applications change the proof state. Following Section 2.1, we represent each goal as $\mathcal{G} = (\mathcal{H}, \mathcal{C})$, where $\mathcal{H}$ contains the local variables and hypotheses, and $\mathcal{C}$ is the target conclusion. Retaining both components makes the assumptions associated with each intermediate goal explicit.

Structural Tree Construction. We first construct a raw tree from the sequence of goal states, tracking the creation and resolution of subgoals. Branching tactics such as `constructor` and `cases` generate multiple subgoals, while `have` introduces an intermediate lemma that may be used by the continuation of the proof. We represent a `have` subgoal and the continuation as sibling branches, with their logical dependence recorded in the continuation's hypotheses.

The raw trace also contains transitions that reflect proof execution rather than mathematical decomposition. As summarized in Algorithm 1, we remove spurious sequential links created when execution returns from a completed branch, promote `have` subgoals to sibling branches, and collapse consecutive non-branching states with high textual similarity. For the latter, we use a trigram Jaccard similarity threshold of 0.7. We also remove unused accumulated hypotheses based on proof dependencies, retaining the premises used by each subgoal. These operations produce a compact proof tree $\mathcal{T}$ whose nodes provide candidate intermediate tasks. Details are in Appendix A.1.

Algorithm 1 Proof-Tree Construction from Lean Goal States

Input: Lean proof script P, Lean workspace $\mathcal{W}$
Initialize Lean LSP client $\mathcal{L}$ in $\mathcal{W}$
$States \leftarrow []$
for each line l in P **do**
  $S_l \leftarrow \mathcal{L}.\texttt{plain_goal_at}(l)$ {Query the goals at this position}
  if S_l is not null **then**
    $States.\texttt{append}((l, S_l))$
  end if
end for
$\mathcal{T}_{\text{raw}} \leftarrow \texttt{BuildRawTree}(States)$ {Track subgoal creation and resolution}
$\mathcal{T} \leftarrow \texttt{RemoveBounceBackChains}(\mathcal{T}_{\text{raw}})$
$\mathcal{T} \leftarrow \texttt{PromoteHaveSiblings}(\mathcal{T})$
$\mathcal{T} \leftarrow \texttt{CollapseSimilarChains}(\mathcal{T}, \text{threshold} = 0.7)$
$\mathcal{T} \leftarrow \texttt{StripAccumulatedHypotheses}(\mathcal{T})$ {Remove unused inherited hypotheses}
Output: Proof tree $\mathcal{T}$

3.2 Reliable Subproblem Synthesis

The reconstructed proof tree provides explicit intermediate goals, but some nodes represent trivial rewrites or Lean-specific bookkeeping. We therefore select goals that involve useful mathematical reasoning and convert them into natural-language training tasks.

Rule-Based and LLM-Based Pruning. We collect the non-root nodes of $\mathcal{T}$ as candidate subproblems, retaining their proof-tree depths for curriculum construction. A rule-based filter removes excessively deep nodes, very short or tautological goals, and tasks with binary answers. These criteria serve as practical filters for low-level proof steps and tasks that are easily solved by guessing; the exact rules, including the filter for explicit `sorry` markers, are provided in Appendix A.2. We then use DeepSeek-V3.2 (Liu et al., 2025) to identify goals that require more than direct substitution, definition unfolding, or routine bookkeeping. The pruning prompt favors goals involving mathematical techniques such as algebraic identities, inequalities, counting arguments, and number-theoretic reasoning (Appendix A.3).

Natural-Language Translation. For each retained goal $\mathcal{G}_i = (\mathcal{H}_i, \mathcal{C}_i)$, we prompt an LLM to produce a self-contained problem $\mathbf{x}_i$ and a reference answer a_i. The prompt instructs the model to preserve the variables, domains, and hypotheses in $\mathcal{H}_i$, and to express the target in $\mathcal{C}_i$ as a concrete computation question. For example, a goal with conclusion $x = c$ can yield a question asking for the

value of x under the stated hypotheses, with c as the reference answer. This conversion makes the intermediate goals compatible with the answer-based reward used for natural-language mathematical reasoning. The translation template is given in Appendix A.4.

The formal goals supply explicit conditions and targets for synthesis, reducing the need for an LLM to infer a decomposition from scratch. The LLM-based pruning and translation are not themselves formally verified.

3.3 Depth-Based Curriculum Learning

The synthesized subproblems provide intermediate training tasks, while the proof tree supplies a basis for ordering them. We use node depth as a practical proxy for difficulty: deeper goals tend to involve more local reasoning, whereas shallower goals combine intermediate steps toward the original problem. This motivates a curriculum that moves from deeper subproblems to the root.

Curriculum Construction. We augment $\mathcal{D}_{\text{base}}$ with the synthesized subproblems to form $\mathcal{D}_{\text{enhanced}}$. Each synthesized example retains the depth of its source node in the reconstructed tree, and original problems are assigned depth 0. Let $\mathcal{D}_d$ denote the examples at depth d, and let $d_{\max}$ be the cutoff at or above which depths are grouped into a single initial phase. The curriculum follows the order

$$\mathcal{D}_{\geq d_{\max}} \longrightarrow \mathcal{D}_{d_{\max}-1} \longrightarrow \cdots \longrightarrow \mathcal{D}_1 \longrightarrow \mathcal{D}_0. \tag{3}$$

We use $d_{\max} = 5$, so training begins with retained nodes at depth 5 or greater and proceeds through decreasing depths to the original problems.

Learning from Intermediate Tasks. We apply this ordering to RL training, using GRPO as described in Section 2.2. For a hard problem on which all sampled responses receive the same incorrect-answer reward, the reward-based group-relative advantages are zero. Accessible intermediate tasks can instead yield a mixture of correct and incorrect responses, providing within-group reward differences that guide policy updates. The model can use this feedback to develop intermediate reasoning skills before training on shallower tasks that require combining them. The final phase returns to the original problems, where the curriculum aims to improve the model's ability to obtain successful rollouts and learn from their rewards.

4 Experiments

Our experiments address three questions: **(1)** Can the subproblems generated by LeanTree improve mathematical reasoning compared with direct RL and synthesis baselines? **(2)** Do they provide more informative reward signals during RL? **(3)** How does depth-based curriculum ordering contribute to performance? We also conduct supplementary analyses of proof depth and empirical difficulty, subproblem fidelity, and synthesis cost, reported in Appendices C.1, C.2, and C.4.

4.1 Experimental Setup

Training Data. We construct the training data from NuminaMath-LEAN (Wang et al., 2025), which pairs mathematical problems with Lean formal statements and proof-related annotations. We filter out pure proof problems and examples without reliable final answers, using LLM-assisted answer extraction where necessary. The resulting original problem set is denoted as **Base Data**. **Our Data** augments Base Data with the natural-language subproblems synthesized by LeanTree from the available proofs of problems the initial model fails to solve. Details of data construction and filtering are provided in Appendix B.1.

Benchmarks and Models. We evaluate on GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021; Lightman et al., 2024), AMC 2023 (Mathematical Association of America, 2023), and AIME 2025 (Mathematical Association of America, 2025). We split MATH500 into levels 1–3 and 4–5 to examine performance at different difficulty levels. We use two backbones: Qwen2.5-7B-Instruct (Qwen et al., 2025), initialized with supervised fine-tuning on Mixture-of-Thoughts (Face, 2025), and DeepSeek-R1-Distill-Llama-8B (DeepSeek-AI et al., 2025), initialized directly from the official distilled checkpoint without additional SFT. **Vanilla** denotes each backbone's initialization before our RL training. Benchmark details are given in Appendix B.2.

Baselines. We compare LeanTree with direct RL on **Base Data** and two synthesis baselines: **Prompt LLM**, which prompts an LLM to decompose hard problems without formal proof structure, and **SwS** (Liang et al., 2025), which synthesizes problems based on model weaknesses. For Qwen, all synthesis methods are trained with both GRPO (Shao et al., 2024) and BAPO (Xi et al., 2025); for Llama, we compare them under GRPO. Within each comparison, methods share the same initialization and optimizer with comparable rollout budgets. Baseline details and offline synthesis costs are reported in Appendices B.3 and C.4.

Training and Evaluation Settings. During RL, we use 8 rollouts per prompt, a maximum response length of 8192, temperature 1.0, learning rate 1×10^{-6}, and a default global batch size of 256. Correct answers receive $+1$; incorrect, unverifiable, or format-violating answers receive -1. For evaluation, we use vLLM (Kwon et al., 2023) and ask models to reason before presenting a final answer in `\boxed{}`, which is checked against the reference after rule-based normalization. We repeat evaluation 8 times on GSM8K and MATH500 and sample 16 responses per problem on AMC 2023 and AIME 2025. We report mean accuracy and standard deviation from evaluation sampling, with an unweighted average over the five benchmark columns. Full training and evaluation settings, including the smaller-batch learning-curve diagnostics, are given in Appendices B.4 and B.5.

4.2 Main Results

Table 1: **Mathematical reasoning accuracy (%).** Each block compares methods with the same backbone and optimizer. Entries report mean $\pm$ standard deviation under the repeated-evaluation protocol. Bold marks the highest mean in each block, including ties.

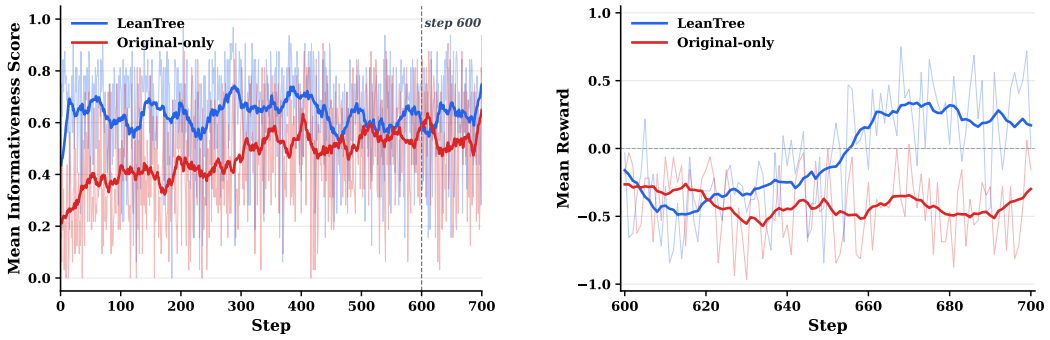
Method	GSM8K	MATH500		AMC 2023	AIME 2025	Average
		L1–3	L4–5			
Qwen2.5-7B-Instruct / GRPO						
Vanilla	89.37 ± 0.18	93.33 ± 0.65	67.94 ± 1.77	60.78 ± 3.92	20.63 ± 3.77	66.41
GRPO + Base Data	91.94 ± 0.14	95.22 ± 0.29	75.38 ± 1.90	62.19 ± 4.50	21.04 ± 3.67	69.15
Prompt LLM	92.94 ± 0.17	94.22 ± 0.28	72.28 ± 2.12	62.50 ± 4.51	22.50 ± 3.44	68.89
SwS	93.05 ± 0.13	95.17 ± 0.21	75.52 ± 1.74	63.28 ± 3.72	22.92 ± 3.70	69.99
LeanTree	$\mathbf{93.39} \pm 0.18$	$\mathbf{96.43} \pm 0.21$	$\mathbf{77.15} \pm 2.64$	$\mathbf{65.00} \pm 4.24$	$\mathbf{23.75} \pm 3.51$	**71.14**
Qwen2.5-7B-Instruct / BAPO						
Vanilla	89.37 ± 0.18	93.33 ± 0.65	67.94 ± 1.77	60.78 ± 3.92	20.63 ± 3.77	66.41
BAPO + Base Data	91.74 ± 0.12	94.59 ± 0.25	74.86 ± 2.11	62.19 ± 4.23	22.71 ± 3.58	69.22
Prompt LLM	93.05 ± 0.15	93.70 ± 0.30	74.71 ± 2.09	63.12 ± 4.00	23.54 ± 3.63	69.62
SwS	92.98 ± 0.09	94.54 ± 0.59	76.81 ± 1.92	63.59 ± 4.42	$\mathbf{24.17} \pm 3.63$	70.42
LeanTree	$\mathbf{93.58} \pm 0.14$	$\mathbf{95.54} \pm 0.42$	$\mathbf{80.15} \pm 2.18$	$\mathbf{65.94} \pm 4.32$	$\mathbf{24.17} \pm 3.23$	**71.88**
DeepSeek-R1-Distill-Llama-8B / GRPO						
Vanilla	85.99 ± 0.63	90.60 ± 0.78	71.04 ± 1.24	66.88 ± 5.12	19.58 ± 3.51	66.82
GRPO + Base Data	86.70 ± 0.66	93.80 ± 0.58	77.43 ± 0.90	78.75 ± 4.59	22.92 ± 3.31	71.92
Prompt LLM	87.45 ± 0.49	93.28 ± 0.73	76.81 ± 1.01	77.50 ± 4.51	22.29 ± 3.67	71.47
SwS	87.68 ± 0.57	93.70 ± 0.79	77.91 ± 1.31	78.91 ± 4.50	24.17 ± 3.63	72.47
LeanTree	$\mathbf{89.77} \pm 0.67$	$\mathbf{95.48} \pm 0.69$	$\mathbf{80.63} \pm 1.41$	$\mathbf{80.94} \pm 4.50$	$\mathbf{25.42} \pm 4.70$	**74.45**

LeanTree improves mathematical reasoning across backbones and optimizers. Table 1 presents the main results. On Qwen, LeanTree improves average accuracy over direct RL on Base Data by 1.99 percentage points with GRPO and 2.66 points with BAPO. It also exceeds SwS by 1.15 and 1.46 points under the same respective optimizers, with gains of 1.63 and 3.34 points on MATH500 (L4–5). On AIME 2025, LeanTree ties SwS at 24.17% under BAPO. With Llama and GRPO, LeanTree achieves 74.45% average accuracy, exceeding direct RL by 2.53 points and SwS by 1.98 points, with the highest mean in all five benchmark columns. These comparisons support the effectiveness of the synthesis and curriculum pipeline beyond a single backbone or optimizer.

LeanTree provides more informative RL signals. Figure 2 compares Qwen/GRPO training on Our Data and Base Data under matched batch sizes, rollout group sizes, and training steps. For each prompt, let $\bar{r}$ be the mean reward over its 8 rollouts. We define the reward-informativeness score as $1 - |\bar{r}|$ and average it over prompts at each step. With rewards in $\{-1, +1\}$, the score is zero when all rollouts have the same outcome: the within-group reward variance and reward-based relative advantages vanish, leaving GRPO without an effective reward-driven learning signal for

that prompt. The score is maximal when correct and incorrect outcomes are balanced, yielding the greatest within-group reward variance and the strongest learning signal under this reward-contrast measure. Figure 2a shows that LeanTree maintains higher informativeness over most of training.

Figure 2b focuses on steps 600–700, when both methods train exclusively on the difficult original root problems. When LeanTree transitions to these problems around step 600, its mean reward experiences a brief dip. Before this transition, the model has already developed prerequisite *sub-skills* by learning from proof-derived subproblems in the earlier curriculum phases. This preparation helps it adapt rapidly: its mean reward recovers, rises sharply, and consistently surpasses the baseline during the later part of this shared root-training phase. This pattern is consistent with bottom-up capability construction, in which the model learns to combine intermediate reasoning skills to solve the original difficult problems. Appendix D provides the per-question reward distribution.



(a) Reward informativeness (steps 0–700).

(b) Mean reward on root problems (steps 600–700).

Figure 2: **Qwen/GRPO training dynamics under matched rollout budgets.** Faint lines show raw measurements; solid lines use smoothing windows of 25 steps in (a) and 15 steps in (b). During steps 600–700, both methods train exclusively on the difficult original root problems.

Benefits extend to held-out root problems. In a separate Qwen/GRPO diagnostic evaluation, we split by original problem ID before synthesis, keeping each root and its derived subproblems in the same split (Appendix C.1). On held-out roots, LeanTree achieves 28.03% accuracy, compared with 24.02% for direct RL and 26.60% for a size-matched Prompt LLM baseline: gains of 4.01 and 1.43 percentage points, respectively. These results support benefits on the original problems beyond increasing training-set size alone. Natural-language conversion fidelity is assessed separately in Appendix C.2.

4.3 Ablations

Depth-based ordering contributes to the gains. We compare the full Qwen/GRPO pipeline with a variant that randomly shuffles the same Our Data. The full curriculum progresses from deeper subproblems toward the original problems, using the order $\geq 5 \to 4 \to 3 \to 2 \to 1 \to 0$. Appendix C.1 evaluates the empirical relationship between depth and difficulty; Appendix C.3 gives the curriculum and ablation evaluation details.

Table 2: **Effect of depth-based ordering with Qwen and GRPO.** Entries report mean ± standard deviation; Full uses the results from Table 1. Bold marks the higher mean accuracy (%).

Strategy	GSM8K	MATH500		AMC 2023	AIME 2025	Average
		L1–3	L4–5			
LeanTree (Random)	92.82 ± 0.12	94.64 ± 0.28	73.19 ± 2.18	62.03 ± 4.17	21.88 ± 2.63	68.91
LeanTree (Full)	**93.39** ± 0.18	**96.43** ± 0.21	**77.15** ± 2.64	**65.00** ± 4.24	**23.75** ± 3.51	**71.14**

Table 2 shows average accuracy of 71.14% for the full curriculum and 68.91% for random ordering, a gain of 2.23 percentage points. The gains are 3.96 points on MATH500 (L4–5), 2.97 points on AMC 2023, and 1.87 points on AIME 2025. These results support ordering intermediate tasks before returning to the original problems.

5 Related Work

Decomposition for Complex Reasoning. Complex reasoning tasks are often addressed via problem decomposition to reduce difficulty. Prevailing approaches to achieve this typically fall into two paradigms: leveraging external stronger models or relying on the model's internal heuristics. On the one hand, approaches utilizing external models attempt to extract rationales from superior teachers to guide smaller models (Hsieh et al., 2023), or leverage rich explanation traces generated by frontier models to enhance reasoning capabilities (Mukherjee et al., 2023). On the other hand, heuristic approaches like Chain-of-Thought (CoT) prompting attempt to elicit intermediate steps directly from the LLMs (Wei et al., 2022). Building on this, Tree-of-Thought (ToT) explores branching reasoning structures for more effective heuristic search (Yao et al., 2023). However, both paradigms suffer from critical limitations, namely the ceiling effect when relying on external stronger models and logical hallucinations in heuristic decomposition. Thus, obtaining structurally correct and reliable problem decompositions without being bottlenecked by external ceilings or misguided by internal hallucinations remains a fundamental challenge.

Formal Systems for Mathematical Reasoning. Recent work has increasingly leveraged formal systems to advance mathematical reasoning. AlphaProof (Hubert et al., 2026) combines reinforcement learning with the Lean theorem prover to discover machine-verifiable formal proofs at the Olympiad level. DeepSeek-Prover-V2 (Ren et al., 2025) further incorporates recursive subgoal decomposition to improve formal theorem proving through reinforcement learning. Another line of work explores autoformalization, translating natural-language mathematical problems into formal representations so that they can benefit from formal verification (Wu et al., 2022). While these approaches primarily target formalization or formal theorem proving, LeanTree instead leverages the structure of existing formal proofs as supervision for improving informal mathematical reasoning.

6 Conclusion, Discussion, and Limitations

Conclusion. We introduced LeanTree, which uses the intermediate reasoning structure of existing Lean proofs to construct reliable subproblems and organize them into a depth-based curriculum. By connecting formal proof structure with natural-language RL, LeanTree offers a new route to making beyond-capability problems learnable. Experiments show higher average mathematical reasoning accuracy than the compared baselines with GRPO on both Qwen2.5-7B-Instruct and DeepSeek-R1-Distill-Llama-8B, and with BAPO on Qwen. The training dynamics and curriculum ablation support the value of combining proof-derived subproblems with progressive learning.

Dependence on stronger teachers. LeanTree separates the source of the decomposition from the model used to express it as training data. Intermediate assumptions, targets, and structure come from existing Lean proofs; the synthesis LLM selects useful goals and converts them into natural-language questions and reference answers. It therefore need not discover the intermediate reasoning steps on its own to supply a decomposition. This makes formal proof structure a promising supervision source near the reasoning frontier, where a reliably stronger teacher may be unavailable. Correctness alone, however, is insufficient: intermediate tasks must remain informative for the learner, producing successful and unsuccessful rollouts rather than uniformly correct or incorrect outcomes. Our experiments use DeepSeek-V3.2 for synthesis and 7B/8B learners; they do not test weaker synthesis models or establish gains for frontier models. Those settings remain open empirical questions.

Limitations. The limited availability of natural-language problems paired with complete Lean proofs constrains the coverage and scale of LeanTree's synthesized supervision, limiting the gains attainable in the current setting (Appendix B.1). In addition, the LLM-based conversion into natural language does not provide an end-to-end formal guarantee. Our fidelity audit found that $298/300$ examples (99.3%) passed all checks, providing empirical support for the high fidelity of the generated subproblems (Appendix C.2). Even with limited proof coverage, LeanTree improves average accuracy over direct RL and the synthesis baselines in all three evaluated backbone–optimizer settings (Section 4.2). These gains demonstrate the practical value of its central contribution: turning existing formal proof structure into intermediate supervision and an effective learning curriculum. Future work will focus on constructing larger and more diverse datasets pairing natural-language problems with complete Lean proofs. We expect broader proof coverage to expand the range of difficult problems that LeanTree can make learnable and enable larger gains in mathematical reasoning.

AI USE STATEMENT

In this work, we used generative AI for data curation, synthetic training-data generation, and quality assessment. LLMs assisted answer extraction and data filtering; DeepSeek-V3.2 was used to prune candidate goals and translate retained Lean goals into natural-language subproblems and reference answers (Section 3.2; Appendices B.1, A.3, and A.4). We used GPT-5.6-Sol as an independent evaluator of subproblem fidelity (Appendix C.2). We also used AI agents and LLMs to assist code writing, refactoring, debugging, manuscript drafting and editing, and reference lookup. To assess the generated data, we audited a stratified sample of 300 subproblems and manually spot-checked a subset of the evaluator's judgments. Benchmark answers were evaluated using the rule-based procedures described in Appendix B.5. The model-based audit provides empirical evidence rather than a formal guarantee of correctness. We take responsibility for the final content of this work, including its text, implementation, scientific claims, reported results, and artifacts produced with generative AI assistance.

ETHICS STATEMENT

This work studies mathematical reasoning using publicly available mathematical datasets and model checkpoints. Potential benefits include improved mathematical problem solving and more inspectable training data. Ethical concerns include errors in synthesized problems or reference answers, overreliance on model-based evaluation, and the misuse of improved reasoning capabilities. We document data sources, filtering, benchmark-overlap removal, and evaluation procedures in Appendix B, and assess synthesis fidelity in Appendix C.2. These checks do not establish an end-to-end correctness guarantee for the natural-language data or trained models. Use and redistribution of source datasets, model checkpoints, and derived artifacts should respect their applicable licenses and terms. We recommend human oversight when applying the resulting models in consequential settings.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 3 and Algorithm 1 describe the LeanTree pipeline, with proof-tree transformations and filtering rules provided in Appendix A. We provide prompts for LLM-based pruning and natural-language translation (Appendices A.3 and A.4), answer extraction and baseline decomposition (Appendices B.1 and B.3), and Qwen evaluation (Appendix B.5). Appendix B.1 documents the source dataset, data-construction procedure, and filtering statistics. Model initialization, baselines, training hyperparameters, and evaluation procedures are described in Section 4.1 and Appendices B.3, B.4, and B.5. Supplementary diagnostic and ablation protocols are given in Appendices C.1 and C.3, and the synthesized-data fidelity audit is described in Appendix C.2. Upon acceptance, we will open-source our implementation, including data-processing, training, and evaluation scripts, together with the experiment configurations.

REFERENCES

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

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In

Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.02406.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Yelong Shen, Ying Nian Wu, and Weizhu Chen. SwS: Self-aware weakness-driven problem synthesis in reinforcement learning for LLM reasoning, 2025. URL https://arxiv.org/abs/2506.08989.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Thang Luong and Edward Lockhart. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. Google DeepMind, July 2025.

Ilia Mahrooghi, Aryo Lotfi, and Emmanuel Abbe. Goldilocks RL: Tuning task difficulty to escape sparse rewards for reasoning, 2026. URL https://arxiv.org/abs/2602.14868.

Mathematical Association of America. American mathematics competitions (AMC) 2023, 2023. URL https://maa.org/student-programs/amc/.

Mathematical Association of America. American invitational mathematics examination (AIME) 2025, 2025. URL https://maa.org/maa-invitational-competitions/.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4, 2023. URL https://arxiv.org/abs/2306.02707.

OpenAI. GPT-5.6 Sol model. https://developers.openai.com/api/docs/models/gpt-5.6-sol, n.d. OpenAI API documentation. Accessed: 2026-09-23.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL https://arxiv.org/abs/2504.21801.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Chengyu Shen, Zhen Hao Wong, Runming He, Hao Liang, Meiyi Qiang, Zimo Meng, Zhengyang Zhao, Bohan Zeng, Zhengzhou Zhu, Bin Cui, and Wentao Zhang. Let's verify math questions step by step, 2025. URL https://arxiv.org/abs/2505.13903.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. 2025. URL http://arxiv.org/abs/2504.11354.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in neural information processing systems*, 35:32353–32368, 2022.

Ningyuan Xi, Hao Xu, Hongsheng Xin, and Ning Miao. TD-Grokking: Learning from zero-reward problems by training-time decomposition, 2026. URL https://arxiv.org/abs/2606.09883.

Zhiheng Xi, Xin Guo, Yang Nan, Enyu Zhou, Junrui Shen, Wenxiang Chen, Jiaqi Liu, Jixuan Huang, Zhihao Zhang, Honglin Guo, Xun Deng, Zhikai Lei, Miao Zheng, Guoteng Wang, Shuo Zhang, Peng Sun, Rui Zheng, Hang Yan, Tao Gui, Qi Zhang, and Xuanjing Huang. BAPO: Stabilizing off-policy reinforcement learning for LLMs via balanced policy optimization with adaptive clipping, 2025. URL https://arxiv.org/abs/2510.18927.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-GRPO: Scaffolded group relative policy optimization for enhancing LLM reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bOwVr0yr7r.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2205.10625.

A MORE DETAILS ABOUT METHOD

This section provides supplementary details regarding the formal proof tree decomposition and proof-grounded subproblem synthesis pipeline introduced in Section 3.

A.1 DETAILED LOGICAL EXPLANATION OF STRUCTURAL TREE TRANSFORMATIONS

In the formal proof decomposition pipeline, transforming the raw execution trace into a semantically meaningful proof tree $\mathcal{T}$ requires several critical structural operations. These transformations are essential to bridge the gap between how a compiler records state changes and how an LLM learns mathematical logic.

A.1.1 REMOVING BOUNCE-BACK CHAINS

In a sequential execution trace of a Lean 4 (de Moura & Ullrich, 2021) proof, completing a focused sub-goal (such as closing a `have` block or finishing a bullet point $\cdot$) causes the compiler's active state to abruptly "bounce back" to the next pending sibling goal or the parent's continuation. If we naively construct a logic graph by linking each state $\mathcal{G}_i$ directly to the chronologically subsequent state $\mathcal{G}_{i+1}$ in the trace, this creates an artificial, non-logical edge connecting the final leaf of a completed sub-proof directly to a higher-level goal.

The `RemoveBounceBackChains` operation detects these erratic structural jumps—typically characterized by a sudden, complete change in the goal target $\mathcal{C}$ or a significant, disjoint drop in the local context hypotheses—and severs these false sequential edges. This operation works in tandem with `have`-promotion (Appendix A.1.2) to ensure that the linear execution trace is correctly unrolled into a proper hierarchical tree $\mathcal{T}$, where branches represent independent logical sub-tasks rather than an erratic chronological path.

A.1.2 PROMOTING HAVE SUB-GOALS TO SIBLINGS

In Lean 4, the `have` tactic is utilized to introduce an intermediate lemma or auxiliary goal before solving the current branch. Intuitively, an intermediate lemma B and the final target C follow a sequential temporal dependency: B must be proven before C can be proven using B. However, in the context of formal logic, the proof trace is modeled as a tree of *subgoals* rather than a chronological execution graph.

When a root goal aims to prove C given premise A, and an intermediate lemma B is introduced via `have h : B := ...`, the root goal logically splits into two distinct, parallel sub-tasks:

- **Subgoal 1:** Prove B under premise A.
- **Subgoal 2:** Prove C under premises A and B.

For the proof compiler, both sub-tasks must be resolved to fulfill the root goal. We represent them as *sibling nodes* sharing the same parent, while retaining B in the continuation's hypotheses to record its logical dependence. If we do not explicitly extract Subgoal 1 as a sibling tree, the raw execution trace would present a deep, uninterrupted chain dedicated to proving B, followed by an abrupt bounce back to Subgoal 2. Promoting the `have` branch separates these intermediate tasks from the parent goal. This representation records the dependency through the local context; depth-based ordering alone does not prescribe an order between sibling nodes at the same depth.

A.1.3 COLLAPSING SIMILAR CHAINS

During a formal proof, automated tactics such as `simp` or `ring` often perform a sequence of minor algebraic rewrites. For instance, an expression might transition through several intermediary states:

$$x + y - x = y \Rightarrow (x - x) + y = y \Rightarrow 0 + y = y \Rightarrow y = y.$$

While necessary for the compiler's rigorous verification, these highly similar, granular steps often offer minimal cognitive value for an LLM learning advanced mathematical reasoning. Retaining them would result in unnecessarily deep sub-branches filled with trivial bookkeeping. By collapsing

consecutive nodes whose textual similarity exceeds a predefined threshold (e.g., trigram Jaccard similarity > 0.7), we compress the proof tree and focus the model's attention on substantial mathematical transitions rather than low-level symbolic rewrites.

A.1.4 STRIPPING ACCUMULATED HYPOTHESES

As a proof progresses deeper into the tree, the local context accumulates hypotheses from parent branches. Consequently, a deep leaf node may inherit a large number of premises, many of which are irrelevant to its localized conclusion.

If these extraneous hypotheses are retained and directly translated into a natural language subproblem, the resulting text becomes cluttered with distracting information. To mitigate this, the `StripAccumulatedHypotheses` operation removes unused inherited assumptions. We analyze the dependency graph of the proof terms to identify and retain the premises invoked by each subgoal. This aims to provide concise formal inputs for translation; the fidelity of the resulting natural-language tasks is assessed separately in Appendix C.2.

A.2 RULE-BASED FILTERING CRITERIA

To efficiently prune overly trivial or uninformative subproblems before invoking computationally expensive LLM evaluations, we apply a rigorous rule-based filter. As described in Section 3.2, each raw subproblem extracted from the proof tree is evaluated against the following criteria, and is immediately discarded if any condition is met:

- **Binary Outcomes:** The target conclusion implies a simple `True` or `False` answer, which is unsuitable for RLVR training due to the 50% random guess probability.
- **Trivial Goal Length:** The formal goal statement string is shorter than 30 characters, typically indicating a trivial algebraic simplification.
- **Explicit Proof Gaps:** The goal-state text contains an explicit `sorry` marker. Filtering this marker does not by itself certify every dependency of a proof containing admitted steps.
- **Tautological Conclusions:** The specific conclusion statement (the text following the $\vdash$ symbol) is shorter than 5 characters, which usually represents tautologies or single variables (e.g., $x = x$).

A.3 LLM-BASED PRUNING PROMPT

To ensure the synthesized dataset provides high-quality learning signals for RL training, we employ an LLM-based evaluator to aggressively prune subgoals that require only trivial cognitive effort. We use the following prompt to filter out raw formal subgoals that only require trivial algebraic manipulations, such as definition unpacking or simple substitution.

LLM-Based Formal Pruning Prompt

```
You are deciding whether a proof intermediate step
is worth keeping as a training subproblem for a math
reasoning model.

**BE STRICT -- PRUNE aggressively.** Only KEEP steps that
represent genuine mathematical insight.

**ALWAYS PRUNE if the step is:**
- A simple True/False check (e.g., "Is X prime?", "Is 2 < 5?",
  "Is this set finite?")
- Trivial arithmetic (e.g., "Compute 2-1",
  "Compute \sqrt{169}")
- Obvious variable substitution or rewriting
- A direct unpacking of a definition (e.g., "Let x = 5,
  compute f(x)")
- Bookkeeping (type checking, finiteness, bounds checking)
```

```
- Nearly identical to another step or to the original
  problem statement
- A step so specific to the proof machinery that it has no
  standalone educational value

**Only KEEP if:**
- The step requires a non-trivial mathematical technique (e.g.,
  algebraic identity, inequality trick, counting argument,
  number theory lemma)
- Solving this step independently would teach a useful skill
  transferable to other problems
- The answer is a concrete non-trivial value (not just
  True/False)

**Original Main Problem:**
{main_problem}

**Intermediate Step:**
{sub_problem}

**Answer:**
{sub_answer}

Output exactly one word: KEEP or PRUNE.
```

A.4 Natural Language Translation Prompt

In the data synthesis phase, we use DeepSeek-V3.2 (Liu et al., 2025) to translate retained formal goal states into standalone natural-language problems. The following prompt instructs the model to preserve the formal hypotheses as problem conditions. These instructions do not guarantee faithful conversion; Appendix C.2 reports an empirical audit.

Natural Language Translation Prompt

```
You are converting a Lean proof step into a
GRPO training data point. GRPO trains models by reinforcement
learning -- the model must produce a concrete, verifiable
answer.

**Lean Goal:**
{goal}

**How to read this:**
- Before `|-`: given hypotheses and context
- After `|-`: the conclusion (what needs to hold)

**Your task:** Produce a self-contained math problem and its
answer.

**Rules:**
1. **Problem style:** Must be a "compute/find/determine"
question, NOT "prove/show".
2. **Answer must be concrete and short:** a number or a simple
   expression like `1 + \sqrt{{2}}`.
   - If the conclusion is an equality like
     `x = 1 + \sqrt{{2}}`, the answer is `1 + \sqrt{{2}}`.
```

```
    - If the conclusion is an inequality such as
      `x + 4/x \ge 4`, convert it into a minimum/maximum
      question only when the retained hypotheses and the formal
      goal establish both the bound and its attainability on
      the exact same domain. Do not turn a non-attainable
      bound into a minimum/maximum question.
    - If the conclusion is `\exists x, ...`, ask for a specific
      value only when the retained hypotheses and the formal
      goal uniquely determine the answer. Do not convert
      existential goals with multiple valid witnesses into
      single-answer questions.
3. **Self-contained:** Include all hypotheses as given
   conditions. No Lean references.
4. **LaTeX for math.**
5. **Answer in simplest form.** No Lean syntax. Use standard
   math notation.

**Output format** (exactly two blocks):
```problem
<the problem statement>
```

```answer
<the answer, as short as possible>
```
```

B More Details about Experiments

This section provides additional experimental details omitted from the main text. In the main paper, we keep the experimental setup concise; here we describe the data construction pipeline, baselines, training configuration, reward design, and evaluation protocol in greater detail.

B.1 Data Construction

Our experiments are based on NuminaMath-LEAN[1], which contains 104,155 mathematical problems with natural-language statements and Lean-related formal fields. Among them, 97,334 examples have answer annotations, and 11,380 examples have formal proof entries labeled `complete` or `with_sorry`. The latter label permits admitted steps and does not establish a fully certified proof. Since RLVR requires automatically checkable final answers, we filter out pure proof problems and examples whose answers cannot be reliably verified. The construction counts below describe the Qwen experiment; they are not separate measurements of the Llama model's failure set.

We construct the training data in three stages. First, we build an answer-verifiable RLVR set from NuminaMath-LEAN. Rule-based filters identify simple numerical, symbolic, and option-style answers, while ambiguous cases are classified with a strong LLM. For examples with missing or unclear answers, we attempt to extract the final answer from the Lean theorem declaration, falling back to LLM-assisted extraction when necessary. This produces 26,055 RLVR-suitable examples.

Second, after data curation, we retain 22,993 original answer-verifiable training problems, which we call **Base Data**. This set is not restricted to problems with usable Lean proofs. We evaluate the Qwen MoT-SFT base model on this set and identify 10,186 incorrectly solved problems. Among these wrong problems, 2,844 have usable Lean proof entries and are eligible for proof-tree decomposition.

Third, LeanTree extracts formal proof trees for these hard problems, flattens non-root nodes into candidate subproblems, prunes trivial nodes, and translates retained formal goals into natural language. The quality-filtered subproblem set contains 4,772 examples. Before the last RLVR suitability and

[1] https://huggingface.co/datasets/AI-MO/NuminaMath-LEAN

answer-quality filter, the enhanced dataset contains 27,765 examples: 22,993 original problems and 4,772 proof-grounded subproblems. After final filtering, the training file used for RL contains 24,636 answer-verifiable examples. Therefore, **Our Data** denotes the filtered merged training set consisting of original problems and synthesized subproblems, rather than the subproblem set alone.

Note that we remove all exact and near-duplicate overlaps with GSM8K, MATH500, AMC 2023, and AIME 2025 using normalized text matching and embedding-based similarity filtering.

Prompt for LLM-assisted answer extraction. To ensure transparency and reproducibility in our data curation process, the exact prompt used for extracting final answers from ambiguous examples or Lean theorem declarations is provided below.

Prompt for LLM-assisted answer extraction

```
You are a Lean 4 code reader. Read the theorem declaration
below and extract the answer from the conclusion
(the part after the last ':' before ':=').

Rules:
- Do NOT solve the problem. Just read the Lean conclusion and
  convert to math notation.
- For 'n = 7', output \boxed{7}.
- For 'P 1 = 87', output \boxed{87}.
- For 'x = 1 / 2 /\ y = -1', output \boxed{1/2, -1}.
- For 'forall x, f x = x^-1', output \boxed{f(x) = 1/x}.
- Only output \boxed{} with the answer, nothing else.
```

Table 3: Data statistics for constructing Base Data and Our Data in the Qwen experiment.

Stage	Number of Examples
NuminaMath-LEAN total	104,155
RLVR-suitable examples	26,055
Original answer-verifiable problems (Base Data)	22,993
Wrong problems under the MoT-SFT base model	10,186
Wrong problems with usable Lean proofs	2,844
Quality-filtered proof-grounded subproblems	4,772
Merged original + subproblem data before final filtering	27,765
Final RL training file (Our Data)	24,636

B.2 Evaluation Benchmarks

We evaluate models on four mathematical reasoning benchmarks with different levels of difficulty.

- **GSM8K** (Cobbe et al., 2021): A grade-school mathematical reasoning benchmark comprising 1,319 test problems. It mainly evaluates arithmetic and multi-step word-problem solving ability. Although it is relatively easier than competition-level benchmarks, it provides a useful measure of whether RL training preserves general mathematical reasoning capability.
- **MATH500** (Hendrycks et al., 2021; Lightman et al., 2024): A widely used subset of the MATH benchmark consisting of 500 problems in total. To better analyze whether LeanTree improves performance on hard reasoning tasks, we additionally split MATH500 into two subsets according to their official difficulty levels:
  - **MATH500 (L1-3)**: Contains 238 problems (43 Level 1, 90 Level 2, and 105 Level 3 problems). This subset mainly evaluates relatively standard mathematical problem solving.
  - **MATH500 (L4-5)**: Contains 262 problems (128 Level 4 and 134 Level 5 problems). This subset focuses on more complex problems that are more likely to expose the reasoning boundary of a 7B model.

- **AMC 2023** (Mathematical Association of America, 2023): This benchmark consists of 40 competition-style mathematical problems from the American Mathematics Competitions. Compared with GSM8K, AMC problems require stronger symbolic manipulation, combinatorial reasoning, and multi-step derivations.
- **AIME 2025** (Mathematical Association of America, 2025): A challenging high-school mathematics competition benchmark containing 30 problems. It is particularly suitable for evaluating whether a method can improve reasoning on hard problems, because many examples require non-trivial mathematical insight and long-horizon reasoning. We use exact integer matching for evaluation on AIME 2025.

B.3 BASELINES

We compare LeanTree with the following baselines. Each method uses the same initialization within a backbone comparison. Qwen comparisons include GRPO and BAPO, while the Llama comparisons use GRPO.

- **Vanilla**: The corresponding initial model before our RL training. For Qwen2.5-7B-Instruct, this is the model after supervised fine-tuning on Mixture-of-Thoughts[2]. For DeepSeek-R1-Distill-Llama-8B, it is the official distilled checkpoint, without additional Mixture-of-Thoughts SFT.
- **Standard RL on Base Data**: This baseline trains the Vanilla model directly on the original answer-verifiable problems using GRPO or BAPO for Qwen and GRPO for Llama. It measures the benefit of RL without synthesized intermediate tasks.
- **Prompt LLM**: A data synthesis baseline that asks a strong LLM to heuristically decompose hard problems into subproblems without formal proof structure. In the Qwen data construction, 433 hard problems are decomposed into 6,919 LLM-generated subproblems, with an average of about 16 subproblems per input problem.
- **SwS** (Liang et al., 2025): A data synthesis baseline that generates additional training problems based on self-identified weaknesses during the RL training process. Both Prompt LLM and SwS are evaluated with GRPO and BAPO for Qwen and with GRPO for Llama, matching the optimizer used by LeanTree in each comparison.

Prompt for LLM-based decomposition.

Prompt for LLM-based decomposition

```
You are an expert math educator. Given a math problem and its
step-by-step solution, your task is to decompose it into
subproblems that a student would need to master in order
to solve the original problem.

## Rules:
1. Each subproblem must be a **self-contained question** with
   a **concrete, verifiable answer** (a number, expression,
   or simple set --- NOT "prove that..." or "show that...").
2. Order subproblems from easiest to hardest (the first ones
   should be simpler building blocks).
3. Focus on **key insights and non-trivial steps** --- skip
   trivial arithmetic, obvious substitutions, and pure
   bookkeeping.
4. Each subproblem should teach a **transferable skill**
   useful for other problems too.

## Original Problem:
{problem}
```

[2] https://huggingface.co/datasets/open-r1/Mixture-of-Thoughts

```
## Solution:
{solution}

## Answer to Original Problem:
{answer}

## Output Format:
For each subproblem, output exactly in this format:
'''subproblem
<the subproblem question, self-contained>
'''
'''answer
<the concrete answer>
'''
```

B.4 Training Setup

We use Qwen2.5-7B-Instruct[3] initialized from the Mixture-of-Thoughts SFT checkpoint, and DeepSeek-R1-Distill-Llama-8B[4] initialized directly from the official distilled checkpoint. The Llama experiments do not apply additional Mixture-of-Thoughts SFT. We use vLLM-based rollout generation and distributed training with Fully Sharded Data Parallel (FSDP) (Zhao et al., 2023). The Qwen runs were conducted on a single compute node with 8 NVIDIA A100 (80GB) GPUs; a standard 100-step Qwen run took approximately 15 minutes per step, or 25 hours in total. These hardware and timing measurements describe the Qwen runs.

For GRPO (Shao et al., 2024), we use a rollout group size of 8, maximum prompt length 2048, maximum response length 8192, rollout temperature 1.0, and top-p 1.0. Unless otherwise specified, the global batch size is 256 and the mini-batch size is 64. We use a learning rate of 1×10^{-6} and PPO (Schulman et al., 2017) clipping threshold 0.2. The KL and entropy coefficients are set to 0.0 in our main GRPO runs.

For the Qwen BAPO comparisons (Xi et al., 2025), we use the same initialization, data conditions, rollout budget, and learning rate 1×10^{-6}. The rollout sampling size is 8, the discount factor is 1.0, and the generalized advantage estimation (GAE) (Schulman et al., 2015) parameter λ is 1.0. We do not report BAPO experiments for Llama. Main benchmark runs train for 100 steps, and we use the final checkpoint at step 100 for evaluation on all benchmarks. The Qwen/GRPO learning-curve diagnostics in Section 4.2 instead use batch size 8 and extend to step 700. The diagnostic methods have matched batch sizes, rollout group sizes, and training-step counts, and both train exclusively on difficult original root problems during steps 600–700.

We use a strictly rule-based reward during training. A generated response receives $+1$ if its final answer is equivalent to the ground-truth answer and satisfies the required answer format. It receives -1 if the answer is incorrect, missing, unverifiable, or violates the required output format.

This reward design matters for interpreting the training curves. If all 8 rollouts for a prompt have the same outcome, their within-group reward variance and reward-based GRPO advantages are zero. A mixture of correct and incorrect responses provides nonzero reward contrast. For binary rewards, the within-group variance is $1 - \bar{r}^2$, where $\bar{r}$ is the mean rollout reward, and is maximal at $\bar{r} = 0$. This is a measure of available reward contrast, not a standalone measure of solution quality or total policy-gradient magnitude.

B.5 Evaluation Setup

We evaluate with vLLM (Kwon et al., 2023), using generation temperature 0.7, top-p 0.95, maximum generation length 8192 tokens, batch size 400, and seed 1. The Qwen evaluation uses bfloat16 precision and four-way tensor parallelism. The Llama evaluation runner uses tensor parallelism 1

[3] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[4] https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B

and splits each model's evaluation data into four shards, with one worker per GPU. We perform 8 evaluation repetitions on GSM8K and MATH500 and sample 16 responses per problem on AMC 2023 and AIME 2025. The reported means and standard deviations summarize accuracy across these repeated samples. The standard deviations describe evaluation sampling variability, not variation across independently retrained models.

For Qwen, we use the Qwen2 chat template below, which guides the model to reason before answering and to put its final answer in `\boxed{}`. Llama uses the model-specific `deepseek-r1-distill-llama` prompt format rather than this Qwen template.

Qwen Evaluation Prompt Template

```
<|im_start|>system
You are a helpful assistant. To answer a query from the user,
please first think about the reasoning process in the mind and
then output your answer. The reasoning process is enclosed
within <think> </think> tags, i.e., <think> reasoning process
here </think> answer here.
<|im_end|>
<|im_start|>user
{question} Please output the final answer within \boxed{}.
<|im_end|>
<|im_start|>assistant
<think>
```

For Qwen, the rule-based answer extraction and verification pipeline (`naive_dapo`) inspects the last 300 characters of the output to extract the final `\boxed{...}` expression. It normalizes the extracted answer by removing units (e.g., cm, meters, dollars), stripping LaTeX formatting commands (e.g., `\text{}`, `\textbf{}`), normalizing shorthand notations for fractions and square roots, and stripping commas from pure numbers. It then compares the normalized strings with the reference answer; outputs without `\boxed{}` receive no credit. The Llama evaluation runner selects the `naive` rule-based scorer. Comparisons within each backbone use the same evaluation configuration.

B.6 Declaration of LLM and AI Agent Usage

LLMs are components of the data pipeline: they assist answer extraction, prune candidate subproblems, and translate retained Lean goals into natural-language tasks. An independent LLM also performs the fidelity audit described in Appendix C.2. These roles are separate from the use of AI agents and LLMs to assist code writing, refactoring, debugging, and manuscript editing. The authors are responsible for checking the resulting implementation, scientific claims, and reported results. Neither coding assistance nor model-based auditing provides a formal guarantee of correctness.

C Additional Experimental Analyses

The following analyses use the Qwen experimental setting.

C.1 Proof Depth and Learning on Root Problems

Deeper subproblems are empirically easier. To test whether proof depth provides a useful difficulty signal, we split the data by original problem ID before synthesis and reserve 20% of the original problems and their associated proof trees for held-out diagnostic evaluation. Each root problem and all subproblems derived from its proof are assigned to the same split, and held-out roots and their derived subproblems are excluded from every method's training data. We then group the held-out examples by proof depth. Depth 0 contains original root problems, and depths ≥ 5 are pooled as in the curriculum. We evaluate the initial Qwen model, GRPO on Base Data, Prompt LLM, and LeanTree on the same held-out examples. For this comparison, Prompt LLM retains the full Base Data training split and randomly subsamples only its synthesized subproblems so that its total training-set size matches Our Data.

Table 4: **Accuracy (%) on held-out Lean-derived problems, stratified by proof depth.** All RL methods use Qwen and GRPO. Prompt LLM is matched to LeanTree in total training-set size. The last two columns give LeanTree's gains in percentage points. Bold marks the highest accuracy at each depth.

Depth	Vanilla	Base-RL	Prompt LLM	LeanTree	$\Delta_{\text{Base-RL}}$	Δ_{Prompt}
0	21.61	24.02	26.60	**28.03**	+4.01	+1.43
1	41.39	43.15	44.94	**46.29**	+3.14	+1.35
2	50.54	52.56	53.94	**55.18**	+2.62	+1.24
3	54.23	55.53	56.49	**57.91**	+2.38	+1.42
4	63.87	65.41	66.05	**67.32**	+1.91	+1.27
≥ 5	70.14	71.65	72.32	**73.25**	+1.60	+0.93

As shown in Table 4, initial-model accuracy increases monotonically from 21.61% at depth 0 to 70.14% at depth ≥ 5. This supports depth as a practical difficulty proxy for the retained subproblems, without implying that every deeper node is easier than its parent.

The largest gain over direct RL occurs at the root. LeanTree improves over Base-RL at every depth. Its largest improvement is on the original root problems, where accuracy rises from 24.02% to 28.03% ($+4.01$ points), compared with a 1.60-point gain at depth ≥ 5. Thus, the benefit extends beyond the relatively accessible intermediate tasks to the original difficult problems. LeanTree also exceeds the size-matched Prompt LLM baseline at every depth, with a 1.43-point gain at the root. This comparison supports the value of the complete synthesis and curriculum pipeline beyond training-set size alone; it does not isolate formal grounding from proof structure or training order.

C.2 Fidelity of Synthesized Subproblems

Formal proof structure supplies the conditions and targets for synthesis, but the LLM-based pruning and natural-language translation are not themselves formally verified. We therefore assess translation fidelity empirically. We randomly sample 300 synthesized subproblems, stratified by proof-tree depth and formal target type, and use GPT-5.6-Sol (OpenAI, n.d.) as an independent evaluator. For each sample, the evaluator receives the exact Lean goal after dependency stripping, the generated natural-language problem, and its reference answer. It checks condition preservation, target equivalence, answer correctness, self-containedness, unsupported additions, and lost dependencies. A sample passes only if all criteria are satisfied.

Overall, $298/300$ examples pass all checks (99.3%), with a 95% Wilson confidence interval (Wilson, 1927) of 97.6–99.8%. Manual spot-checks of a subset of the evaluator's judgments agree with the stated criteria. The two remaining examples fail at least one of the audit criteria. This audit supports the reliability of the conversion process while demonstrating a nonzero translation-error rate; it is an empirical fidelity assessment, not a formal certificate for the natural-language training data.

C.3 Curriculum and Ablation Protocol

We set $d_{\max} = 5$ and train in the order $\geq 5 \to 4 \to 3 \to 2 \to 1 \to 0$, where depth 0 contains original problems and retained nodes at depth 5 or greater form the first phase. The main benchmark runs use 100 training steps, whereas the Qwen/GRPO diagnostics extend to step 700 with batch size 8; both diagnostic methods train exclusively on difficult original root problems during steps 600–700. The held-out results in Appendix C.1 support depth as a practical difficulty signal, but neither those results nor the random-order ablation establish that the chosen cutoff, phase count, or phase boundaries are optimal.

The curriculum ablation in Table 2 compares the same Qwen training data under random ordering and depth-based ordering. Both variants use the same evaluation protocol: 8 repetitions on GSM8K and MATH500, and 16 sampled responses per problem on AMC 2023 and AIME 2025, with mean accuracy and standard deviation reported for each benchmark. The Full row uses the results from Table 1. Full has average accuracy 71.14%, compared with 68.91% for Random, a gain of 2.23 percentage points.

C.4 Offline Synthesis Cost

The main additional cost of LeanTree is offline data construction. In the Qwen synthesis experiment, Prompt LLM uses approximately 1.52 million tokens, LeanTree 56.91 million, and SwS 1.014 billion. Thus, LeanTree uses about $37.4\times$ as many synthesis tokens as Prompt LLM and approximately $1/17.8$ of SwS's synthesis tokens. The LeanTree synthesis run incurred approximately US$23 in DeepSeek API charges; this figure refers to that run's API bill, not total training or local proof-processing cost. Once constructed, the dataset can be reused across GRPO and BAPO training, and no Lean interaction is required during RL rollout or inference. These results quantify the trade-off between additional offline synthesis effort and the observed reasoning gains.

D Detailed Analysis of Reward Distribution

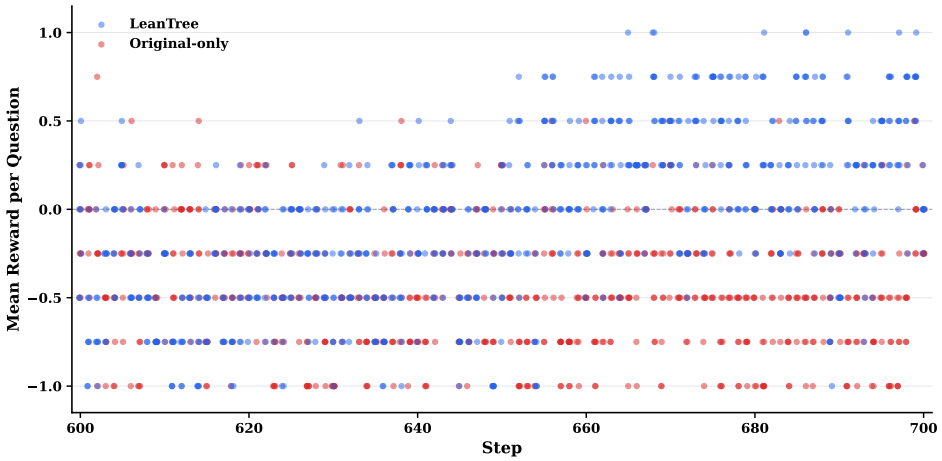


Figure 3: **Scatter plot of the mean reward per question during training steps 600–700.** Each point represents the mean reward of 8 rollouts for a single question in a batch.

To complement the macroscopic, aggregated learning curves presented in Section 4.2, Figure 3 provides a granular scatter plot of the mean reward per question (averaging the 8 rollouts per question in a batch) across training steps 600 to 700. While the line charts in the main text illustrate the overall trend, this scatter plot shows the distribution of per-question rollout outcomes. Because correct and incorrect responses receive $+1$ and -1, respectively, a mean reward near 0 (e.g., 4 correct and 4 incorrect rollouts) maximizes within-group reward variance. Conversely, points located exactly at $+1$ (all correct) or -1 (all incorrect) have zero reward variance and zero reward-based group-relative advantages. Higher mean reward indicates more successful rollouts, but it need not imply greater reward contrast: an all-correct group also has zero variance.

During this window, the Base Data baseline has a higher concentration of points at or near -1 than LeanTree, indicating more groups with few or no successful rollouts. After the transition to root problems, LeanTree's distribution shifts toward higher mean rewards, particularly later in the window. This pattern is consistent with intermediate-task training helping subsequent optimization on the original problems. The scatter plot describes rollout outcomes; it does not establish that particular sub-skills have been mastered or isolate the causal contribution of each pipeline component.