

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*. After that, we conduct extensive experiments (about 0.2 million GPU hours) in a unified protocol, deconstructing training prompts and rollouts, reward shaping, and optimization strategies. In particular, a key finding is to train on relatively easier prompts. We distill all findings into *valuable insights* and *practical guidelines*, and further validate them across multiple LLMs, demonstrating their 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).

To address this issue, one mainstream method is to incentivize efficient reasoning via Reinforcement Learning (RL) with reward shaping (Ma et al.,

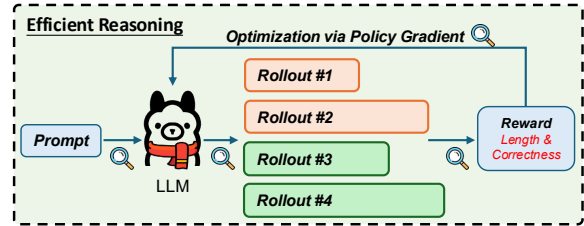


Figure 1: General pipeline for efficient reasoning via RL. The key is to promote short and accurate thinking trajectories via reward design. In this paper, we provide systematic insights (🔍) considering data, reward, and optimization.

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 ranging from 2k to 32k. The effectiveness of different strategies is budget-dependent, exhibiting

070 distinct or even contradictory behaviors.

071 Through extensive ablation studies, we further
072 deconstruct the impact of data difficulty, rollout
073 number, reward, and optimization strategies. No-
074 tably, we find that training on relatively easier
075 prompts provides a denser positive reward signal,
076 which is essential for stable reasoning distillation.
077 More rollouts contribute to better performance, but
078 also bring heavier training costs. For the reward
079 assignment, we compare the strategy to assign a
080 negative reward or mask corresponding rollouts.
081 Moreover, we further explore the off-policy strat-
082 egy with different staleness to speed up the rea-
083 soning refinement stage. We distill all the findings
084 into valuable insights and practical guidelines, and
085 evaluate them on more LLMs for robustness and
086 generalization. In summary, our contributions are
087 as follows:

- 088 • We identify and characterize the two-stage
089 paradigm for efficient reasoning, i.e., length
090 adaptation followed by reasoning refinement.
- 091 • We introduce fine-grained metrics, providing
092 a more comprehensive understanding of train-
093 ing dynamics.
- 094 • We provide a systematic exploration of the
095 training recipe, offering practical insights into
096 data, reward, and optimization that signifi-
097 cantly improve the efficiency of CoT models.

098 2 Preliminary

099 2.1 Experimental Setup

100 **Prompts.** RL methods have been demonstrated
101 as an effective way for reasoning. Given an input
102 prompt x from a dataset \mathcal{D} , the LLM policy π_θ gen-
103 erates a set of N reasoning trajectories (rollouts)
104 $\{y_1, y_2, \dots, y_N\}$. The objective is to update π_θ us-
105 ing policy gradients derived from reward signals
106 upon rollouts. Data quality is critical for efficient
107 reasoning distillation (Guo et al., 2025). Therefore,
108 we employ the popular DeepScaleR benchmark¹.

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

$$114 R_{\text{vanilla}}(x, y_i) = \mathbb{I}(y_i \text{ is correct}). \quad (1)$$

¹[Dataset link](#)

To enforce efficient reasoning, we employ reward
shaping to incentivize conciseness without sacrific-
ing accuracy. In this work, we focus on the Trunca-
tion Strategy as a strong baseline, which applies a
hard constraint on the trajectory length:

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

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

Evaluation Protocol. To capture the nuances of
efficient reasoning, we introduce fine-grained eval-
uation metrics beyond simple accuracy.

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

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 generation capabilities using 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) on 64
NVIDIA H200 GPUs. 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 2, the training dynamics
exhibit a distinct chronological structure, which we
characterize as a two-stage paradigm:

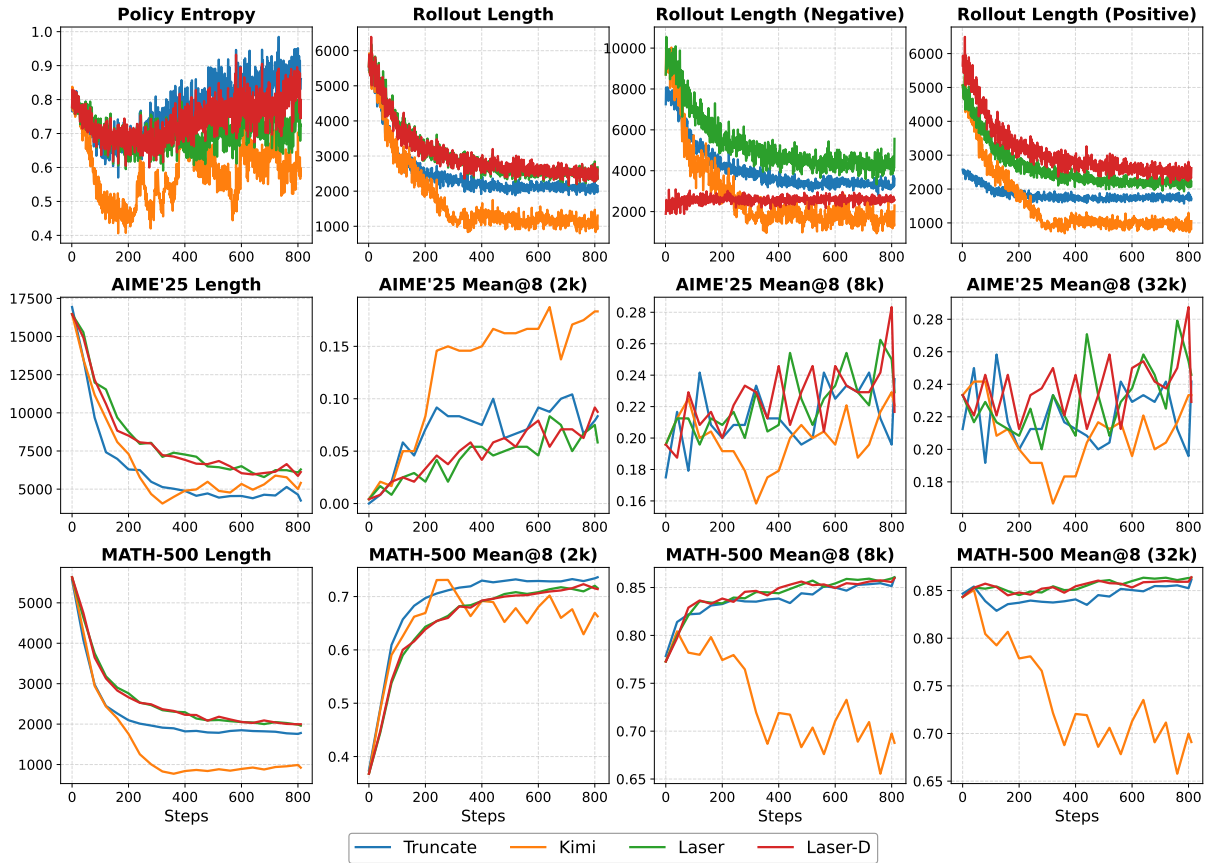


Figure 2: Training dynamics of various reward shaping methods on DeepSeek-R1-Distill-Qwen-1.5B. All of them follow the two-stage paradigm. The behaviors are distinct when evaluated under different token budgets.

Stage I: Length Adaptation. The initial phase is dominated by the optimization of constraint satisfaction. 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, valid trajectories. In this stage, the primary learning signal encourages the model to meet the length constraint.

Stage II: Reasoning Refinement. Once the rollout length stabilizes within the target budget, the training enters a stationary phase regarding length, shifting focus to logic optimization. In this stage, the length curves plateau, demonstrating that the model has successfully adapted to the hard constraints. Crucially, the performance metrics (e.g., Mean@8) continue to evolve or recover. This indicates that the model is learning to compress the Chain-of-Thought (CoT) and increase the infor-

mation density of each token to improve accuracy without violating the length budget.

2.3 Distinct Behaviors across Token Budgets

Contrary to previous single-budget evaluations, Figure 2 also reveals that model behaviors are highly sensitive to the token constraints, exhibiting distinct and often 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), these same methods often suffer from Reasoning Collapse, permanently sacrificing reasoning depth for brevity. In contrast, more balanced strategies (e.g., Laser) exhibit 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 strict efficiency can severely harm the model’s upper-bound reasoning capability, necessitating the multi-budget evaluation protocol.

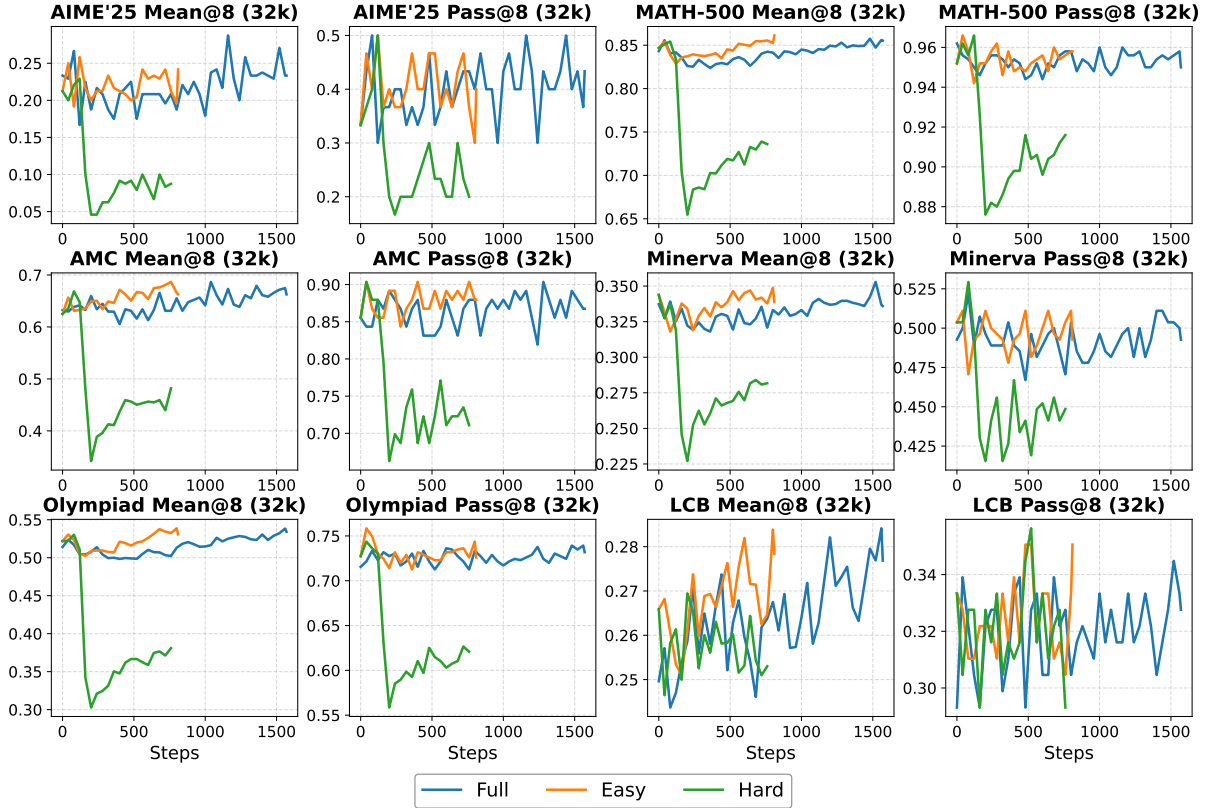


Figure 3: Performance training on all prompts and easy/hard counterparts (rollout $L_R = 16k$, target $L_T = 4k$).

3 Experiments and Guidelines

In this section, we systematically investigate the impact of training data, reward assignment, and optimization strategies. We select truncation as a minimalist proxy to isolate the effects of data and optimization dynamics, avoiding the confounding factors introduced by complex reward shaping.

3.1 Data: Prompt and Rollout

Impact of Prompt Difficulty. Efficient reasoning optimization is primarily a post-training alignment phase, aimed at distilling existing reasoning capabilities into concise formats rather than acquiring novel knowledge. Consequently, the difficulty of training prompts plays a pivotal role in determining the density of positive reward signals. To investigate this, we classify DeepScaleR prompts based on the backbone model’s pass rate over $N = 8$ rollouts: DeepScaleR-Easy (pass rate > 0.5) and DeepScaleR-Hard (pass rate ≤ 0.5).

As illustrated in Figure 3, the training dynamics exhibit stark differences depending on data composition. Training exclusively on Hard prompts results in catastrophic failure. The policy entropy spikes drastically, and the rollout length col-

lapses prematurely. Consequently, downstream performance metrics (e.g., AMC and Olympiad Mean@8) 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.

Conversely, training on Easy prompts 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 using only a subset of the available data, the Easy split achieves asymptotic performance comparable to (or even slightly exceeding) the Full dataset. Please refer to Appendix B for results and analysis under more settings. Qualitative analysis in Appendix D shows that the model does not merely truncate tokens but transitions to a more symbolic and expert-like reasoning style, even when trained on easier instances.

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 . Fixing the training data to the stable *DeepScaleR-Easy* split, we vary $N \in \{8, 12, 16, 24\}$ with a

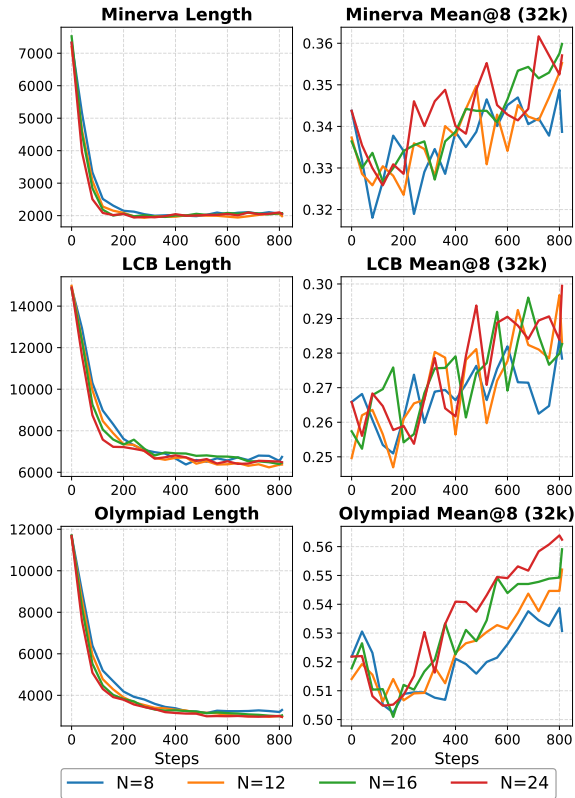


Figure 4: Performance with various rollouts N using DeepScaleR-Easy.

rollout limit $L_R = 16k$ and a target length $L_T = 4k$ trained on DeepScaleR-Easy.

As shown in Figure 4, increasing N yields observable benefits. Larger N significantly speeds up the *Length Adaptation* phase. With more attempts, the policy discovers high-reward (short and correct) trajectories earlier, causing the length curve (e.g., $N = 24$, red line) to decay faster than the baseline ($N = 8$, blue line), although all settings converge to a similar length floor. Meanwhile, larger N leads to a more robust *Reasoning Refinement* stage. In mathematical benchmarks (Minerva, Olympiad), 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 . It is worth noting that while increasing N improves performance, it introduces a linear increase in computational overhead during the rollout phase. Please refer to Appendix C for results under more settings.

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 resource allows.*

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 by Correctness and Length.

The art of RLHF often lies in the design of negative signals. In the standard truncation strategy (Vanilla), the penalty is uniform: both *incorrect* responses and *overlong correct* responses are treated as negative samples ($R = 0$). An intuitive idea is to mask the overlong correct rollouts rather than as negative samples. 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 5, the training dynamics reveal that different masking strategies exhibit distinct behaviors. Improper strategies will lead to unintended consequences, categorized into three distinct failure modes:

1) The trap that short is correct (-I, -L&C-S&I). When we mask penalties for incorrect answers based on length, the model will be misled by incorrect causal relationships.

For masking all incorrect (-I), we only penalize overlong and correct rollouts. 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 length 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.

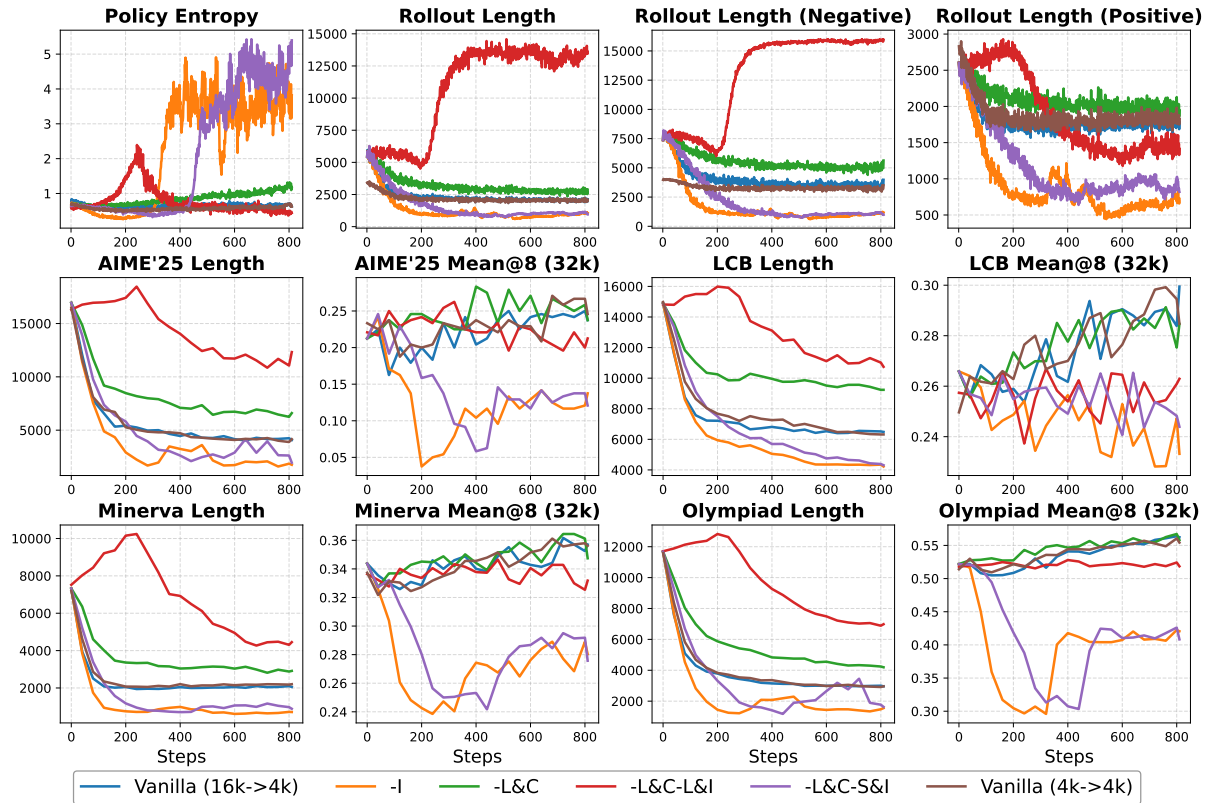


Figure 5: Performance for various reward strategies on negative rollouts (rollout $L_R = 16k$, target $L_T = 4k$, $N = 24$). We also visualize $L_R = 4k$, $L_T = 4k$ for comparison.

For masking overlong correct and short incorrect rollouts (-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. In this case, the LLMs will hack the length bias regardless of the correctness of negative overlong rollouts.

2) Short rollouts only (-L). A particularly interesting phenomenon occurs in the strategy **-L&C-L&I** (red line), where we remove the penalty for all overlong trajectories. It means that the LLMs are optimized with short correct and short incorrect rollouts. Meanwhile, 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 (no penalty). Consequently, the outputs on the benchmark turn out to be much longer. Interestingly, these outputs are almost incorrect, but 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 than baselines and outperform the vanilla baseline. It is a trade-off for length control and performance.

Finally, we compare these complex shaping strategies against a simple baseline: sampling at target length (i.e., $L_R = L_T$). The Vanilla ($L_R = 4k \rightarrow L_T = 4k$) setting (brown line) restricts the exploration space to the target budget from the outset. Unlike the masking strategies, which struggle to balance signals, this hard constraint forces the model to learn efficiency within a valid search space. We can observe that it achieves the optimal Pareto frontier: matching the high accuracy of lenience strategies while maintaining the minimal token consumption of aggressive strategies.

Insights towards Reward on Negative Rollouts: Do not penalize correct but overlong rollouts leads to higher performance, but also slightly longer outputs. Sampling at target length ($L_R = L_T$) achieves a better trade-off.

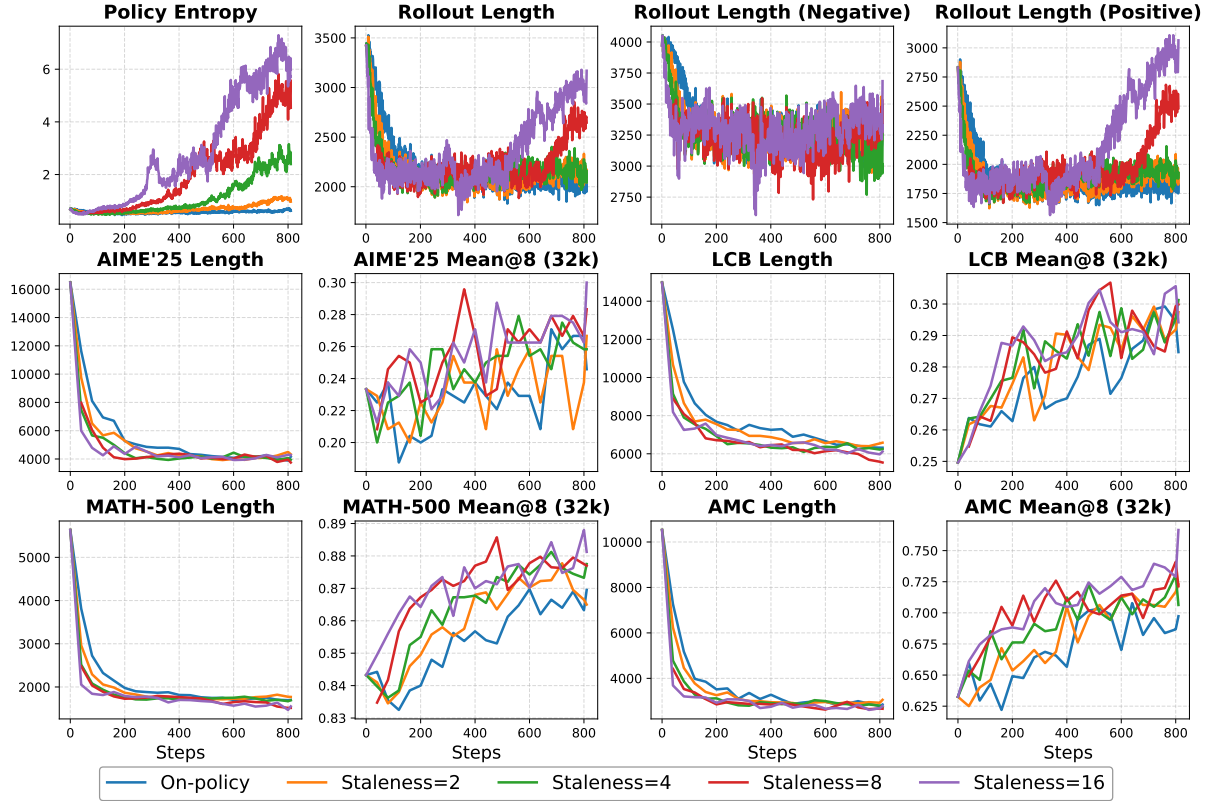


Figure 6: Performance for off-policy strategy with various staleness (i.e., 2,4,8,16).

3.3 Off-policy Optimization

In traditional RL training, off-policy updates (utilizing stale data) can significantly accelerate the training process but often introduce instability or divergence (Zheng et al., 2025; Huang et al., 2025). Meanwhile, we can observe that the performance keeps growing at the reasoning refinement stage. To investigate this, 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 6, the results reveal a nuanced trade-off. Increasing staleness acts as a powerful accelerator for the entire optimization stage. The *Length Adaptation* stage is significantly shortened. As staleness increases, the rollout length decays more rapidly (shifting towards the bottom-left), allowing the model to satisfy the token budget earlier. Consequently, the model enters the *Reasoning Refinement* stage sooner. Downstream metrics show that high-staleness models (e.g., $S = 16$, purple line) achieve higher accuracy faster than the on-policy baseline (blue line).

However, larger staleness also introduces latent instability risks. Contrary to findings that excessive off-policy updates lead to catastrophic model col-

lapse, our experiments maintain high performance. We attribute this robustness to our optimized setup (easy prompts and high N), which ensures sufficient and effective reward signals. However, signs of instability emerge in the training statistics:

- **Entropy explosion:** As shown in the *Policy Entropy* subplot, high staleness (e.g., 16) causes a dramatic surge in entropy after 400 steps.
- **Length rebound:** Correlated with the entropy spike, the training rollout length for high staleness begins to drift upwards again, indicating that the policy model is struggling to maintain the strict efficiency constraint.

Insights towards Off-policy Optimization: *Appropriate staleness can accelerate convergence without harming accuracy (before 800 steps), but also introduces latent instability, manifested as rising entropy and uncontrolled length drift. A moderate staleness is recommended (e.g., 4) to balance training efficiency with policy stability.*

4 Extensive Analysis

4.1 Evaluated on More LLMs

Method	Mean@8	Pass@8	Length
Qwen3-0.6B			
Vanilla	13.33	26.67	14.9k
Ours	21.25	46.67	10.8k
Δ	7.92 \uparrow	20.00 \uparrow	4.1k \downarrow
Qwen3-4B-Instruct-2507			
Vanilla	43.75	66.67	9.3k
Ours	49.58	73.33	8.2k
Δ	5.83 \uparrow	6.66 \uparrow	1.1k \downarrow

Table 2: Performance comparison on Qwen3 Models.

To verify the universality of our derived guidelines, we extend our evaluation to the Qwen3 family, including Qwen3-0.6B and Qwen3-4B-Instruct-2507. Based on the insights, we strictly align the rollout limit with the target budget: setting $L_R = L_T = 8k$ for the 0.6B model and $L_R = L_T = 16k$ for the 4B model. Crucially, we maintain a high reward density by sampling $N = 24$ trajectories for each prompt.

As shown in Table 2, the experimental results demonstrate the robustness of our strategy across different model scales. For Qwen3-0.6B, our method achieves a dramatic performance boost. The Pass@8 surges by +20.00% and Mean@8 by +7.92%, all while reducing the average response length by 4.1k tokens. This suggests that efficient reasoning is particularly critical for smaller models to navigate the solution space effectively. For the stronger Qwen3-4B-Instruct, we observe a consistent trend. The method yields a +5.83% increase in Mean@8 and a +6.66% increase in Pass@8, accompanied by a 1.1k reduction in token consumption.

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 does not merely truncate output, but actively incentivizes the model to reorganize its CoTs into a more streamlined and expert-like format.

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). 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 after comparison 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, i.e., the length adaptation and reasoning refinement. 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. Moreover, we distill all findings into valuable insights and practical guidelines. A key finding is to train on relatively easier prompts, which contribute to sufficient and effective rewards for optimization. Evaluation of more LLMs further demonstrates the robustness and generalization. For future work, finding an optimal target length is a valuable topic.

473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521

Limitation

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

Domain diversity of training prompts. Our training experiments primarily rely on the DeepScaleR benchmark, which is concentrated on mathematical reasoning. Consequently, the impact of utilizing other data domains (e.g., code generation, common sense reasoning) for training remains underexplored. However, we argue that this does not undermine the validity of our conclusions. As demonstrated in our extensive evaluations, the efficient reasoning patterns distilled from mathematics (e.g., *length adaptation* and *reasoning refinement*) successfully generalize to out-of-domain tasks like LiveCodeBench. This suggests that the underlying capability of eliminating redundancy and optimizing thinking trajectories is transferable across domains. Nevertheless, incorporating domain-specific training data (e.g., training directly on code repositories) remains a promising avenue to further maximize efficiency in specific verticals.

Fixed vs. adaptive target length. In our current protocol, we employ a fixed target length (L_T) to incentivize conciseness. While this serves as an effective and controlled baseline for revealing the *two-stage paradigm*, it may not be the optimal strategy for all queries. Complex problems may inherently require longer reasoning budgets, while simpler ones can be compressed further. Future work could explore adaptive length thresholds that dynamically adjust the budget conditioned on the input difficulty, thereby achieving a more flexible trade-off between efficiency and performance.

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*.

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. 522
523
524
525

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*. 526
527
528

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. 529
530
531
532
533

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*. 534
535
536
537

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. 538
539
540
541
542
543
544
545
546

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*. 547
548
549
550
551

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*. 552
553
554
555
556

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*. 557
558
559
560
561
562

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*. 563
564
565
566
567
568

Kimi, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chen-zhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*. 569
570
571
572
573

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo

577	Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. <i>Advances in neural information processing systems</i> , 35:3843–3857.	631
578		632
579		633
580		634
581	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. <i>arXiv preprint arXiv:2505.15612</i> .	635
582		636
583		637
584		638
585		
586	Haozhe Ma, Zhengding Luo, Thanh Vinh Vo, Kuankuan Sima, and Tze-Yun Leong. 2024. Highly efficient self-adaptive reward shaping for reinforcement learning. <i>arXiv preprint arXiv:2408.03029</i> .	639
587		640
588		641
589		642
590	Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. Cot-valve: Length-compressible chain-of-thought tuning. <i>arXiv preprint arXiv:2502.09601</i> .	643
591		644
592		645
593		646
594	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. <i>arXiv preprint arXiv:2402.03300</i> .	647
595		648
596		649
597		650
598		651
599		652
600	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. <i>arXiv preprint arXiv:2503.04472</i> .	653
601		654
602		655
603		656
604		657
605	Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. <i>arXiv preprint arXiv:2408.03314</i> .	658
606		659
607		660
608		661
609	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. <i>arXiv preprint arXiv:2502.03275</i> .	
610		
611		
612		
613		
614	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. <i>arXiv preprint arXiv:2503.16419</i> .	
615		
616		
617		
618		
619		
620	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. <i>Advances in Neural Information Processing Systems</i> , 37:32630–32652.	
621		
622		
623		
624		
625	Lilian Weng. 2025. Why we think . <i>lilianweng.github.io</i> .	
626		
627	Taiqiang Wu, Runming Yang, Tao Liu, Jiahao Wang, and Ngai Wong. 2025a. Revisiting model interpolation for efficient reasoning. <i>arXiv preprint arXiv:2510.10977</i> .	
628		
629		
630		
	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. <i>arXiv preprint arXiv:2502.07266</i> .	
	Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. <i>arXiv preprint arXiv:2502.12067</i> .	
	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. <i>arXiv preprint arXiv:2503.13288</i> .	
	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. <i>arXiv preprint arXiv:2505.09388</i> .	
	Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. <i>arXiv preprint arXiv:2502.03373</i> .	
	Qiyang 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. <i>arXiv preprint arXiv:2503.14476</i> .	
	Haizhong Zheng, Jiawei Zhao, and Beidi Chen. 2025. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? <i>arXiv preprint arXiv:2510.01161</i> .	

Appendix

A Detailed Results on Reward Engineering

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

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

The reward function can be rewritten as the sum of the correct reward and the incorrect penalty:

$$R_{\text{Kimi}}(x, y_i) = \mathbb{I}(y_i \text{ is correct}) \cdot (1.5 - \tilde{L}(y_i)) + \mathbb{I}(y_i \text{ is incorrect}) \cdot \min(0, 0.5 - \tilde{L}(y_i)). \quad (4)$$

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

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

They also propose a variant to encourage exploration. This variant adds a term to incentivize long incorrect trajectories (exploration). It is the sum of the standard Laser reward and the exploration bonus:

$$R_{\text{Laser-D}}(x, y_i) = \mathbb{I}(y_i \text{ is correct}) \cdot (1 + \alpha \cdot \mathbb{I}(L(y_i) < L_T)) + \mathbb{I}(y_i \text{ is incorrect}) \cdot (\alpha \cdot \mathbb{I}(L(y_i) \geq L_T)). \quad (6)$$

Figures 7 and 8 present the training dynamics on additional benchmarks, including AIME’25, MATH-500, AMC (Figure 7) and Minerva Math, Olympiad Bench, LiveCodeBench (Figure 8).

Consistent with our main analysis, we observe the following patterns:

- **Universal two-stage paradigm:** Across all benchmarks, the training process strictly follows the *length adaptation* phase followed by the *reasoning refinement* phase. The rollout length (top row) consistently decays exponentially before stabilizing.
- **Budget sensitivity:** The performance decoupling phenomenon remains pervasive. For instance, on challenging tasks like Olympiad Bench, aggressive strategies (e.g., Kimi) often show rapid gains at stricter budgets (2k) but suffer from stagnation or collapse at generous budgets (32k), whereas the Truncation baseline maintains a more balanced recovery.

These extensive results reinforce the necessity of our proposed fine-grained evaluation protocol: relying on a single token budget or a single dataset is insufficient to capture the true efficacy of efficient reasoning strategies.

B Detailed Results for Data Selection

In the main text, we demonstrated that training on easier prompts leads to more stable optimization. Here, we extend this analysis to our full evaluation suite, comprising AIME’25, MATH-500, AMC, Minerva Math, Olympiad Bench, and LiveCodeBench.

As illustrated in Figures 9 and 10, the results consistently validate our hypothesis regarding reward density. 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. This confirms that sparse reward signals from difficult prompts impede effective reasoning distillation.

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 LiveCodeBench (LCB), the Easy split achieves a competitive pass rate with significantly lower variance than the Hard split.

The fact that relatively easier prompts suffice for high performance even on difficult test sets (like Olympiad Bench) suggests that the core mechanism of efficient reasoning via RL is a transferable skill that does not strictly require training on the hardest problem instances.

C Detailed Results for More Rollouts

Figures 11 and 12 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 constraint (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 strictly positive effect on the convergence speed and asymptotic performance of the Mean@8 metric. This confirms that increasing the density of the reward signal is a robust strategy for stabilizing efficient reasoning, independent of the exploration horizon.

2) Task-dependent sensitivity. The magnitude of improvement varies significantly across datasets. On highly 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 to break through the reasoning barrier 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. In these regimes, the model likely knows how to solve the problem, and higher N helps it learn to do so more reliably within the length constraint.

3) Stability vs. peak capability (Mean@8 vs. Pass@8). A critical observation is the divergence between Mean@8 and Pass@8 metrics. While Mean@8 shows significant improvement with larger N , the Pass@8 metric often remains stagnant or improves only marginally. Pass@8 measures the model's ability to generate at least one correct solution (peak capability). The lack of improvement here implies that scaling N does not necessarily enable the model to solve new problems that were previously unsolvable. Mean@8 measures the expected correctness across samples (consistency). The rise in Mean@8 indicates that scaling N effectively reduces the variance of the policy.

Conclusion: Increasing N primarily acts as a stabilizer, reducing the likelihood of reasoning errors on solvable problems rather than expanding the model's upper-bound intelligence.

D Detailed Cases

Tables 3 and 4 indicate two examples from the training prompts. 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 stark 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. The reasoning transitions from a hesitant narrative to a precise, symbolic derivation (e.g., integrating formulas like $V = \frac{1}{3}Bh$ directly into the calculation flow). This demonstrates that our method does not merely truncate output, but actively incentivizes the model to reorganize its CoTs into a more streamlined and expert-like format.

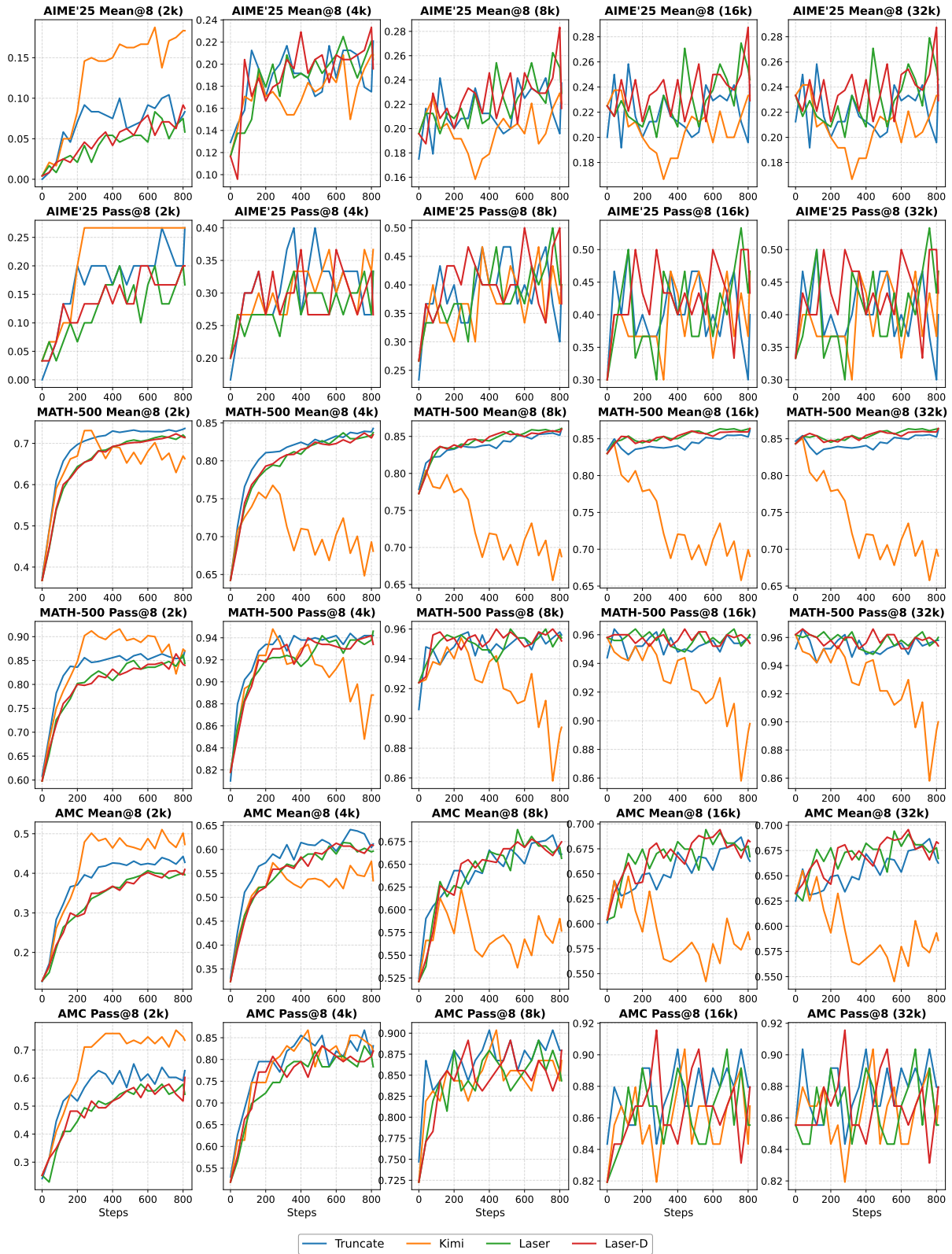


Figure 7: Training dynamics of various reward shaping methods on AIME'25, MATH-500, and AMC.

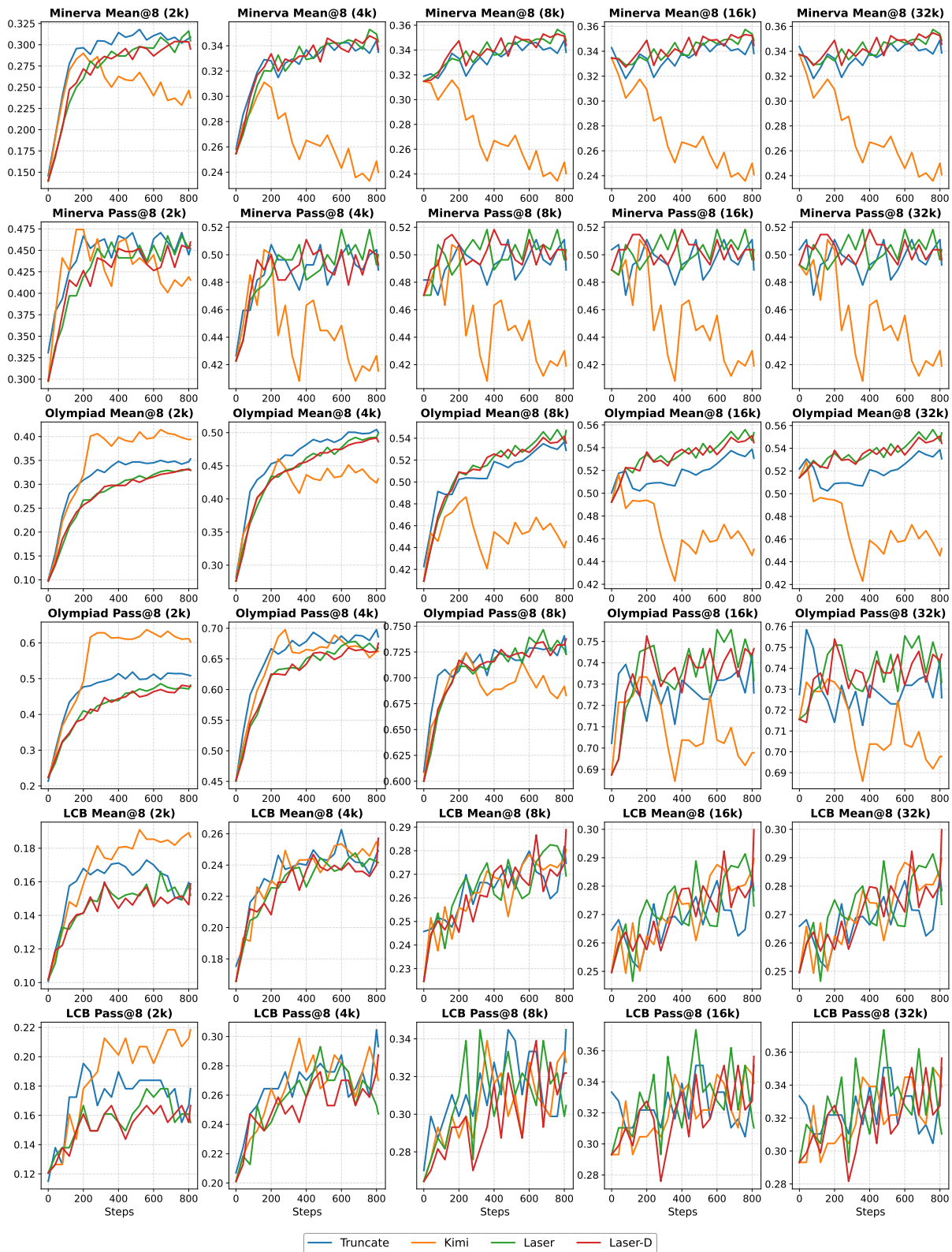


Figure 8: Training dynamics of various reward shaping methods on Minerva Math, Olympiad Bench, and Live-CodeBench.

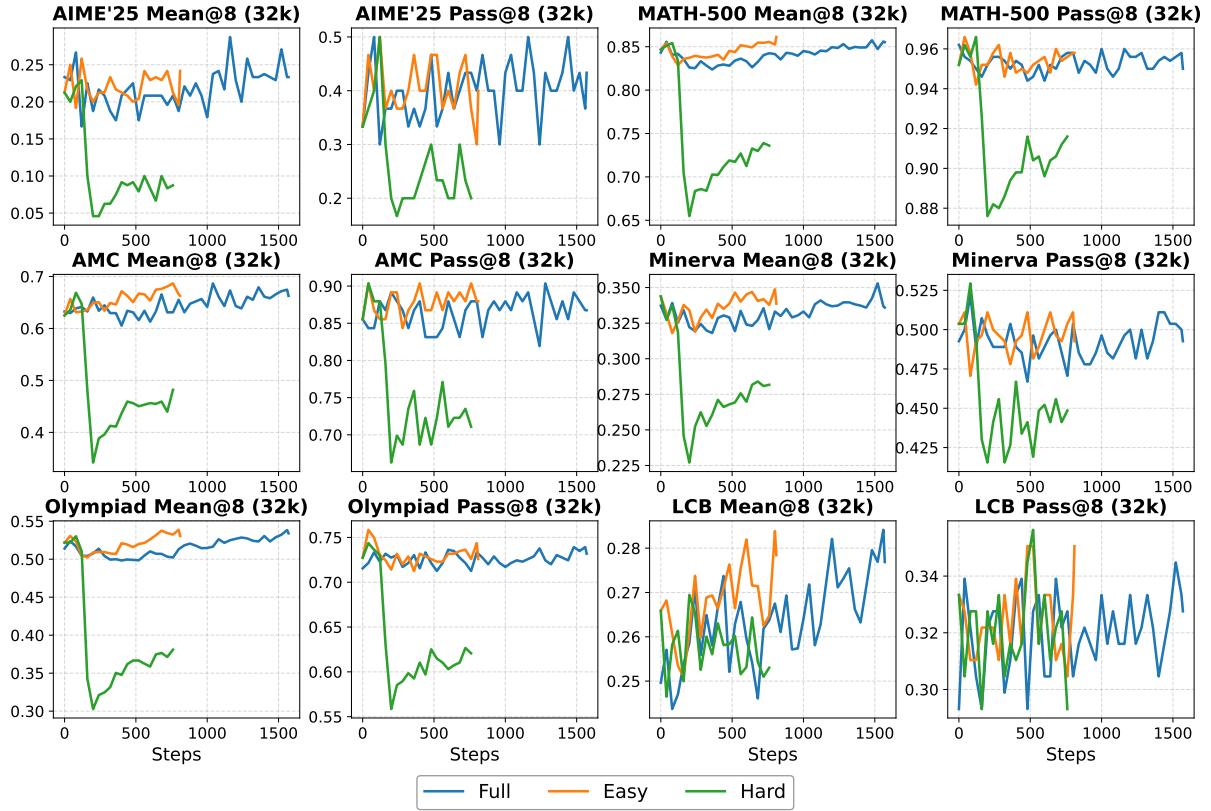


Figure 9: Performance training on various training prompts (rollout $L_R = 16k$, target $L_T = 8k$).

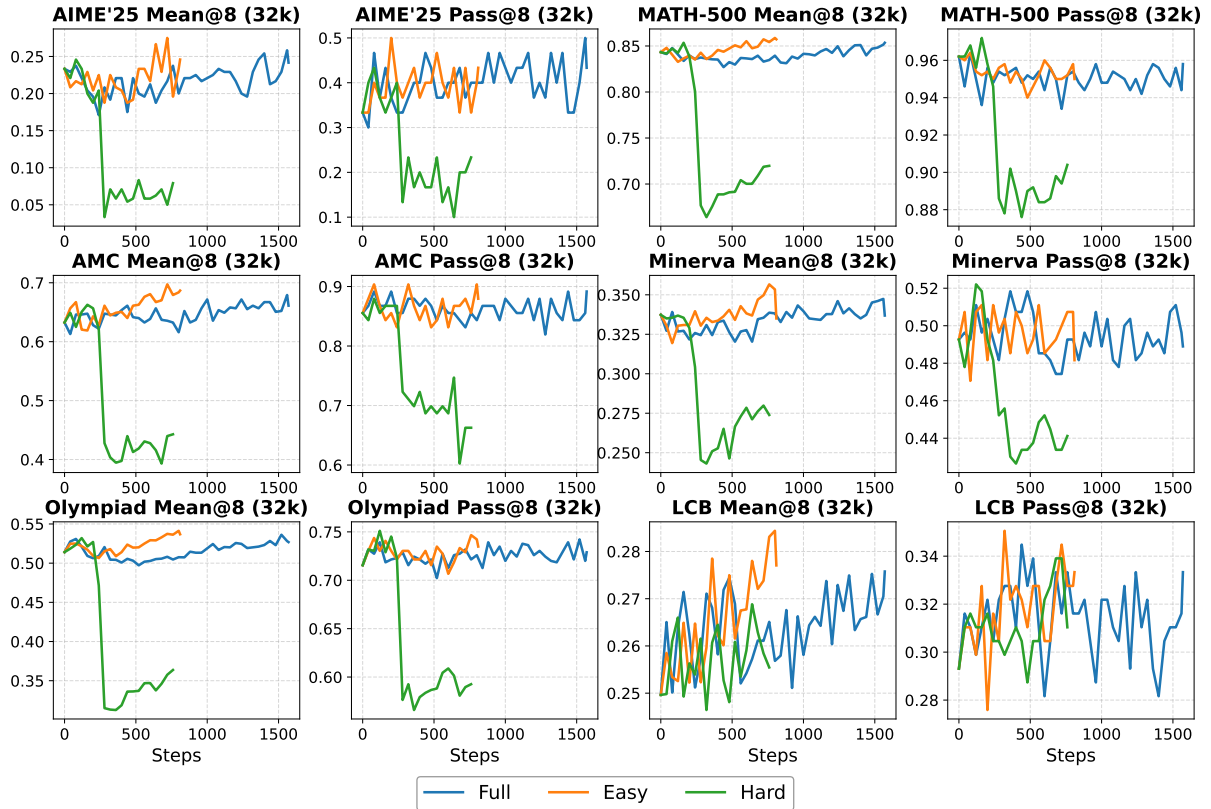


Figure 10: Performance training on various training prompts (rollout $L_R = 16k$, target $L_T = 4k$).

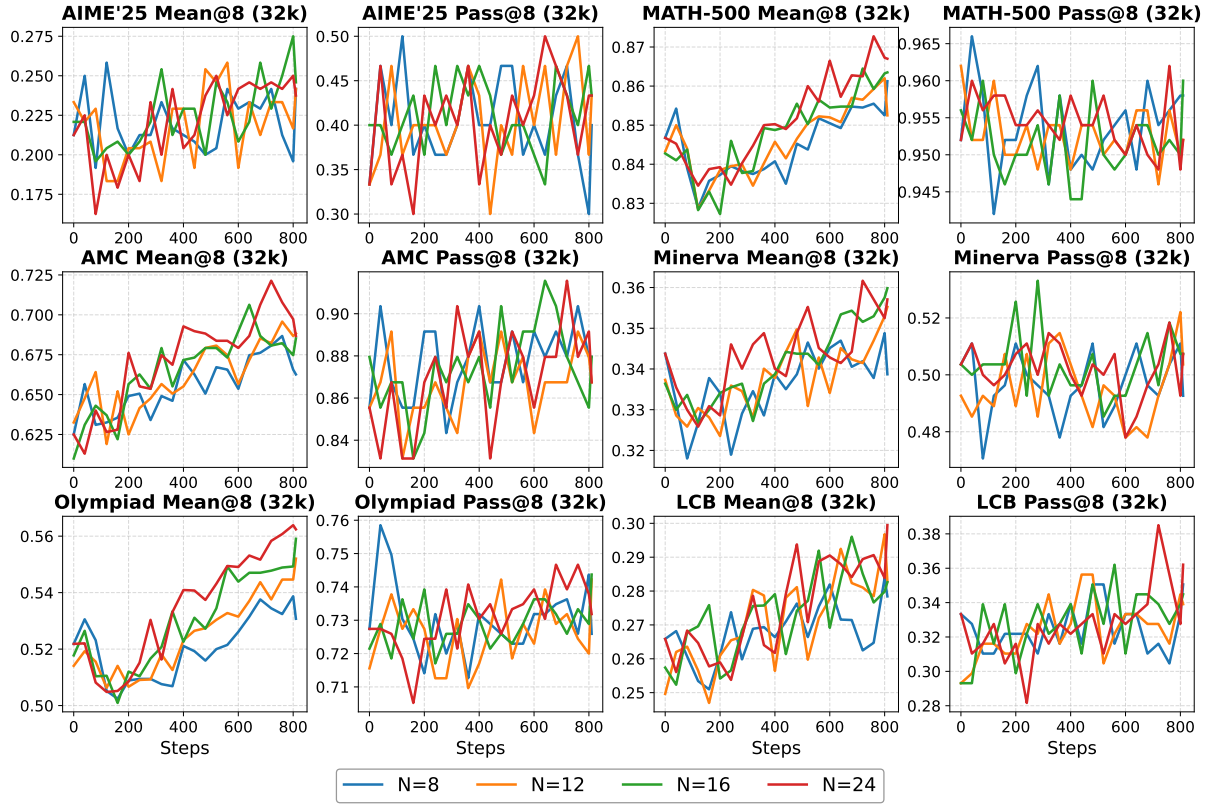


Figure 11: Performance training on various rollouts N ($L_R = 16k$, $L_T = 4k$).

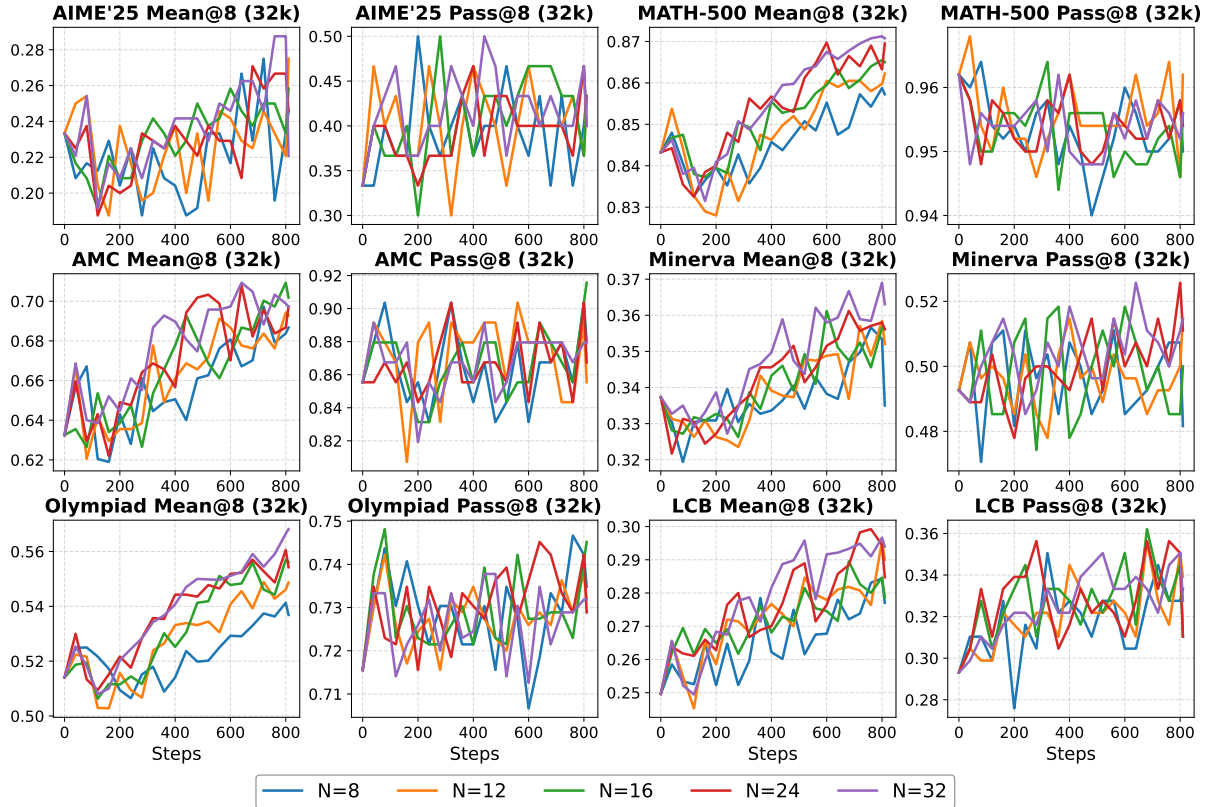


Figure 12: Performance training on various rollouts N ($L_R = 4k$, $L_T = 4k$).

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: $\binom{31}{28} = \frac{31 \times 30 \times 29 \times 28!}{28! \times 3!}$ Oh, I see! The $(28!)$ terms cancel out in

numerator and denominator: $= \frac{31 \times 30 \times 29}{3!}$ Now, $(3! = 3 \times 2 \times 1 = 6)$, so substituting that in: $= \frac{31 \times 30 \times 29}{6}$

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: - $930 \times 20 = 18,600$ - $930 \times 9 = 8,370$ 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: $26,970 \div 6$. Dividing step by step: - 6 goes into 26 four times ($4 \times 6 = 24$), remainder 2. - Bring down the 9: 29. 6 goes into 29 four times ($4 \times 6 = 24$), remainder 5. - Bring down the 7: 57. 6 goes into 57 nine times ($9 \times 6 = 54$), 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 $26,970 \div 6$ as $(26,000 + 970) \div 6$. $26,000/6 = 4,333.333...$ $970/6 \approx 161.666...$ 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 `\boxed{4495}`. 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 $(31! = 31 \times 30 \times 29 \times 28!)$: $\binom{31}{28} = \frac{31 \times 30 \times 29 \times 28!}{28! \times 3!} = \frac{31 \times 30 \times 29}{3!}$

Since $(3! = 6)$, we have: $= \frac{31 \times 30 \times 29}{6}$ 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: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ 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: $\binom{31}{3} = \frac{31!}{3!(31-3)!} = \frac{31!}{3! \cdot 28!}$ Simplifying, since $(31! =$

$31 \times 30 \times 29 \times 28!)$: $\frac{31 \times 30 \times 29 \times 28!}{3! \times 28!} = \frac{31 \times 30 \times 29}{3!}$ Calculate the denominator $(3! = 6)$: Therefore, $\frac{31 \times 30 \times 29}{6}$ Computing numerator: $31 \times 30 = 930$; $930 \times 29 = ?$ Compute: 930×29 : Breakdown: $= 930 \times (30 - 1) = 930 \times 30 - 930 \times 1 = 27900 - 930 = 26970$ Then divide by 6: $26970 \div 6 = ?$ Calculate: $26970 \div 6 = 4495$ Wait perhaps: $6 \times 4495 = 26970$, 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, $\binom{31}{3} = \frac{31!}{3!(31-3)!} = \frac{31 \times 30 \times 29}{3 \times 2 \times 1}$ Calculating: $31 \times 30 \times 29 = 26970$ Divide by $(6) :$

$26970/6 = 4495$ Thus, `\boxed{4495}`

Table 3: One case for vanilla and our trained LLM

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	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>So, putting it all together, $V = 65$ cubic units. That should be the volume of the cone.</p> <p>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.</p> <p>So, I think I did that correctly. The volume is 65 cubic units.</p> <p>Final Answer The volume of the cone is $\boxed{65}$ cubic units.</p> <p>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.</p> <p>Given: - The area of the base ($B = 30$) square units. - The height ($h = 6.5$) units.</p> <p>We substitute these values into the formula:</p> $V = \frac{1}{3} \times 30 \times 6.5$ <p>First, calculate $\frac{1}{3} \times 30$:</p> $\frac{1}{3} \times 30 = 10$ <p>Next, multiply the result by 6.5:</p> $10 \times 6.5 = 65$ <p>Thus, the volume of the cone is $\boxed{65}$ cubic units.</p>
Ours	<p>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: $V = \frac{1}{3} \times 30 \times 6.5$</p> <p>Calculate: $30 \times 6.5 = 195$ Then, $V = \frac{195}{3} = 65$ Answer: $\boxed{65}$ cubic units.</p>

Table 4: One case for vanilla and our trained LLM