LEARN TO REASON EFFICIENTLY ON REAL PATHS VIA TREE-STRUCTURED POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large reasoning models often waste substantial test-time compute, and recent length-aware reinforcement learning addresses this issue by rewarding short and correct trajectories. We reveal a previously overlooked training–inference mismatch in the prevailing trajectory-level paradigm for length-aware RL: positive learning signals are concentrated on trajectories whose early reasoning is already well directed, whereas inference requires the model to continue from its own potentially erroneous or exploratory prefixes. Through prefix-level continuation analysis, we empirically verify this mismatch and show that conditioning reinforcement learning on self-generated prefixes substantially improves both reasoning accuracy and efficiency. To address this issue, we propose ***T**ree-structured **L**ength-aware **P**olicy **O**ptimization* (**TLPO**), which performs prefix-conditioned credit assignment by comparing continuations branching from the same self-generated prefix and optimizes their efficiency with segment-level length-aware rewards. Across three representative reasoning models and five benchmarks, TLPO reduces average reasoning length by 56.0% to 80.3% while improving average accuracy by 0.5 to 4.1 percentage points, with out-of-domain evaluations showing consistent gains on scientific reasoning and coding tasks. These results more broadly establish prefix-conditioned optimization over self-generated reasoning paths as an effective principle for training efficient reasoning models.

1 INTRODUCTION

Recent advancements have demonstrated that large language models (LLMs) can achieve strong reasoning capabilities through test-time scaling (Snell et al., 2024; Wu et al., 2025). However, these models often fail to utilize test-time compute efficiently, producing unnecessarily long reasoning traces with redundant steps and excessive self-verification, leading to high token usage and inference latency (Chen et al., 2024; Cuadron et al., 2025). To improve reasoning efficiency, prior work has explored strategies such as early exit (Ma et al., 2025; Dai et al., 2025), CoT compression (Cheng et al., 2025; Xia et al., 2025), and length-aware reinforcement learning (RL) (Aggarwal & Welleck, 2025; Yuan et al., 2025; Liu et al., 2025b). Among these directions, RL-based methods have emerged as a major paradigm for optimizing test-time reasoning efficiency (Yue et al., 2025). In particular, length-aware RL methods typically sample multiple responses for each problem and use length-aware rewards to assign larger learning signals to trajectories that are both correct and concise.

However, we identify a previously overlooked training–inference mismatch in trajectory-level optimization: **the prefixes that receive strong positive learning signals during training are systematically more favorable than the broader set of self-generated prefixes that the model must condition on at inference time.** As illustrated in Figure 1, existing RL-based efficient reasoning methods (Arora & Zanette, 2025; Su & Cardie, 2025; Hou et al., 2025a; Liu et al., 2025b) typically compare complete trajectories sampled for the same problem. Under these objectives, shorter and correct trajectories tend to receive larger relative advantages, thereby concentrating positive updates on reasoning paths whose early prefixes are associated with more accurate and efficient continuations. At inference time, however, generation proceeds autoregressively from the model's own prefixes, which can contain exploratory steps, detours, or intermediate mistakes. Such prefixes are a natural part of difficult reasoning rather than exceptional failure cases. Consequently, selecting short and correct trajectories among training rollouts does not provide equally direct learning signals for continuation behavior across the broader prefix distribution encountered during inference.

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.

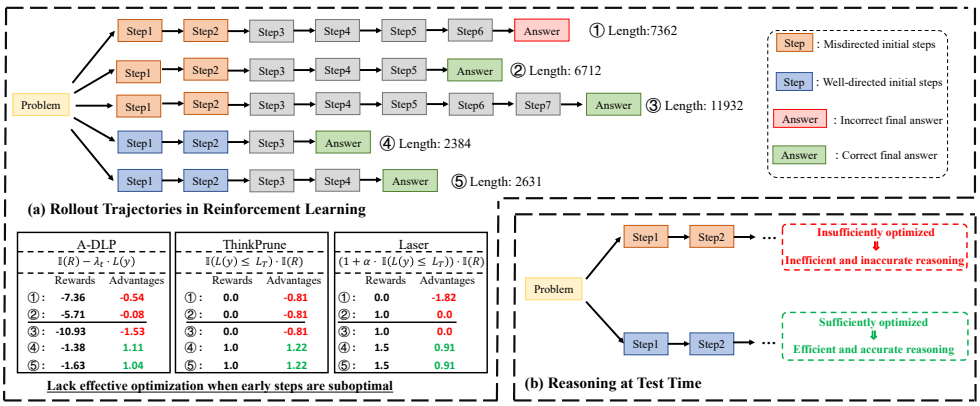


Figure 1: Illustration of the training–inference gap in RL-based efficient reasoning. (a) During training, multiple reasoning trajectories are sampled and compared using relative rewards or advantages. Although some trajectories with suboptimal early steps eventually reach correct answers with relatively efficient reasoning (trajectory ②), they receive non-positive advantages due to the presence of shorter trajectories that start in the correct direction (trajectories ④, ⑤). (b) During inference, the model may encounter misdirected or exploratory early prefixes, for which trajectory-level objectives provide weaker optimization signals, potentially leading to inefficient or inaccurate continuation.

We empirically characterize this mismatch by examining the continuation quality of prefixes associated with different training advantages. We find that prefixes from positive-advantage trajectories, and especially those from the highest-advantage trajectories within each rollout group, lead to substantially higher continuation accuracy and shorter continuation lengths than prefixes naturally generated at inference time. This analysis provides direct empirical evidence that trajectory-level credit assignment systematically favors a comparatively high-quality subset of self-generated prefixes. We further find that explicitly conditioning RL optimization on self-generated prefixes substantially improves both reasoning accuracy and efficiency, suggesting that efficient reasoning can benefit from directly optimizing continuation behavior under the prefixes actually induced by the model itself.

Motivated by these observations, we propose ***T**ree-structured **L**ength-aware **P**olicy **O**ptimization* (**TLPO**). TLPO constructs tree-structured rollouts in which multiple continuations branch from the same self-generated reasoning prefix. Rather than comparing complete trajectories only from the original problem, TLPO estimates segment-level advantages among continuations that share the same prefix. This prefix-conditioned credit assignment shifts optimization from selecting globally favorable trajectories at the problem level to directly learning efficient continuation behavior under prefixes induced by the model, including both favorable and suboptimal reasoning states. Across three representative reasoning models and five reasoning benchmarks, TLPO reduces average reasoning length by 56.0%–80.3% while improving average accuracy by 0.5–4.1 percentage points. Moreover, TLPO substantially suppresses the long tail of unsuccessful generations, showing that its efficiency gains extend beyond successful trajectories to failed reasoning processes as well.

Our contributions are threefold. First, we identify and empirically characterize a previously underexplored training–inference mismatch in trajectory-level efficient-reasoning RL: positive learning signals are disproportionately concentrated on trajectories with favorable early prefixes, whereas inference requires continuation from a broader distribution of self-generated prefixes. **Second**, we introduce TLPO, a tree-structured policy optimization framework that performs prefix-conditioned credit assignment among continuations sharing the same self-generated reasoning prefix. **Third**, extensive experiments across three reasoning models and five benchmarks demonstrate consistent and substantial gains in the accuracy–efficiency trade-off.

2 Gaps between Training and Inference

To better understand the training–inference discrepancy in RL-based efficient reasoning, we first analyze the correlation between CoT length and reasoning accuracy in models trained with length-aware RL (Section 2.1). We then characterize the distribution shift in early CoT between training and

inference (Section 2.2). Finally, we examine whether conditioning RL on realistic, self-generated reasoning prefixes improves reasoning accuracy and efficiency (Section 2.3). In our experiments, we select Laser-L2048-1.5B (Liu et al., 2025b) as a representative LLM trained with length-aware RL.

2.1 Correlation between Length and Accuracy

When solving the same problem, models may produce responses of varying lengths, reflecting different reasoning behaviors, such as more careful deliberation, self-correction, or frequent shifts in reasoning direction. To better understand these behaviors and their impact in models trained with RL-based efficient reasoning, we investigate how response length relates to accuracy for the same problem. To this end, we analyze the correlation between response length and accuracy by sampling 2048 responses per problem from Laser-1.5B. As shown in Figure 2, we observe **a clear negative correlation between CoT length and accuracy**, both across the entire dataset and at the level of individual problems with varying difficulty.

We further conduct a case analysis of CoTs with different lengths for the same problem. We find that variations in response length and accuracy typically arise from two distinct reasoning cases. In the first case, the model follows a correct reasoning direction at an early stage and solves the problem efficiently through a smooth reasoning process, resulting in short and accurate responses. In the second case, the model starts from an incorrect direction or makes errors during intermediate steps, and subsequently attempts to recover by switching reasoning paths and performing self-correction. This process often leads to longer outputs—sometimes several times longer than successful solutions—yet remains prone to yielding incorrect answers. These observations suggest that longer and unsuccessful generations are often associated with reasoning processes that deviate from a favorable direction early on or undergo repeated correction. This motivates us to examine whether such prefixes receive the same optimization signals as the more favorable prefixes associated with short and correct trajectories. Additional results are provided in Appendix A.1.

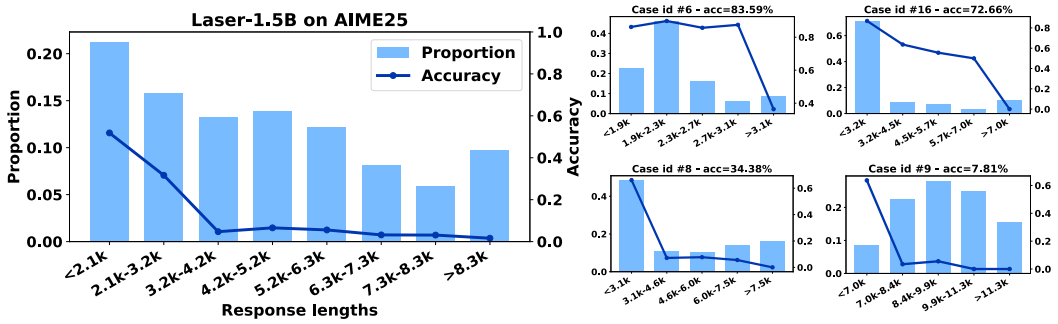
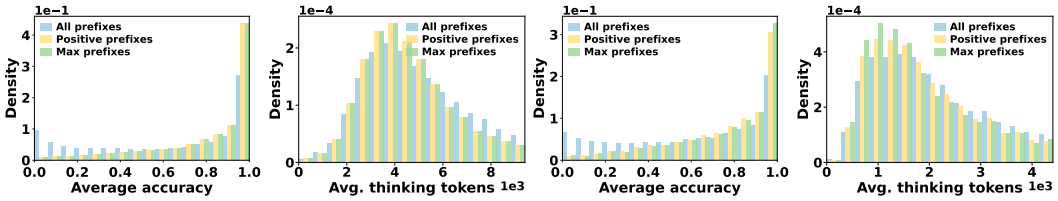


Figure 2: Correlation between CoT length and reasoning performance. Bars indicate the distribution over CoT length intervals, and the line shows average accuracy. Results are shown for the full dataset (left) and individual problems (right).

2.2 Prefix Mismatch between Training Signals and Inference

The observations above show that unsuccessful and excessively long generations are often associated with less favorable early reasoning paths. We next investigate whether trajectory-level RL assigns positive learning signals uniformly across the self-generated prefixes encountered by the model, or instead concentrates them on a comparatively favorable subset. Specifically, we consider outcome-based RL with binary rewards and length-aware RL, represented by DeepScaler-1.5B (Luo et al., 2025b) and Laser-1.5B (Liu et al., 2025b), respectively. For each model, we compare three types of prefixes, each corresponding to the first 20% of the full response: (1) prefixes whose associated trajectories have positive advantage during training (*Positive prefixes*); (2) prefixes corresponding to trajectories with the highest advantage within each training group (*Max prefixes*); and (3) early prefixes naturally generated at inference time without advantage-based filtering (*All prefixes*). To evaluate the quality of these prefixes, we sample 16 continuations from each prefix and measure their average response length and accuracy.



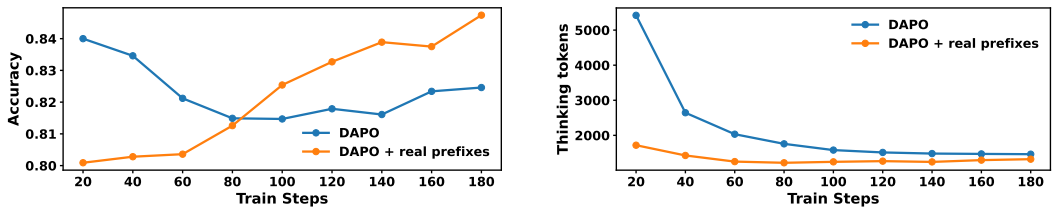
(a) Accuracy of DeepScaleR-1.5B (b) Think length of DeepScaleR-1.5B (c) Accuracy of Laser-1.5B (d) Think length of Laser-1.5B

Figure 3: Histograms of expected accuracy and response length for different types of prefixes from DeepScaler-1.5B and Laser-1.5B. *All prefixes* denote prefixes generated at inference time. *Positive prefixes* correspond to prefixes from trajectories with positive advantage during training. *Max prefixes* correspond to prefixes from trajectories with the highest advantage within each group.

As shown in Figure 3, for models trained with both outcome-based RL and length-aware RL, prefixes extracted from positive-advantage trajectories consistently exhibit higher continuation accuracy and shorter continuation lengths than prefixes naturally generated at inference time. Furthermore, under length-aware RL, prefixes associated with the highest training advantages show an even stronger shift toward high continuation accuracy and short responses. These results demonstrate that **trajectory-level positive learning signals are disproportionately concentrated on a comparatively favorable subset of self-generated prefixes**, whereas inference requires the model to continue from a broader prefix distribution. In other words, trajectory-level optimization does not provide equally direct learning signals for continuation behavior across the prefixes the model may encounter during autoregressive generation. This observation motivates explicitly conditioning policy optimization on self-generated reasoning prefixes.

2.3 Training with Realistic CoT Prefixes

Motivated by the above observation that trajectory-level positive learning signals are concentrated on comparatively favorable prefixes, we next examine whether conditioning RL on a broader set of self-generated prefixes improves reasoning efficiency and accuracy. Specifically, for each problem, we first sample a response from the model and obtain a reasoning prefix by truncating the response at a proportion uniformly drawn from [0,0.5]. Conditioned on this prefix, we then sample multiple continuations, compute their advantages based on the final outcomes, and reinforce these continuation trajectories. We implement this strategy using DAPO with an 8k truncation length on DeepSeek-R1-Distill, and compare it against standard trajectory-level sampling without prefix conditioning.



(a) Accuracy over training steps (b) Response lengths over training steps

Figure 4: Performance of RL-based efficient reasoning with and without conditioning on self-generated prefixes on MATH500 using DeepSeek-R1-Distill-Qwen-1.5B.

As shown in Figure 4, conditioning efficient reasoning RL on realistic, self-generated prefixes leads to rapid compression of generation length and a steady improvement in accuracy as training progresses. Notably, this prefix-conditioned training significantly outperforms conventional efficient reasoning RL that samples trajectories from scratch. These results demonstrate that **alleviating the training–inference gap by training the model to reinforce continuation trajectories conditioned on realistic, potentially suboptimal reasoning prefixes can substantially improve both efficiency and accuracy**. Additional implementation details and results are provided in Appendix A.2.

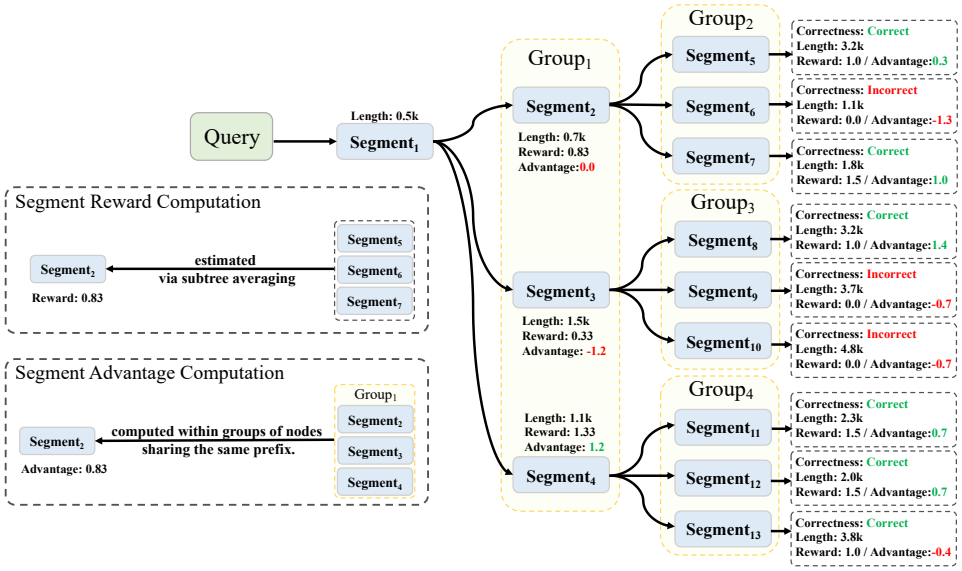


Figure 5: The framework of TLPO. For each query, TLPO performs tree-structured rollouts using reasoning segments as nodes, expanding multiple trajectories via tree-based sampling. Segment nodes that share the same prefix are grouped for advantage estimation. This design enables TLPO to optimize efficient reasoning behaviors under high-quality prefixes (segments$_{11,12}$ with positive advantages), while also providing direct learning signals for efficient continuation under suboptimal self-generated prefixes (segment$_8$ with a positive advantage).

2.4 Takeaways

Our analysis reveals a systematic mismatch in trajectory-level efficient-reasoning RL: positive learning signals are concentrated on comparatively favorable prefixes, while inference requires continuation from a broader set of self-generated prefixes. Directly conditioning RL on these realistic prefixes substantially improves both reasoning accuracy and efficiency. These findings highlight a broader principle for efficient reasoning: models should learn not only to favor short successful trajectories, but also to continue efficiently from the reasoning states they actually encounter.

3 Methods

As discussed above, efficient reasoning requires learning from realistic reasoning paths, including exploratory or erroneous prefixes. To this end, we propose ***T**ree-structured **L**ength-aware **P**olicy **O**ptimization* (**TLPO**), which uses tree-structured sampling to condition optimization on self-generated prefixes with segment-level advantage estimation.

3.1 Flexible Tree-structured Generation

TLPO employs a tree-structured generation process where each node corresponds to a reasoning segment. Given an input query q, we treat q as the root at depth 0 and construct a tree of reasoning segment nodes with depth D. At depth 1, we sample k_1 full responses from the model and obtain k_1 segment nodes by truncating each response at a proportion of its generated length uniformly drawn from $[a_1, b_1]$. For each segment node at depth $i \in \{1, \ldots, D-1\}$, we continue generation from its prefix to sample k_{i+1} responses, and truncate them using proportions uniformly drawn from $[a_{i+1}, b_{i+1}]$. At depth D, corresponding to the leaf nodes, we allow the model to generate full reasoning trajectories until an `[EOS]` token is produced or a maximum length constraint is reached. This yields $\sum_{m=1}^{D-1} \prod_{i=1}^{m} k_i$ intermediate segment nodes and $\prod_{i=1}^{D} k_i$ leaf nodes. In our experiments, we use $D = 3$ with branching factors $(k_1, k_2, k_3) = (1, 8, 8)$ and truncation ranges $[0, 0.4]$ and $[0.2, 0.6]$ for the first two depths. To ensure a fair comparison with standard trajectory-level RL methods, we reduce the training batch size such that the total rollout compute remains comparable.

3.2 Length-aware Reward Strategy under Tree-structured Sampling

Under TLPO, the rollout process samples a tree with D layers and a total of n nodes, where each node corresponds to a reasoning segment. For the i-th segment node, denoted by s_i, the input consists of the original query together with the concatenation of segments along the path from the root to this node, forming a prefix of length $L_{\text{pre}}(s_i)$. The node outputs a reasoning segment, after which its subtree further extends this segment to form multiple complete trajectories that reach a final answer. This tree-structured generation process implies that reward design must explicitly account for the length of the existing prefix. Accordingly, we design a length-aware reward function that conditions on the available remaining context under a given prefix.

We first consider the case where a segment s_i is generated until an `[EOS]` token is produced, corresponding to leaf nodes in the tree. Let $L_{\text{pre}}(s_i)$ denote the length of the input prefix for segment s_i, and let $L(s_i)$ denote the length of the generated segment. We assume a maximum context window length of $L_{\max}$. Following prior work (Liu et al., 2025b), which demonstrates the effectiveness of step-function rewards based on target length, we define the reward for segment s as

$$R(s_i) = \mathbb{I}_{\text{correct}}(s_i)\Big(1 + \beta\mathbb{I}\Big(L(s_i) \le \alpha\big(L_{\max} - L_{\text{pre}}(s_i)\big)\Big)\Big), \tag{1}$$

where $\mathbb{I}_{\text{correct}}$ is an indicator function that evaluates whether the final answer is correct, as determined by rule-based evaluation. Here, α controls the target length relative to the remaining context budget, and β sets the short-response bonus. Incorrect answers receive zero reward.

For non-leaf segment nodes, the model does not directly generate a final answer. Therefore, the reward for an intermediate segment must be estimated based on the outcomes of its descendant trajectories. For a given non-leaf segment node s_i, we denote by $\mathcal{T}(s_i)$ the set of all continuation trajectories that start from s_i and extend it to a final answer at the leaf nodes. We then estimate the quality of s_i by aggregating the correctness and length statistics of trajectories in $\mathcal{T}(s_i)$. Specifically, the reward for a non-leaf segment node s_i is defined as

$$R(s_i) = \frac{1}{|\mathcal{T}(s_i)|}\sum_{\tau\in\mathcal{T}(s_i)} \mathbb{I}_{\text{correct}}(\tau)\Big(1 + \beta\mathbb{I}\Big(L(s_i) + L(\tau) \le \alpha\big(L_{\max} - L_{\text{pre}}(s_i)\big)\Big)\Big), \tag{2}$$

which can be viewed as a Monte Carlo estimate of the expected reward for segment s_i under its subtree. Eqs. equation 1 and equation 2 assign rewards to all nodes in the TLPO tree.

3.3 Segment-level Advantage Estimation and Parameter Update

After computing length-aware rewards under tree-structured sampling, we estimate group-wise advantages and update the policy parameters following DAPO (Yu et al., 2025). Specifically, we group segment nodes that share the same prefix, i.e., those that have the same parent node. For a parent node u, we define the corresponding group as $\mathcal{G}(u) = \{\, s_i \mid p(s_i) = u \,\}$, where $p(s_i)$ denotes the parent of segment node s_i. For each segment $s_i \in \mathcal{G}(u)$, the advantage is computed by normalizing its reward relative to other segments branching from the same prefix:

$$\hat{A}(s_i) = \frac{R(s_i) - \text{mean}\big(\{R(s_j) \mid s_j \in \mathcal{G}(u)\}\big)}{\text{std}\big(\{R(s_j) \mid s_j \in \mathcal{G}(u)\}\big)}. \tag{3}$$

If all continuations within a group yield identical rewards (e.g., all rewards are zero), the resulting advantages are zero and provide no gradient signal. Such degenerate groups are filtered out during training. The computed advantage is then broadcast to all response tokens within the segment.

Formally, for each query $q \in \mathcal{D}$, we sample a tree consisting of $n = \sum_{m=1}^{D}\prod_{i=1}^{m} k_i$ segment nodes, which we denote by the set $\mathcal{S}$. The optimization objective can be written as:

$$\mathcal{J}_{\text{TLPO}}(\theta) = \mathbb{E}_{[q\sim\mathcal{D},\mathcal{S}\sim\pi_{\theta_{\text{old}}}]}\Bigg[\frac{1}{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{S}|}\frac{1}{|s_i|}\sum_{t=1}^{|s_i|}\min \left(\frac{\pi_\theta^{i,t}}{\pi_{\theta_{\text{old}}}^{i,t}}\,\hat{A}(s_i),\ \text{clip}\Big(\frac{\pi_\theta^{i,t}}{\pi_{\theta_{\text{old}}}^{i,t}},\ 1-\varepsilon_{\text{low}},\ 1+\varepsilon_{\text{high}}\Big)\hat{A}(s_i)\right)\Bigg], \tag{4}$$

Table 1: Experimental results on four mathematical reasoning benchmarks and the out-of-domain GPQA-Diamond benchmark. Acc denotes accuracy and Length denotes average reasoning tokens. Avg. reports mean accuracy and reasoning length across all five benchmarks. Best and second-best results within each base-model setting are highlighted in bold and underline, respectively.

Methods	**AIME24**		**AIME25**		**MATH500**		**AMC**		**GPQA**		**Avg.**	
	Acc	Length	Acc	Length	Acc	Length	Acc	Length	Acc	Length	Acc	Length
DeepSeek-R1-Distill-Qwen-1.5B												
Original	29.79	16717	21.88	16093	84.34	5524	70.16	10048	34.41	10108	48.12	11698
AdaptThink	27.70	6305	19.60	5833	80.80	1450	66.60	2931	35.98	8217	46.14	4947
LASER-L2048	25.80	4877	19.20	4497	82.20	1485	72.00	2539	34.85	3956	46.81	3471
LASER-L8192	28.30	6659	23.50	6104	85.60	2747	74.50	4153	34.25	5216	49.23	4976
ThinkPrune	25.40	5975	21.70	5526	83.50	1964	72.50	3068	35.61	4077	47.74	4122
LACONIC	24.37	7632	20.42	6947	82.06	2257	72.50	3911	32.26	2777	46.32	4705
SmartThinker	24.17	9624	22.08	8233	83.83	2668	72.34	4472	34.53	5722	47.39	6144
TLPO	**32.92**	**2953**	**27.29**	**2874**	**87.46**	**1341**	**76.41**	**2003**	**37.06**	**2338**	**52.23**	**2302**
DeepSeek-R1-Distill-Qwen-7B												
Original	52.92	13543	36.88	14849	92.63	3956	90.31	6352	48.93	7116	64.33	9163
AdaptThink	53.12	10541	38.33	11467	91.13	2045	87.03	3974	47.76	5841	63.47	6774
LASER-D-L4096	51.67	5703	41.25	5694	93.01	1785	88.28	2800	47.66	3817	64.37	3960
LASER-DE-L4096	49.38	5052	35.63	5233	92.97	1672	88.12	2566	48.33	3451	62.89	3595
TLPO	**53.87**	**3254**	**41.28**	**3439**	**94.01**	**1495**	**91.18**	**2391**	**50.05**	**2731**	**66.08**	**2662**
DeepScalerR-1.5B-Preview												
Original	**41.30**	9156	**30.20**	9755	**89.00**	3122	80.80	4940	32.89	7833	54.84	6961
ThinkPrune	33.10	5325	26.70	5325	87.50	1924	79.50	2972	35.70	4273	52.50	3964
LACONIC	37.92	5431	28.12	4874	87.35	2106	81.41	3142	34.79	3817	53.92	3874
TLPO	38.80	**3803**	29.80	**3572**	88.50	**1733**	**82.50**	**2502**	**37.25**	**3705**	**55.37**	**3063**

where $\pi^{i,t} = \pi(s_{i,t} \mid q, \mathrm{Pre}(s_i), s_{i,<t})$ denotes the conditional probability of the t-th token in continuation s_i given the query q and the prefix $\mathrm{Pre}(s_i)$. The models π_θ and $\pi_{\theta_{\text{old}}}$ correspond to the training policy and the sampling policy, respectively. The clipping thresholds ε_{low} and $\varepsilon_{\text{high}}$ are hyperparameters used to bound the importance sampling ratio for stable optimization. The complete training procedure is summarized in Appendix B.1.

4 Experiments

4.1 Experimental Setup

We evaluate TLPO on three representative reasoning models: DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025a), and DeepScalerR-1.5B-Preview (Luo et al., 2025b). All models are trained on the 40K-example DeepScaleR-Preview-Dataset (Luo et al., 2025b) for 200 optimization steps with a learning rate of 1×10^{-6}. Unless otherwise specified, TLPO uses a three-layer rollout tree with branching factors $(1, 8, 8)$ and a batch size of 128. We evaluate on four mathematical reasoning benchmarks—AIME24, AIME25 (MAA Committees), MATH500 (Hendrycks et al., 2021), and AMC2023 (AI-MO, 2024)—together with the out-of-domain GPQA-Diamond benchmark (Rein et al., 2023). We compare against released checkpoints of representative efficient-reasoning methods, including AdaptThink (Zhang et al., 2025), ThinkPrune (Hou et al., 2025a), LASER and its adaptive variants (Liu et al., 2025b), LACONIC (Liu et al., 2026), and SmartThinker (Hu et al., 2026), re-evaluating all methods under the same inference protocol. Implementation details, computational overhead, baseline configurations, and evaluation settings are provided in Appendices B.2 and B.3.

4.2 Main Results

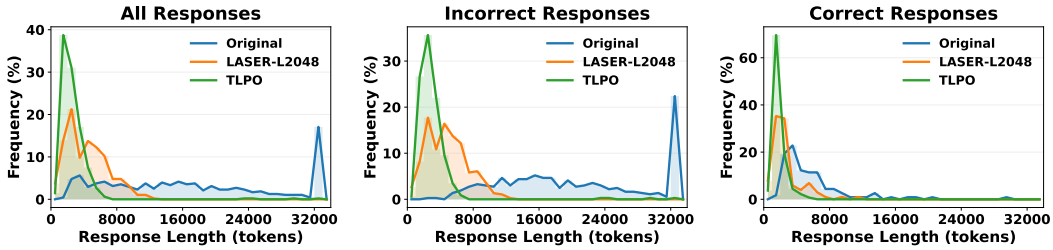
As shown in Table 1, TLPO consistently improves the accuracy–efficiency trade-off across all three base models, reducing average reasoning length by 56.0%–80.3% while improving accuracy by 0.5–

4.1 percentage points. It achieves both the highest average accuracy and shortest average reasoning length in every base-model setting.

On the distilled reasoning models, TLPO improves average accuracy by 4.11 and 1.75 percentage points on DeepSeek-R1-Distill-Qwen-1.5B and 7B, respectively, while reducing average reasoning length by 80.3% and 70.9%. More importantly, across both model scales, TLPO is the only method among all compared baselines that reduces reasoning length on every benchmark without sacrificing accuracy on any of the five evaluation sets. TLPO remains effective on DeepScalerR-1.5B-Preview, where further compression is more challenging because the base model has already undergone extensive reinforcement learning. Even in this setting, TLPO reduces the average reasoning length by 56.0% while slightly improving the average accuracy by 0.53 percentage points, outperforming existing efficient-reasoning baselines in the overall accuracy–efficiency trade-off.

Notably, TLPO is trained exclusively on mathematical reasoning data, yet its improvements also transfer to the out-of-domain GPQA-Diamond benchmark. Across the three base models, TLPO improves GPQA-Diamond accuracy by 1.12–4.36 percentage points while reducing reasoning length by 52.7%–76.9%. This suggests that TLPO does not merely learn domain-specific shortcuts for mathematical problems, but improves the efficiency of continuation reasoning under self-generated prefixes more generally. We provide broader out-of-domain evaluations in Appendix B.4. To further isolate the source of these gains, we provide controlled component ablations in Appendix B.6 and comparisons with representative tree-structured and segment-level RL methods in Appendix B.7.

4.3 Reasoning-Length Distributions



(a) All responses. (b) Incorrect responses. (c) Correct responses.

Figure 6: Response-length distributions on AIME25 for DeepSeek-R1-Distill-Qwen-1.5B, LASER-L2048-1.5B, and TLPO, shown for (a) all responses, (b) incorrect responses, and (c) correct responses.

To better understand how TLPO changes reasoning behavior, we compare the response-length distributions of the original model, LASER-L2048, and TLPO, separately for all, incorrect, and correct responses in Figure 6. TLPO shifts the distribution toward shorter reasoning sequences and suppresses long-tailed generations. Importantly, this effect is particularly pronounced for unsuccessful reasoning: while the original model and LASER still produce long incorrect trajectories, TLPO concentrates failed responses within a shorter range. Specifically, compared with LASER-L2048, TLPO reduces the 90th-percentile length of incorrect responses by 49.0%, from 8,406 to 4,290 tokens.

This result provides behavioral evidence consistent with our central hypothesis. Trajectory-level length-aware RL concentrates positive learning signals on globally short and correct trajectories, leaving continuation behavior over the broader set of self-generated reasoning states less directly optimized. In contrast, TLPO performs prefix-conditioned credit assignment over self-generated reasoning paths. Consequently, its efficiency gains extend beyond successful trajectories: even when reasoning ultimately fails, TLPO substantially reduces prolonged and unproductive computation.

4.4 Performance under Varying Token Budgets

Since there exists a trade-off between accuracy and response length, it is important to evaluate model performance across different token budgets. We train TLPO under varying maximum context window sizes $\{4k, 6k, 8k, 16k\}$, and examine how accuracy varies with the average thinking length. We compare TLPO with six efficient RL baselines: Laser, Laser-D, Laser-DE (Liu et al., 2025b), ThinkPrune (Hou et al., 2025a),

AdaptThink (Zhang et al., 2025), and Leash-1.5B (Li et al., 2025a), evaluated on AIME24. All models are trained on DeepSeek-R1-Distill-Qwen-1.5B. For Laser, Laser-D, Laser-DE, and ThinkPrune, we report results under different hyperparameter settings using released checkpoints, illustrating accuracy–length trade-offs.

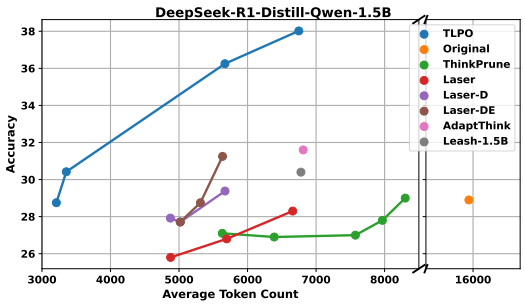


Figure 7: Trade-off between accuracy and response length across various methods on AIME24. Results are obtained under different training hyperparameter settings (p-value < 0.05).

As shown in Figure 7, TLPO achieves higher accuracy as generation length increases, and consistently exhibits a better Pareto-optimal frontier than other methods. Notably, although TLPO is designed to assess the model's performance under efficient reinforcement learning conditioned on self-generated reasoning prefixes sampled from its inference-time distribution, and is therefore trained without adapting the length penalty to problem difficulty, it still outperforms length-aware reinforcement learning methods such as Laser-DE, which explicitly adjust training objectives based on difficulty.

5 Related Work

Recent reasoning models achieve strong performance through test-time scaling (Snell et al., 2024; Welleck et al., 2024), but often suffer from overthinking and excessive computation (Chen et al., 2024; Wang et al., 2025a). Prior approaches improve reasoning efficiency through early exit (Yang et al., 2025; Qiao et al., 2025; Yong et al., 2025; Kapoor et al., 2024; Dai et al., 2025) or CoT compression, including prompting, pruning, and rewriting (Renze & Guven, 2024; Aytes et al., 2025; Han et al., 2025; Xia et al., 2025; Song et al., 2025; Cui et al., 2025; Wang et al., 2025b; Luo et al., 2025a). More recently, a growing body of work has leveraged reinforcement learning to promote reasoning efficiency via reward shaping (Gao et al., 2025; Shen et al., 2025; Song & Zheng, 2025; Liu et al., 2025a). L1 (Aggarwal & Welleck, 2025) is the first RL-based approach for training reasoning language models under explicit length constraints, incorporating length penalties into the reward function. ALP (Xiang et al., 2025) adapts length penalties to problem difficulty, while ADLP (Su & Cardie, 2025) introduces learnable parameters to better balance reasoning accuracy and length. Laser (Liu et al., 2025b) presents a unified framework for efficient reasoning, employing step-function rewards based on target lengths to reduce reasoning length without sacrificing performance.

Despite their differences, existing length-aware RL methods primarily optimize complete trajectories, while tree-structured and segment-level RL methods mainly use branching to improve exploration, credit assignment, or sampling efficiency (Hou et al., 2025b; Guo et al., 2025b; Li et al., 2025b). In contrast, we identify a training–inference mismatch specific to trajectory-level efficient-reasoning RL and formulate efficient reasoning as *prefix-conditioned continuation optimization*. Rather than treating tree sampling as a generic exploration mechanism, TLPO uses it to construct competing continuations from the same self-generated prefix, enabling relative credit assignment over the reasoning states actually induced by the model. Its length-aware objective is further conditioned on the remaining reasoning budget under each prefix, so that efficiency is optimized at the continuation level rather than only at the level of complete trajectories.

6 Conclusion

We identify a training–inference mismatch in trajectory-level efficient-reasoning RL: positive learning signals are concentrated on comparatively favorable prefixes, while inference requires continuation from a broader set of self-generated reasoning states. We address this mismatch with TLPO, which performs prefix-conditioned continuation optimization through tree-structured sampling and segment-level length-aware credit assignment. Across multiple models and benchmarks, TLPO consistently improves reasoning accuracy and efficiency while suppressing prolonged unsuccessful generations. These results highlight prefix-conditioned optimization over self-generated reasoning paths as an effective principle for efficient reasoning.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing and improving the clarity of the manuscript. We did not use generative AI tools for research ideation, experimental design, data analysis, or generating scientific claims. All AI-assisted edits were carefully reviewed and verified by the authors. We take full responsibility for the final content of this work, including any text produced with the aid of generative AI.

ETHICS STATEMENT

This work focuses on improving the reasoning efficiency of large language models and does not involve human subjects, personal data, or sensitive user information. The models, datasets, and benchmarks used in our experiments are publicly available research resources.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed method, training procedure, hyperparameter settings, evaluation protocols, and computational setup in the main paper and appendix. All base models, training datasets, and evaluation benchmarks used in our experiments are publicly available.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

AI-MO. Amc 2023, 2024. URL https://huggingface.co/datasets/AI-MO/aimo-validation-amc.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

Zhengxiang Cheng, Dongping Chen, Mingyang Fu, and Tianyi Zhou. Optimizing length compression in large reasoning models. *arXiv preprint arXiv:2506.14755*, 2025.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, et al. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *arXiv preprint arXiv:2502.08235*, 2025.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, Suhang Wang, Yue Xing, Jiliang Tang, and Qi He. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18581–18597, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.956. URL https://aclanthology.org/2025.findings-acl.956/.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *arXiv preprint arXiv:2505.07686*, 2025.

Jiaxuan Gao, Shu Yan, Qixin Tan, Lu Yang, Shusheng Xu, Wei Fu, Zhiyu Mei, Kaifeng Lyu, and Yi Wu. How far are we from optimal reasoning efficiency? *arXiv preprint arXiv:2506.07104*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *arXiv preprint arXiv:2505.23564*, 2025b.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1274. URL https://aclanthology.org/2025.findings-acl.1274/.

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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint arXiv:2504.01296*, 2025a.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. Treerl: Llm reinforcement learning with on-policy tree search. *arXiv preprint arXiv:2506.11902*, 2025b.

Chenzhi Hu, Qinzhe Hu, Yuhang Xu, Junyi Chen, Ruijie Wang, Shengzhong Liu, Jianxin Li, Fan Wu, and Guihai Chen. Smartthinker: Progressive chain-of-thought length calibration for efficient large language model reasoning. *arXiv preprint arXiv:2603.08000*, 2026.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Sanyam Kapoor, Nate Gruver, Manley Roberts, Katie Collins, Arka Pal, Umang Bhatt, Adrian Weller, Samuel Dooley, Micah Goldblum, and Andrew G Wilson. Large language models must be taught to know what they don't know. *Advances in Neural Information Processing Systems*, 37: 85932–85972, 2024.

Yanhao Li, Lu Ma, Jiaran Zhang, Lexiang Tang, Wentao Zhang, and Guibo Luo. Leash: Adaptive length penalty and reward shaping for efficient large reasoning model. *arXiv preprint arXiv:2512.21540*, 2025a.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, et al. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling. *arXiv preprint arXiv:2508.17445*, 2025b.

Chang Liu, Yiran Zhao, Lawrence Liu, Yaoqi Ye, Csaba Szepesvári, and Lin F Yang. Laconic: Length-aware constrained reinforcement learning for llm. *arXiv preprint arXiv:2602.14468*, 2026.

Hanbing Liu, Lang Cao, Yuanyi Ren, Mengyu Zhou, Haoyu Dong, Xiaojun Ma, Shi Han, and Dongmei Zhang. Bingo: Boosting efficient reasoning of llms via dynamic and significance-based reinforcement learning. *arXiv preprint arXiv:2506.08125*, 2025a.

Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. Learn to reason efficiently with adaptive length-based reward shaping. *arXiv preprint arXiv:2505.15612*, 2025b.

Feng Luo, Yu-Neng Chuang, Guanchu Wang, Hoang Anh Duy Le, Shaochen Zhong, Hongyi Liu, Jiayi Yuan, Yang Sui, Vladimir Braverman, Vipin Chaudhary, et al. Autol2s: Auto long-short reasoning for efficient large language models. *arXiv preprint arXiv:2505.22662*, 2025a.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025b. Notion Blog.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025.

MAA Committees. Aime problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

Ziqing Qiao, Yongheng Deng, Jiali Zeng, Dong Wang, Lai Wei, Guanbo Wang, Fandong Meng, Jie Zhou, Ju Ren, and Yaoxue Zhang. ConCISE: Confidence-guided compression in step-by-step efficient reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8010–8029, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.405. URL `https://aclanthology.org/2025.emnlp-main.405/`.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 476–483. IEEE, 2024.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. DAST: Difficulty-adaptive slow-thinking for large reasoning models. In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2322–2331, Suzhou (China), November 2025. Association for Computational Linguistics. ISBN 979-8-89176-333-3. doi: 10.18653/v1/2025.emnlp-industry.160. URL `https://aclanthology.org/2025.emnlp-industry.160/`.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Jiwon Song, Dongwon Jo, Yulhwa Kim, and Jae-Joon Kim. Reasoning path compression: Compressing generation trajectories for efficient llm reasoning. *arXiv preprint arXiv:2505.13866*, 2025.

Mingyang Song and Mao Zheng. Walk before you run! concise llm reasoning via reinforcement learning, 2025. URL `https://arxiv.org/abs/2505.21178`.

Jinyan Su and Claire Cardie. Thinking fast and right: Balancing accuracy and reasoning length with adaptive rewards. *arXiv preprint arXiv:2505.18298*, 2025.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, et al. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*, 2025a.

Yunhao Wang, Yuhao Zhang, Tinghao Yu, Can Xu, Feng Zhang, and Fengzong Lian. Adaptive deep reasoning: Triggering deep thinking when needed. *arXiv preprint arXiv:2505.20101*, 2025b.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *Transactions on Machine Learning Research*, 2024.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3351–3363, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.165. URL https://aclanthology.org/2025.emnlp-main.165/.

Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning. *arXiv preprint arXiv:2506.05256*, 2025.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.

Xixian Yong, Xiao Zhou, Yingying Zhang, Jinlin Li, Yefeng Zheng, and Xian Wu. Think or not? exploring thinking efficiency in large reasoning models via an information-theoretic lens. *arXiv preprint arXiv:2505.18237*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Danlong Yuan, Tian Xie, Shaohan Huang, Zhuocheng Gong, Huishuai Zhang, Chong Luo, Furu Wei, and Dongyan Zhao. Efficient rl training for reasoning models via length-aware optimization. *arXiv preprint arXiv:2505.12284*, 2025.

Linan Yue, Yichao Du, Yizhi Wang, Weibo Gao, Fangzhou Yao, Li Wang, Ye Liu, Ziyu Xu, Qi Liu, Shimin Di, et al. Don't overthink it: A survey of efficient r1-style large reasoning models. *arXiv preprint arXiv:2508.02120*, 2025.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.184. URL https://aclanthology.org/2025.emnlp-main.184/.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

A Gaps between Training and Inference

A.1 Case Analysis of CoTs with Different Lengths

In this section, we present a case study comparing responses with varying lengths of CoT reasoning generated from multiple samples of the same problem, as illustrated in Figures 8, 9, 10, 11. When the model adheres to a correct reasoning pathway with high-quality initial steps, it frequently converges to the correct answer using a relatively concise CoT. Conversely, when the initial reasoning direction is misguided, the model often produces significantly longer CoTs containing numerous additional tokens, yet still fails to reach the correct solution. This observation underscores the critical role of early reasoning quality and demonstrates the inefficiency of prolonged reasoning when the initial trajectory is flawed.

longer and shorter response in AIME25

Problem: Six points A, B, C, D, E, and F lie in a straight line in that order. Suppose that G is a point not on the line and that $AC = 26$, $BD = 22$, $CE = 31$, $DF = 33$, $AF = 73$, $CG = 40$, and $DG = 30$. Find the area of $\triangle BGE$.?

Shorter Response: <think >Okay, so I've got this geometry problem here. Let me try to visualize it first. There are six points A, B, C, D, E, F lying on a straight line in that order. There's another point G not on this line. We're given several distances: AC = 26, BD = 22, CE = 31, DF = 33, AF = 73, CG = 40, DG = 30. We need to find the area of triangle BGE.Hmm, so all these points are on a straight line, which probably means we can assign coordinates to them. Since they're on a line, maybe I can place them on the x-axis for simplicity and use coordinates to find distances. Let me set up a coordinate system where the line is the x-axis.Let me assign coordinates to points A, B, C, D, E, F on the x-axis. Let's say:Let's define the points on the x-axis as follows:Let's set point A at coordinate 0. Then since AC =26, point C would be at 26.Now, BD = 22. BD is the distance between B and D. Let's denote the position of B as x, so the position of D would be $x + 22$.Similarly, CE = 31. Since C is at 26, E would be at $26 + 31 = 57$.DF =33. Since D is at $x + 22$, F is at $x + 22 + 33 = x + 55$.AF =73. Since A is at 0, F is at 73. So, since F is also $x + 55$, we have $x + 55 = 73$. So $x = 73 - 55 = 18$. Therefore, point B is at 18, C at 26, D at $18 + 22 = 40$, E at 57, F at 73. So summarizing: A:0, B:18, C:26, D:40, E:57, F:73. Now, point G is somewhere not on the line, so it's somewhere in the plane, but the triangle BGE is formed by points B, G, E. We have CG=40 and DG=30. So let's assign coordinates: Let me let's denote point G as (h, k), since it's not on the x-axis. Then, CG =40: distance from C (26,0) to G (h,k) is $\sqrt{(h-26)^2 + k^2} = 40$.Similarly, DG=30: distance from D (40,0) to G is $\sqrt{(h-40)^2 + k^2} = 30$. So we have two equations:1. $(h-26)^2 + k^2 = 40^2 = 1600$.2. $(h-40)^2 + k^2 = 30^2 = 900$.Subtracting equation 2 from equation1:$(h-26)^2 - (h-40)^2 = 1600 - 900 = 700$. Let's compute:$(h-52h+676) - (h-80h+1600) = 700$. Simplify:$(-52h+676) - (-80h+1600) = 700$.Which is:$(-52h+676+80h-1600) = 700$.$(28h-924) = 700$.$28h = 700+924 = 1624$. Thus, h=$1624/28$. Let's compute this: Divide numerator and denominator by 4: $406/7 = 58$.So h=58. Then plug back into equation 1:$(58-26)^2+k^2 = 1600$. So $32^2+k^2 = 1600$.$32^2 = 1024$. So $k^2 = 1600 - 1024 = 576$. Thus k=24 or k=-24. Since the area of triangle BGE depends on the absolute value of k, which is 24. Therefore, the area is $(1/2) * base * height$. Point B is at (18,0), E is at (57,0), G is at (58,24) or (58,-24). Either way, the height is $|k| = 24$.The base is BE: from 18 to 57 is 39 units.Thus, area is $(1/2) * 39 * 24 = 93$. Half of that is 468. So the area of triangle BGE is 468.**Final Answer** $\boxed{468}$ </think >

We place the points A, B, C, D, E, F on the x-axis with coordinates:- A : 0 - B : 18 - C : 26 - D : 40 - E : 57 - F : 73. Given distances:- $AF = 73$ - $BD = 22$ - $CE = 31$ - $DF = 33$ - $CG = 40$ - $DG = 30$. We need to find the coordinates of point G. Using the distances from C and D:1. Distance from $C(26, 0)$ to $G(h, k)$ is 40:$\sqrt{(h-26)^2 + k^2} = 40 \implies (h-26)^2 + k^2 = 16002$. Distance from $D(40, 0)$ to $G(h, k)$ is 30: $\sqrt{(h-40)^2 + k^2} = 30 \implies (h-40)^2 + k^2 = 900$ Subtracting these equations:$(h-26)^2 - (h-40)^2 = 700$ Expanding and simplifying:$28h - 924 = 700 \implies 28h = 1624 \implies h = 58$ Substituting $h = 58$ back:$(58-26)^2 + k^2 = 1600 \implies 32^2 + k^2 = 1600 \implies k^2 = 576 \implies k = 24$ or -24 The height is $|k| = 24$. The base BE is 39 units. The area of $\triangle BGE$ is:Area $= \frac{1}{2} \times 39 \times 24 = 468$Thus, the area of $\triangle BGE$ is $\boxed{468}$. (Correct)

Figure 8: Case Analysis of CoTs with Different Lengths in AIME25

longer and shorter response in AIME25

Longer Response: <think >Okay, let's try to solve this problem step by step. First, I'll try to visualize the problem. There are six points on a straight line: A, B, C, D, E, F. Then there's a point G not on this line. We have several distances given between these points and G. Our goal is to find the area of triangle BGE.Let me list down all the given distances: AC = 26, BD = 22, CE = 31, DF = 33, AF = 73, CG = 40, DG = 30. Hmm, okay. So since all points are on a straight line, we can assign coordinates to each point. Let's assume the line is the x-axis for simplicity.Let me denote the positions of points A, B, C, D, E, F with variables. Let's set the origin at point A. So let's say A is at position $x = 0$.Let's denote the positions as follows:Let's assign coordinates as A = 0.Let's let AB = a, BC = b, CD = c, DE = d, EF = e. But maybe it's better to assign variables in order. (Error1) **Let's denote the positions as:A = 0,B = x,C = x + AC = x + 26,D = C + BD = (x + 26) + 22 = x + 48,E = D + CE = (x + 48) + 31 = x + 79,F = E + DF = (x + 79) + 33 = x + 112.**But we also have AF = 73. Since A is at 0, and F is at $x + 112$. So AF is the distance from A to F, which is x + 112 - 0 = x + 112. But it's given that AF = 73. Therefore,$x + 112 = 73 => x = 73 - 112 = -39$.So A is at 0, B is at x = -39, C is at -39 + 26 = -13, D is at -13 + 22 = 9, E is at 9 + 31 = 40, F is at 40 + 33 = 73.(Error2) **Wait, but AF is given as 73. So if A is at 0 and F is at 73, then AF is 73 units. That makes sense.So let's write down the coordinates:A: 0 B: -39 C: -13 D: 9 E: 40 F: 73.** Okay, that seems consistent.Now, point G is not on the line, but we know distances from G to some points:CG = 40,DG = 30.Hmm, so G is a point such that distance from C (-13) is 40, and distance from D (9) is 30. So perhaps G is somewhere off the line. Since the line is our x-axis, G has coordinates (g, h), where $h \neq 0$.Given that, we can set up equations.From C: distance from G to C is $\sqrt{(g-(-13))^2 + (h-0)^2} = \sqrt{(g+13)^2 + h^2} = 40$. Similarly, from D: $\sqrt{(g-9)^2 + h^2} = 30$ So we have two equations:1. $(g+13)^2 + h^2 = 40^2 = 16002$. $(g-9)^2 + h^2 = 30^2 = 900$ Subtracting equation 2 from equation 1:$(g+13)^2 - (g-9)^2 = 1600 - 900 = 700$ Expanding:$[g^2+26g+169] - [g^2-18g+81] = 700$ Simplify:$g^2+26g+169-g^2+18g-81 = 700$ So, $44g + 88 = 700, 44g = 700 - 88 = 612, g = 612/44 = 143/11 \approx 13.0$ Wait, 612 divided by 44 is 14.045...? Wait, $44 * 14 = 616$, which is more than 612.Wait, $44 * 14 = 616$, so $612 - 44 * 13 = 612 - 572 = 40$.So $612/44 = 13 + 40/44 = 13 + 10/11 = 13.909$ approx.Wait, maybe it's better to write $612/44$. Let's divide numerator and denominator by 4: $612/44 = 153/11 \approx 13.909$. But perhaps exact fractions would be better. So $g = 612/44 = 153/11$.Now, let's compute h from equation 2:$(g-9)^2 + h^2 = 900$ So $h^2 = 900 - (g-9)^2$ First compute $(g-9) = (153/11 - 99/11) = 54/11$ So $(54/11)^2 = 2916/121$ Thus $h^2 = 900 - 2916/121$ Convert 900 to 121 denominator: $900 = 900 * 121/121 = 108900/121$So $h^2 = (108900 - 2916)/121 = 105984/121$ Thus $h = \sqrt{105984/121} = sqrt105984/11$ Let me compute $\sqrt{105984}$:Divide by 16: $105984/16 = 6624$ Again divide by 16: $6624/16 = 414$ Wait, $16 * 16 = 256.16 * 414 = 6624$. So $16 * 16 * 16 = 4096$? Wait, maybe this is getting too complicated.Alternatively, perhaps I made a miscalculation earlier.Wait, maybe it's better to compute $h^2 = 900 - (g-9)^2 = 900 - ((153/11 - 99/11)^2) = 900 - (54/11)^2 = 900 - 2916/121$.Convert 900 to 121 denominator: $900 * 121 = 108900$.So $108900 - 2916 = 105984$.Thus $h^2 = 105984/121$.Compute $\sqrt{105984}$:105984 divided by 16 is 6624.6624 divided by 16 is 414.414 is $2 * 207 = 2 * 3 * 69 = 2 * 3 * 3 * 23$. So $\sqrt{105984} = \sqrt{16 * 16 * 414} = 16 * \sqrt{414}$.Wait, maybe not. Alternatively, perhaps h is $\sqrt{105984/121} = \sqrt{105984}/11$.But maybe I don't need to compute h exactly.Wait, perhaps I can find coordinates of G as (153/11, h), with h being $\sqrt{105984}/11$.

Figure 9: Case Analysis of CoTs with Different Lengths in AIME25

longer and shorter response in AIME25

But maybe I can just keep it as h for now.So coordinates of G are $(153/11, \sqrt{105984}/11)$.Wait, perhaps exact value is not necessary for the area. Let's think about triangle BGE.Point B is at (-39,0), E is at 40,0, so BE is the segment from (-39,0) to (40,0), which is 79 units long. The height of triangle BGE would be the distance from G to line BE, which is the y-coordinate of G, which is h. Thus, area of triangle BGE is $(base * height)/2 = (79 * h)/2$. We need to compute h. From above, $h^2 = 105984/121$=105984 divided by 121.Compute $105984/121$.Divide 105984 by 121:$121 * 876 = 121 * (800 + 76) = 121 * 800 = 96800, 121 * 76 = 9196$, so total $96800+9196 = 105,996$.But 105984 is 12 less than that. Thus, $105984 = 121*(876)-12$.Thus, $h^2 = (121 * 876 - 12)/121 = 876 - 12/121$.Thus $h^2 = 876 - 12/121$.But $\sqrt{876 - 12/121}$ is messy.Wait, perhaps this is incorrect.Wait, perhaps I made a mistake in calculation earlier.Wait, let's compute g correctly.From the equations:$(g + 13)^2 + h^2 = 1600, (g - 9)^2 + h^2 = 900$ Subtracting: $(g + 13)^2 - (g - 9)^2 = 700$Which expands to $44g + 88 = 700$ So $44g = 612$,$g = 61244 = 15311 \approx 13.909$.So correct.Then $h^2 = 900 - (153/11 - 9)^2$Compute $(153/11 - 99/11) = 54/11$So $(54/11)^2 = 2916/121 \approx 24.066$Then $h^2 = 900 - 2916/121$ Convert 900 to over 121: $900 * 121 = 108900$,$108900 - 2916 = 105984$ So $h^2 = 105984/121$ Therefore $h = \sqrt{105984}/11$Compute $\sqrt{105984}$:Let me factor it:$105984 = 16 * 6624, 6624 = 16 * 414, 414 = 2 * 3 * 69 = 2 * 3 * 3 * 23$So $\sqrt{105984} = \sqrt{16 * 16 * 414} = 16 * \sqrt{414} = 16 * \sqrt{9 * 46} = 16 * 3 * \sqrt{46} = 48 * sqrt46$Thus $h = 48 * \sqrt{46}/11$.Therefore, $h = 48\sqrt{46}/11$.Thus, area of triangle BGE is $(base * height)/2 = 79 * (48\sqrt{46}/11)/2 = 79 * (24\sqrt{46}/11) = (79 * 24\sqrt{46})/11$.Compute $79 * 24 : 79 * 20 = 1580, 79 * 4 = 316, total = 1580 + 316 = 1896$.Thus $area = (1896\sqrt{46})/11$.So the answer is $1896\sqrt{46}/11$, which is the same as $(79*24\sqrt{46})/11$.So, probably that's the answer.**Final Answer** $\boxed{\frac{1896\sqrt{46}}{11}}$ </think >

Given six points A, B, C, D, E, F lie on a straight line in that order, and a point G not on this line. We know the distances between these points and G:- $AC = 26$ - $BD = 22$ - $CE = 31$ - $DF = 33$ - $AF = 73$ - $CG = 40$ - $DG = 30$. We need to find the area of triangle $\triangle BGE$.First, we assign coordinates to each point on the x-axis. Let $A = 0$. We then express the positions of the points in terms of x:- $A = 0$ - $B = x$ - $C = x + 26$ - $D = x + 48$ - $E = x + 79$ - $F = x + 112$. Given $AF = 73$, we find $x = -39$. Thus, the coordinates are:- $A = 0$ - $B = -39$ - $C = -13$ - $D = 9$ - $E = 40$ - $F = 73$. Next, we determine the coordinates of G. Using the distances $CG = 40$ and $DG = 30$, we set up the equations:1. $(g + 13)^2 + h^2 = 1600$ 2. $(g - 9)^2 + h^2 = 900$.Subtracting these equations, we find $g = \frac{153}{11}$. Using this, we compute h:- $h^2 = 900 - \left(\frac{54}{11}\right)^2 = \frac{105984}{121}$,$h = \frac{48\sqrt{46}}{11}$. The area of triangle $\triangle BGE$ is calculated using the base $BE = 79$ and height h: Area = $\frac{1}{2} \times 79 \times h = \frac{79}{2} \times \frac{48\sqrt{46}}{11} = \frac{1896\sqrt{46}}{11}$. Thus, the area of $\triangle BGE$ is: $\boxed{\frac{1896\sqrt{46}}{11}}$.(Incorrect)

(**Error:** 1.Misunderstood the length of the line segment provided in the problem, treating it as the distance between adjacent points for calculation. 2.When the mistake was discovered, did not correct it promptly; instead, I continued with the flawed approach.)

Figure 10: Case Analysis of CoTs with Different Lengths in AIME25

longer and shorter response in MATH500

Problem: For the eight counties listed below, what was the median number of students in 2005?

Number of Students per County

COUNTY	2001	2002	2003	2004	2005
Aiken	124	141	130	143	136
Bamberg	17	15	15	14	11
Barnwell	25	22	26	28	29
Berkeley	583	557	554	553	524
Calhoun	15	12	10	18	11
Cherokee	19	13	18	13	19
Chesterfield	46	18	13	22	29
Colleton	64	49	52	46	41

Shorter Response: <think >Okay, so I need to find the median number of students. The table lists eight counties with their student numbers. Since we're looking for the median for 2005, I need to focus on the '2005' column.First, let me recall that the median is the middle value when the numbers are arranged. Since there are eight counties, the median will be the average of the 4th and 5th values after sorting the data.So, let's list out the 2005 numbers:Aiken: 136 Bamberg: 11 Barnwell: 29 Berkeley: 524 Calhoun:11 Cherokee:19 Chesterfield:29 Colleton:41. Let me arrange them in ascending order.11, 11, 19, 29, 29, 41, 136, 524.So that's correct. Since there are eight numbers, the median will be the average of the 4th and 5th terms.That is, $(29 + 29)/2 = 29$. So the median is 29.**Final Answer**The median number is $\boxed{29}$. </think >
To find the median number of students in 2005, we need to look at the '2005' column of the given table and arrange the numbers in ascending order. The numbers for 2005 are: 136, 11, 29, 524, 11, 19, 29, 41.Arranging them in ascending order, we get: 11, 11, 19, 29, 29, 41, 136, 524.Since there are eight numbers, the median is the average of the 4th and 5th values. Therefore, the median is: $\frac{29+29}{2} = 29$.The median number of students in 2005 is $\boxed{29}$. (Correct)

Longer Response: <think >Okay, so I need to find the median number of students per county in 2005 from the given table. Let me recall what median is. The median is the middle value in a sorted list of numbers. If there's an odd number of observations, it's the middle one. If even, it's the average of the two middle ones.Looking at the table, each county has five data points from 2001 to 2005. So for each county, we have five numbers. We need to focus on 2005 numbers.So first, I need to extract the 2005 values for each county. Let me list them out:,- Aiken: 136,- Bamberg: 11,- Barnwell:29,- Berkeley:524,- Calhoun:11,- Cherokee:19,- Chesterfield:29,- Colleton:41.So now, let's list them in order from smallest to largest.First, list all 2005 numbers:11,11,13,18,19,21? Wait, hold on, no.Wait, the 2005 values are:Aiken:136 ,Bamberg:11,Barnwell:29,Berkeley:524,Calhoun:11 ,Cheerokee:19 ,Chesterfield:29 ,Colleton:41. So writing all of them:11 (Bamberg), 11 (Calhoun),13 (Chevron?), wait no, hold on. (Error) **Wait, let's list them all:Looking at 2005 numbers:Aiken:136,Bamberg:11,Barnwell:29,Berkeley:524,Calhoun:11,Cheerokee:19, Chesterfield:29,Colleton:41.So the numbers in 2005 are:136,11,29,524,11,19,29,41.Now, order them:11,11,13,19,29,29,41,136.**So in order:11,11,13,19,29,29,41,136.That's 8 numbers, so an even count.The median is the average of the 4th and 5th terms.Wait, counting them:$1:11, 2:11, 3:13, 4:19, 5:29, 6:29, 7:41, 8:136$. So the two middle numbers are the 4th and 5th:19 and 29. Average is $(19 + 29)/2 = 48/2 = 24$.Therefore, median is 24.**Final Answer**The median number of students in 2005 is $\boxed{24}$. </think >
To find the median number of students in 2005 for the eight counties, we first extract the 2005 values from the table:- Aiken: 136,- Bamberg: 11,- Barnwell: 29,- Berkeley: 524,- Calhoun: 11,- Cherokee: 19,- Chesterfield: 29,- Colleton: 41. Next, we list these values in ascending order:11, 11, 13, 19, 29, 29, 41, 136.Since there are 8 values (an even number), the median is the average of the 4th and 5th values. The 4th value is 19 and the 5th value is 29. The median is calculated as follows:Median $= \frac{19+29}{2} = \frac{48}{2} = 24$.Thus, the median number of students in 2005 is $\boxed{24}$. (Incorrect)

(**Error:** Fabricated a non-existent value "13")

Figure 11: Case Analysis of CoTs with Different Lengths in MATH500

A.2 Training with Realistic CoT Prefixes

In this section, we provide additional implementation details for experiments on RL-based efficient reasoning, with and without conditioning on self-generated CoT prefixes. For a fair comparison, both settings use the same maximum generation length of 8k tokens. We adopt the length-penalty formulation from DAPO, with an overlong buffer length of 4096 and an overlong penalty factor set to 1. The training batch size is set to 128, with 8 samples generated per prompt. All models are trained using a learning rate of $1e{-}6$.

B Experiments

B.1 Detailed Training Algorithm

Algorithm 1 **TLPO**: Tree-structured Length-aware Policy Optimization

Input initial policy π_θ; task prompts $\mathcal{D}$; training steps s; tree depth D; branching factors $\{k_d\}_{d=1}^{D}$; truncation ranges $\{[a_d, b_d]\}_{d=1}^{D-1}$; max length $L_{\max}$; reward hyperparameters α, β; clipping thresholds $\varepsilon_{\texttt{low}}, \varepsilon_{\texttt{high}}$; update steps μ
for step = 1,...,s **do**
Sample a mini-batch $\mathcal{D}_b$ from $\mathcal{D}$
for each query $q \in \mathcal{D}_b$ **do**
for depth $d = 1, \ldots, D$ **do**
Sample k_d segment nodes s from each node at depth $d-1$ using $\pi_{\theta_{\text{old}}}$
Truncate each segment s at $u \sim \text{Unif}[a_d, b_d]$ if $d < D$; otherwise generate s to `[EOS]` or max length
Compute node rewards $\{R(s)\}$ according to Eq. equation 1 and Eq. equation 2
Compute group-wise advantages $\{\hat{A}(s)\}$ according to Eq. equation 3
Discard groups with zero reward variance
for iteration = 1,...,μ **do**
Update the policy model π_θ by maximizing $\mathcal{J}_{\text{TLPO}}(\theta)$ (Eq. equation 4)
Update the old policy $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
Output updated policy π_θ

Algorithm 1 summarizes the complete training procedure of TLPO. For each mini-batch of queries, TLPO first constructs tree-structured rollouts using the current sampling policy, where intermediate generations are truncated to form self-generated reasoning prefixes and leaf nodes are generated to completion. It then computes length-aware rewards for all segment nodes, estimates group-wise advantages among sibling segments sharing the same prefix, and updates the policy using the segment-level objective defined in Eq. 4. Groups with identical rewards are discarded since they provide no effective learning signal.

B.2 Training Details and Efficiency

In this section, we provide additional training details for TLPO. All models are trained on the DeepScaleR-Preview-Dataset (Luo et al., 2025b) for 200 optimization steps with a learning rate of 1×10^{-6}. Unless otherwise specified, TLPO uses a three-layer rollout tree with branching factors $(1, 8, 8)$ and a prompt batch size of 128.

For experiments based on DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B, we set the maximum context window length to 6k tokens. The reward hyperparameters are set to $\alpha = 0.35$ and $\beta = 0.5$. The clipping thresholds are $\varepsilon_{\texttt{low}} = 0.2$ and $\varepsilon_{\texttt{high}} = 0.28$.

For experiments based on DeepScalerR-1.5B-Preview, the maximum context window length is set to 8k tokens. The reward hyperparameters are set to $\alpha = 0.5$ and $\beta = 0.5$. The clipping thresholds are $\varepsilon_{\texttt{low}} = 0.2$ and $\varepsilon_{\texttt{high}} = 0.28$.

We further compare the per-step training time between TLPO and DAPO under settings with the same total number of rollouts. Specifically, we evaluate DAPO with a batch size of 512 and 16

rollout samples per prompt, and TLPO with a batch size of 128 and $1 \times 8 \times 8$ rollout samples under tree-structured sampling. Experiments are conducted on a 16-node H20 GPU cluster. Under these settings, a single training step of TLPO takes 2059 seconds, while a training step of DAPO takes 2945 seconds. This difference arises because TLPO reuses self-generated prefixes across multiple branches during rollout generation, thereby reducing rollout computation. Overall, TLPO achieves a more favorable Pareto frontier of reasoning efficiency while also improving training efficiency.

We also measure peak GPU memory usage on 8 NVIDIA A100-PCIE-40GB GPUs. Under the default configuration, TLPO reaches a peak memory usage of 39,580 MB, compared to 35,460 MB for DAPO. This indicates that TLPO introduces only a modest memory overhead. During rollout generation, TLPO reuses shared prefixes in the tree and therefore requires less memory than independently generating complete trajectories for all leaf nodes. During training, TLPO introduces additional segment nodes from the non-leaf parts of the tree, which increases memory usage. However, this additional cost is limited compared with the number of complete leaf-level trajectories, keeping the overall memory increase small.

B.3 Evaluation Details

In this section, we provide additional details on the evaluation settings. All evaluations are conducted with a maximum generation length of 32,768 tokens. We use a sampling temperature of 0.6 and generate 32 responses per prompt. For nucleus sampling, the top-p parameter is set to 0.95. Statistical significance is assessed using paired bootstrap resampling over benchmark problems, comparing per-problem accuracy and response length between TLPO and each baseline. We report significance at $p < 0.05$.

B.4 Out-of-Domain Evaluation

TLPO is trained exclusively on mathematical reasoning data. To evaluate its out-of-domain generalization, we further test DeepSeek-R1-Distill-Qwen-1.5B on GPQA-Diamond (Rein et al., 2023) and three code benchmarks: HumanEval (Chen et al., 2021), LiveCodeBench (Jain et al., 2025), and BigCodeBench (Zhuo et al., 2025). These benchmarks cover substantially different task distributions, including scientific reasoning, standard function-level code generation, recent contamination-resistant competitive programming, and practical code generation involving complex instructions and compositional function use. In particular, BigCodeBench extends beyond short self-contained code synthesis toward more task-oriented coding with multiple functions used compositionally, providing complementary evidence toward tool-oriented reasoning, although it does not constitute a full long-horizon agentic evaluation. We compare TLPO with the original model, LASER-L2048 (Liu et al., 2025b), and AdaptThink (Zhang et al., 2025).

Table 2: Out-of-domain evaluation of DeepSeek-R1-Distill-Qwen-1.5B. We report accuracy (%) and average reasoning length. Avg. denotes the unweighted mean across the four benchmarks. Best and second-best results are highlighted in bold and underline, respectively.

	HumanEval		**LiveCodeBench**		**BigCodeBench**		**GPQA-Diamond**		**Avg.**	
Method	Acc.	Length	Acc.	Length	Acc.	Length	Acc.	Length	Acc.	Length
Original	40.90	8309	25.88	13227	7.50	5768	34.41	10108	27.17	9353
AdaptThink	51.20	6062	27.18	10118	<u>8.31</u>	3014	<u>35.98</u>	8217	30.67	6853
LASER-L2048	<u>56.70</u>	<u>5002</u>	<u>27.44</u>	<u>5931</u>	7.80	<u>2213</u>	34.85	<u>3956</u>	<u>31.70</u>	<u>4276</u>
TLPO	**60.40**	**4081**	**29.05**	**4023**	**10.42**	**2018**	**37.06**	**2338**	**34.23**	**3115**

As shown in Table 2, TLPO consistently transfers its accuracy–efficiency gains beyond the mathematical training distribution, achieving both the highest accuracy and the shortest reasoning length on all four out-of-domain benchmarks. On average, TLPO improves accuracy from 27.17% to 34.23% over the original model while reducing reasoning length from 9,353 to 3,115 tokens, corresponding to a 7.06 percentage-point accuracy gain and a 66.7% length reduction. TLPO also consistently outperforms both LASER-L2048 and AdaptThink in average accuracy and reasoning length.

The consistent gains across scientific reasoning, recent competitive programming, and practical code generation with compositional function use suggest that TLPO learns a general capability for efficient continuation reasoning rather than domain-specific shortcuts for mathematical problems.

B.5 Training Dynamics of TLPO

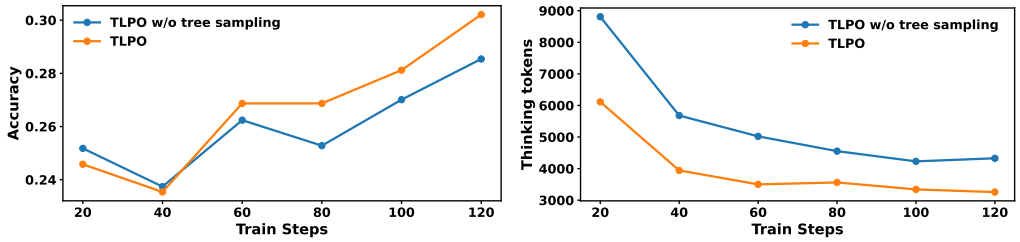


Figure 12: Training dynamics of TLPO with and without tree-structured sampling.

Figure 12 illustrates the training dynamics of TLPO, comparing variants with and without tree-structured sampling. With tree sampling enabled, the model exhibits more consistent improvements in reasoning accuracy while simultaneously reducing reasoning length as training progresses. These results suggest that performing efficient reinforcement learning conditioned on self-generated reasoning prefixes—drawn from the model's inference-time distribution—facilitates more effective optimization, leading to a more favorable Pareto frontier between reasoning accuracy and efficiency.

B.6 Ablation and Sensitivity Analysis

We conduct ablation studies to examine the contributions of the main components of TLPO and its sensitivity to different tree structures and length-reward configurations.

Main components. We first ablate the two key components of TLPO: tree-structured prefix-conditioned sampling and the length-aware reward. For the variant without tree sampling, we replace the $(1, 8, 8)$ tree with independent root-level rollouts while keeping the number of terminal continuations matched, such that the comparison does not benefit from additional rollout samples. For the variant without the length penalty, we retain tree-structured sampling and segment-level advantage estimation, but remove the explicit length-aware reward.

Table 3: Component ablation of TLPO on AIME24.

Method	**Acc.** (%)	**Length**
Original	29.79	16717
TLPO w/o Tree Sampling	28.54	4327
TLPO w/o Length Bonus	31.42	4533
TLPO	**32.92**	**2953**

As shown in Table 3, removing tree-structured sampling degrades both accuracy and reasoning efficiency, reducing accuracy from 32.92% to 28.54% and increasing the average response length from 2,953 to 4,327 tokens. This result indicates that optimizing continuations conditioned on self-generated prefixes provides substantially more effective credit assignment than independent trajectory-level rollouts under a matched terminal-rollout budget.

Removing the length-aware reward has a different effect. The resulting model retains relatively strong accuracy (31.42%) but produces substantially longer responses, with the average length increasing from 2,953 to 4,533 tokens. Equivalently, adding the length-aware objective to the tree-based variant further reduces reasoning length by 34.9% while improving accuracy by 1.50 percentage points. These results suggest complementary roles for the two components: tree-structured prefix-conditioned optimization is critical for learning effective continuation behavior, while the length-aware reward explicitly encourages efficient reasoning under each prefix.

Table 4: Sensitivity of TLPO to tree branching factors and the length-reward hyperparameter α. We report accuracy (%) and average response length on AIME24, AIME25, and MATH500. All TLPO variants use 64 leaf rollouts and are trained for at most 120 optimization steps.

	AIME24		AIME25		MATH500		Avg.	
Method	Acc.	Length	Acc.	Length	Acc.	Length	Acc.	Length
Trajectory-level length-aware RL								
LASER-L2048	25.8	4877	19.2	4497	82.2	1485	42.4	3620
LASER-L8192	28.3	6659	23.5	6104	85.6	2747	45.8	5170
Default configuration								
TLPO $(1, 8, 8)$, $\alpha = 0.35$	32.1	3207	24.6	2651	87.0	1356	47.9	2405
Branching-factor ablation								
TLPO $(4, 4, 4)$, $\alpha = 0.35$	31.7	2161	24.4	1774	86.1	789	47.4	1575
TLPO $(1, 16, 4)$, $\alpha = 0.35$	31.5	3525	27.1	2786	87.2	1307	48.6	2539
TLPO $(1, 4, 16)$, $\alpha = 0.35$	32.7	4025	25.6	3199	87.2	1468	48.5	2897
Length-reward hyperparameter ablation								
TLPO $(1, 8, 8)$, $\alpha = 0.2$	31.3	4318	23.1	2713	86.1	1044	46.8	2692
TLPO $(1, 8, 8)$, $\alpha = 0.5$	31.9	4501	26.9	3913	88.1	1909	49.0	3441

Sensitivity to tree structure and length reward. We further study the sensitivity of TLPO to the allocation of tree branching factors and the length-reward hyperparameter α. For the tree structure, we compare different branching-factor allocations while fixing the number of leaf rollouts to 64. For the length-aware reward, we vary α, which controls the target continuation length relative to the remaining token budget. The bonus coefficient β is fixed to 0.5 throughout these experiments, following prior length-aware RL settings (Liu et al., 2025b). Due to computational constraints, all sensitivity variants are trained for at most 120 optimization steps. We additionally report LASER-L2048 and LASER-L8192 as trajectory-level length-aware RL references.

Table 4 shows that TLPO remains effective across substantially different tree structures. Although different branching allocations result in different operating points on the accuracy–length trade-off, all tested configurations achieve higher average accuracy while producing shorter responses than the trajectory-level LASER baselines. The balanced $(4, 4, 4)$ structure yields the shortest average responses, whereas the asymmetric $(1, 16, 4)$ and $(1, 4, 16)$ configurations achieve slightly higher average accuracy. In particular, the strong performance of $(4, 4, 4)$ also shows that TLPO remains effective when the first-level segments are directly optimized rather than used only as sampled conditioning prefixes.

We observe a similarly stable trend when varying α. A larger α relaxes the target-length constraint and generally shifts the model toward longer but more accurate responses, whereas a smaller α imposes a stronger efficiency constraint. Importantly, both tested settings retain a favorable accuracy–efficiency trade-off relative to the trajectory-level baselines. Overall, these results indicate that the effectiveness of TLPO does not rely on a narrowly tuned branching structure or length-reward target.

B.7 Comparison with Tree-Structured RL Methods

Since TLPO employs tree-structured sampling and segment-level credit assignment, we additionally compare it with representative tree-structured and segment-level policy optimization methods. Specifically, we consider SPO-tree (Guo et al., 2025b) and TreePO (Li et al., 2025b). To control major experimental factors, all methods in this comparison use the same DeepSeek-R1-Distill-Qwen-1.5B base model, training data, training-time maximum generation length, branching budget, and evaluation protocol.

As shown in Table 5, SPO-tree and TreePO also produce substantially shorter reasoning trajectories than the original model, partly because they are trained with the same maximum generation length as TLPO. However, unlike TLPO, these methods are not specifically designed to optimize reasoning efficiency and do not incorporate an explicit length-aware objective. Under the same controlled setting, TLPO achieves both higher accuracy and shorter reasoning length than SPO-tree and TreePO

Table 5: Controlled comparison with tree-structured and segment-level RL methods. All methods use the same base model, training data, training-time maximum generation length, branching budget, and evaluation protocol.

	AIME25		MATH500	
Method	**Acc.** (%)	**Length**	**Acc.** (%)	**Length**
Original	21.88	16093	84.34	5524
SPO-tree	<u>24.92</u>	4522	<u>87.08</u>	2045
TreePO	23.75	<u>3925</u>	84.99	<u>1765</u>
TLPO	**27.29**	**2874**	**87.46**	**1341**

on both benchmarks. Compared with SPO-tree, TLPO improves accuracy by 2.37 percentage points on AIME25 and 0.38 points on MATH500, while reducing reasoning length by 36.4% and 34.4%, respectively. Compared with TreePO, TLPO improves accuracy by 3.54 and 2.47 percentage points, while reducing reasoning length by 26.8% and 24.0%.

Together with the component ablation in Table 3, these results show that the gains of TLPO cannot be explained by tree-structured or segment-level optimization alone. Prior tree-structured RL methods primarily use branching to improve exploration or sample efficiency, rather than to address the training–inference mismatch in efficient reasoning. In contrast, TLPO explicitly identifies this mismatch and combines prefix-conditioned tree optimization with a length-aware objective to optimize efficient continuations under realistic self-generated prefixes. To our knowledge, TLPO is the first to formulate tree-structured policy optimization specifically for efficient reasoning, and the results demonstrate substantial gains over both trajectory-level efficient-reasoning methods and generic tree-structured RL baselines.

C Limitations.

Due to computational constraints, our experiments are limited to the model scales and reasoning-oriented benchmarks considered in this work. We have not yet fully explored how TLPO scales to larger models or generalizes to broader domains, such as agentic tasks involving long-horizon interaction and tool use. A more systematic study across model scales and task families remains an important direction for future work.

D Impact Statement

This paper presents work aimed at advancing the field of reasoning with large language models. As with all research involving large language models, there are inherent risks, including the spread of misinformation and the propagation of societal biases.

E Case Analysis of TLPO

Here we present a case study comparing ThinkPrune, Laser-L2048, and our proposed method in Figures 13, 14, 15, and 16. The evaluation employs checkpoints derived from two base models: Deepscaler-Preview-1.5B and DeepSeek-R1-Distill-Qwen1.5B. Responses for comparison were sampled from these checkpoints on identical problem instances. In contrast to existing efficient reasoning approaches, our method attains the greatest reduction in reasoning length without compromising answer correctness.

Problem

Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.

Response from DeepScaler

<think >
Okay, so I need to find the sum of all integer bases $b > 9$ such that 17_b divides 97_b. Hmm, let's break this down step by step. First, I know that numbers in base b can be converted to decimal (base 10) by expanding each digit multiplied by b raised to the power of its position. Since both numbers are in base b, let's convert them to base 10. Starting with 17_b. In base b, the first digit from the right is the units place (b^0), and the next is the base (b^1). So, 17_b would be equal to $1 \times b + 7 \times 1$ in decimal. That is, $b + 7$.
Similarly, 97_b would be $9 \times b + 7 \times 1$ in decimal. So, $9b + 7$. The problem states that 17_b is a divisor of 97_b, which means 97_b divided by 17_b should be an integer. In decimal terms, that means $(9b + 7)$ divided by $(b + 7)$ is an integer.So,mathematically, we can write: $\frac{9b+7}{b+7}$ is an integer. Let me denote this as k, where k is an integer. Therefore, we have:$9b + 7 = k(b + 7)$.Let me rearrange this equation:$9b + 7 = kb + 7k$. Bringing all terms to one side:$9b + 7 - kb - 7k = 0$. Factor terms with b:$(9 - k)b + (7 - 7k) = 0$. Let me solve for b:$(9 - k)b = 7k - 7$. Thus,$b = \frac{7k-7}{9-k}$. Since b must be an integer greater than 9, let's consider the expression: $b = \frac{7(k-1)}{9-k}$. Simplify numerator and denominator: Note that $7(k - 1)$ and $(9 - k)$ must both be integers, and since $b > 9$, we need to find integer values of k such that b is an integer greater than 9.
Let me note that $b > 9$, so let's write $b = \frac{7(k-1)}{9-k}$. Let's denote $m = 9 - k$, so $k = 9 - m$. Substituting back:$b = \frac{7((9-m)-1)}{m} = \frac{7(8-m)}{m} = \frac{56-7m}{m} = \frac{56}{m} - 7$. Since b must be an integer, m must be a divisor of 56. Also, since $b > 9$, let's find the possible values of m such that $b = \frac{56}{m} - 7 > 9$. So,$\frac{56}{m} - 7 > 9\frac{56}{m} > 1656 > 16mm < \frac{56}{16}m < 3.5$ Since m is a positive integer divisor of 56 (because $m = 9 - k$, and k is positive integer because $9b + 7$ is positive, so k must be positive as well).
Wait, actually, m can be negative or positive, but since $b > 9$, let's see. But perhaps it's better to note that since b must be an integer greater than 9, let's look for integer values m such that $\frac{56}{m} - 7$ is integer and greater than 9. Therefore, $\frac{56}{m}$ must be an integer because b is integer. Thus, m must be a positive divisor of 56. So, let's list all positive divisors of 56: 1, 2, 4, 7, 8, 14, 28, 56. But since $m < 3.5$, as we saw earlier, but m is a positive divisor, so possible m values are 1, 2, 4, 7, 8. Wait, but let's verify:

. . . . . .

The sum of all such integer bases is $\boxed{70}$. (Correct)

[Total Tokens: 2619]

Response from ThinkPrune

<think >
Okay, let me try to figure out this problem step by step. The question is: Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.First, I need to understand what 17_b and 97_b represent in decimal. Since these are numbers in base b, I should convert them into base 10 to work with them more easily.Starting with 17_b: In base b, this is equal to $1 \times b + 7 \times 1 = b + 7$ in decimal.Similarly, 97_b: In base b, this is equal to $9 \times b + 7 \times 1 = 9b + 7$ in decimal.The problem states that 17_b is a divisor of 97_b. So in decimal terms, this means that $b+7$ divides $9b+7$.

. . . . . .

The final answer is $\boxed{70}$ (Correct)

[Total Tokens: 2124]

Response from Ours

<think >
Okay, so I've got this problem: Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b. First, I know that when a number is written in base b, each digit represents a power of b. So let's write that down:$17_b = 1 \times b + 7 = b + 7$ $97_b = 9 \times b + 7 = 9b + 7$ Now, the problem says that 17_b is a divisor of 97_b. If $b + 7$ divides $9b + 7$, then the remainder when $9b + 7$ is divided by $b + 7$ must be zero.So let's do polynomial division or use the remainder theorem. The remainder when dividing $9b + 7$ by $b + 7$ can be found by substituting $b = -7$ into $9b + 7$.Let's compute that:$9(-7) + 7 = -63 + 7 = -56$.So:$|-56|$ must be divisible by $b + 7$ .Since $b > 9$, $b + 7 > 16$. So possible values for $b + 7$ are 28 and 56.Therefore,Case 1: $b + 7 = 28 \rightarrow b = 21$Case 2: $b + 7 = 56 \rightarrow b = 49$ So possible bases are 21 and 49.Therefore, the sum is 21 +49=70.**Final Answer** The sum is $\boxed{70}$.
</think >
To solve the problem, we start by converting the base b numbers 17_b and 97_b into their decimal equivalents.$17_b = 1 \times b + 7 = b + 7$ $97_b = 9 \times b + 7 = 9b + 7$ We need $b + 7$ to be a divisor of $9b + 7$. Using the remainder theorem, we find the remainder when $9b + 7$ is divided by $b + 7$:The remainder is $9(-7) + 7 = -63 + 7 = -56$ Thus, $b + 7$ must divide 56.Since $b > 9$, $b + 7$ must be a divisor of 56 greater than 16. The divisors of 56 greater than 16 are 28 and 56. Therefore:$b + 7 = 28$ gives $b = 21$ $b + 7 = 56$ gives $b = 49$ The possible bases are 21 and 49. Summing these bases:$21 + 49 = 70$ Thus, the sum is $\boxed{70}$. (Correct)

[Total Tokens: 1204]

Figure 13: Case Analysis of Responses from DeepScaler,ThinkPrune and Ours in AIME25

Problem

Evaluate $\lceil\sqrt{20}\rceil^2$.

Response from DeepScaler

<think >
Okay, so I need to evaluate the expression $\lceil\sqrt{20}\rceil^2$. Hmm, let's break this down step by step. First, I remember that the ceiling function, denoted by $\lceil x\rceil$, gives the smallest integer that is greater than or equal to x. So, if I can figure out what $\sqrt{20}$ is approximately, I can then apply the ceiling function to it. Once I have that ceiling value, I need to square it to get the final answer. Alright, let's start by calculating $\sqrt{20}$. I know that $\sqrt{16}$ is 4 because $4^2 = 16$, and $\sqrt{25}$ is 5 because $5^2 = 25$. Since 20 is between 16 and 25, $\sqrt{20}$ should be somewhere between 4 and 5. To get a better approximation, maybe I can think about how close 20 is to 16 and 25.Alternatively, perhaps using a calculator would be more straightforward, but since I don't have one handy, maybe I can approximate it. Let's see, 4.4 squared is 19.36, because $4.4^2 = (4 + 0.4)^2 = 16 + 2 * 4 * 0.4 + 0.16 = 16 + 3.2 + 0.16 = 19.36$. Then, 4.5 squared is 20.25, since $4.5 * 4.5 = 20.25$. Wait, that's interesting because 20 is less than 20.25. So, if $\sqrt{20}$ is between 4.4 and 4.5.But wait, let's verify that. If I take 4.4 squared, it's 19.36. Then, 4.4 squared is less than 20, so I need to find how much more beyond 4.4 it is. Let's compute $4.4^2 = 19.36$, which is less than 20. Then, perhaps trying 4.43 squared. Wait, maybe another approach.Alternatively, perhaps use linear approximation or think in terms of differences. Since $4.4^2 = 19.36$, which is 0.64 less than 20. Then, each additional 0.01 increases the square by approximately $2 * 4.4 * 0.01 + (0.01)^2$. Wait, maybe this is getting too complicated.Alternatively, perhaps just compute $4.4^2 = 19.36$, so $4.4^2 = 19.36$, and $4.5^2 = 20.25$. Since 20 is between 19.36 and 20.25, it means that $\sqrt{20}$ is between 4.4 and 4.5. To get a better approximation, let's see how much more beyond 4.4 we need to get from 19.36 to 20.The difference between 20 and 19.36 is 0.64. Since each increment of 0.01 in x increases x^2 by approximately $2*4.4*0.01 = 0.088$. Wait, perhaps this isn't the best way.Alternatively, perhaps just compute 4.4^2 is 19.36, so$4.4 + (0.64)/(2 * 4.4 + something)$. Wait, perhaps this is getting too convoluted. Maybe it's better to note that since $4.4^2 = 19.36$ and $4.5^2 = 20.25$, and 20 is between these two, so $\sqrt{20}$ is between 4.4 and 4.5.
But perhaps for the ceiling function, we just need to know whether $\sqrt{20}$ is less than 5 or not. Wait, since 20 is less than 25, so $\sqrt{20}$ is less than 5, but greater than 4. So, $\lceil\sqrt{20}\rceil$ would be the smallest integer greater than or equal to $\sqrt{20}$.

. . . . . .

Thus, the value of $\lceil\sqrt{20}\rceil^2$ is $\boxed{25}$. (Correct)

[Total Tokens: 4825]

Response from ThinkPrune

<think >
Okay, so I have this problem: evaluate $\lceil\sqrt{20}\rceil^2$. Hmm, I need to figure out what this means and then compute it step by step. First, let me recall what the notation $\lceil x\rceil^2$ means. I think it represents the ceiling function, which is the smallest integer greater than or equal to x. So, if I have the ceiling of $\sqrt{20}$, that would be the next integer above $\sqrt{20}$.Alright, so let me break it down. First, compute $\sqrt{20}$. I know that $\sqrt{16}$ is 4 and $\sqrt{25}$ is 5, so $\sqrt{20}$ should be somewhere between 4 and 5. To get a better approximation, let me calculate it.I remember that 4.4 squared is 19.36 and 4.5 squared is 20.25. Since 20 is between 19.36 and 20.25, $\sqrt{20}$ must be between 4.4 and 4.5.

. . . . . .

Therefore, the value is $\boxed{25}$. (Correct)

[Total Tokens: 2005]

Response from Ours

<think >
Okay, so I have this problem: evaluate the ceiling of the square root of 20, and then square that result. Hmm. Let's break it down step by step.First, I need to find the square root of 20.
I remember that $\sqrt{20}$ is the same as $\sqrt{4 * 5}$ which is $2 * \sqrt{5}$. And $\sqrt{5}$ is approximately 2.236, so $2 * 2.236$ is about 4.472. So $\sqrt{20} \approx 4.472$.Now, the ceiling function, denoted by ceil(x), gives the smallest integer greater than or equal to x. So since $\sqrt{20}$ is approximately 4.472, the ceiling would be the next integer, which is 5. Because 4.472 is between 4 and 5, but the ceiling function rounds it up to the next integer.So, $ceil(\sqrt{20}) = 5$.Now, the problem says to square this result. So 5 squared is 25. Therefore, the final answer should be 25.
Wait, let me just verify:
1. Compute $\sqrt{20}$: yes, it's about 4.472.
2. Apply ceiling function: since 4.472 is not an integer, ceiling goes to the next integer, which is 5.
3. Square 5: $5 * 5 = 25$.
Final Answer The value is $\boxed{25}$.
</think >
To evaluate $\lceil\sqrt{20}\rceil^2$:
1. Calculate the square root of 20: $\sqrt{20} \approx 4.472$.
2. Apply the ceiling function to 4.472, which results in 5 because the ceiling function rounds up to the next integer.
3. Square the result: $5^2 = 25$.
Thus, the final answer is $\boxed{25}$. (Correct)

[Total Tokens: 580]

Figure 14: Case Analysis of Responses from DeepScaler,ThinkPrune and Ours in MATH500

Problem

The 9 members of a baseball team went to an ice cream parlor after their game. Each player had a singlescoop cone of chocolate, vanilla, or strawberry ice cream. At least one player chose each flavor, and the number of players who chose chocolate was greater than the number of players who chose vanilla, which was greater than the number of players who chose strawberry. Let N be the number of different assignments of flavors to players that meet these conditions. Find the remainder when N is divided by 1000.

Response from DeepSeek-R1-Distill-Qwen1.5B

<think >
Alright, so I've got this problem here about 9 baseball team members going to an ice cream parlor. Each of them had a singlescoop cone, and the possible flavors were chocolate, vanilla, or strawberry. The conditions are that each flavor was chosen by at least one player, chocolate was chosen by more people than vanilla, and vanilla was chosen by more people than strawberry. I need to find the number of different assignments of flavors to players that meet these conditions, and then find the remainder when that number is divided by 1000.
Okay, let's break this down. So, each of the 9 players can choose chocolate, vanilla, or strawberry. So, without any restrictions, the total number of assignments would be 3^9, which is 19683. But we have some constraints: each flavor must be chosen by at least one player, chocolate is chosen more than vanilla, and vanilla is chosen more than strawberry.Hmm. So, first, maybe I should figure out how many ways there are to assign the flavors without considering the order, and then adjust for the constraints.Wait, but the problem is about assigning flavors with order, since each player is distinct. So, it's more like counting the number of colorings with 3 colors, each color appearing at least once, and with the additional condition that chocolate $>$ vanilla $>$ strawberry.I think I need to model this with some combinatorial methods.First, the problem is similar to counting the number of onto functions from a set of 9 elements (players) to a set of 3 elements (flavors), with the added constraints on the counts of each flavor.But the constraints are not only that each flavor is used at least once, but also that the counts satisfy chocolate $>$ vanilla $>$ strawberry.
So, perhaps I can think of this as a problem of counting the number of ordered triples (c, v, s) where $c + v + s = 9, c, v, s \geq 1$, and $c > v > s$.
Once I have the number of such triples, I can multiply by the number of assignments for each triple. But wait, no, because each assignment is a specific assignment of flavors to players, so for each such triple, the number of assignments is $9!/(c!v!s!)$, right? Because we need to assign each player to a flavor, considering the counts.
So, perhaps the total number is the sum over all c, v,s with $c + v + s = 9, c, v, s \geq 1, c > v > sof(9!)/(c!v!s!)$.

. . . . . .

Thus, the remainder when N is divided by 1000 is $\boxed{16}$. (Correct)

[Total Tokens: 10228]

Response from LASER-L2048

<think >
Alright, so I'm trying to solve this problem. There are 9 players. The conditions are:1. Each player has a single scoop.2. At least one player chose each flavor.3. Chocolate was chosen by more players than vanilla, which was chosen by more players than strawberry.

. . . . . .

$S = 1$:V must be at least 2 and C must be at least 3.Possible values for V are 2 and 3.If $V = 2$, then $C = 6$. The number of assignments is $\frac{9!}{1!2!6!} = 252$.If $V = 3$, then $C = 4$. The number of assignments is $\frac{9!}{1!3!4!} = 1260$.$S = 2$: V must be at least 3 and C must be at least 4.The only possible value for V is 3, making $C = 4$. The number of assignments is $\frac{9!}{2!3!4!} = 126$.Summing the valid assignments:Total assignments = 252 + 1260 + 126 = 1512.Finally, the remainder when 1512 is divided by 1000 is:$\boxed{512}$. (Incorrect)

[Total Tokens: 1765]

Response from Ours

<think >
Okay, so here's this problem about the baseball team and ice cream.So let's break this down. Let me denote the number of chocolate, vanilla, and strawberry lovers as c, v, s respectively. So, $c+v+s = 9$, and $c > v > s \geq 1$. First triple $(6, 2, 1)$:Number of assignments: 252.Second triple: $(5, 3, 1)$.504.Third triple: $(4, 3, 2)$.Number of assignments:1260.Thus total N=252+504+1260=2016.Wait, let's compute:$252 + 504 = 756$; $756 + 1260 = 2016$.
2016 divided by 1000 is 2 with remainder 16.
*Final Answer** $\boxed{16}$
</think >
We denote the number of chocolate, vanilla, and strawberry lovers as c, v, and s respectively. The conditions are $c + v + s = 9$, $c > v > s \geq 1$, and each flavor is chosen by at least one player.
We consider possible values for s:
1. $s = 1$: Possible triples: (6, 2, 1) and (5, 3, 1). For (6, 2, 1): $\frac{9!}{6!2!1!} = 252$ For (5, 3, 1): $\frac{9!}{5!3!1!} = 504$
2.$s = 2$:Possible triple: (4, 3, 2). $\frac{9!}{4!3!2!} = 1260$
Summing these assignments:$252 + 504 + 1260 = 2016$
Finding the remainder:$2016 \mod 1000 = 16$ Thus, the final answer is $\boxed{16}$. (Correct)

[Total Tokens: 1960]

Figure 15: Case Analysis of Responses from DeepSeek-R1-Distill-Qwen1.5B,LASER-L2048 and Ours in AIME25

Problem

Find the number of integer values of k in the closed interval $[-500, 500]$ for which the equation $\log(kx) = 2\log(x+2)$ has exactly one real solution.

Response from DeepSeek-R1-Distill-Qwen1.5B

<think >
Alright, so I have this problem here: I need to find the number of integer values of k in the closed interval $[-500, 500]$ for which the equation $\log(kx) = 2\log(x+2)$ has exactly one real solution. Hmm, okay, let's try to break this down step by step.
First, I know that logarithmic equations can sometimes be tricky, but maybe I can simplify this equation using logarithmic properties. Let me recall that $2\log(x+2)$ can be rewritten as $\log((x+2)^2)$. So, substituting that back into the original equation, we get:$\log(kx) = \log((x+2)^2)$.Since logarithms are one-to-one functions, if $\log(A) = \log(B)$, then $A = B$ provided that both A and B are positive. So, I can set the arguments equal to each other:$kx = (x+2)^2$
Now, expanding the right-hand side, we have:$kx = x^2 + 4x + 4$.Let me bring all terms to one side to form a quadratic equation:$x^2 + 4x + 4 - kx = 0$.Simplify the terms with x:$x^2 + (4-k)x + 4 = 0$.So, the equation simplifies to:$x^2 + (4-k)x + 4 = 0$
Now, this is a quadratic equation in terms of x. For the original logarithmic equation to have a real solution, the argument of the logarithm must be positive. That means:1. $kx > 0$, 2. $x + 2 > 0 \rightarrow x > -2$. So, x must satisfy both $kx > 0$ and $x > -2$. Let's consider these conditions. First, let's think about the quadratic equation $x^2 + (4-k)x + 4 = 0$. The number of real solutions depends on the discriminant. The discriminant D is given by:$D = (4-k)^2 - 16$. Simplify that:$D = (16 - 8k + k^2) - 16 = k^2 - 8k$.So, $D = k(k-8)$.
For the quadratic to have exactly one real solution, the discriminant must be zero. So, set $D = 0$:$k(k-8) = 0$ This gives two solutions: $k = 0$ and $k = 8$.But wait, we also have to make sure that the resulting equation from the logarithm is valid, i.e., the arguments must be positive. So, let's analyze each case separately.
Case 1: If $k = 0$, the original equation becomes $\log(0 \cdot x) = 2\log(x+2)$. But $\log(0 \cdot x)$ is undefined because you can't have $0 \cdot x = 0$ for x not equal to zero, and $\log(0)$ is undefined. So, $k = 0$ is not a valid solution. Therefore, we can disregard $k = 0$.
Case 2: If $k = 8$, the original equation becomes $\log(8x) = 2\log(x+2)$. Let's see if this equation has a solution.First, $8x > 0$ implies $x > 0$.Second, $x + 2 > 0$ implies $x > -2$.So, combining both, $x > 0$.Now, let's solve the equation:$\log(8x) = 2\log(x+2)$.Using logarithm properties, $2\log(x+2) = \log((x+2)^2)$.

. . . . . .

Thus, the number of k is:$\boxed{14}$. (Incorrect)

[Total Tokens: 19171]

Response from LASER-L2048

<think >
Okay, I need to find the number of integer values of k in the closed interval $[-500, 500]$ such that the equation $\log(kx) = 2\log(x+2)$has exactly one real solution. Hmm, let's break this down step by step.
First, let's recall the properties of logarithms. The equation is log(kx) = 2 log(x + 2). I can rewrite this equation using logarithm properties. Remember that 2 log(a) is log(a²), so:$\log(kx) = log((x+2)^2)$
Since log is a one-to-one function, if log(A) = log(B), then A = B, provided that A and B are positive. Therefore:$kx = (x+2)^2$.
But we also need to ensure that the arguments of the logarithms are positive.

. . . . . .

The final answer is $\boxed{993}$. (Incorrect)

[Total Tokens: 4683]

Response from Ours

<think >
Alright, so I need to figure out how many integer values of k in the interval $[-500, 500]$ make the equation $\log(kx) = 2\log(x+2)$ have exactly one real solution. Hmm, okay, let's break this down step by step.
First, I remember that logarithmic equations often require considering their domains. The logarithm function is only defined for positive arguments, so I need to make sure that both sides of the equation are valid.Looking at the equation $\log(kx) = 2\log(x+2)$, the arguments inside the logs must be positive. So,1. $kx > 0$, 2. $x + 2 > 0$ which implies $x > -2$
Also, since the left side is $\log(kx)$, kx must be positive. So depending on the sign of k, x will have different conditions.
Let me rewrite the equation using logarithm properties. I know that $2\log(x+2) = \log((x+2)^2)$. So the equation becomes:$\log(kx) = \log((x+2)^2)$.If the logs are equal, then their arguments must be equal (since log is injective, meaning one-to-one). So,$kx = (x+2)^2$
Alright, now we have a quadratic equation:$(x+2)^2 = kx$. Let's expand the left side:$x^2 + 4x + 4 = kx$. Bring all terms to one side:$x^2 + (4-k)x + 4 = 0$

. . . . . .

Hence, the total number of integer values is:$500 + 1 = 501$. The final answer is:$\boxed{501}$ (Correct)

[Total Tokens: 3513]

Figure 16: Case Analysis of Responses from DeepSeek-R1-Distill-Qwen1.5B,LASER-L2048 and Ours in MATH500