The Art of Efficient Reasoning: Data, Reward, and Optimization

Anonymous ACL submission

Abstract

Large Language Models (LLMs) consistently benefit from scaled Chain-of-Thought (CoT) reasoning, but also suffer from heavy computational overhead. To address this issue, efficient reasoning aims to incentivize short yet accurate thinking trajectories, typically through reward shaping with Reinforcement Learning (RL). In this paper, we systematically investigate the mechanics of efficient reasoning for LLMs. For comprehensive evaluation, we advocate for more fine-grained metrics, including length distribution conditioned on correctness and performance across a wide spectrum of token budgets ranging from 2k to 32k. First, we reveal that the training process follows a two-stage paradigm: *length adaptation* and *reasoning refinement*. Through extensive experiments (about 0.2 million GPU hours) in a unified protocol, we deconstruct training prompts and rollouts, reward shaping, and optimization strategies. A central finding is to maintain a sufficient density of positive reward signals and avoid the short-is-correct trap. Moreover, the learned length bias generalizes across domains and difficulty levels. We distill these findings into valuable insights and practical guidelines, and validate them across the Qwen3 models ranging from 0.6B to **30B**, demonstrating the robustness and generalization.

1 Introduction

Large language models (LLMs), such as Qwen3 (Yang et al., 2025) and DeepSeek-R1 (Guo et al., 2025), have revolutionized the field of natural language processing (NLP) due to their superior reasoning capabilities. One key insight for such success is the consistently scaled Chain-of-thought (CoT) thinking during inference (Snell et al., 2024). Despite the advancement, longer CoT also introduces significant trade-offs, such as high latency for real-world deployments (Sui et al., 2025; Wu et al., 2025a).

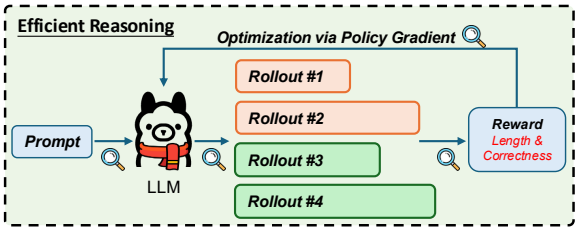

Figure 1: General pipeline for efficient reasoning via RL. The key is to promote short and accurate thinking trajectories via reward design. In this paper, we provide systematic insights (🔍) considering data, reward, and optimization.

To address this issue, one mainstream method is to incentivize efficient reasoning via Reinforcement Learning (RL) with reward shaping (Ma et al., 2024; Kimi et al., 2025; Liu et al., 2025). As shown in Figure 1, the core idea is to incentivize efficient reasoning by allocating the rewards based on rollout length and correctness. For instance, one important principle is that shorter correct CoTs should receive higher rewards than longer correct CoTs (Yeo et al., 2025). However, previous methods are almost exclusively focused on reward design (Hou et al., 2025; Liu et al., 2025), while overlooking the broader training recipe, including data composition and optimization strategy.

In this paper, we propose to *systematically* investigate the mechanics of efficient reasoning in a unified experimental protocol. Our analysis reveals that the training dynamics follow a two-stage paradigm, i.e., 1) *length adaptation*, where the model rapidly adapts to token constraints; and 2) *reasoning refinement*, where it optimizes performance within the length scope. For comprehensive observations, we advocate for more fine-grained metrics. Specifically, we propose to compare the length distribution conditioned on correctness for the training prompts. Meanwhile, for the downstream benchmarks, we argue to record the performance across a wide spectrum of token budgets

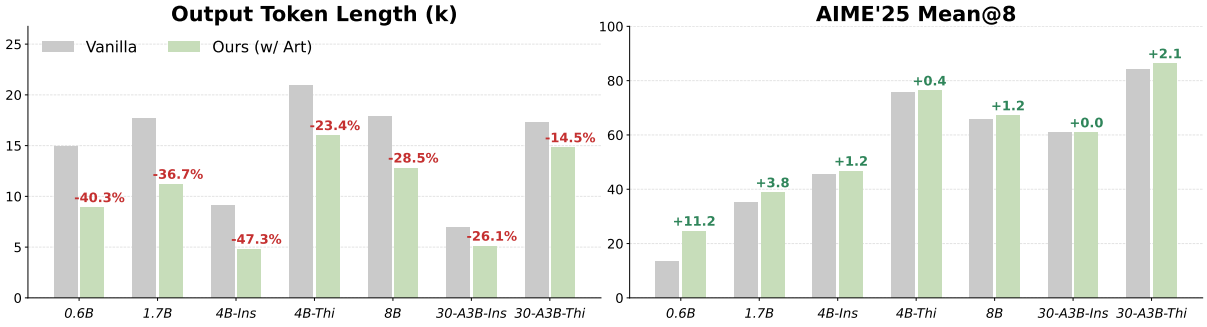

Figure 2: Performance comparison on AIME'25 for Qwen3 models ranging from 0.6B to 30B.

ranging from 2k to 32k. The effectiveness of different strategies is budget-dependent, exhibiting distinct or even contradictory behaviors. Importantly, we find that the **learned length bias can be generalized across domains**, i.e., training on mathematical prompts works well on the code task.

Through extensive ablation studies, we further deconstruct the impact of data difficulty, rollout number, reward, and optimization strategies. Notably, we find that **training on relatively easier prompts** provides a denser positive reward signal, which is essential for stable reasoning distillation. More rollouts contribute to better performance, but also bring heavier training costs. For the reward assignment, we compare the strategy to assign a negative reward or mask corresponding rollouts. Moreover, we further explore the off-policy strategy with different staleness to speed up the reasoning refinement stage. We distill all the findings into valuable insights and practical guidelines, and evaluate them on the Qwen3 series. Figure 2 demonstrates that our strategy reduces the CoT length by 15% to 47.3% while preserving or improving Mean@8 and achieving favorable trade-offs overall. In summary, our contributions are as follows:

- We identify and characterize the two-stage paradigm for efficient reasoning, i.e., length adaptation followed by reasoning refinement.
- We introduce fine-grained metrics, providing a more comprehensive understanding of training dynamics.
- We provide a systematic exploration of the training recipe, offering practical insights into data, reward, and optimization that significantly improve the efficiency of CoT models.

2 Preliminary

2.1 Experimental Setup

Prompts. RL methods have been demonstrated as an effective way for reasoning. Given an input prompt x from a dataset $\mathcal{D}$, the LLM policy π_θ generates a set of N reasoning trajectories (rollouts) $\{y_1, y_2, \ldots, y_N\}$. The objective is to update π_θ using policy gradients derived from reward signals upon rollouts. Data quality is critical for LLM reasoning (Guo et al., 2025). Therefore, we employ the popular DeepScaleR as training prompts[1].

Reward Engineering. In standard outcome-supervised RL, the reward function focuses solely on correctness. Let $\mathbb{I}(\cdot)$ denote the indicator function which returns 1 if the condition holds and 0 otherwise. The vanilla reward is defined as:

$$R_{\text{vanilla}}(x, y_i) = \mathbb{I}(y_i \text{ is correct}). \tag{1}$$

To enforce efficient reasoning, we apply reward shaping to incentivize concise yet accurate rollouts. In this work, we select the truncation strategy as a baseline with

$$R_T(x, y_i) = \mathbb{I}(y_i \text{ is correct}) \cdot \mathbb{I}(L(y_i) \leq L_T), \tag{2}$$

where $L(y_i)$ denotes the token length of the i-th rollout, and L_T represents the target length. We first compare this approach against various baselines such as Kimi-1.5 (Kimi et al., 2025) and Laser (Liu et al., 2025). Further details are provided in Appendix A.

Evaluation Protocol. To capture the nuances of efficient reasoning, we advocate for more fine-grained metrics as follows:

[1] Dataset link

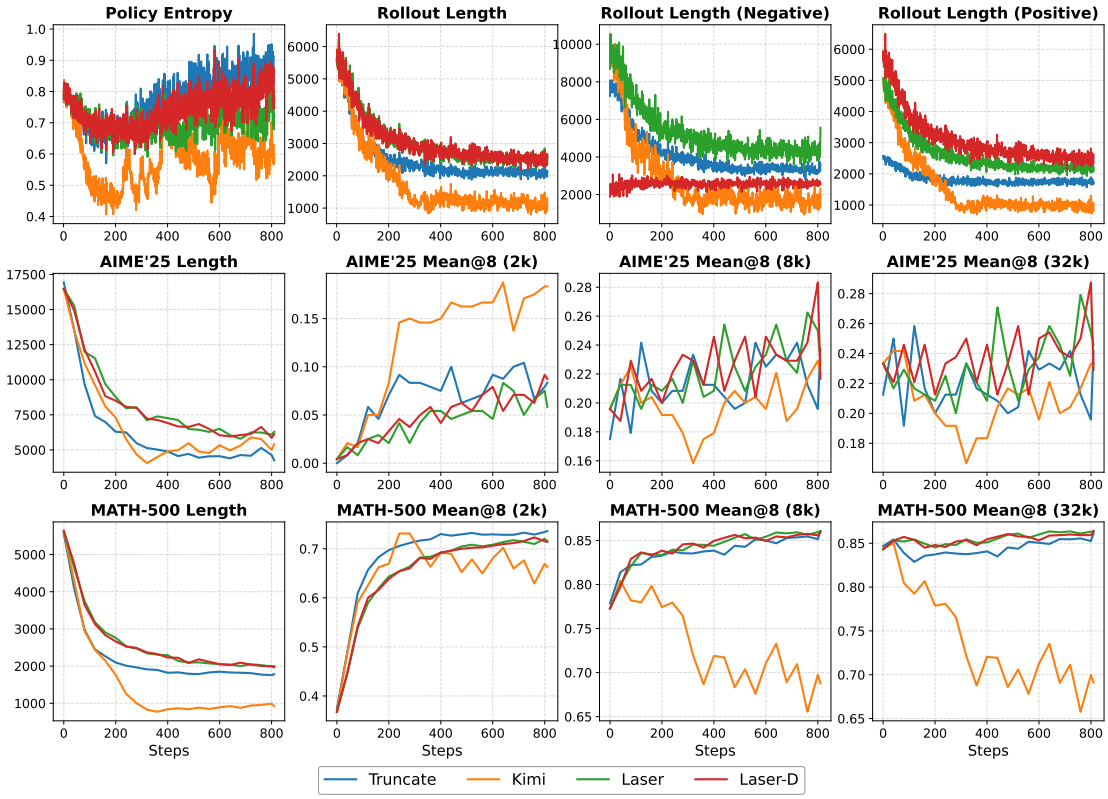

Figure 3: Training dynamics of various reward shaping methods on `DeepSeek-R1-Distill-Qwen-1.5B`. All of them follow the two-stage paradigm, including constraint satisfaction and optimization within a feasible region. The behaviors are distinct when evaluated under different token budgets.

- **Training dynamics:** We monitor the length distribution conditioned on correctness to visualize how the model trades off verbosity for precision.
- **Budget-aware benchmarking:** For downstream tasks, we report performance across a wide spectrum of inference token budgets ($\mathcal{B} \in \{2\text{k}, 4\text{k}, 8\text{k}, 16\text{k}, 32\text{k}\}$).

We report Pass@8 and Mean@8 metrics across standard mathematical reasoning benchmarks: AIME'25 (AIME, 2025), AMC (AMC, 2025), MATH-500 (Hendrycks et al., 2021), Minerva Math (Lewkowycz et al., 2022), and Olympiad Bench (He et al., 2024). Additionally, we assess code capabilities via LiveCodeBench (LCB) (Jain et al., 2024).

Training Implementation. We use `DeepSeek-R1-Distill-Qwen-1.5B` as the backbone model. RL training is performed using Group Relative Policy Optimization (GRPO) (Shao et al., 2024). The learning rate is set to 1×10^{-6} with a clip-high ratio of 0.28 following Yu et al. (2025). During the rollout phase, we use a batch size of 128 and sample $N = 8$ trajectories per prompt with a maximum length of 16k ($L_R = 16k$). The target length L_T is 4k.

2.2 Two-stage Paradigm

As illustrated in Figure 3, the training dynamics exhibit a two-stage paradigm:

Stage I: Length Adaptation. The optimization of constraint satisfaction dominates this initial phase. Driven by the length penalty, the model rapidly adjusts its output distribution to avoid zero-reward truncation. As shown in the *Rollout Length* curves, the average token consumption undergoes a precipitous decline (e.g., from ~6k to ~2k), exhibiting an exponential decay pattern. Simultaneously, the *Policy Entropy* decreases significantly, indicating that the model is converging towards a subspace of shorter and valid trajectories.

Stage II: Reasoning Refinement. Once the rollout length stabilizes within the target budget, the training enters a stationary phase regarding length, shifting focus to performance optimization. In this stage, the length curves plateau, demonstrating that the model has successfully adapted to the hard constraints on output length. Crucially, the perfor-

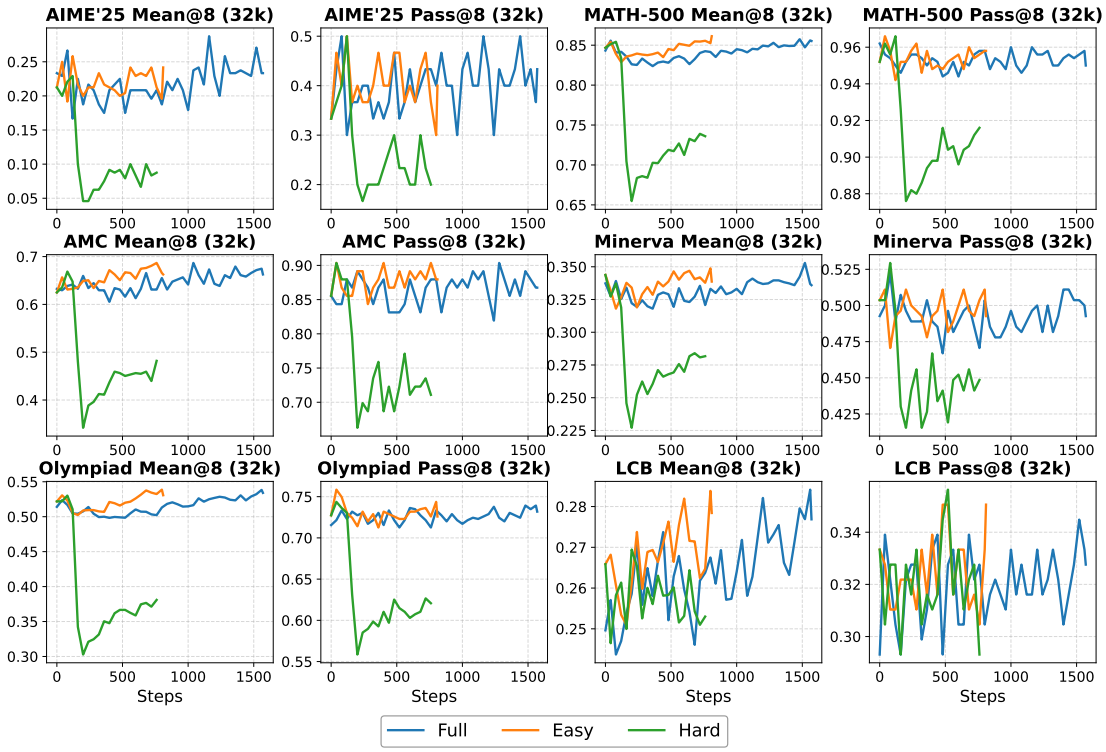

Figure 4: Performance training on all prompts and easy/hard counterparts (rollout $L_R = 16k$, target $L_T = 4k$).

mance metrics (e.g., Mean@8) continue to evolve or recover. At the same time, the *Policy Entropy* increases with such exploration. This indicates that the model is learning to increase the information density of each token to improve accuracy without violating the length budget.

2.3 Distinct Behaviors across Token Budgets

Figure 3 also reveals that model behaviors are highly sensitive to the token constraints, exhibiting distinct and even contradictory trends.

Under a strict budget (2k), performance is dominated by length adaptation. Aggressive penalties (e.g., Kimi) excel here by forcing the model to fit the narrow context window. However, under a generous budget (32k), such a strategy suffers from a reasoning collapse issue. In contrast, Laser exhibits a U-shaped trajectory at 32k, which initially drops due to compression, but subsequently recovers through reasoning refinement. This decoupling phenomenon highlights a critical trade-off: over-optimizing for efficiency can severely harm the upper-bound reasoning capability, necessitating the multi-budget evaluation protocol (Qi et al., 2025).

2.4 Generalization across Domains

As detailed in Figures 9 and 10, the dynamic on code task (i.e., LiveCodeBench) is similar to other mathematical tasks, though the training prompts are math-related only. For example, on the LCB benchmark, the Kimi strategy performs best under a 2k budget but is comparable at larger budgets, which is similar to other benchmarks. The similarity proves that learned length bias can be generalized across domains. We further provide extensive results on private benchmarks in Appendix F.

3 Experiments and Guidelines

3.1 Data: Prompt and Rollout

Impact of Prompt Difficulty. The difficulty of training prompts plays a pivotal role in determining the density of positive reward signals. To investigate this, we split DeepScaleR prompts based on the pass rate over $N = 8$ rollouts into DeepScaleR-Easy (pass rate > 0.5) and DeepScaleR-Hard (pass rate ≤ 0.5).

As illustrated in Figure 4, the training dynamics exhibit stark differences. Training exclusively on hard prompts results in catastrophic failure. The policy entropy spikes drastically, and the rollout length collapses prematurely. Consequently, downstream performance metrics (e.g., Mean@8 on AMC and Olympiad Bench) degrade significantly. This suggests that when the model struggles to generate correct answers, the RL signal becomes **dominated by the length penalty on incorrect rollouts, leading to reasoning collapse**. We attribute such

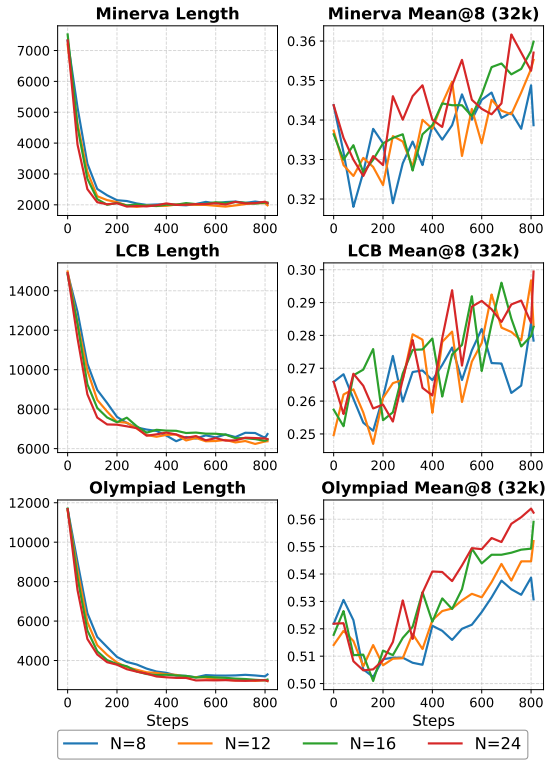

Figure 5: Performance with various rollouts N using DeepScaleR-Easy.

an issue to the **sparsity of positive samples**, which leads to the overfitting on short output length. Conversely, training on the easier counterpart yields the most stable trajectory. The policy entropy remains low and stable, indicating consistent positive reinforcement. The rollout length adapts smoothly to the target budget. Crucially, despite training on easy prompts, the performance on relatively tough tasks (e.g., AIME'25) is comparable to (or even slightly exceeding) training on the full dataset.

Please refer to Appendix B for results and analysis under more settings. Appendix D further indicates cases for reasoning collapse from baselines and paradigm shift from our strategy.

Impact of Rollout Number N. Given that the density of valid reward signals is crucial, we further investigate the impact of the rollout number N. We set $N \in \{8, 12, 16, 24\}$ with a rollout limit $L_R = 16k$ and a target length $L_T = 4k$ on the DeepScaleR-Easy prompts.

As shown in Figure 5, increasing N yields observable benefits that significantly speed up the *Length Adaptation* phase. With a larger N, it is easier to discover short and correct trajectories, promoting the length curve (e.g., $N = 24$, red line) to decay faster than the baseline ($N = 8$, blue line), though all settings converge to a similar length floor. Meanwhile, larger N leads to a more robust *Reasoning Refinement* stage. In mathematical benchmarks, the model recovers its reasoning capabilities faster and achieves a higher asymptotic Mean@8.

However, this advantage is task-dependent. On the LiveCodeBench (LCB) coding task, the performance gap between $N = 8$ and $N = 24$ is marginal, suggesting that the complexity of code generation may require distinct exploration strategies beyond simply scaling N. Please refer to Appendix C for results under more settings and discussion on *wall-time comparison*.

> **Insights towards Training Data**: *The key is to ensure sufficient and effective rewards. Training on easier prompts allows LLMs to focus on length reduction without compromising performance. Larger rollout N would be better if computational resources allow.*

3.2 Reward on Negative Rollouts

Strategy	Correct		Incorrect	
	Short	Long	Short	Long
Vanilla	1	0	0	0
-I	1	0	-	-
-L&C	1	-	0	0
-L&C-S&I	1	-	-	0
-L&C-L&I	1	-	0	-

Table 1: Reward for different strategies on negative rollouts. − denotes masking out.

The art of RL is to utilize the negative signals. In the standard truncation strategy (denoted as Vanilla), both *incorrect* responses and *overlong correct* responses are treated as negative samples ($R = 0$). An alternative way is to mask these negative samples rather than setting the reward to 0. To investigate this, we conduct a fine-grained ablation study by masking specific subsets of negative rollouts, as detailed in Table 1.

As illustrated in Figure 6, the training dynamics reveal that different masking strategies exhibit distinct behaviors, categorized into three distinct failure modes:

1) The trap that short is correct (-I, -L&C-S&I). When the correctness is coupled with length, the model will be misled by such incorrect causal relationships. For masking all incorrect rollouts (-I), we only penalize overlong and correct rollouts.

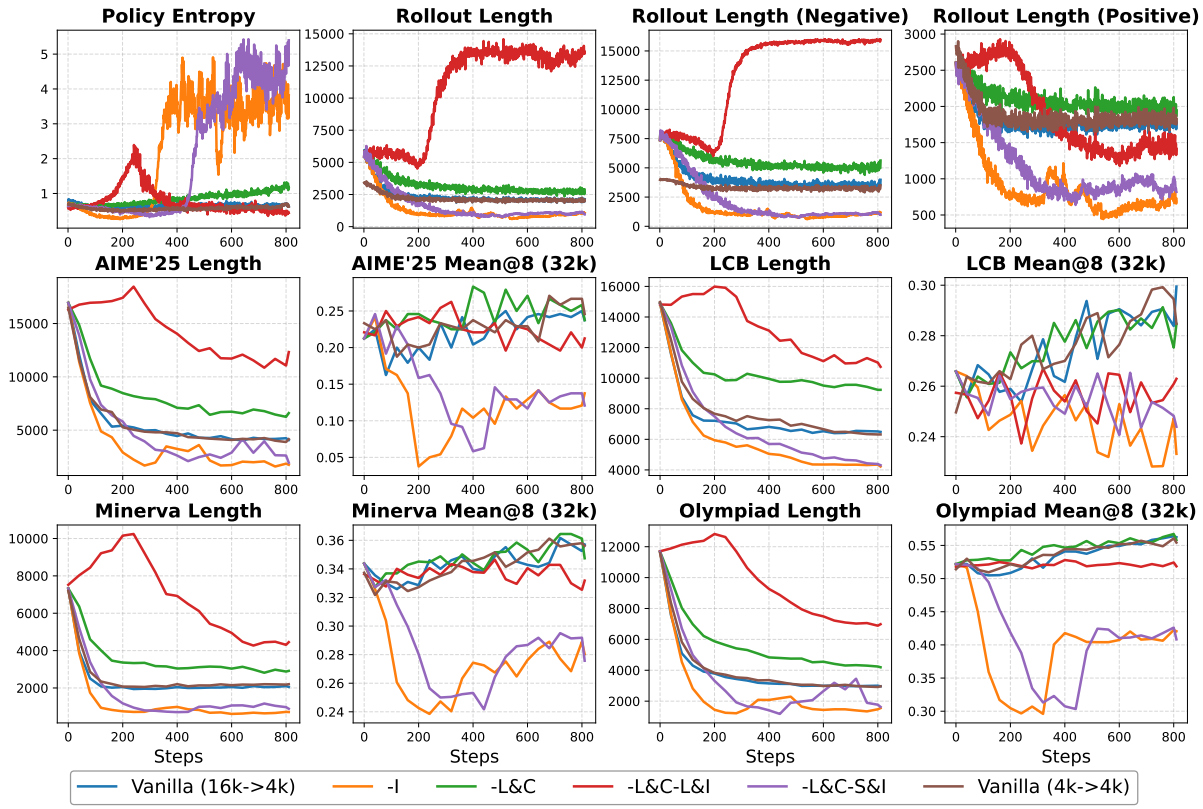

Figure 6: Performance for various reward strategies on negative rollouts (rollout $L_R = 16k$, target $L_T = 4k$, $N = 24$). We also visualize $L_R = 4k$, $L_T = 4k$ for comparison.

The training signal only contains a) short and correct rollouts with positive reward and b) overlong and correct rollouts with negative reward. In this way, the model would hack this bias to generate short output. As shown by the orange line, the policy entropy explodes after 400 steps and rollout length collapses precipitously. The model abandons reasoning entirely to satisfy the length constraint. Meanwhile, for -L&C-S&I, the dynamics are similar. The training signal only contains a) short and correct rollouts with positive reward and b) overlong and incorrect rollouts with negative reward.

2) Short rollouts only. A particularly interesting phenomenon occurs in the strategy **-L&C-L&I** (red line), where we mask all overlong trajectories. It means that the LLMs are optimized with short correct and short incorrect rollouts exclusively. In this setting, the overlong outputs are masked, without either positive or negative rewards. After 200 steps (reasoning refinement stage), the LLMs hack this and begin to generate overlong outputs. Interestingly, the models *do not* collapse.

3) Do not penalize overlong but correct rollouts (-L&C). The **-L&C** strategy (green line) masks overlong but correct rollouts instead of penalizing them. On the downstream benchmarks, the LLMs will generate longer outputs and also outperform the vanilla baseline, indicating a trade-off for length control and performance.

Length of Negative Samples. Additionally, we compare these complex shaping strategies against a simple baseline sampling at target length (i.e., $L_R = L_T = 4k$, brown line). Compared to Vanilla ($L_R = 16k, L_T = 4k$), the positive samples are roughly the same (correct rollouts that are less than $4k$) while the negative samples are much shorter (4k vs. 6k). We can observe that it achieves the optimal Pareto frontier. We attribute such success to avoiding the harmful explicit length bias trap that short is correct. Typically, the positive rollouts are shorter than the negative ones, which **implicitly encourages** the model to be short yet accurate.

> **Insights towards Reward on Negative Rollouts**: *Sampling at target length ($L_R = L_T$) achieves a better trade-off via avoiding the length trap.*

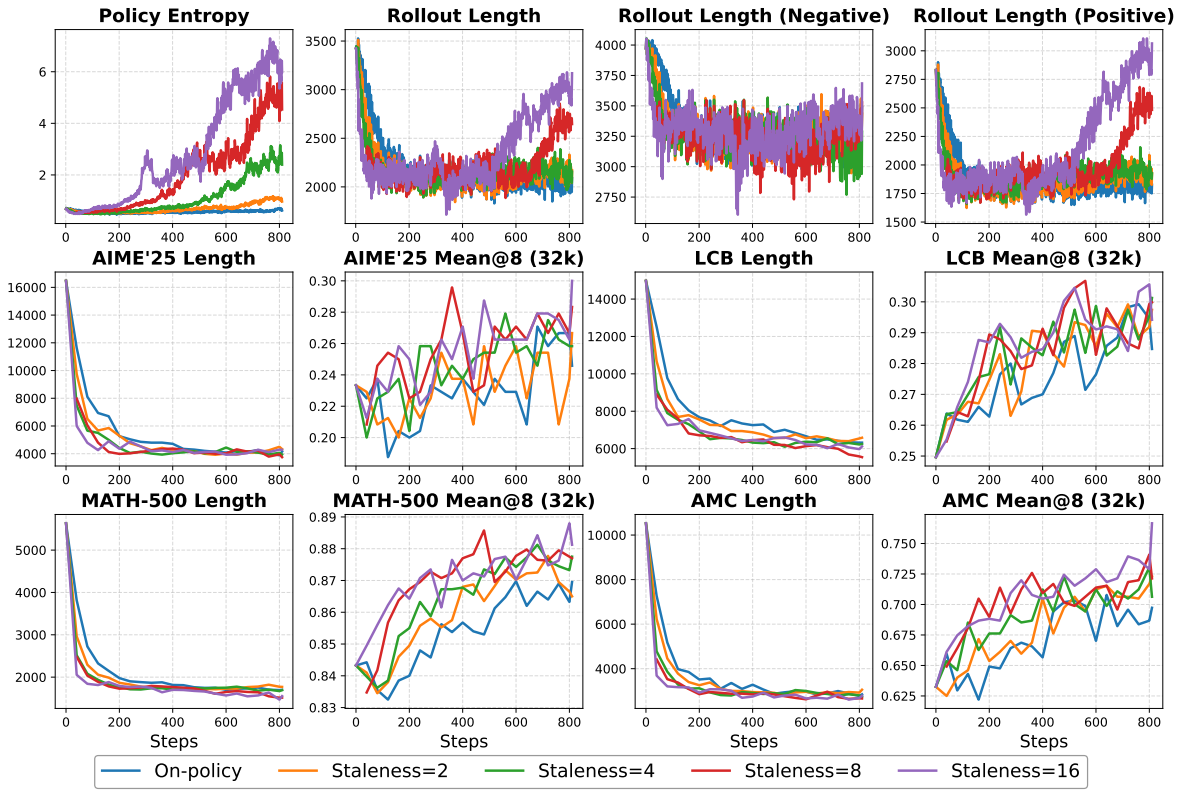

Figure 7: Performance for off-policy strategy with various staleness (i.e., 2,4,8,16).

3.3 Off-policy Optimization

Off-policy optimization via introducing staleness can significantly accelerate the training process but also introduces instability (Zheng et al., 2025; Huang et al., 2025). Meanwhile, we can observe that the performance keeps growing at the reasoning refinement stage. Hence, we conduct experiments with varying staleness degrees $S \in \{2, 4, 8, 16\}$ using the robust setting derived from our previous guidelines ($L_R = L_T = 4k, N = 24$).

As illustrated in Figure 7, the results reveal a trade-off between performance and stability. Firstly, the *Length Adaptation* stage is significantly shortened. As staleness increases, the rollout length decays more rapidly, allowing the model to satisfy the token budget earlier. After that, the model enters the *Reasoning Refinement* stage and learns quickly. As a result, performance on downstream tasks shows that high-staleness models (e.g., $S = 16$) performs better than on-policy baseline.

Contrary to findings that excessive off-policy updates lead to catastrophic model collapse, our experiments maintain high performance. We attribute this robustness to our optimized setup (easy prompts and large N), which ensures sufficient and effective reward signals. However, larger staleness also introduces *potential instability risks*: i) *Entropy explosion:* As shown in the *Policy Entropy* subplot, high staleness (e.g., 16) causes a dramatic surge in entropy after 400 steps. For the on-policy baseline, the entropy remains almost the same as initialization. ii) *Length rebound:* The positive rollout length on training prompts under high staleness begins to drift upwards again, indicating that the policy model is struggling to balance efficiency and effectiveness.

Insights towards Off-policy Optimization: *Appropriate staleness can accelerate convergence without harming accuracy (before 800 steps), but also introduces latent instability, such as rising entropy. We suggest the on-policy strategy, especially for larger and more fragile LLMs.*

4 Extensive Analysis

4.1 Evaluated on More LLMs

To verify the universality of our derived guidelines, we extend our evaluation to the Qwen3 family, ranging from 0.6B to 30B. Please refer to Appendix E for detailed hyperparameters. After

training, we evaluate these compressed LLMs on AIME'25 (AIME, 2025).

As shown in Table 2, the experimental results demonstrate the robustness of our strategy across different model scales. Specifically, on the `Qwen3-0.6B`, our method significantly boosts the Mean@8 from 13.33 to 24.58, while reducing the average response length from 14.9k to 8.9k. Meanwhile, our approach improves the Pass@8 to 36.67 compared to the vanilla score of 26.67. A similar trend is observed in larger models. For instance, on `Qwen3-4B-Instruct-2507`, our method maintains a superior Pass@8 of 70.00 and increases the Mean@8 from 45.42 to 46.67, while drastically compressing the length from 9.1k to roughly 4.8k.

4.2 Case Study

To qualitatively validate the impact of our strategy, we compare the reasoning trajectories of the vanilla and optimized models. Please refer to Appendix D for detailed examples and analysis. In short, our method not only encourages short outputs but also incentivizes the model to reorganize its CoTs into a more streamlined and expert-like format.

Method	Mean@8↑	Pass@8↑	Length↓
`Qwen3-0.6B`			
Vanilla	13.33	26.67	14.9k
Ours	24.58	36.67	8.9k
`Qwen3-1.7B`			
Vanilla	35.00	60.00	17.7k
Ours	38.75	60.00	11.2k
`Qwen3-4B-Instruct-2507`			
Vanilla	45.42	66.67	9.1k
Ours	46.67	70.00	4.8k
`Qwen3-4B-Thinking-2507`			
Vanilla	75.83	90.00	20.9k
Ours	76.25	86.67	16.0k
`Qwen3-8B`			
Vanilla	65.83	86.67	17.9k
Ours	67.08	83.33	12.8k
`Qwen3-30B-A3B-Instruct-2507`			
Vanilla	60.83	83.33	6.9k
Ours	60.83	76.67	5.1k
`Qwen3-30B-A3B-Thinking-2507`			
Vanilla	84.17	96.67	17.3k
Ours	86.25	96.67	14.8k

Table 2: Performance on AIME'25 for Qwen3 models.

5 Related Work

5.1 Efficient Reasoning

Efficient reasoning methods aim to mitigate the overthinking phenomenon (Wu et al., 2025b; Sui et al., 2025) and reduce the prohibitive inference costs associated with long-form CoT (Wu et al., 2025a; Cui et al., 2024). One prominent approach trains long CoTs to be short using SFT (Xia et al., 2025; Ma et al., 2025) or RL (Hou et al., 2025; Shen et al., 2025; Liu et al., 2025; Liang et al., 2025). Parallel research directions explore architectural innovations, such as reasoning within latent spaces (Hao et al., 2024; Su et al., 2025) or more efficient decoding (Sun et al., 2024; Xu et al., 2025). We refer the readers to Feng et al. (2025) for more details. Unlike works that propose novel architectures, we focus on the mechanics of *RL-based* efficiency optimization.

5.2 Reward Shaping Methods

The philosophy for reward shaping is to incentivize short yet accurate rollouts via allocating training rewards (Weng, 2025). The first principle is to promote shorter responses and penalize longer responses among correct ones (Kimi et al., 2025; Hou et al., 2025; Aggarwal and Welleck, 2025). Meanwhile, we can also penalize longer for incorrect answers (Kimi et al., 2025). Contrary to that, Liu et al. (2025); Yeo et al. (2025) argue to promote longer incorrect rollouts to encourage exploration. Despite these advancements, existing studies often evaluate reward functions in isolation. In this paper, we select the simplest truncation strategy and conduct extensive experiments in a unified protocol.

6 Conclusion

In this work, we first reveal that the training dynamics of efficient reasoning follow a two-stage paradigm. Meanwhile, we advocate for more fine-grained metrics. Based on these, we further deconstruct the impact of data difficulty, rollout number, reward on negative rollouts, and optimization strategies. All findings are distilled into valuable insights and practical guidelines. A key finding is to train on relatively easier prompts for sufficient and effective rewards. The learned length bias for reasoning can be generalized across domains and difficulty levels. Evaluation of more LLMs up to 30B further demonstrates the robustness and generalization.

Limitation and Future Work

In this work, we systematically investigate the mechanics of efficient reasoning. However, there are several limitations for future research.

Domain diversity. In this paper, we train on DeepScaleR, containing mathematical reasoning prompts, and validate on math and coding benchmarks. One future work is to evaluate on more domains, such as creative writing. Another interesting topic is whether training on more diverse prompts contributes to better performance.

Adaptive length. In this paper, we employ fixed rollout and target length (i.e., L_R and L_T). One intuitive idea is to set the length adaptively based on the prompts and current LLMs. For target length L_T, we conduct experiments on a toy setting detailed in Appendix G. For adaptive rollout length L_R, we can pre-compute and set the rollout engine correspondingly.

Evaluate on larger models. In this paper, we conduct extensive experiments (about 0.2 million GPU hours) in a unified protocol on the `DeepSeek-R1-Distill-Qwen-1.5B`. Moreover, we extend our evaluation to the Qwen3 family, such as `Qwen3-30B-A3B-Instruct-2507`. However, due to the limited GPUs, we do not evaluate on extremely large LLMs such as `Qwen3-235B-A22B-Instruct-2507`. We leave it for future work.

More fine-grained supervision. In this paper, we focus on efficient reasoning via reward shaping-based RL training. The principle is to incentivize short yet accurate thinking trajectories. We do not apply fine-grained refinement to the CoTs. Meanwhile, we humans simplify the previous experience by making and employing useful tools, such as a notebook and a calculator. Therefore, how to create useful tools and reuse them during reasoning remains a valuable topic.

References

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

AIME. 2025. Aime. In *American invitational mathematics examination*.

AMC. 2025. Amc. In *American mathematics competitions*.

Anonymous. 2025. Deepscaler: Effective RL scaling of reasoning models via iterative context lengthening. In *Submitted to The Fourteenth International Conference on Learning Representations*. Under review.

Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, and 1 others. 2024. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 958–979.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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

Guanhua Huang, Tingqiang Xu, Mingze Wang, Qi Yi, Xue Gong, Siheng Li, Ruibin Xiong, Kejiao Li, Yuhao Jiang, and Bo Zhou. 2025. Low-probability tokens sustain exploration in reinforcement learning with verifiable reward. *arXiv preprint arXiv:2510.03222*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Kimi, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

Tian Liang, Wenxiang Jiao, Zhiwei He, Jiahao Xu, Haitao Mi, and Dong Yu. 2025. Deepcompress: A dual reward strategy for dynamically exploring and compressing reasoning chains. *arXiv preprint arXiv:2510.27419*.

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

Haozhe Ma, Zhengding Luo, Thanh Vinh Vo, Kuankuan Sima, and Tze-Yun Leong. 2024. Highly efficient self-adaptive reward shaping for reinforcement learning. *arXiv preprint arXiv:2408.03029*.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. Cot-valve: Length-compressible chain-of-thought tuning. *arXiv preprint arXiv:2502.09601*.

Penghui Qi, Zi-Yan Liu, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Optimizing anytime reasoning via budget relative policy optimization. *ArXiv*, abs/2505.13438.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *arXiv preprint arXiv:2503.04472*.

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

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. 2025. Token assorted: Mixing latent and text tokens for improved language model reasoning. *arXiv preprint arXiv:2502.03275*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. 2024. Fast best-of-n decoding via speculative rejection. *Advances in Neural Information Processing Systems*, 37:32630–32652.

Lilian Weng. 2025. Why we think. *lilianweng.github.io*.

Taiqiang Wu, Runming Yang, Tao Liu, Jiahao Wang, and Ngai Wong. 2025a. Revisiting model interpolation for efficient reasoning. *arXiv preprint arXiv:2510.10977*.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025b. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Jun Liu, Qika Lin, and Zhiyong Wu. 2025. ϕ-decoding: Adaptive foresight sampling for balanced inference-time exploration and exploitation. *arXiv preprint arXiv:2503.13288*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. 2025. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? *arXiv preprint arXiv:2510.01161*.

Appendix

A Detailed Results on Reward Engineering

For Kimi-1.5 (Kimi et al., 2025), they first define a normalized length term:

$$\tilde{L}(y_i) = \frac{L(y_i) - L_{\min}}{L_{\max} - L_{\min}}, \tag{3}$$

where $L_{\max}$ and $L_{\min}$ are the maximum and minimum length within rollout group. Then the reward function can be rewritten as the sum of the correct reward and the incorrect penalty:

$$\begin{aligned} R_{\text{Kimi}}(x, y_i) = &\ \mathbb{I}(y_i \text{ is correct}) \cdot \left(1 + \alpha(0.5 - \tilde{L}(y_i))\right) \\ &+ \mathbb{I}(y_i \text{ is incorrect}) \cdot \min\left(0, \alpha(0.5 - \tilde{L}(y_i))\right). \end{aligned} \tag{4}$$

For Laser (Liu et al., 2025), it can be viewed as a base reward for correctness plus a bonus if the length condition is met:

$$R_{\text{Laser}}(x, y_i) = \mathbb{I}(y_i \text{ is correct}) \cdot \left(1 + \alpha \cdot \mathbb{I}(L(y_i) < L_T)\right). \tag{5}$$

They also propose a variant to encourage exploration by adding an extra term to incentivize longer trajectories. Specifically, it adds an exploration bonus for incorrect rollouts:

$$\begin{aligned} R_{\text{Laser-D}}(x, y_i) = &\ \mathbb{I}(y_i \text{ is correct}) \cdot \left(1 + \alpha \cdot \mathbb{I}(L(y_i) < L_T)\right) \\ &+ \mathbb{I}(y_i \text{ is incorrect}) \cdot \left(\alpha \cdot \mathbb{I}(L(y_i) \geq L_T)\right). \end{aligned} \tag{6}$$

In this paper, we set α to 0.4. Figures 9 and 10 present the training dynamics on additional benchmarks, including AIME'25, MATH-500, AMC (Figure 9), and Minerva Math, Olympiad Bench, LiveCodeBench (Figure 10).

We can find that the training process strictly follows a two-stage paradigm, i.e., the *length adaptation* phase followed by the *reasoning refinement* phase. The rollout length (top row) consistently decays before stabilizing. Meanwhile, the performance shows distinct behaviors across various token budgets. For instance, a more aggressive strategy Kimi often exhibits gains at stricter budgets (2k), but suffers from stagnation or collapse at generous budgets (32k), whereas the truncation baseline maintains a more balanced recovery. Without results under larger token budgets, the observations and conclusions will be biased. These extensive results reinforce the necessity of our proposed fine-grained evaluation protocol.

B Detailed Results for Data Selection

In the main paper, we demonstrate that training on easy prompts allows LLMs to focus on length reduction without compromising performance, which is attributed to sufficient and effective rewards. Here, we provide details on all benchmarks and more settings.

As illustrated in Figures 11 and 12, the results consistently validate our hypothesis regarding reward density under various settings. Across almost all benchmarks, the model trained exclusively on DeepScaleR-Hard (green line) exhibits significant instability. For instance, on AIME'25 and Olympiad Bench, the performance often fluctuates violently or collapses entirely after the initial adaptation phase.

Meanwhile, the model trained on DeepScaleR-Easy (orange line) consistently matches or rivals the performance of the model trained on the full dataset (blue line). Notably, on the relatively tough AIME'25 and LCB, the performance is also comparable and even better, though we train on high-pass prompts. In short, the learned length bias for reasoning can be **generalized across domain and difficulty**.

C Detailed Results for More Rollouts

Detailed Results. Figures 13 and 14 indicate the results on more rollouts under two different settings ($L_R = 16k, L_T = 4k$ and $L_R = 4k, L_T = 4k$). We observe three key phenomena:

1) Consistent benefits across settings. Regardless of the rollout length (L_R), the advantage of scaling N remains consistent. Whether allowing for long-context exploration ($16k$) or strictly constraining the search space ($4k$), increasing N yields a positive effect on the convergence and asymptotic performance. It indicates that increasing the density of the reward signal is a robust strategy for efficient reasoning.

2) Task-dependent sensitivity. The performance improvement varies significantly across datasets. On difficult benchmarks such as AIME'25 and LiveCodeBench (LCB), the performance gap between $N = 8$ and $N = 32$ is relatively narrow. This suggests that for problems requiring complex multi-step reasoning or code synthesis, simply increasing the number of rollouts is insufficient if the base model lacks the fundamental capability to solve

the problem. Conversely, on relatively easier or intermediate tasks like AMC and MATH-500, larger N leads to substantial performance separation on Mean@8.

3) Mean@8 vs. Pass@8. While Mean@8 shows significant improvement with larger N, the Pass@8 metric often remains stagnant or improves only marginally. Pass@k measures the ability to generate at least one correct solution under k trials. It implies that scaling N does not necessarily enable the model to solve new problems that were previously unsolvable. Meanwhile, Mean@k measures the expected correctness. The rise in Mean@8 indicates that scaling N effectively reduces the variance of the policy.

In conclusion, increasing N for training prompts primarily acts as a stabilizer, reducing the variance of the policy on solvable problems rather than expanding the upper-bound. However, it also brings challenges, i.e., increased computing overheads.

Wall-time Comparison. Rollout stage is the primary compute bottleneck in RL training, and the compute cost is almost linear related to the rollout size. As illustrated in Figures 13 and 14, when compute cost is almost matched (e.g., $N = 8$ at Step 600 vs. $N = 24$ at Step 200), $N = 8$ may initially appear more step-efficient. However, our research focuses on the asymptotic performance upper bound. Models trained with $N = 24$ achieve higher final accuracy compared to $N = 24$ even after extensive steps. We prioritize this performance ceiling for complex reasoning tasks.

D Case Study

Easy and hard prompts. Tables 4 and 5 indicate two examples with corresponding responses when trained on DeepScaleR-Easy and DeepScaleR-Hard. On DeepScaleR-Hard, the model suffers from the reasoning collapse issue, leading to over-short outputs. One behavior is to omit the double-checking process. We attribute such an issue to the **sparsity of positive samples**, which leads to the overfitting on length. Meanwhile, the **group normalization** also amplifies such an effect (larger advantage value). For instance, the advantage of positive samples in group $\{1, 0, 0, 0\}$ is larger than group $\{1, 1, 1, 0\}$.

Vanilla and our strategy. Tables 6 and 7 indicate two examples with corresponding responses from vanilla and our strategy. For our method, we set $L_R = L_T = 4k, N = 24$ with staleness of 16. As observed in the vanilla response, the model exhibits typical conversational redundancy: it frequently uses filler phrases (e.g., "Hmm," "Let me think"), re-states the problem premise unnecessarily, and performs verbose arithmetic decomposition. In contrast, the model trained with our efficient reasoning strategy undergoes a fundamental stylistic shift. It eliminates conversational fluff and directly adopts a dense and mathematically formal structure. Reasoning transitions from a hesitant narrative to a precise symbolic derivation (e.g., integrating formulas such as $V = \frac{1}{3}Bh$ directly into the calculation flow). It demonstrates that our method not only encourages short outputs but also actively incentivizes the model to reorganize its CoTs into a more streamlined and expert-like format.

E Experimental Setting for Qwen3 Models

Based on our insights, we strictly align the rollout limit with the target budget, including $L_R = L_T = 8k$ for `Qwen3-0.6B`, $L_R = L_T = 10k$ for `Qwen3-1.7B`, and $L_R = L_T = 16k$ for `Qwen3-4B-Thinking-2507` and `Qwen3-30B-A3B-Thinking-2507`. In particular, we set $L_R = 8k, L_T = 6k$ for `Qwen3-4B-Instruct-2507` and $L_R = 10k, L_T = 8k$ for `Qwen3-30B-A3B-Instruct-2507`, aiming to incentivize shorter outputs better since original rollouts are already short and concise. Crucially, we maintain a high reward density by sampling $N = 24$ trajectories for each prompt from DeepScaleR-Easy. Also, we *do not use the off-policy strategy* due to the potential instability. The training batch sizes are 128, 128, 32, 32, 32, 16, and 16, respectively. In experiments, checkpoints are selected by monitoring convergence on reward curves, similarly to early stopping.

F Results on Private Benchmark

We further evaluate our compressed `Qwen3-30B` models on a private out-of-distribution benchmark covering 10 diverse domains.

First, although the model is trained on mathematical prompts, the learned compression patterns generalize well across domains. For example, our `Qwen3-30B-A3B-Thinking-2507` model reduces CoT length by 15% on AIME'25 (shown in Table 2) and achieves a comparable 15% reduction on the private benchmark.

Benchmark	Qwen3-30B-A3B-Instruct-2507			Qwen3-30B-A3B-Thinking-2507		
	Mean@4↑	**Pass@4↑**	**Length↓**	**Mean@4↑**	**Pass@4↑**	**Length↓**
Domain #1	27.5 / 28.0	40.9 / 40.9	7145 / 4881	34.2 / 34.1	50.0 / 49.8	11536 / 10237
Domain #2	35.1 / 36.0	51.7 / 50.7	7258 / 5182	42.2 / 41.6	68.8 / 68.4	11744 / 10847
Domain #3	20.5 / 21.0	31.9 / 29.8	3985 / 3102	27.0 / 26.2	34.0 / 34.7	8826 / 8278
Domain #4	15.4 / 15.8	24.0 / 25.0	7695 / 3176	16.4 / 14.7	28.9 / 26.2	4726 / 4292
Domain #5	40.6 / 40.6	53.3 / 55.4	6487 / 4500	57.6 / 58.6	78.2 / 78.2	2598 / 2351
Domain #6	6.9 / 6.8	10.8 / 10.8	12322 / 9481	18.3 / 16.7	33.8 / 30.4	21346 / 18115
Domain #7	25.4 / 24.4	43.8 / 40.3	897 / 799	25.5 / 25.2	21.2 / 20.8	4962 / 4312
Domain #8	41.5 / 40.6	52.5 / 51.2	1429 / 1309	29.6 / 29.5	45.6 / 44.8	13912 / 11726
Domain #9	49.4 / 48.2	66.6 / 67.5	1100 / 991	37.2 / 37.5	51.5 / 52.5	3302 / 2943
Domain #10	14.1 / 13.0	27.4 / 24.9	6835 / 5069	47.3 / 45.6	63.3 / 60.0	2492 / 2311
Average	25.2 / 24.9	37.6 / 36.9	5622 / 3735	30.7 / 30.0	44.6 / 43.1	9285 / 8077

Table 3: Performance on private out-of-distribution benchmarks covering various domains. Each cell reports *Vanilla / Ours*. Our proposed model thinks shorter while maintaining comparable performance.

Second, the compressed model consistently shortens the reasoning chain while preserving performance. As shown in Table 3, our method substantially reduces token usage with minimal impact on both Mean@4 and Pass@4 across all domains.

G Adaptive Target Length L_T

For adaptive target length, we set L_T as the 90th quantile of correct rollouts. As shown in Figure 8, the strategy of adaptive target leads to short answers and thus better performance on 4k budget. We attribute it to a more negative signal since 10% of correct answers are always negative samples. However, on 32k budget, the performance is comparable or even worse. Also, there are lots of spikes on the KL divergence curve, indicating rapidly changing gradients and potential instability. We leave it for future work to explore more settings.

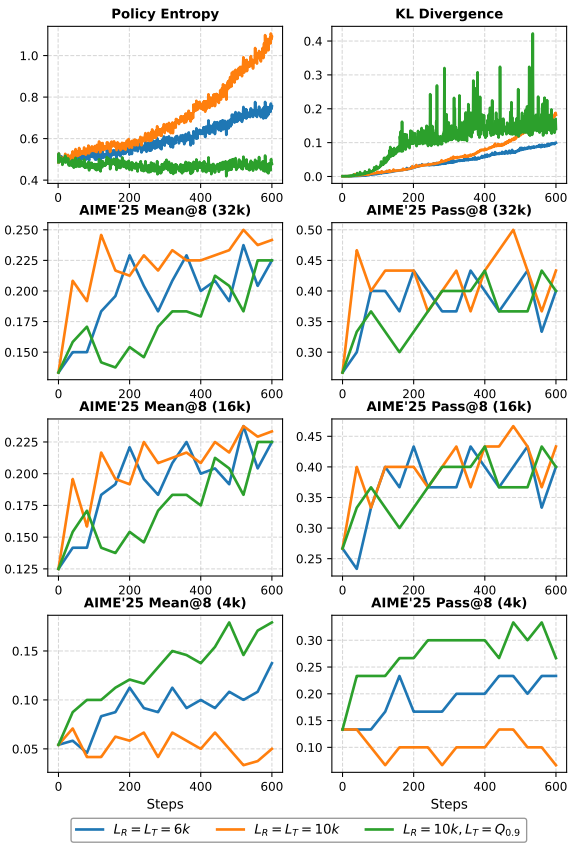

Figure 8: Performance on `Qwen3-0.6B` when adaptively setting target length L_T as 90th qualities.

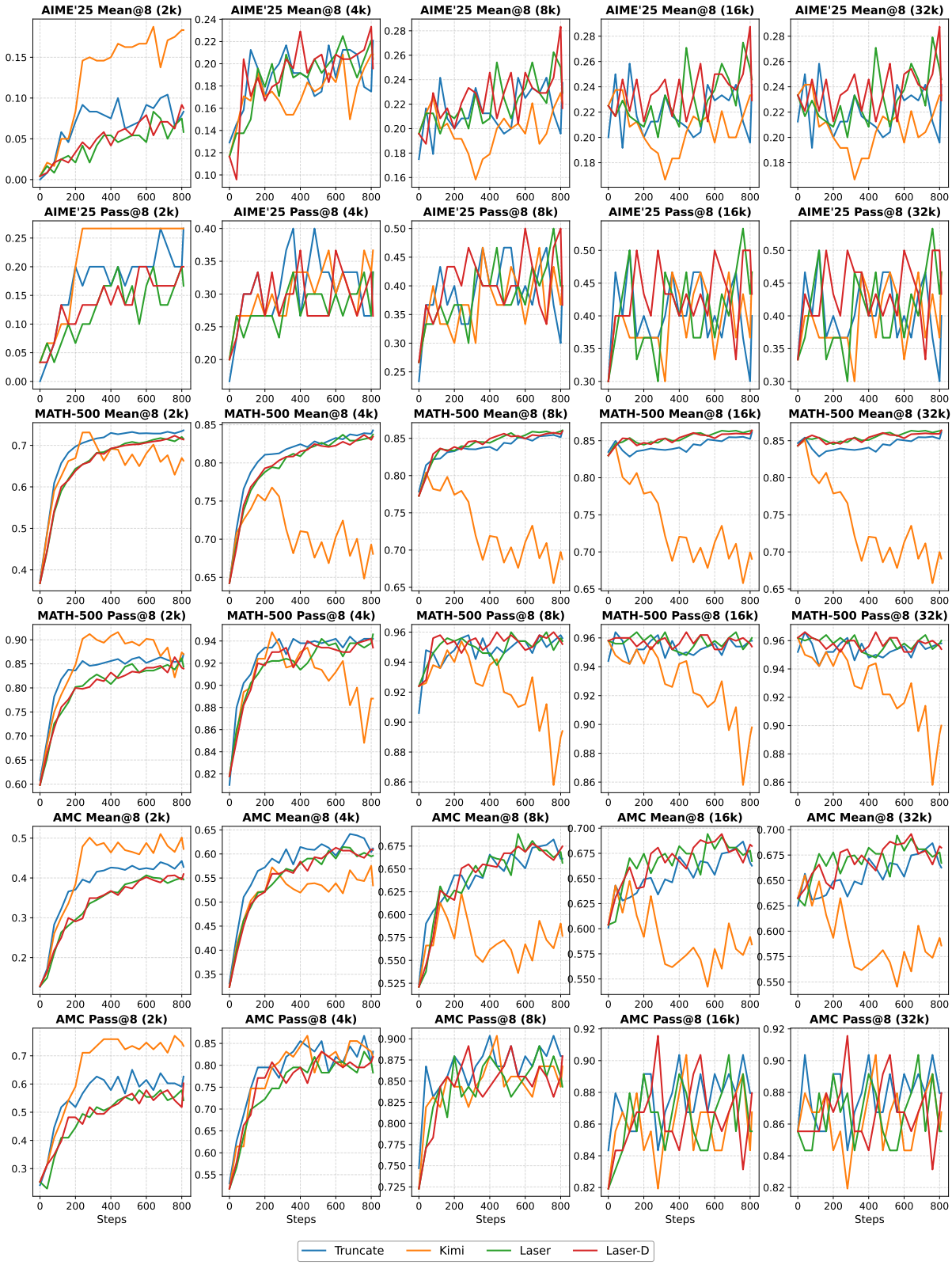

Figure 9: Training dynamics of various reward shaping methods on AIME'25, MATH-500, and AMC.

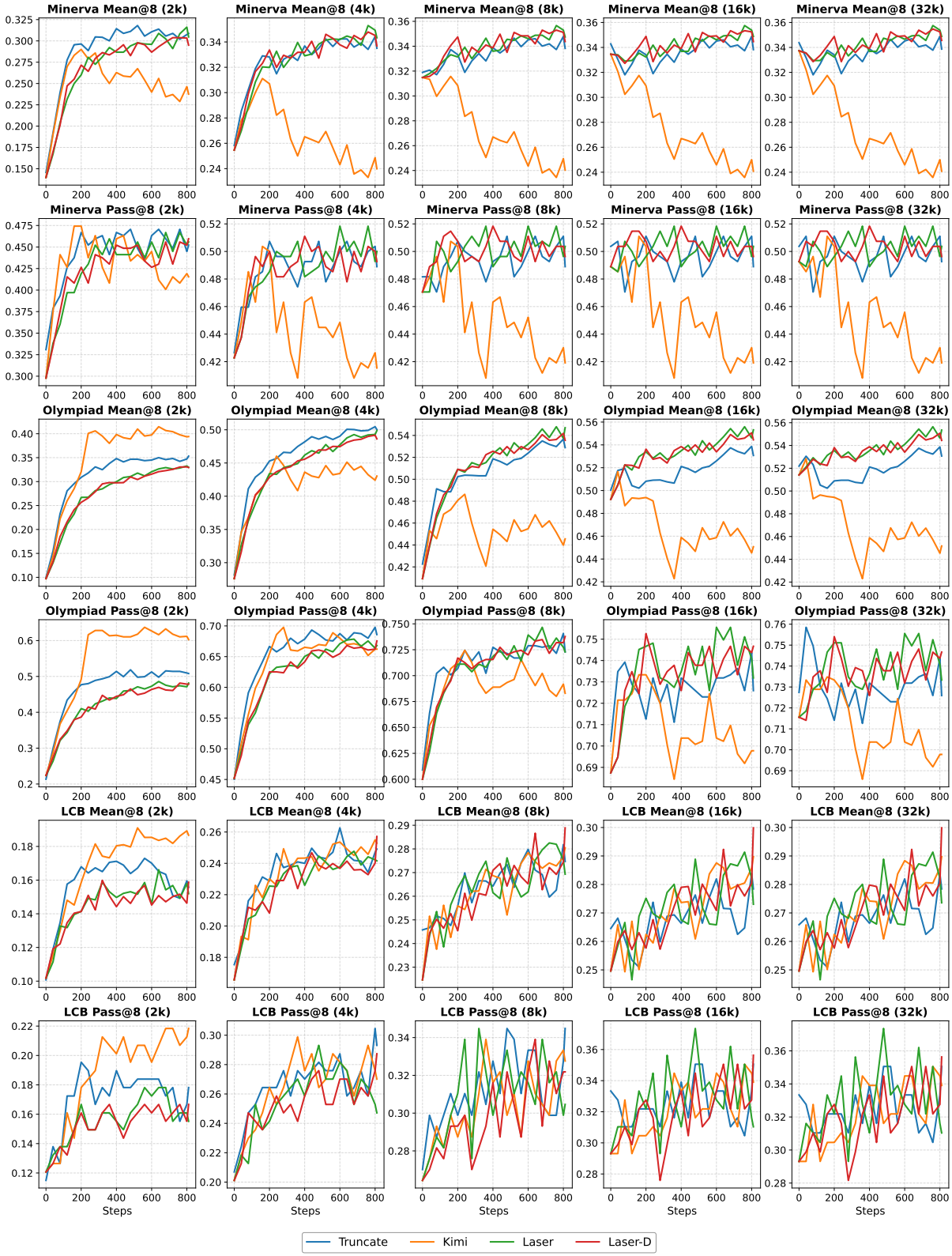

Figure 10: Training dynamics of various reward shaping methods on Minerva Math, Olympiad Bench, and LiveCodeBench.

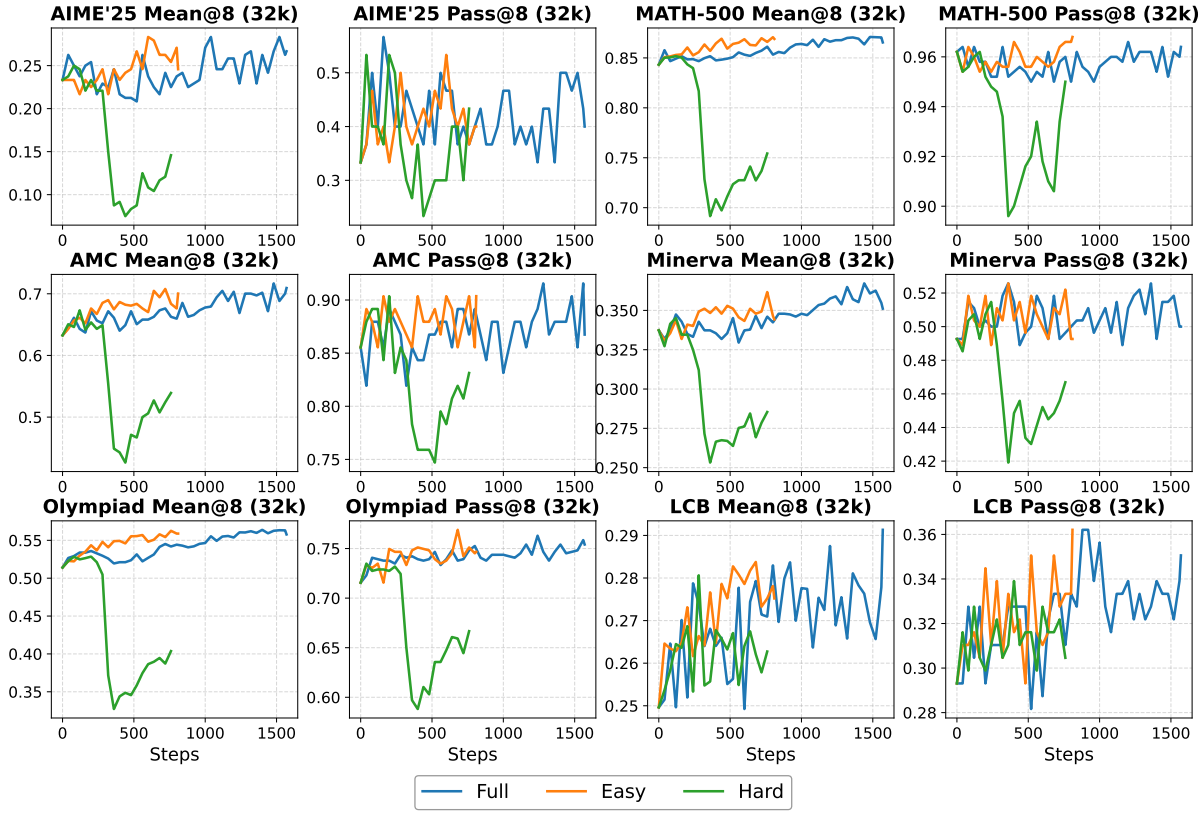

Figure 11: Performance training on various training prompts (rollout $L_R = 16k$, target $L_T = 8k$).

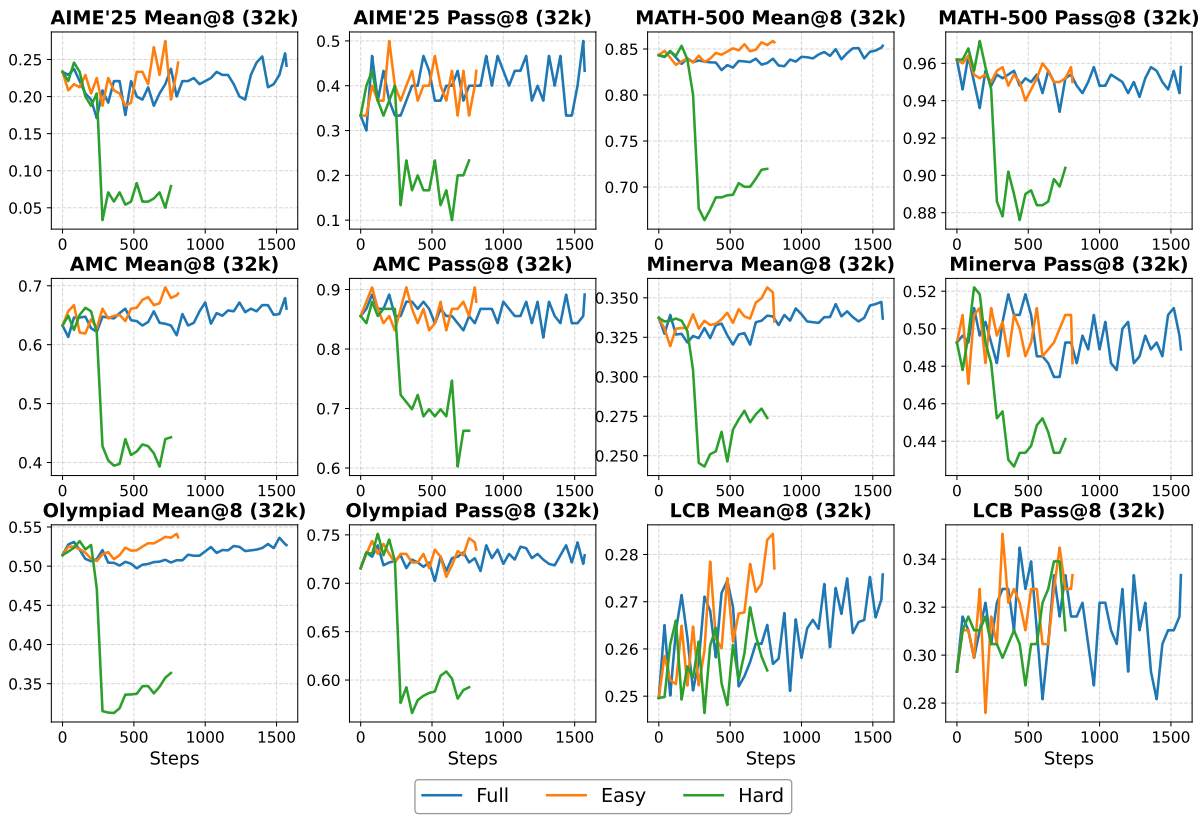

Figure 12: Performance training on various training prompts (rollout $L_R = 4k$, target $L_T = 4k$).

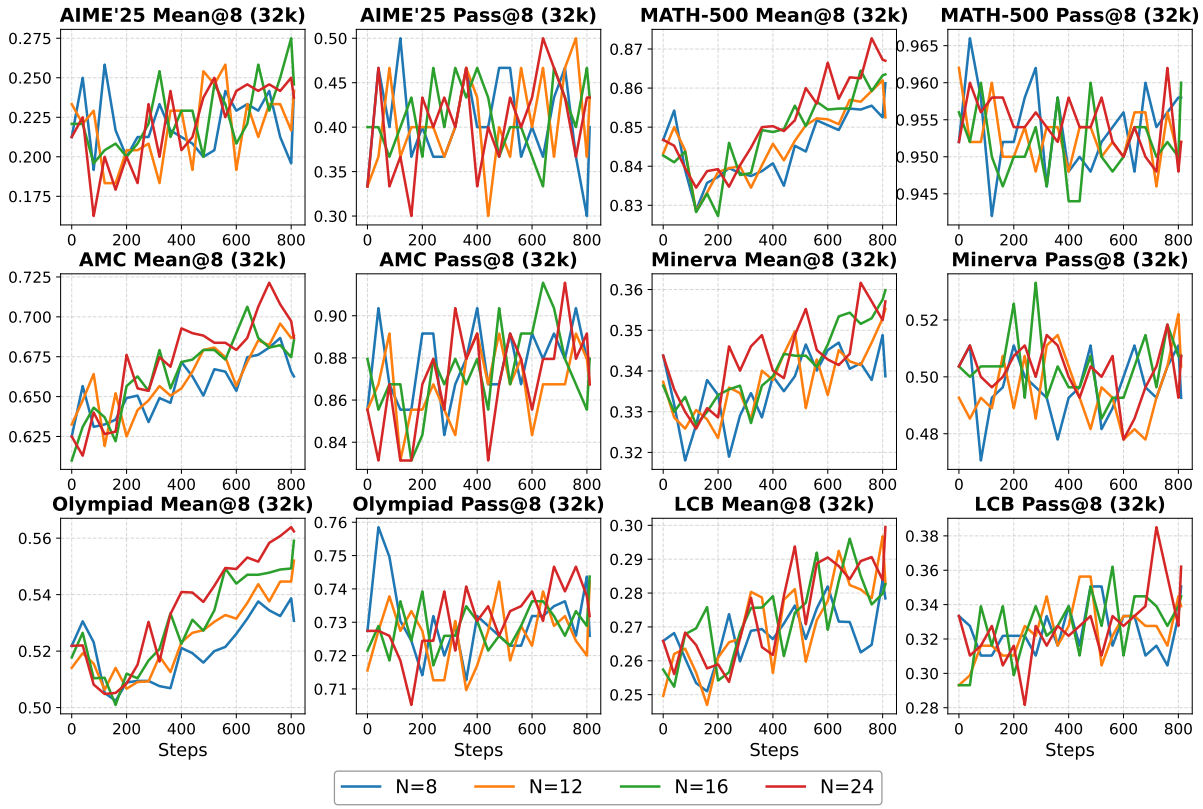

Figure 13: Performance training on various rollouts N (L_R = $16k$, L_T = $4k$).

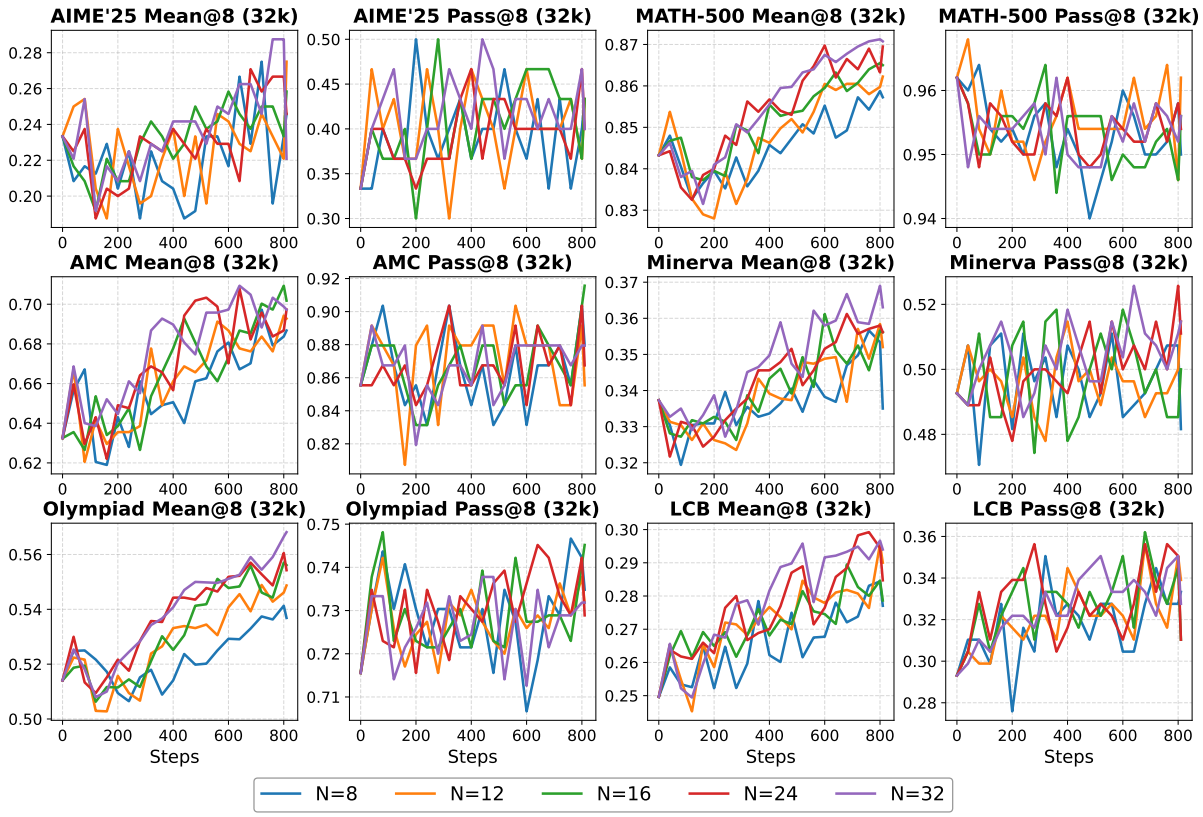

Figure 14: Performance training on various rollouts N (L_R = $4k$, L_T = $4k$).

Question	Let $\mathcal{R}$ be the region in the complex plane consisting of all complex numbers z that can be written as the sum of complex numbers z_1 and z_2, where z_1 lies on the segment with endpoints 3 and $4i$, and z_2 has magnitude at most 1. What integer is closest to the area of $\mathcal{R}$? Let's think step by step and output the final answer within \boxed{}
Hard	To determine the area of the region $\mathcal{R}$, I first identify the segments involved. The first segment runs from 3 to $4i$, and the second segment encompasses all points within a unit circle centered at the origin. By translating these segments into the complex plane and applying appropriate transformations, I calculate the combined area as the sum of their respective areas. To find the area of the region $\mathcal{R}$ in the complex plane: Understanding the Components: Segment from 3 to 4i: This is a line segment in the complex plane with endpoints at 3 (which is (3, 0)) and 4i (which is (0, 4)). The length of this segment is $\sqrt{(3)^2 + (4)^2} = 5$. Region defined by z_2: This is a disk (or circle) centered at the origin with radius 1. Calculating the Area: The region $\mathcal{R}$ is the set of all points $z = z_1 + z_2$, where z_1 lies on the segment from 3 to 4i and z_2 lies within the unit disk. This effectively translates the unit disk to every point along the segment z_1 and sums the areas. Since the unit disk has an area of $\pi \times (1)^2 = \pi$, the total area of $\mathcal{R}$ is π. Finding the Closest Integer: The area is approximately 3.1416, so the closest integer is 3. Final Answer: $\boxed{3}$ (Incorrect)
Easy	Okay, so I need to find the area of the region R in the complex plane. R consists of all complex numbers z that can be written as $z = z1 + z2$, where $z1$ lies on the segment with endpoints 3 and 4i, and z2 has magnitude at most 1. Then, I have to find the integer closest to this area. First, let me visualize this. The region R is the Minkowski sum of the segment from 3 to 4i and the disk of radius 1. So, if I think about it, R should be a rectangle with semicircles on both ends. But let me break it down step by step. The segment from 3 to 4i: In the complex plane, 3 is the point (3,0) and 4i is the point (0,4). So this is a straight line connecting (3,0) to (0,4). The length of this segment can be calculated, but maybe we don't need it right away. z1 lies on this segment. So z1 can be parameterized. Let me define z1 as a point on the line segment between 3 and 4i. Let's parameterize z1 with a parameter t from 0 to 1. So, $z1(t) = (3 - 3t) + (4i - 4it) = (3(1 - t), 4(1 - t))$. Wait, actually, in complex terms, if I let t vary from 0 to 1, when t=0, z1 = 3, and when t=1, z1 = 4i. Alternatively, z1 can be expressed as $3 + (4i - 3)i$, where i is a parameter from 0 to 1. Hmm, maybe that's more complicated. Alternatively, the equation of the line from (3,0) to (0,4). The slope is $(4 - 0)/(0 - 3) = -4/3$. So equation is $y = (-4/3)x + b$. When $x = 3, y = 0 : 0 = (-4/3)(3) + b$ => $0 = -4 + b$ => $b = 4$. So equation is $y = (-4/3)x + 4$. So, for any point z1 on the segment, z1 can be written as x + yi where x and y satisfy $y = (-4/3)x + 4$, and x ranges from 0 to 3? Wait, no, the segment goes from (3,0) to (0,4). So when x decreases from 3 to 0, y increases from 0 to 4. So, for any z1 on the segment, we can write $z1 = (x, y)$ where $y = (-4/3)x + 4, x \in [0, 3]$. Alternatively, parametric equations: Let's define t as a parameter from 0 to 1. $z1(t) = (3 - 3t, 0 + 4t)$ because at t=0, we get (3,0), and at t=1, we get (0,4). So in complex terms, $z1 = (3 - 3t) + (4t)i$. Then, z2 is a complex number with $\lvert z2 \rvert \leq 1$, which is a disk of radius 1 centered at the origin. Since R is the Minkowski sum of the segment and the disk, it is like sliding the disk along every point of the segment. So geometrically, it's a rectangle with length equal to the length of the segment and width 2 (diameter of the disk). But with semicircular ends because the disk is only of radius 1. Wait, actually, no. The Minkowski sum of a line segment and a disk is called an "offset" of the disk by the segment. So it's a rectangle with length equal to the segment length and width equal to twice the radius (since each end contributes a semicircle). But in this case, the radius is 1, so each end is a semicircle. Wait, but the segment is from (3,0) to (0,4), so the Minkowski sum would be the set of all points within distance 1 from the segment. So it's like a rectangle with length equal to the segment length and width 2, but with semicircular ends of radius 1. Therefore, the area would be the area of the rectangle plus the area of the two semicircles. Area of rectangle: length * width. The length is the distance between 3 and 4i, which is $\sqrt{(3 + 4)} = 5$. The width is 2 (since radius is 1, diameter is 2). So area of rectangle is $5 * 2 = 10$. Area of two semicircles: Each semicircle has area $(1/2)\pi(1)$, so two of them is π. Therefore, total area is $10 + \pi \approx 10 + 3.1416 \approx 13.1416$. So the area is approximately 13.1416, so the closest integer is 13. Wait, but let me verify again. The Minkowski sum of a line segment and a disk is indeed a rectangle with length equal to the segment's length and width equal to 2r, and with two semicircular ends. So that's correct. The length of the segment is 5, as calculated before. So area is $5 \times 2 + \pi \times (1)^2 = 10 + \pi$. So yes, the area is $10 + \pi \approx 13.1416$. Therefore, the integer closest to the area is 13. Final Answer $\boxed{13}$ `Double check:` The region $\mathcal{R}$ consists of all complex numbers (z) that can be written as $z = z_1 + z_2$, where z_1 lies on the segment with endpoints (3) and (4i), and z_2 has magnitude at most (1). The segment from (3) to (4i) in the complex plane is a straight line connecting the points ((3, 0)) and ((0, 4)). This line has a length of (5) (calculated using the distance formula). The region ($\mathcal{R}$) is the Minkowski sum of this segment and the disk of radius (1). This results in a shape that is a rectangle with length (5) and width (2) (since the radius of the disk is (1)), plus two semicircular ends each with radius (1). The area of the rectangle is ($5 \times 2 = 10$). The area of the two semicircular ends is ($\pi \times 1^2 = \pi$). Therefore, the total area of ($\mathcal{R}$) is $10 + \pi \approx 10 + 3.1416 \approx 13.1416$. The integer closest to the area of ($\mathcal{R}$) is $\boxed{13}$. (Correct)

Table 4: Case comparison when trained on DeepScaleR-Easy and DeepScaleR-Hard using truncation strategy.

Question	The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where B is the area of the base and h is the height. The area of the base of a cone is 30 square units, and its height is 6.5 units. What is the number of cubic units in its volume? Let's think step by step and output the final answer within \boxed{}
Hard	To find the volume of the cone, I'll use the formula V = (1/3) * B * h. The area of the base, B, is 30 square units, and the height, h, is 6.5 units. First, I'll calculate one-third of the base area: (1/3) * 30 = 10. Next, I'll multiply this result by the height: 10 * 6.5 = 65. Therefore, the volume of the cone is 65 cubic units. To find the volume of the cone, we can use the formula: $V = \frac{1}{3} \times B \times h$ Where: - (B) is the area of the base = 30 square units - (h) is the height = 6.5 units Plugging in the values: $V = \frac{1}{3} \times 30 \times 6.5$ First, calculate one-third of 30: $\frac{1}{3} \times 30 = 10$ Then multiply by 6.5: $10 \times 6.5 = 65$ Therefore, the volume of the cone is: $\boxed{65}$ cubic units. (Correct)
Easy	Okay, so I need to find the volume of a cone. The formula given is V = (1/3)Bh, where B is the area of the base and h is the height. Hmm, let me recall what each part means. First, the area of the base, which is B. The problem says the area of the base is 30 square units. So B is 30. Then, the height h is 6.5 units. Got that. So plugging these values into the formula: V = (1/3) * B * h. So that would be (1/3) * 30 * 6.5. Let me compute that step by step. First, I can compute 30 multiplied by 6.5. Let me do that: 30 * 6.5. Hmm, 30 * 6 is 180, and 30 * 0.5 is 15, so adding those together gives 195. So 30 * 6.5 is 195. Then, take that result and multiply by (1/3). So (1/3) * 195. Dividing 195 by 3. Well, 3 goes into 195 sixty-five times because 360 is 180, and 35 is 15, so 60 + 5 = 65. So (1/3)*195 is 65. So V = 65 cubic units. Let me just double-check that. Alternatively, maybe I can compute it differently. Since V = (1/3) * B * h, with B = 30 and h = 6.5. Compute B * h first: 30 * 6.5. As above, that's 195. Then multiply by 1/3: 195 / 3 = 65. Yep, same result. So the volume is 65 cubic units. I think that's correct. Final Answer The volume of the cone is $\boxed{65}$ cubic units. `Double check:` The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where B is the area of the base and h is the height. The area of the base B is 30 square units, and the height h is 6.5 units. First, we compute the product of (B) and (h): $30 \times 6.5 = 195$ Next, we multiply this result by $\frac{1}{3}$: $\frac{1}{3} \times 195 = 65$ Thus, the volume of the cone is $\boxed{65}$ cubic units. (Correct)

Table 5: Case comparison when trained on DeepScaleR-Easy and DeepScaleR-Hard using truncation strategy. The model suffers from the reasoning collapse issue when trained on DeepScaleR-Hard, e.g., the double check part is skipped for shorter length.

Question	Compute $\binom{31}{28}$. Let's think step by step and output the final answer within \boxed{}
Vanilla	Okay, so I need to compute the binomial coefficient ($\binom{31}{28}$). Hmm, binomial coefficients are those numbers that come up in combinations, right? They represent the number of ways to choose a subset of items from a larger set, disregarding the order. First, I remember that the formula for a binomial coefficient is ($\binom{n}{k} = \frac{n!}{k!(n-k)!}$). So, in this case, (n = 31) and (k = 28). Let me write down the formula with these values: $\binom{31}{28} = \frac{31!}{28!(31-28)!}$ Simplifying the denominator: 31 - 28 = 3, so it becomes $\frac{31!}{28!3!}$ Hmm, but calculating factorials for such large numbers seems tedious. Maybe there's a way to simplify this without computing the entire factorials. I remember that sometimes, in binomial coefficients, you can simplify by canceling out terms in the numerator and denominator. Let me think about how to do that. Since $(31! = 31 \times 30 \times 29 \times 28!)$, I can rewrite the numerator as: $31! = 31 \times 30 \times 29 \times 28!$ So substituting back into the equation: $\binom{31}{28} = \frac{31\times30\times29\times28!}{28!\times3!}$ Oh, I see! The (28!) terms cancel out in numerator and denominator: = $\frac{31\times30\times29}{3!}$ Now, $(3! = 3\times2\times1 = 6)$, so substituting that in: = $\frac{31\times30\times29}{6}$ Alright, now I need to compute this. Let me compute the numerator first: 31 multiplied by 30 is 930, and 930 multiplied by 29. Hmm, 930 times 29. Let me do that step by step: - 930 x 20 = 18,600 - 930 x 9 = 8,370 Adding them together: 18,600 + 8,370 = 26,970 So the numerator is 26,970. The denominator is 6. So now, 26,970 divided by 6. Let's do that: 26,970 ÷ 6. Dividing step by step: - 6 goes into 26 four times (4x6=24), remainder 2. - Bring down the 9: 29. 6 goes into 29 four times (4x6=24), remainder 5. - Bring down the 7: 57. 6 goes into 57 nine times (9x6=54), remainder 3. - Bring down the 0: 30. 6 goes into 30 five times. So putting it all together: 4, 4, 9, 5. So it's 4,495. Wait, let me double-check that division because I might have messed up the steps. Alternatively, I can write 26,970 ÷ 6 as (26,000 + 970) ÷ 6. 26,000/6 = 4,333.333... 970/6≈161.666... Adding them together: 4,333.333...+161.666... = 4,495. Yep, that matches. So the result is 4,495. Therefore, ($\binom{31}{28} = 4,495$). Final Answer The value of ($\binom{31}{28}$) is \boxed4495. To compute the binomial coefficient ($\binom{31}{28}$), we use the formula: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ Substituting (n = 31) and (k = 28): $\binom{31}{28} = \frac{31!}{28!(31-28)!} = \frac{31!}{28!3!}$ We simplify the factorials by recognizing that ($31! = 31 \times 30 \times 29 \times 28!$): $\binom{31}{28} = \frac{31\times30\times29\times28!}{28!\times3!} = \frac{31\times30\times29}{3!}$ Since ($3! = 6$), we have: = $\frac{31\times30\times29}{6}$ Calculating the numerator: $31 \times 30 = 930][930 \times 29 = 26970$ Dividing by the denominator: $\frac{26970}{6} = 4495$ Thus, the value of ($\binom{31}{28}$) is ($\boxed{4495}$).
Ours	To compute the binomial coefficient ($\binom{31}{28}$), I'll start by recalling that the binomial coefficient formula is given by: $\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!}$ Here, (n = 31) and (k = 28). Alternatively, since ($\binom{n}{k} = \binom{n}{n-k}$), perhaps I can simplify the expression by choosing the smaller (k) value to make calculation easier. Calculating (31 - 28 = 3), which means: $\binom{31}{28} = \binom{31}{3}$ Now compute ($\binom{31}{3}$): Using the formula: $\binom{31}{3} = \frac{31!}{3!\cdot(31-3)!} = \frac{31!}{3!\cdot28!}$ Simplifying, since ($31! = 31 \times 30 \times 29 \times 28!$): $\frac{31\times30\times29\times28!}{3!\times28!} = \frac{31\times30\times29}{3!}$ Calculate the denominator ($3! = 6$): Therefore, $\frac{31\times30\times29}{6}$ Computing numerator: 31 × 30 = 930; 930 × 29 = ? Compute: 930 × 29: Breakdown: = 930 × (30 - 1) = 930 × 30 - 930 × 1 = 27900 - 930 = 26970 Then divide by 6: 26970 ÷ 6 = ? Calculate: 26970 ÷ 6 = 4495 Wait perhaps: 6 × 4495 = 26970, yes. Therefore, the value becomes 4495. Thus, $\boxed{4495}$ To compute ($\binom{31}{28}$), we use the property ($\binom{n}{k} = \binom{n}{n-k}$): $\binom{31}{28} = \binom{31}{3}$ Compute ($\binom{31}{3}$): Using the formula, $\binom{31}{3} = \frac{31!}{3!\cdot(31-3)!} = \frac{31\times30\times29}{3\times2\times1}$ Calculating: $31 \times 30 \times 29 = 26970$ Divide by (6) : $26970/6 = 4495$ Thus, $\boxed{4495}$

Table 6: Case I for vanilla and our trained LLM. Both answers are correct.

Question	The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where B is the area of the base and h is the height. The area of the base of a cone is 30 square units, and its height is 6.5 units. What is the number of cubic units in its volume? Let's think step by step and output the final answer within \boxed{}
Vanilla	Okay, so I need to find the volume of a cone. The formula given is V = (1/3)Bh, where B is the area of the base and h is the height. Hmm, let's break this down step by step. First, I know the area of the base is 30 square units. That's B. Then, the height is 6.5 units, so h is 6.5. The formula is straightforward, but I should make sure I plug the values correctly. So, I can write it out as V = (1/3) * B * h. Substituting the given values, that becomes V = (1/3) * 30 * 6.5. Let me compute this step by step. First, let me calculate 1/3 of 30. Hmm, 30 divided by 3 is 10. So, 1/3 * 30 = 10. Now, I need to multiply that result by 6.5. So, 10 * 6.5. Let me think, 10 * 6 is 60, and 10 * 0.5 is 5, so 60 + 5 is 65. Therefore, 10 * 6.5 is 65. So, putting it all together, V = 65 cubic units. That should be the volume of the cone. Wait, let me double-check to make sure I didn't make any calculation errors. 1/3 of 30 is indeed 10, correct. Then, 10 multiplied by 6.5. Yes, 6.5 is the same as 13/2, so 10(13/2) is (10/1)(13/2) = (10*13)/2 = 130/2 = 65. Yep, that's right. So, I think I did that correctly. The volume is 65 cubic units. Final Answer The volume of the cone is $\boxed{65}$ cubic units. The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where (B) is the area of the base and (h) is the height. Given: - The area of the base (B = 30) square units. - The height (h = 6.5) units. We substitute these values into the formula: $V = \frac{1}{3} \times 30 \times 6.5$ First, calculate $\frac{1}{3} \times 30$: $\frac{1}{3} \times 30 = 10$ Next, multiply the result by 6.5: $10 \times 6.5 = 65$ Thus, the volume of the cone is ($\boxed{65}$) cubic units.
Ours	To find the volume of the cone, I'll use the formula ($V = \frac{1}{3}Bh$). Given that the base area (B) is 30 square units and the height (h) is 6.5 units. First, multiply these values together: ($30 \times 6.5 = 195$) Then, take one-third of that product: ($\frac{1}{3} \times 195 = 65$) Therefore, the volume of the cone is 65 cubic units. Given: - Area of the base ((B)) = 30 square units - Height ((h)) = 6.5 units The formula for the volume (V) of a cone is: $V = \frac{1}{3}Bh$ Plugging in the given values: $V = \frac{1}{3} \times 30 \times 6.5$ Calculate: $30 \times 6.5 = 195$ Then, $V = \frac{195}{3} = 65$ Answer: ($\boxed{65}$) cubic units.

Table 7: Case II for vanilla and our trained LLM. Both answers are correct.