

Anti-Length Shift: Dynamic Outlier Truncation for Training Efficient Reasoning Models

Anonymous ACL submission

Abstract

Large reasoning models enhanced by reinforcement learning with verifiable rewards have achieved significant performance gains by extending their chain-of-thought. However, this paradigm incurs substantial deployment costs as models often exhibit excessive verbosity on simple queries. Existing efficient reasoning methods relying on explicit length penalties often introduce optimization conflicts and leave the generative mechanisms driving overthinking largely unexamined. In this paper, we identify a phenomenon termed length shift where models increasingly generate unnecessary reasoning on trivial inputs during training. To address this, we introduce Dynamic Outlier Truncation (DOT), a training-time intervention that selectively suppresses redundant tokens. This method targets only the extreme tail of response lengths within fully correct roll-out groups while preserving long-horizon reasoning capabilities for complex problems. To complement this intervention and ensure stable convergence, we further incorporate auxiliary KL regularization and predictive dynamic sampling. Experimental results across multiple model scales demonstrate that our approach significantly pushes the efficiency-performance Pareto frontier outward. Notably, on the AIME-24, our method reduces inference token usage by 78% while simultaneously increasing accuracy compared to the initial policy and surpassing state-of-the-art efficient reasoning methods.

1 Introduction

A widely used way to improve the reasoning capability of large language models (LLMs) is to make them “think longer” (Snell et al., 2024). Recent reasoning models (OpenAI et al., 2024; DeepSeek-AI et al., 2025a; Yang et al., 2025) have systematized this idea by combining mid-training data refinement with post-training reinforcement learning with verifiable rewards (RLVR) (Zhang et al., 2025a), yielding policies that allocate more com-

putation at inference time via longer reasoning trajectories. This paradigm delivers substantial gains on challenging benchmarks, making long chain-of-thought (CoT) (Wei et al., 2022) nearly a standard component of state-of-the-art models.

Nonetheless, this paradigm incurs substantial deployment costs. Long-CoT policies often expend excessive tokens on trivial queries, manifesting as repetitive restatements, backtracking, and post-hoc self-checks that rarely alter the final answer (Chen et al., 2025). Given that modern reasoning models are predominantly post-trained via RLVR, recent efforts on efficient reasoning naturally targets the RL objective and encodes brevity by directly coupling response length with reward. These methods commonly introduce explicit length-aware shaping through intra-group comparisons (Luo et al., 2025a; Team et al., 2025a; Shen et al., 2025a; Cheng et al., 2025), explicit budget adherence (Aggarwal and Welleck, 2025), or thresholded rewards for correctness under a target length (Hou et al., 2025; Zhang et al., 2025b; Liu et al., 2025b; Wen et al., 2025; Liu et al., 2025a), demonstrating that significant token reduction can be achieved with little or no accuracy loss. Despite these empirical successes, such shaping induces optimization conflicts, as gradients for length reduction often diverge from accuracy maximization. This misalignment hinders convergence and suppresses exploration, forcing reliance on heuristic schedules or sensitive tuning to sustain a fragile trade-off. More importantly, existing methods primarily penalize overthinking as an outcome, leaving its generative mechanisms under-investigated. This motivates a mechanism-centric treatment that reduces response length without sacrificing the model’s exploration capabilities.

In this paper, we first provide an empirical analysis of why reasoning models become verbose on easy inputs during mid and post-training. We term this phenomenon *length shift*: prompts that are already solved correctly tend to elicit longer re-

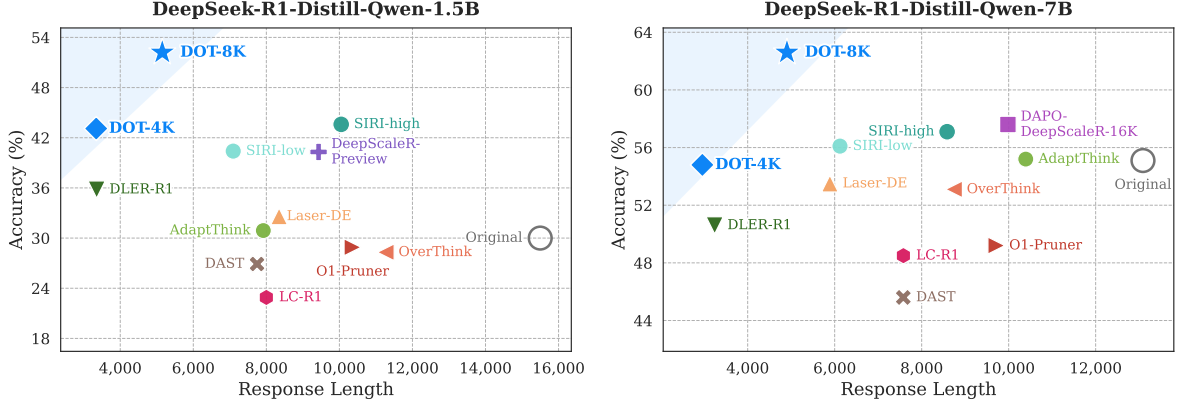


Figure 1: Performance-efficiency comparison on AIME-24 across two model scales.

sponses as training progresses. We find that length shift is accompanied by a higher propensity to emit reasoning words such as verification and hesitation markers; these behaviors are useful under uncertainty, but under a shared policy across difficulty regimes they are over-triggered on trivial queries and inflate length. This calls for an asymmetric intervention that selectively prunes habituated verbosity on trivial inputs, while strictly insulating hard queries from restrictive length constraints.

To achieve this, we propose **Dynamic Outlier Truncation (DOT)**. DOT identifies redundancy solely within fully correct rollout groups, using group-wise length statistics to truncate trajectories exceeding a dynamic threshold $T = \mu + \alpha \cdot \sigma$. Since T is determined a posteriori, DOT leaves hard queries unconstrained during generation; meanwhile, its reliance on dynamic distribution statistics minimizes susceptibility to reward hacking. Further analysis shows that although DOT affects only a tiny fraction of responses (e.g., $\sim 0.5\%$) it can still induce global entropy reduction. We therefore introduce auxiliary KL regularizer (Cui et al., 2025) to prevent premature entropy collapse, and propose a predictive dynamic sampling strategy to avoid late-stage training being dominated by all-correct queries. As shown in Fig. 1, DOT moves the Pareto frontier outward by a large margin. For example, on DeepSeek-R1-Distill-Qwen-1.5B, DOT uses only 21.6% of the average tokens on AIME-24 while improving pass@1 from 30.0% to 43.1%. Moreover, compared with state-of-the-art efficient reasoning methods (e.g., SIRI), DOT uses only 33.3% of the average tokens on AIME-24 with essentially the same accuracy. In summary, we make the following contributions:

- An empirical analysis of the length shift phenomenon, revealing that redundancy in reasoning

models stems from the over-triggering of reasoning words on trivial queries during training.

- Dynamic Outlier Truncation (DOT), a training-time intervention based on group-wise statistics that selectively truncates redundant rollouts in all-correct groups, minimizing reward hacking while preserving long-horizon reasoning capacity.
- Results across 1.5B, 7B and 32B models demonstrate that our training recipe generalizes robustly across scales and consistently pushes the efficiency–performance Pareto frontier outward on challenging reasoning benchmarks.

2 Preliminary

In this work, we ground our analysis and discussion in the framework of Group Relative Policy Optimization (GRPO) (Shao et al., 2024). GRPO estimates advantages in a group-relative manner and avoids fitting an explicit value function. For each query–answer pair (q, a) , the behavior policy $\pi_{\theta_{\text{old}}}$ samples a rollout group of G responses $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | q)$. Let R_i denote the reward assigned to o_i . The advantage for the i -th response is obtained by normalizing rewards within the group:

$$\hat{A}_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G)}. \quad (1)$$

GRPO updates the policy by maximizing the following objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a) \sim \mathcal{D}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | q)} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \min \left(r_{i,t}(\theta) \hat{A}_i, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i \right) \right], \quad (2)$$

where ϵ is the clipping range of importance sampling ratio:

$$r_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t} | q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} | q, o_{i,<t})}. \quad (3)$$

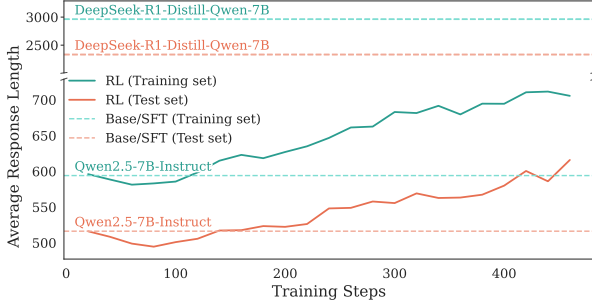


Figure 2: Evolution of average response length on all-correct queries during RL and SFT training.

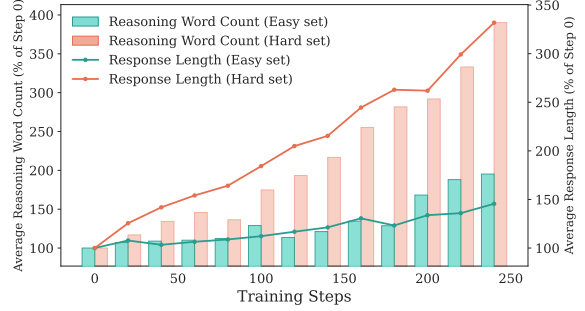


Figure 3: Co-evolution of reasoning word count and response length on test problems of varying difficulty.

In this paper, an explicit KL penalty to a reference policy is omitted following DAPO (Yu et al., 2025), which encourages exploration during RL.

3 Empirical Analysis

In this section, we attempt to answer a fundamental question through empirical analysis: *why do reasoning models produce increasingly long responses on problems that can already be solved easily?* Our key observation is that length growth persists even on all-correct queries for which GRPO yields zero advantages, indicating that overthinking arises from a global shift in policy behavior rather than any need to improve correctness on those problems.

Concretely, under RL with GRPO, we construct an all-correct set consisting of queries that are already solved correctly at the start of training, where each rollout group satisfies $\{R_i\}_{i=1}^G = \mathbf{1}$. For such queries, Eq. (1) implies that the group-wise reward has zero variance, yielding $\hat{A}_i = 0$ for all responses. Consequently, these prompts contribute zero gradient signal under Eq. (2) for improving correctness, and thus should not be systematically driven to become longer. Counterintuitively, Fig. 2 shows the opposite: both SFT (as a proxy for mid-training) and RL (post-training), starting from the same base model, increase the average response length $\mathbb{E}[|o|]$ on this all-correct set, and the trend consistently holds on both training and test splits, indicating a genuine distributional shift instead of overfitting.

To go beyond this phenomenological trend and uncover the mechanism behind length shift, we ask what internal behavioral change accompanies this drift and could plausibly mediate it. Specifically, we track the model’s propensity to emit a set of reasoning words, lexical markers of hesitation, verification, and reflection (e.g., “however” and “wait”) that often precede a new reasoning segment (Hu et al., 2025; Huang et al., 2025). Fig. 3

shows that during RL these markers increase over training steps on both easy (pass@1 = 1.0) and hard (pass@1 ≤ 0.6) subsets, accompanied by a co-evolving increase in $\mathbb{E}[|o|]$. The hard subset exhibits a larger growth, consistent with the view that reasoning words supports exploration and helps resolve complex problems (Yu et al., 2025). However, the concurrent upward drift on the easy subset supports a different and more concerning implication: learning on hard queries increases the global prior of emitting reasoning words under a shared policy, and once such words are generated, they tend to initiate additional reasoning spans that compound into longer trajectories even when further deliberation is unnecessary. Overall, this empirical analysis suggests that overthinking on trivial inputs can be viewed as cross-difficulty policy interference, where behaviors that are beneficial for handling uncertainty on hard problems are overly activated on easy ones. This perspective motivates interventions that suppress only the redundant long tail on already-solved queries, while preserving exploration capacity where it is genuinely needed.

4 Methodology

In this section, we propose a simple RL training recipe to counter length shift while preserving the model’s exploration ability. Motivated by the analysis in Sec. 3, we depart from prior approaches that explicitly incorporate response length into the reward. Instead, we introduce **Dynamic Outlier Truncation (DOT)**, a training-time intervention that trims only the extreme length tail after rollouts are sampled, and only on prompts that the policy already solves reliably (as illustrated in Fig. 9). Furthermore, by inspecting the resulting training dynamics, we incorporate two targeted auxiliary techniques that further improve the stability and scalability of RL training.

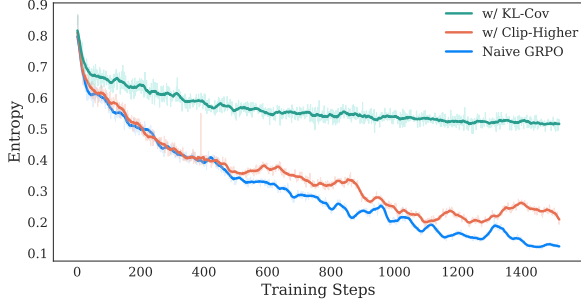


Figure 4: Evolution of policy entropy during training.

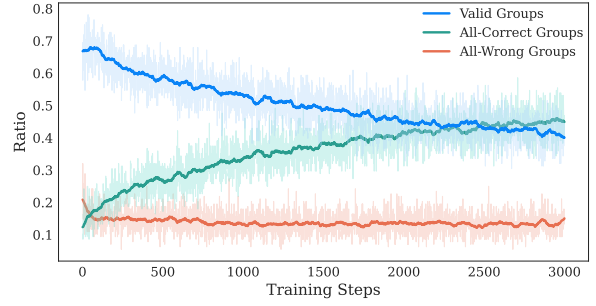


Figure 5: Evolution of group ratios during training.

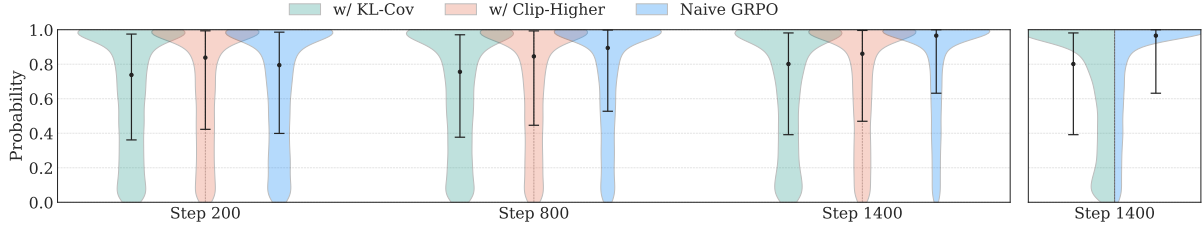


Figure 6: Distribution of observed sampling probabilities of reasoning words on AIME-24/25 during training.

4.1 Dynamic Outlier Truncation (DOT)

DOT operates on rollout groups in GRPO-style RL. Given a query q with a rollout group $\{o_i\}_{i=1}^G$ and rewards $\{R_i\}_{i=1}^G$, we apply DOT only when $\{R_i\}_{i=1}^G = \mathbf{1}$, i.e., all sampled responses are correct under the task reward. In GRPO, such groups exhibit no residual learning pressure for correctness (their group-relative advantages vanish due to zero reward variance), so an overly long trajectory should be understood as redundancy rather than uncertainty-driven exploration.

To identify redundant verbosity without introducing a length objective into the reward, DOT uses a-posteriori, group-wise statistics. Let $L_i = |o_i|$ denote the response length in tokens. We compute the group mean and standard deviation,

$$\mu_L = \text{mean}(\{L_i\}_{i=1}^G), \quad \sigma_L = \text{std}(\{L_i\}_{i=1}^G), \quad (4)$$

and define an outlier cutoff

$$T(q) = \lfloor \mu_L + \alpha \cdot \sigma_L \rfloor. \quad (5)$$

Using a “three-sigma”-style threshold makes DOT target only statistical outliers, leaving typical rollouts unchanged.

To avoid unstable behavior when σ_L becomes small late in training, we apply truncation only when the potential reduction is non-trivial, $L_i - T(q) \geq m$. This margin m prevents frequent minor edits from injecting gradient noise.

Formally, DOT can be viewed as a post processing on sampled rollouts:

$$\hat{o}_i = \begin{cases} o_{i,1:T(q)}, & \text{if } \{R_j\}_{j=1}^G = \mathbf{1} \text{ and } L_i - T(q) \geq m \\ o_i, & \text{else} \end{cases} \quad (6)$$

After truncation, we recompute the reward on the modified rollouts, ensuring that any dependence on later tokens is reflected in the task reward, while leaving the standard GRPO update unchanged.

4.2 Stabilizing and Scaling RL Training

Although DOT affects only a small fraction of rollouts within the all correct groups, it introduces subtle yet critical shifts in the training dynamics.

Structural Degeneration of the Reasoning Subspace.

Even rare interventions in the tail are associated with a rapid decline in policy entropy (Fig. 9). This trend persists even when employing the asymmetric clipping strategy (Clip-Higher) from DAPO, suggesting that the observed decline is not merely an instance of indiscriminate entropy collapse. Instead, it reflects a structural degeneration of the reasoning subspace. Once redundant long tails are removed on already-solved prompts, the policy update aggressively optimizes for efficiency by rapidly driving the reasoning words that originally facilitated exploratory branching into a near-deterministic regime (see Fig. 6). To counteract this behavior, we incorporate KL-Cov (Cui et al., 2025) as a regularizer. This method identifies the specific subset of tokens exhibiting high covariance between their log-probabilities and advantage estimates, which serves as a statistical signature for aggressive policy shifts. By imposing a targeted KL penalty $\mathbb{D}_{KL}(\pi_\theta || \pi_{old})$ exclusively on these high-covariance tokens, KL-Cov effectively curbs the drastic updates that lead to premature determinism, thereby maintaining exploration stability without

constraining the entire policy distribution.

Sampling Inefficiency in Evolving Distributions.

When training stronger base models or scaling RL to longer schedules, more prompts gradually become all correct. Consequently, although DOT affects a small absolute number of responses, these truncated trajectories increasingly dominate the RL training dynamics (Fig. 4). While previous work like DAPO addresses this via dynamic sampling, such approaches incur significant synchronization overheads and latency. In our context, the challenge is further compounded because DOT dynamically reactivates a subset of zero-gradient groups by re-computing rewards on truncated outputs, making the effective group ratio difficult to tune with fixed oversampling. To address this, we propose Predictive Dynamic Sampling. Instead of relying on costly iterative generation or wasteful fixed oversampling, we estimate the required oversampling factor based on the historical ratio of effective groups. This allows us to perform efficient single-round sampling that adapts automatically as the model improves and stabilizes the effective batch size. Formally, we present Predictive Dynamic Sampling with DOT in Algorithm 1 (Appendix B).

5 Experiments

In this section, we introduce the experimental setup and evaluate both the performance and generation efficiency of our method on challenging reasoning benchmarks, with additional analyses and results provided in Appendix C and E.

5.1 Experimental Setup

Datasets. We train our models on the dataset of DeepScaleR-Preview (Luo et al., 2025b) and evaluate on four challenging math benchmarks: AIME-24¹, AIME-25², AMC (AMC-22 and AMC-23)³, and MATH-500 (Hendrycks et al., 2021). Following the official recommendation, we use the prompt template: “Please reason step by step, and put your final answer within `\boxed{}`.” During evaluation, we decode with temperature $t = 0.6$, $top_p = 0.95$ and $top_k = 20$, and set the maximum generation budget to 32,768 tokens to avoid premature truncation. To reduce variance, we sample 32 outputs per problem and report average pass@1 accuracy. We adopt the Qwen-Math (Yang et al., 2024) evaluation tool to extract boxed final

¹HuggingFace dataset: Maxwell-Jia/AIME_2024

²HuggingFace dataset: yentinglin/aime_2025

³HuggingFace dataset: AI-MO/aimo-validation-amc

answers and compute accuracy, and additionally report the average generated length.

Baselines. We conduct experiments with the DeepSeek-R1-Distill-Qwen family as the base model, covering three scales (1.5B, 7B and 32B). We compare DOT against recent methods that target reasoning performance and efficiency, including DeepScaleR-Preview, OverThink, DAST, O1-Pruner, LCR1, Laser, AdaptThink, DLER-R1, and SIRI. For baselines with released checkpoints, we re-evaluate them under our unified protocol (same prompt template, decoding configuration, token budget, and the Qwen-Math evaluation tool) to ensure a fair comparison. Detailed descriptions of these methods are provided in Appendix F.

Implementation Details. We instantiate DOT with a standard GRPO-based RL pipeline, and do not employ auxiliary tricks such as token-level loss or asymmetric clipping (Yu et al., 2025). Rewards are computed using the rule-based math verifier from Luo et al. (2025b). Our implementation is built on verl (Sheng et al., 2025), using FSDP (Zhao et al., 2023) for distributed training and SGLang (Zheng et al., 2024) to serve rollouts efficiently. For policy optimization, following Wen et al. (2025), we sample 32 rollouts per prompt, with a batch size of 128 and a mini-batch size of 32, i.e., the off-policy staleness is 4. Unless otherwise stated, we cap the maximum training-time generation length at 8K tokens (DOT-4K is trained with a stricter 4K-token cap), setting the outlier threshold $\alpha = 3$ in Eq. (5) and the reduction margin $m = 32$ in Eq. (6). Models are trained with a sampling temperature of 1.0 and a learning rate of 1×10^{-6} . For the KL-Cov regularization, we adopt the default parameter setting from Cui et al. (2025). A complete list of training configurations is provided in Appendix G.

5.2 Main Results

Significant Extension of the Pareto Frontier.

As presented in Table 1, DOT consistently pushes the efficiency–performance Pareto frontier outward across varying model scales and benchmarks. In contrast to prior length-penalty methods (e.g., O1-Pruner, Laser) that often trade accuracy for brevity, or threshold-based methods (e.g., SIRI) that necessitate larger token budgets to sustain performance, DOT achieves substantial gains in both dimensions simultaneously. For instance, at the 1.5B scale, DOT-8K improves AIME-24 accuracy from 30.0%

Method	AIME-24		AIME-25		AMC		MATH-500	
	Acc	Length	Acc	Length	Acc	Length	Acc	Length
<i>DeepSeek-R1-Distill-Qwen-1.5B</i>								
<i>Original</i>	30.0	15498	23.5	15604	64.1	10316	84.0	5483
DeepScaleR-Preview (Luo et al., 2025b)	40.3	9430	30.2	9778	73.8	5538	88.9	3102
OverThink* (Chen et al., 2025)	28.3	11269	—	—	—	—	81.2	4131
DAST* (Shen et al., 2025a)	26.9	7745	—	—	—	—	83.0	2428
O1-Pruner* (Luo et al., 2025a)	28.9	10361	—	—	—	—	82.2	3212
LC-R1 (Cheng et al., 2025)	22.9	8000	21.0	7961	60.7	4568	81.8	2362
Laser-DE-L4096 (Liu et al., 2025b)	32.6	8349	23.6	7839	67.5	4994	84.8	2763
AdaptThink (Zhang et al., 2025b)	30.9	7917	23.3	8166	63.0	3710	82.5	1964
DLER-R1 (Liu et al., 2025a)	35.8	<u>3354</u>	25.6	<u>3101</u>	73.5	<u>2544</u>	87.1	1777
SIRI-low* (Wen et al., 2025)	40.4	7093	29.6	6509	74.6	4700	87.7	2881
SIRI-high* (Wen et al., 2025)	43.6	10049	<u>32.2</u>	9739	75.9	7396	88.4	4633
DOT-4K (Ours)	43.1	3342	29.2	2979	<u>77.5</u>	2281	<u>89.2</u>	1249
DOT-8K (Ours)	52.2	5151	34.2	5143	80.6	3140	89.9	<u>1423</u>
<i>DeepSeek-R1-Distill-Qwen-7B</i>								
<i>Original</i>	55.1	13088	39.9	14240	82.5	7668	92.2	4026
DAPO-DeepScaleR	<u>57.6</u>	9983	40.8	10705	84.5	6508	92.5	3658
OverThink* (Chen et al., 2025)	53.1	8744	—	—	—	—	89.4	2435
DAST* (Shen et al., 2025a)	45.6	7578	—	—	—	—	89.6	2162
O1-Pruner* (Luo et al., 2025a)	49.2	9719	—	—	—	—	86.6	2534
LC-R1 (Cheng et al., 2025)	48.5	7580	35.6	7984	79.2	3765	90.1	1536
Laser-DE-L4096 (Liu et al., 2025b)	53.5	5890	37.4	6324	83.0	3381	92.6	1883
AdaptThink (Zhang et al., 2025b)	55.2	10393	38.3	11723	81.5	5177	91.0	2008
DLER-R1 (Liu et al., 2025a)	50.6	<u>3241</u>	33.6	<u>3357</u>	83.5	<u>2262</u>	92.4	1438
SIRI-low* (Wen et al., 2025)	56.1	6122	41.5	6386	85.8	4015	93.5	2452
SIRI-high* (Wen et al., 2025)	57.1	8585	<u>45.4</u>	9106	<u>86.7</u>	5773	<u>93.7</u>	3378
DOT-4K (Ours)	54.8	2958	41.1	2835	86.1	1836	93.4	1008
DOT-8K (Ours)	62.6	4903	48.5	5464	87.6	2779	94.3	<u>1293</u>
<i>DeepSeek-R1-Distill-Qwen-32B</i>								
<i>Original</i>	<u>72.4</u>	10299	<u>56.0</u>	12385	<u>88.9</u>	6578	94.3	3557
Laser-DE-L8192* (Liu et al., 2025b)	70.8	6785	—	—	—	—	93.2	2314
DOT-4K (Ours)	65.3	2622	52.5	2782	87.4	1472	<u>94.5</u>	861
DOT-8K (Ours)	73.2	<u>4151</u>	59.6	<u>5301</u>	90.6	<u>2786</u>	95.0	<u>1369</u>

Table 1: Performance comparison on AIME-24, AIME-25, AMC, and MATH-500 benchmarks. We report pass@1 accuracy (%) and average response length (tokens). (For methods marked with an asterisk, we cite results from Wen et al. (2025) and Liu et al. (2025b) as public checkpoints are unavailable.)

(Original) to 52.2%, while reducing the average response length by over 66%. Notably, concurrent work DLER (Liu et al., 2025a) shares our philosophy of avoiding explicit length-aware reward shaping, thus also achieving extreme length compression. However, by employing a dynamic truncation strategy that eliminates redundancy without stifling the exploration essential for complex reasoning, our DOT-4K achieves equivalent extreme brevity while further lifting accuracy by 7.3%. Moreover, compared with the state-of-the-art method SIRI-high, DOT-8K achieves a 8.6% absolute accuracy gain on AIME-24 while consuming only half the inference budget. This trend holds robustly on the 7B model, where DOT-8K establishes a new state-of-the-art with 62.6% accuracy, surpassing the original model by 7.5% while using only $\sim 37\%$ of the response length. Synthesizing results across all

four benchmarks, DOT successfully realizes adaptive reasoning, i.e., allocating extended computation to challenging problems while aggressively pruning redundancy on simpler ones.

Scalability to Larger Models. The benefits of DOT generalize effectively to the 32B scale. DOT-8K consistently achieves superior accuracy across all benchmarks while reducing token consumption by $\sim 60\%$. This substantial efficiency gain suggests that DOT enables larger models to exhibit a distinct scaling trend in “per-token intelligence”.

Generalization to Out-of-Distribution Domains. DOT demonstrates robust transferability to out-of-distribution code generation tasks. As shown in Table 2, our method consistently outperforms all baselines across both datasets. This indicates that DOT has internalized a universal pattern of concise

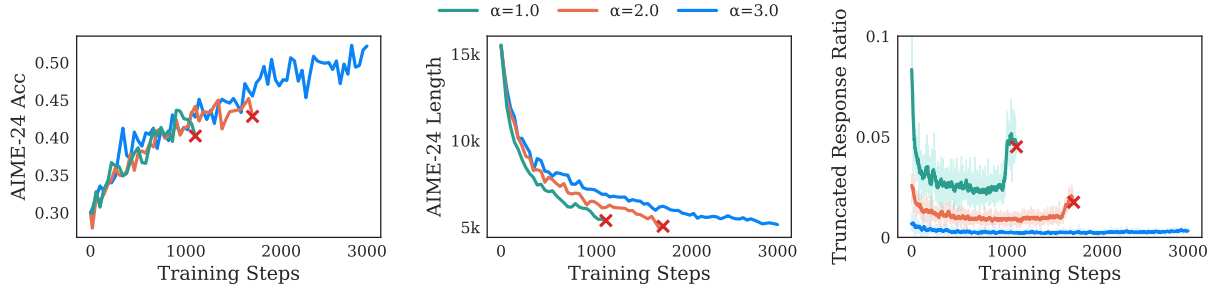


Figure 7: Impact of the hyperparameter of truncation threshold α . We report the curves for pass@1 accuracy, average response length, and the ratio of truncated responses. The ‘X’ markers indicate training collapse.

Method	HumanEval		LiveCodeBench	
	Acc	Length	Acc	Length
<i>DeepSeek-R1-Distill-Qwen-1.5B</i>				
Original	64.7	4377	16.4	13706
DeepScaleR-Preview	69.6	4657	21.0	10076
LC-R1	59.8	2814	15.1	11128
Laser-DE-L4096	64.5	2372	17.5	6223
AdaptThink	64.4	3859	17.7	11117
DLER-R1	68.2	2350	20.8	4132
DOT-4K (Ours)	<u>70.5</u>	2306	<u>21.7</u>	<u>4481</u>
DOT-8K (Ours)	70.7	2860	22.6	6903
<i>DeepSeek-R1-Distill-Qwen-7B</i>				
Original	81.9	3265	31.8	9718
LC-R1	81.2	2173	31.4	6634
Laser-DE-L4096	82.9	2118	<u>33.0</u>	6051
AdaptThink	81.6	2862	32.2	8767
DLER-R1	82.9	2118	<u>33.0</u>	6050
DOT-4K (Ours)	<u>85.0</u>	1474	<u>33.0</u>	3988
DOT-8K (Ours)	85.1	2019	34.8	<u>5979</u>

Table 2: Performance comparison on code generation benchmarks (HumanEval (Chen et al., 2021) and LiveCodeBench(v6) (Jain et al., 2024)).

reasoning, capable of adaptively allocating computational budget commensurate with the intrinsic difficulty of diverse problems.

5.3 Ablation Study

To investigate the contribution of each component in our training recipe, we conduct an ablation study as summarized in Table 3. The most critical finding lies in the effectiveness of Dynamic Outlier Truncation. Removing this mechanism results in significantly longer responses while yielding almost identical accuracy, demonstrating that our method successfully mitigates overthinking while preserving reasoning capabilities. In contrast, omitting Group-Conditional Truncation causes a marked decline in performance, suggesting that applying a uniform threshold is detrimental to solving complex problems that necessitate extended CoTs. Regarding optimization objectives, incorporating the token-level loss into the GRPO leads to increased

Variant	AIME-24		AMC	
	Acc	Length	Acc	Length
DOT-8K	52.2	5151	80.6	3140
w/o Dynamic Outlier Truncation	51.5	6879	81.3	4851
w/o Group-Conditional Truncation	47.8	5071	79.7	3014
w/o Minimum Truncation Margin	50.0	5696	77.3	2902
w/o KL-Cov	47.9	6057	79.3	3856
w/o Predictive Dynamic Sampling	48.9	5637	79.1	3576
w/ Token-Level Loss	49.5	6334	79.4	3964

Table 3: Performance comparison of different variants on DeepSeek-R1-Distill-Qwen-1.5B.

verbosity. This occurs because token-level aggregation inherently assigns larger gradient magnitude to longer responses, thereby exacerbating the model’s tendency towards length inflation. Moreover, the removal of the KL-Cov or Predictive Dynamic Sampling negatively impacts both accuracy and efficiency, verifying their necessity in stabilizing the training process. Furthermore, we examine the hyperparameter of truncation threshold α in Fig 7. While moderate settings ensure stability, overly aggressive thresholds (e.g., $\alpha = 1.0, 2.0$) trigger training collapse. This confirms that a three-sigma deviation serves as a statistically robust boundary for identifying genuine outliers, whereas tighter bounds risk truncating valid reasoning steps essential for convergence.

5.4 Training Dynamics

Fig. 8 shows the training evolution of DOT-8K, illustrating simultaneous gains in both reasoning accuracy and token efficiency.

Decoupling Length from Performance. Our method inverts the detrimental length shift. While benchmark accuracy steadily improves (Fig. 8a, c), response length monotonically decreases (Fig. 8b, d). Rising reward (Fig. 8f) accompanied by reduced length verifies that the policy eliminates redundancy while preserving reasoning capacity.

Systemic Correction via Minimal Intervention. Substantial efficiency gains are achieved through

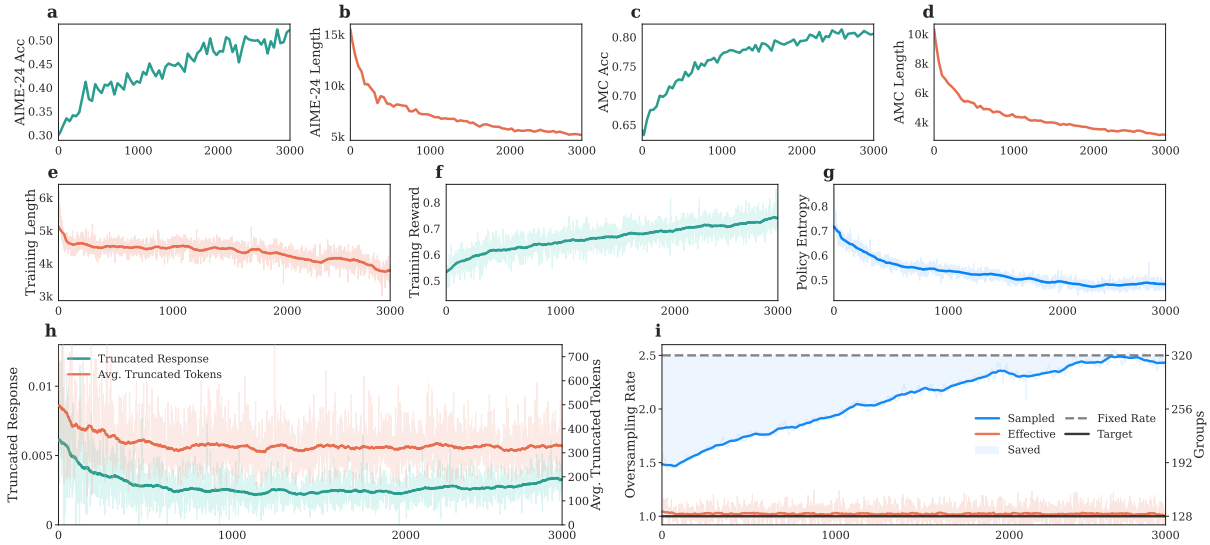


Figure 8: Metric curves monitoring the DOT-8K training process on DeepSeek-R1-Distill-Qwen-1.5B.

minimal interventions. *Fig. 8h* shows that DOT affects $< 0.5\%$ of generated responses, yet this suffices to drive a global length reduction (*Fig. 8e*). By targeting only extreme outlier tails in all-correct groups, DOT corrects the shared policy prior responsible for redundancy rather than merely suppressing local symptoms.

Optimization Stability and Efficiency. Training remains robust against collapse, with policy entropy exhibiting a controlled decline (*Fig. 8g*) to sustain exploration. Furthermore, *Fig. 8i* demonstrates the precision of our predictive dynamic sampling. The effective batch size adheres strictly to the target with negligible variance. By adapting the oversampling rate to the improving pass rate, our predictive dynamic sampling eliminates the computational waste of fixed rate strategies of standard dynamic sampling.

6 Related Work

Recent reasoning models have demonstrated exceptional problem-solving abilities through extended CoT. However, they frequently exhibit the overthinking phenomenon (Chen et al., 2025), incurring high computational costs on simple tasks. While comprehensive literature also covers reasoning compression and adaptive routing (see Appendix I for extended discussion), RL has emerged as one of the most pivotal paradigm for explicitly refining reasoning efficiency.

Early approaches in this domain, such as L1 (Agarwal and Welleck, 2025) and O1-Pruner (Luo et al., 2025a), impose explicit length penalties to harmonize token budgets. To address structural

inefficiencies, LC-R1 (Cheng et al., 2025) employs dual rewards to prune invalid verification steps, while DLER (Liu et al., 2025a) optimizes truncation penalties to prevent training collapse. Recognizing that optimal length varies by complexity, DAST (Shen et al., 2025a) and Laser (Liu et al., 2025b) introduce difficulty-aware shaping, dynamically adjusting penalties to tolerate longer chains for complex queries. Other works explore autonomous mode switching (Zhang et al., 2025b) or interleaved training schedules (Wen et al., 2025) to optimize the trade-off.

In this work, we move beyond generic length penalties to analyze the generative mechanisms driving overthinking, proposing a targeted training recipe that significantly pushes the Pareto frontier of reasoning efficiency.

7 Conclusion

In this paper, we identified length shift as the primary driver of redundancy in reasoning models, revealing how uncertainty-driven exploration on complex problems inadvertently causes verbosity on trivial inputs. To address this, we introduced Dynamic Outlier Truncation (DOT), a training-time intervention that selectively eliminates statistical outliers in all-correct groups. By complementing this mechanism with entropy stabilization and predictive dynamic sampling, DOT achieves a significant reduction in response length while preserving, and often enhancing accuracy. Our results across multiple scales demonstrate that DOT effectively pushes the efficiency–performance Pareto frontier outward, offering a simple, robust and scalable paradigm for training efficient reasoning models.

540 Limitations

541 Despite significantly extending the efficiency-
542 performance Pareto frontier, our study acknowl-
543 edges certain limitations. As a training-time in-
544 tervention rooted in RL, the effectiveness of DOT
545 is inherently bounded by the quality of the train-
546 ing data and the initial policy. Applying DOT
547 to further refine state-of-the-art reasoning models
548 (e.g., DeepSeek-V3.2 (DeepSeek-AI et al., 2025b),
549 Qwen3-235B-A22B-Thinking (Yang et al., 2025))
550 poses challenges, as these models have typically un-
551 dergone extensive post-training on datasets vastly
552 exceeding the scale of open-source counterparts.
553 Consequently, they exhibit extremely low policy en-
554 tropy on open-source datasets, making it nearly im-
555 possible to extract further gains using current pub-
556 lic resources. Nevertheless, given DOT’s simplic-
557 ity and plug-and-play nature, it holds substantial
558 promise for application in the post-training phase
559 of next-generation foundation models, particularly
560 when leveraged with high-quality in-house data or
561 integrated with test-time reinforcement learning
562 strategies (Zuo et al., 2025).

563 Furthermore, our current investigation does not
564 yet encompass agentic tasks. The core principle
565 of length shift likely extends to these scenarios,
566 manifesting as redundant tool invocations or cyclic
567 planning steps. Although we leave this exploration
568 for future work, extending the DOT mechanism
569 to prune redundancy in agent trajectories, thereby
570 optimizing action spaces rather than merely token
571 spaces, represents a critical necessity for building
572 efficient and scalable agents.

573 References

574 Pranjali Aggarwal and Sean Welleck. 2025. L1: Con-
575 trolling how long a reasoning model thinks with rein-
576 forcement learning. *Preprint*, arXiv:2503.04697.

577 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan,
578 Henrique Ponde de Oliveira Pinto, Jared Kaplan,
579 Harri Edwards, Yuri Burda, Nicholas Joseph, Greg
580 Brockman, Alex Ray, Raul Puri, Gretchen Krueger,
581 Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela
582 Mishkin, Brooke Chan, Scott Gray, and 39 others.
583 2021. Evaluating large language models trained on
584 code.

585 Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He,
586 Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi
587 Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang,
588 Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Do
589 not think that much for 2+3=? on the overthinking of
590 o1-like llms. *Preprint*, arXiv:2412.21187.

Zhengxiang Cheng, Dongping Chen, Mingyang Fu,
and Tianyi Zhou. 2025. Optimizing length com-
pression in large reasoning models. *Preprint*,
arXiv:2506.14755.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan,
Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan,
Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng,
Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and
Ning Ding. 2025. The entropy mechanism of rein-
forcement learning for reasoning language models.
Preprint, arXiv:2505.22617.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang,
Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang,
Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhi-
hong Shao, Zhuoshu Li, Ziyi Gao, and 181 others.
2025a. Deepseek-r1: Incentivizing reasoning capa-
bility in llms via reinforcement learning. *Preprint*,
arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin,
Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao
Wu, Bowei Zhang, Chaofan Lin, Chen Dong,
Chengda Lu, Chenggang Zhao, Chengqi Deng, Chen-
hao Xu, Chong Ruan, Damai Dai, Daya Guo, De-
jian Yang, and 245 others. 2025b. Deepseek-v3.2:
Pushing the frontier of open large language models.
Preprint, arXiv:2512.02556.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025.
Thinkless: LLM learns when to think. In *The Thirty-
ninth Annual Conference on Neural Information Pro-
cessing Systems*.

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

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. *Preprint*,
arXiv:2504.01296.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xi-
angyu Zhang, and Heung-Yeung Shum. 2025. Open-
reasoner-zero: An open source approach to scaling up
reinforcement learning on the base model. *Preprint*,
arXiv:2503.24290.

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. *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. Live-
codebench: Holistic and contamination free evalu-
ation of large language models for code. *Preprint*,
arXiv:2403.07974.

649	Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. 2025. Think only when you need with large hybrid-reasoning models . In <i>The Thirty-ninth Annual Conference on Neural Information Processing Systems</i> .	706
650		707
651		
652		
653		
654		
655	Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. 2025. ThinkSwitcher: When to think hard, when to think fast . In <i>Findings of the Association for Computational Linguistics: EMNLP 2025</i> , pages 5185–5201, Suzhou, China. Association for Computational Linguistics.	
656		
657		
658		
659		
660		
661	Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. 2025a. Dler: Doing length penalty right - incentivizing more intelligence per token via reinforcement learning . <i>Preprint</i> , arXiv:2510.15110.	
662		
663		
664		
665		
666		
667		
668	Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. 2025b. Learn to reason efficiently with adaptive length-based reward shaping . <i>Preprint</i> , arXiv:2505.15612.	
669		
670		
671		
672		
673	Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025a. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning . <i>Preprint</i> , arXiv:2501.12570.	
674		
675		
676		
677		
678	Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025b. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl . Notion Blog.	
679		
680		
681		
682		
683	Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. CoT-valve: Length-compressible chain-of-thought tuning . In <i>Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 6025–6035, Vienna, Austria. Association for Computational Linguistics.	
684		
685		
686		
687		
688		
689		
690	OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card . <i>Preprint</i> , arXiv:2412.16720.	
691		
692		
693		
694		
695		
696		
697	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models . <i>Preprint</i> , arXiv:2402.03300.	
698		
699		
700		
701		
702		
703	Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025a. Dast: Difficulty-adaptive slow-thinking for large reasoning models . <i>Preprint</i> , arXiv:2503.04472.	706
704		707
705		
	Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025b. Codi: Compressing chain-of-thought into continuous space via self-distillation . <i>Preprint</i> , arXiv:2502.21074.	708
		709
		710
		711
	Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework . In <i>Proceedings of the Twentieth European Conference on Computer Systems</i> , EuroSys '25, page 1279–1297, New York, NY, USA. Association for Computing Machinery.	712
		713
		714
		715
		716
		717
		718
	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>Preprint</i> , arXiv:2408.03314.	719
		720
		721
		722
	Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, and 77 others. 2025a. Kimi k1.5: Scaling reinforcement learning with llms . <i>Preprint</i> , arXiv:2501.12599.	723
		724
		725
		726
		727
		728
		729
		730
	Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, and 77 others. 2025b. Kimi k1.5: Scaling reinforcement learning with llms . <i>Preprint</i> , arXiv:2501.12599.	731
		732
		733
		734
		735
		736
		737
		738
	Fanqi Wan, Longguang Zhong, Ziyi Yang, Ruijun Chen, and Xiaojun Quan. 2024. Fusechat: Knowledge fusion of chat models . <i>Preprint</i> , arXiv:2408.07990.	739
		740
		741
	Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models . In <i>Advances in Neural Information Processing Systems</i> , volume 35, pages 24824–24837. Curran Associates, Inc.	742
		743
		744
		745
		746
		747
		748
	Haoming Wen, Yushi Bai, Juanzi Li, and Jie Tang. 2025. Siri: Scaling iterative reinforcement learning with interleaved compression . <i>Preprint</i> , arXiv:2509.25176.	749
		750
		751
	Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. TokenSkip: Controllable chain-of-thought compression in LLMs . In <i>Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing</i> , pages 3351–3363, Suzhou, China. Association for Computational Linguistics.	752
		753
		754
		755
		756
		757
	Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less . <i>Preprint</i> , arXiv:2502.18600.	758
		759
		760

761 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,
762 Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,
763 Chengen Huang, Chenxu Lv, Chujie Zheng, Day-
764 iheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao
765 Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41
766 others. 2025. [Qwen3 technical report](#). *Preprint*,
767 arXiv:2505.09388.

768 An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao,
769 Bowen Yu, Chengpeng Li, Dayiheng Liu, Jian-
770 hong Tu, Jingren Zhou, Junyang Lin, Keming Lu,
771 Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang
772 Ren, and Zhenru Zhang. 2024. [Qwen2.5-math tech-
773 nical report: Toward mathematical expert model via
774 self-improvement](#). *Preprint*, arXiv:2409.12122.

775 Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan,
776 Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan,
777 Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin,
778 Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan
779 Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and
780 16 others. 2025. [Dapo: An open-source llm re-
781 inforcement learning system at scale](#). *Preprint*,
782 arXiv:2503.14476.

783 Charlie Zhang, Graham Neubig, and Xiang Yue.
784 2025a. [On the interplay of pre-training, mid-training,
785 and rl on reasoning language models](#). *Preprint*,
786 arXiv:2512.07783.

787 Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and
788 Juanzi Li. 2025b. [Adaptthink: Reasoning models can
789 learn when to think](#). *Preprint*, arXiv:2505.13417.

790 Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo,
791 Chien-Chin Huang, Min Xu, Less Wright, Hamid
792 Shojanazeri, Myle Ott, Sam Shleifer, Alban Des-
793 maison, Can Balioglu, Pritam Damania, Bernard
794 Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Math-
795 ews, and Shen Li. 2023. [Pytorch fsdp: Experi-
796 ences on scaling fully sharded data parallel](#). *Preprint*,
797 arXiv:2304.11277.

798 Yuzhong Zhao, Yue Liu, Junpeng Liu, Jingye Chen, Xun
799 Wu, Yaru Hao, Tengchao Lv, Shaohan Huang, Lei
800 Cui, Qixiang Ye, Fang Wan, and Furu Wei. 2025.
801 [Geometric-mean policy optimization](#). *Preprint*,
802 arXiv:2507.20673.

803 Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui
804 Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong
805 Liu, Rui Men, An Yang, Jingren Zhou, and Jun-
806 yang Lin. 2025. [Group sequence policy optimization](#).
807 *Preprint*, arXiv:2507.18071.

808 Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue
809 Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos
810 Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark
811 Barrett, and Ying Sheng. 2024. [Sglang: Efficient
812 execution of structured language model programs](#).
813 *Preprint*, arXiv:2312.07104.

814 Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu
815 Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xin-
816 wei Long, Ermo Hua, Biqing Qi, Youbang Sun,
817 Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen

Zhou. 2025. [Ttrl: Test-time reinforcement learning](#).
Preprint, arXiv:2504.16084.

818
819

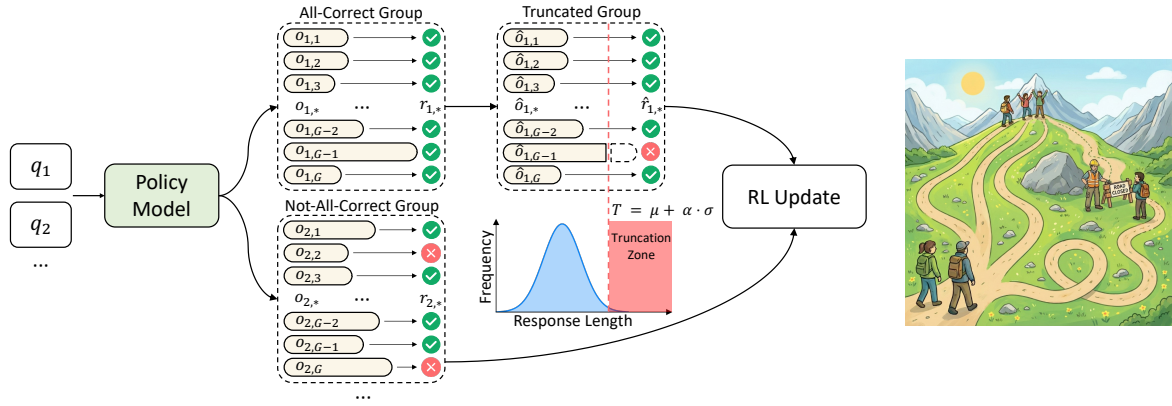


Figure 9: Illustration of Dynamic Outlier Truncation (DOT). By truncating statistical outliers in all-correct groups during GRPO-style RL, DOT creates a negative signal for redundancy without hindering exploration.

Algorithm 1: Predictive Dynamic Sampling (with DOT)

Input: training prompts \mathcal{D} ; target prompt batch size B ; rollout group size G ; history window W

Initialize: history buffer \mathcal{H} storing recent effective group ratio

- 1 **for** step = 1, ..., M **do**
- 2 $\bar{p} \leftarrow \text{mean}(\mathcal{H}), s_p \leftarrow \text{std}(\mathcal{H})$ // effective group ratio stats
- 3 $\gamma \leftarrow \max\left(1.0, \frac{1}{\bar{p}}(1 + s_p)\right)$ // predicted oversampling rate
- 4 Sample a prompt batch $\mathcal{B} \subset \mathcal{D}$ with $|\mathcal{B}| = \lceil \gamma B \rceil$
- 5 Sample G rollouts for each $q \in \mathcal{B}$ and compute rewards
- 6 Apply DOT and recompute rewards
- 7 $\mathcal{B}_{\text{eff}} \leftarrow \{q \in \mathcal{B} \mid \text{std}(\{R_i(q)\}_{i=1}^G) > 0\}$ // keep non-zero std groups
- 8 $\mathcal{U} \leftarrow$ Assemble exactly B groups from \mathcal{B}_{eff} by prompt-level dropping and masked padding
- 9 Execute an RL update on \mathcal{U}
- 10 $p \leftarrow |\mathcal{B}_{\text{eff}}|/|\mathcal{B}|$ // effective group ratio
- 11 $\mathcal{H} \leftarrow \text{Tail}_W(\mathcal{H} \oplus [p])$ // sliding-window update

Output: trained policy model

A Visual Illustration of Dynamic Outlier Truncation

In *Sec. 4*, we introduce Dynamic Outlier Truncation (DOT) as a training-time intervention to curb redundancy. To complement the formal definition, *Fig. 9* provides a schematic overview of the process. It visually demonstrates how DOT selectively identifies and truncates statistical outliers exclusively within "all-correct" rollout groups, thereby creating a precise negative signal for redundancy without hindering the exploration of unsolved queries.

B Formal Description of Algorithms

In *Sec. 4*, we propose Predictive Dynamic Sampling, which estimates the required oversampling factor based on historical pass rates to stabilize the effective batch size. Formally, the detailed algorithm is presented in Algorithm 1.

C Case Study and Detailed Analysis

In this section, we provide both qualitative examples and quantitative statistical analyses to further illustrate the impact of our approach.

Fig. 10 through *15* present a series of side-by-side case studies across multiple benchmarks, comparing the reasoning trajectories of the original long-CoT policy with our DOT-optimized model. These cases consistently demonstrate that while the original model often falls into redundant verification loops and overthinking, our model arrives at the same correct answer through a significantly more concise and purposeful path without sacrificing logical rigor.

As a supplement to these qualitative case studies, *Fig. 16* through *18* provide a macro-statistical perspective on the response length distributions and differences across various model scales and

Method	AIME-24		AIME-25		AMC		MATH-500	
	Acc	Length	Acc	Length	Acc	Length	Acc	Length
DOT-8K	52.2	5151	34.2	5143	80.6	3140	89.4	1423
w/o KL-Cov (Cui et al., 2025)	47.9	6057	33.3	6029	79.3	3856	90.5	1932
w/ GMPO (Zhao et al., 2025)	48.1	5745	34.3	5814	80.5	3550	90.7	1762
w/ GSPO (Zheng et al., 2025)	52.7	5007	37.1	5006	81.2	3163	90.6	1565

Table 4: Performance comparison on DeepSeek-R1-Distill-Qwen-1.5B. We examine the integration of DOT with advanced policy optimization strategies, where KL-Cov is explicitly disabled.

datasets. Our analysis shows that the response length ratio (Original / Ours) remains consistently high across different difficulty levels, confirming that the efficiency gains of DOT are not limited to simple queries but remain effective across the entire complexity spectrum. Furthermore, the scatter plots reveal a strong correlation between the original policy’s response length/variance and the resulting token savings. This indicates that DOT is particularly effective at optimizing policies that are inherently redundant, guiding the model toward a more efficient reasoning distribution during training while maintaining stable and high-quality performance at inference time.

D Orthogonality with Advanced Policy Optimization Algorithms

To investigate the orthogonality of DOT with state-of-the-art policy optimization algorithms, we integrated it with representative algorithmic enhancements (Zhao et al., 2025; Zheng et al., 2025) involving entropy control and importance-sampling estimation. The results in Table 4 demonstrate that DOT, functioning as an intervention on generated rollouts, synergizes effectively with these algorithm-level improvements. Notably, this combination simultaneously amplifies performance and token efficiency, highlighting the substantial potential of DOT to synergize with advanced optimization techniques to drive the next frontier of efficient and scalable reasoning.

E Additional Training Dynamics

Fig. 19 through 23 provide supplementary visualizations of the training evolution across various model scales (1.5B, 7B, 32B) and hyperparameter settings. These results consistently demonstrate that our training recipe remains robust and stable under diverse conditions. Across all settings, the policy successfully decouples reasoning performance from response length, achieving a steady

increase in accuracy while simultaneously eliminating redundant verbosity.

F Detailed Descriptions on Baselines

To comprehensively evaluate the effectiveness of our proposed method, we compare it against a diverse set of state-of-the-art baselines:

- **DeepScaleR-Preview** (Luo et al., 2025b): A strong unconstrained baseline that enhances reasoning capabilities through iterative context lengthening (scaling from 8K to 24K tokens) during reinforcement learning.
- **OverThink** (Chen et al., 2025): A data-centric approach that addresses the “overthinking” phenomenon. It employs efficiency metrics to identify and filter out non-essential steps (such as redundant post-answer verification) to construct concise supervised fine-tuning data.
- **DAST** (Shen et al., 2025a): A difficulty-adaptive framework that introduces a Token Length Budget (TLB) metric. It dynamically adjusts reward shaping during RL to penalize verbosity on simple queries while tolerating longer reasoning chains for complex problems.
- **O1-Pruner** (Luo et al., 2025a): An off-policy optimization method that introduces a length-harmonizing reward. It aligns the student model’s length distribution with a concise reference model while penalizing accuracy degradation to reduce inference overhead.
- **LC-R1** (Cheng et al., 2025): A structural compression framework utilizing a dual-reward mechanism. It combines a global length penalty with a specific “compress reward” to target and prune invalid thinking loops or redundant self-verifications.
- **Laser-DE** (Liu et al., 2025b): A dynamic length-based reward shaping method. The DE variant

931 (Dynamic & Exploration) specifically encour- 975
932 ages exploration by relaxing penalties for incor- 976
933 rect responses while strictly constraining correct 977
934 ones to be concise. 978

- 935 • **AdaptThink** (Zhang et al., 2025b): A mode- 980
936 switching framework that trains the model to ex- 981
937 plicitly select between “Thinking” (long CoT) 982
938 and “NoThinking” (direct answer) modes based 983
939 on the estimated complexity of the input query. 984
- 940 • **DLER-R1** (Liu et al., 2025a): An optimization- 985
941 focused approach that refines simple truncation 986
942 penalties. It mitigates reward collapse issues in 987
943 RL through batch-wise reward normalization and 988
944 dynamic sampling strategies. 989
- 945 • **SIRI** (Wen et al., 2025): A training schedul- 990
946 ing method that alternates between compression 991
947 and expansion phases. We compare against both 992
948 **SIRI-low** and **SIRI-high** checkpoints to eval- 993
949 uate performance at different points along its 994
950 efficiency-accuracy Pareto frontier. 995

951 G Detailed Training Configurations

952 We conduct all experiments on NVIDIA H800 996
953 and H20 GPUs. For 1.5B models, we use either 997
954 16×H800 or 32×H20; for 7B models, we use 998
955 32×H800 or 64×H20; and for 32B models, we 999
956 use 64×H800 or 128×H20. The typical wall-clock 1000
957 training time reported here is measured on H800:
958 approximately 7 days for 1.5B models, and around
959 10 days for both 7B and 32B models; runs on H20
960 are generally slower. The detailed training configu-
961 rations are summarized in Table 5.

962 H List of Reasoning Words

963 In Sec. 3 and Sec. 4, we analyzed the behavioral
964 propensity of the reasoning model to emit specific
965 lexical markers to investigate the length shift phe-
966 nomenon. These markers typically signal the onset
967 of reasoning steps, self-correction, or verification.
968 Table 6 provides the comprehensive list of these
969 reasoning words.

970 I Additional Related Work

971 In addition to the RL-based efficiency methods dis-
972 cussed in the main text, current research primarily
973 addresses reasoning costs through reasoning com-
974 pression and adaptive routing.

Reasoning Compression and Model Merging.

To reduce inference latency, compression methods
transfer reasoning capabilities to compact formats.
Beyond standard distillation, approaches like To-
kenSkip (Xia et al., 2025) and Chain of Draft (Xu
et al., 2025) prune semantic redundancies. Further
pushing compression, CODI (Shen et al., 2025b)
maps explicit reasoning steps into continuous latent
representations. Alternatively, model merging tech-
niques, such as FuseO1 (Wan et al., 2024), Kimi
k1.5 (Team et al., 2025b) and CoT-Valve (Ma et al.,
2025), interpolate weights between reasoning and
non-reasoning models to balance verbosity without
extensive retraining.

Adaptive Routing and Mode Switching.

Dy-
namic routing frameworks allocate compute based
on query complexity. Systems like ThinkSwitcher
(Liang et al., 2025) utilize lightweight classifiers
to toggle between “Fast” and “Slow” thinking
modes. Similarly, Thinkless (Fang et al., 2025)
and LHRMs (Jiang et al., 2025) train models to
autonomously determine the necessity of reasoning
via special control tokens.

J Use of AI Assistants

In this paper, AI assistants are used for literature
retrieval and grammar checking.

Parameter	Value
algorithm.adv_estimator	grpo
actor_rollout_ref.actor.loss_agg_mode	seq-mean-token-mean
actor_rollout_ref.actor.policy_loss.loss_mode	kl_cov
actor_rollout_ref.actor.policy_loss.kl_cov_ratio	0.002
actor_rollout_ref.actor.policy_loss.ppo_kl_coef	1.0
data.train_batch_size	128
actor_rollout_ref.actor.ppo_mini_batch_size	32
actor_rollout_ref.actor.ppo_epochs	1
data.max_prompt_length	4096
data.max_response_length	4096/8192
actor_rollout_ref.actor.optim.lr	1.0×10^{-6}
actor_rollout_ref.rollout.temperature	1.0
actor_rollout_ref.rollout.n	32
actor_rollout_ref.actor.clip_ratio_low	0.2
actor_rollout_ref.actor.clip_ratio_high	0.2

Table 5: Training configuration specified in verl.

Reasoning Words					
aha	all	alternatively	although	analyze	appears
assume	but	check	clearly	compare	confirm
consider	contradiction	correct	define	ensure	even
finally	hence	hmm	hold on	however	if
implies	initially	instead	instead of	invalid	let
maybe	might be	moreover	need to find	next	oh
oops	otherwise	perhaps	reflect	reflection	so
some	split	suggests	then	therefore	this means that
thus	valid	verify	wait	well	

Table 6: The full list of reasoning words used in our empirical analysis.

Question and Ground Truth

Question: There exist real numbers x and y , both greater than 1, such that $\log_x(y^x) = \log_y(x^{4y}) = 10$. Find xy .

Ground Truth:

25

Response from DOT

959 Tokens

Okay, so we have this problem where we need to find real numbers x and y , both greater than 1, such that:

$$\log_x(y^x) = 10 \quad \text{and} \quad \log_y(x^{4y}) = 10$$

We need to find xy .

Hmm. Let me rewrite these logarithms using logarithm properties. First equation: $\log_x(y^x) = 10$. Using the power rule, log base x of (y^x) is $x \cdot \log_x(y)$. So:

$$x \cdot \log_x(y) = 10$$

Similarly, $\log_y(x)$ can be written as $\ln y / \ln x$, so:

$$x \cdot (\ln y / \ln x) = 10 \implies (x / \ln x) \cdot \ln y = 10$$

Let me note that as equation (1).

Now the second equation: $\log_y(x^{4y}) = 10$. Again, using power rule: $4y \cdot \log_y(x) = 10$. Similarly, $\log_y(x)$ is $\ln x / \ln y$, so:

$$4y \cdot (\ln x / \ln y) = 10 \implies (4y / \ln y) \cdot \ln x = 10$$

Let me note that as equation (2).

So now we have two equations:

$$1) (x / \ln x) \cdot \ln y = 10 \quad 2) (4y / \ln y) \cdot \ln x = 10$$

From equation (1): $\ln y = (10 \cdot \ln x) / x$. Let me call this equation (1a). From equation (2): $(4y / \ln y) \cdot \ln x = 10$.

Let me rearrange equation (2):

$$4y \cdot (\ln x / \ln y) = 10$$

But notice that $(\ln x / \ln y)$ is equal to $\log_y x$, which is reciprocal of $\log_x y$. Wait, maybe substitute $\ln y$ from equation (1a). From equation (1a): $\ln y = (10 \ln x) / x$. Therefore, $\ln y = (10/x) \cdot \ln x$.

So, let's compute $\ln x / \ln y$:

$$\frac{\ln x}{\ln y} = \frac{\ln x}{[(10/x) \cdot \ln x]} = \frac{\ln x}{(10 \ln x) / x} = \ln x \cdot \frac{x}{10 \ln x} = \frac{x}{10}$$

So, $\ln x / \ln y = x/10$.

Therefore, equation (2): $4y \cdot (\ln x / \ln y) = 10$. Substitute $\ln x / \ln y = x/10$:

$$4y \cdot (x/10) = 10$$

So: $(4xy)/10 = 10 \implies 4xy = 100 \implies xy = 25$. Therefore, the answer is 25.

25

Response from R1-Distill-Qwen-1.5B 22514 Tokens

Alright, so I have this problem here... I remember that logarithms can be rewritten using the power rule. Starting with the first equation: $\log_x(y^x) = 10 \implies x \cdot \log_x(y) = 10$. Similarly, the second equation: $\log_y(x^{4y}) = 10 \implies 4y \cdot \log_y(x) = 10$.

Let me denote $\log_x(y) = a$. Then, $\log_y(x) = 1/a$. So, substituting back into the equations:

$$1) x \cdot a = 10 \implies x = 10/a$$

$$2) 4y \cdot (1/a) = 10 \implies 4y = 10a \implies y = 5a/2$$

Now, I need another equation to relate x and y . I know that $a = \log_x(y)$. So, $x^a = y$. Substituting x and y :

$$(10/a)^a = 5a/2$$

Let me take natural logarithm on both sides:

$$a \cdot \ln(10/a) = \ln(5a/2)$$

$$\implies a \ln(10) - a \ln(a) = \ln(5) + \ln(a) - \ln(2)$$

... (omitted: attempting to solve the transcendental equation numerically using Newton-Raphson)

... (omitted: trial and error with integer values $t = 1, 2, 3$ leading to contradictions)

Wait, let me think differently. From the first equation, $\log_x(y^x) = 10 \implies x \cdot (\ln y / \ln x) = 10$. From the second equation, $\log_y(x^{4y}) = 10 \implies 4y \cdot (\ln x / \ln y) = 10$.

Let me denote $u = \ln x, v = \ln y$. 1. $x \cdot (v/u) = 10 \implies e^u \cdot (v/u) = 10$ 2. $4y \cdot (u/v) = 10 \implies 4e^v \cdot (u/v) = 10$

From (1): $e^u = 10u/v$. From (2): $e^v = 10v/(4u) = 5v/(2u)$. Let's compute e^u/e^v :

$$\frac{e^u}{e^v} = \frac{10u/v}{5v/2u} = \frac{10u}{v} \cdot \frac{2u}{5v} = \frac{20u^2}{5v^2} = \frac{4u^2}{v^2}$$

So $e^{u-v} = 4(u/v)^2$. Let $k = u/v$. Then $u = kv$. $e^{kv-v} = 4k^2 \implies e^{v(k-1)} = 4k^2$.

... (omitted: substitution $u = kv$ and solving the resulting exponential equation for k)

Wait, let's look at the equations again. $x \cdot \log_x y = 10$ $4y \cdot \log_y x = 10$ Multiply them:

$$(x \log_x y) \cdot (4y \log_y x) = 10 \cdot 10$$

$$4xy(\log_x y \cdot \log_y x) = 100$$

We know that $\log_x y \cdot \log_y x = 1$. So:

$$4xy \cdot 1 = 100 \implies 4xy = 100 \implies xy = 25$$

Let me verify this. If $xy = 25$, then $y = 25/x$. Substitute into first equation:

$$x \log_x(25/x) = 10 \implies x(\log_x 25 - 1) = 10$$

$$\log_x(25^x) = x + 10 \implies 25^x = x^{x+10}$$

This seems solvable. Let's try $x = 5$. $25^5 = (5^2)^5 = 5^{10}$. RHS: $5^{5+10} = 5^{15}$. $5^{10} \neq 5^{15}$. So $x = 5$ is not the solution.

... (omitted: extensive re-verification with $x = 10, 100, 32$ and checking constraints)

Let's re-solve $x \log_x y = 10$ and $4y \log_y x = 10$. We found $xy = 25$. Let's check if there exist such x, y . $x(\ln y / \ln x) = 10 \implies \ln y = (10 \ln x) / x$. We also have $y = 25/x$. Take \ln : $\ln y = \ln 25 - \ln x$. So $\frac{10 \ln x}{x} = \ln 25 - \ln x$. $(10/x + 1) \ln x = \ln 25 \implies \frac{10+x}{x} \ln x = \ln 25$.

Let $f(x) = \frac{10+x}{x} \ln x - \ln 25$. $f(1) < 0$. $f(5) = 3 \ln 5 - 2 \ln 5 = \ln 5 > 0$. So there is a solution between 1 and 5.

So yes, real numbers x and y exist. The value required is xy . Calculation: $4xy = 100 \implies xy = 25$.

25

Figure 10: Case study from AIME-24.

Question and Ground Truth

Question: Sixteen chairs are arranged in a row. Eight people each select a chair in which to sit so that no person sits next to two other people. Let N be the number of subsets of the 16 chairs that could be selected. Find the remainder when N is divided by 1000.

Ground Truth:

907

Response from DOT

1695 Tokens

Okay, so I have this problem with 16 chairs arranged in a row. Eight people need to sit such that no one is next to two others. Hmm, so that means no three people are sitting consecutively? Yes, if three people sit in chairs 1, 2, 3, then the person in chair 2 is next to two others. So the condition is equivalent to selecting 8 chairs such that no three are consecutive.

This is a combinatorial problem where we need to count the number of binary strings of length 16 with exactly 8 ones, such that there are no three consecutive ones. We can model this using blocks. Let's denote the number of blocks as m . Each block is either size 1 or 2 (since size 3 is forbidden). Let t be the number of blocks of size 1 and s be the number of blocks of size 2. Total ones: $t \cdot 1 + s \cdot 2 = 8$. Total blocks: $t + s$.

Between these blocks, we need at least one zero separating them. Total zeros: $16 - 8 = 8$. Zeros used as separators: $(t + s) - 1$. Remaining zeros: $8 - (t + s - 1) = 9 - (t + s)$. We distribute these remaining zeros into $t + s + 1$ positions (before, between, after blocks). Using stars and bars, the number of ways is:

$$\binom{(9 - (t + s)) + (t + s + 1) - 1}{(t + s + 1) - 1} = \binom{9}{t + s}$$

Now we iterate over possible values of s (blocks of size 2): Since $2s \leq 8$, s can range from 0 to 4. For each s , $t = 8 - 2s$. Number of ways to arrange the blocks themselves: $\binom{t+s}{s} = \binom{8-s}{s}$. Total ways for a fixed s : $\binom{8-s}{s} \cdot \binom{9}{8-s}$.

Let's calculate the sum for $s = 0$ to 4: $s = 0$: $\binom{8}{0} \binom{9}{8} = 1 \cdot 9 = 9$. $s = 1$: $\binom{7}{1} \binom{9}{7} = 7 \cdot 36 = 252$. $s = 2$: $\binom{6}{2} \binom{9}{6} = 15 \cdot 84 = 1260$. $s = 3$: $\binom{5}{3} \binom{9}{5} = 10 \cdot 126 = 1260$. $s = 4$: $\binom{4}{4} \binom{9}{4} = 1 \cdot 126 = 126$.

Total sum: $9 + 252 + 1260 + 1260 + 126 = 2907$. The remainder modulo 1000 is 907.

907

Response from R1-Distill-Qwen-32B 17061 Tokens

Alright, so I have this problem where there are 16 chairs arranged in a row... The condition is that no person is sitting next to two other people. That means no three people can be sitting consecutively. So, the problem reduces to finding the number of ways to choose 8 chairs out of 16 such that no three are consecutive.

Let's define a recursive function $f(n, k)$ as the number of ways to choose k chairs from n with no three consecutive. The recurrence relation is:

$$f(n, k) = f(n - 1, k) + f(n - 2, k - 1) + f(n - 3, k - 2)$$

This seems correct. I need to compute $f(16, 8)$.

... (omitted: lengthy manual computation of DP table for $n=1$ to 10, checking base cases and encountering index errors)

Wait, let me think about another approach. Perhaps using generating functions? The generating function for binary strings with no "111" pattern is related to $\frac{1}{1-x-x^2}$. But here we have a fixed number of ones (8) and a fixed length (16). Maybe the generating function $G(n) = 1 + x + x^2$ corresponds to coefficients? No, that's for compositions. The number of such subsets is the coefficient of x^8 in some polynomial related to length 16.

... (omitted: exploring the generating function $G(n) = G(n - 1) + xG(n - 2) + x^2G(n - 3)$ and attempting to calculate coefficients)

Let's try the Combinations with Blocks approach again. This seems more robust. Let's denote the number of blocks as m . Each block is either size 1 or 2. Let t be the number of blocks of size 2. Then $m - t$ is blocks of size 1. Total people: $2t + (m - t) = t + m = 8 \implies m = 8 - t$. Wait, t is number of size 2 blocks. So t can range from 0 to 4. For each t , the total number of blocks is $8 - t$.

Number of ways to arrange the blocks (order of 1s and 2s):

$$\binom{\text{total blocks}}{\text{blocks of size 2}} = \binom{8-t}{t}$$

Now we need to place these $8 - t$ blocks into the 16 chairs with at least one empty chair between them. Let x_0, x_1, \dots, x_{8-t} be the number of empty chairs in the gaps. Total empty chairs is 8. Constraint: $x_i \geq 1$ for the internal gaps $1 \leq i \leq 8 - t - 1$. Let $y_i = x_i - 1$ for these internal gaps. The equation becomes a stars and bars problem. Total sum of empty chairs available to distribute freely:

$$8 - (\text{number of internal gaps}) = 8 - (8 - t - 1) = t + 1$$

The number of bins is $(8 - t) + 1 = 9 - t$. So we are distributing $t + 1$ items into $9 - t$ bins. Number of solutions is $\binom{(t+1) + (9-t) - 1}{(9-t) - 1} = \binom{9}{8-t}$.

So total ways for a fixed t is $\binom{8-t}{t} \binom{9}{8-t}$. Let's sum over $t = 0$ to 4.

... (omitted: calculation of each term: $t=0 \rightarrow 9$, $t=1 \rightarrow 252$, $t=2 \rightarrow 1260$, $t=3 \rightarrow 1260$, $t=4 \rightarrow 126$)

Summing them up: 2907 . Total $N = 2907$. Find remainder when divided by 1000: $2907 \pmod{1000} = 907$.

Let me double check the DP calculation for a small case, say $n=4, k=2$. Using blocks: $t=0$ (blocks of 1,1): $\binom{2}{0} \binom{3}{2} = 3$. $t=1$ (block of 2): $\binom{1}{1} \binom{3}{1} = 3$. Total = 6. Matches manual count: 1,3, 1,4, 2,4, 1,2, 2,3, 3,4. Wait, 1,2 valid? Yes.

So the formula holds. Final answer is 907.

907

Figure 11: Case study from AIME-25.

Question and Ground Truth

Question: The infinite product

$$\sqrt[3]{10} \cdot \sqrt[3]{\sqrt[3]{10}} \cdot \sqrt[3]{\sqrt[3]{\sqrt[3]{10}}} \dots$$

evaluates to a real number \sqrt{m} . What is the value of m ?

Ground Truth:

10.0

Response from DOT

464 Tokens

Okay, so I need to evaluate this infinite product:

$$\sqrt[3]{10} \cdot \sqrt[3]{\sqrt[3]{10}} \cdot \sqrt[3]{\sqrt[3]{\sqrt[3]{10}}} \dots$$

And it's supposed to equal \sqrt{m} . I have to find m .

Let me try to understand the structure of this product. First, each term is a cube root of the previous term. Let's write the terms out: First term:

$$\sqrt[3]{10} = 10^{1/3} \text{ Second term: } \sqrt[3]{\sqrt[3]{10}} = (10^{1/3})^{1/3} = 10^{1/9}$$

$$\text{Third term: } \sqrt[3]{\sqrt[3]{\sqrt[3]{10}}} = (10^{1/9})^{1/3} = 10^{1/27}$$

So in general, the n -th term is $10^{1/3^n}$. Therefore, the infinite product is:

$$\prod_{n=1}^{\infty} 10^{1/3^n} = 10^{\sum_{n=1}^{\infty} (1/3)^n}$$

Because when you multiply exponents with the same base, you add the exponents.

So sum of exponents is $\sum_{n=1}^{\infty} (1/3)^n$. That's a geometric series with first term $a = 1/3$ and common ratio $r = 1/3$. Sum $S = \frac{a}{1-r} = \frac{1/3}{1-1/3} = \frac{1/3}{2/3} = \frac{1}{2}$.

Therefore, the product is $10^{1/2} = \sqrt{10}$. But wait, the problem says it evaluates to \sqrt{m} . So $\sqrt{m} = \sqrt{10}$, hence $m = 10$.

Wait, let me double-check. Yes, the exponents add up to $1/2$, so $10^{1/2}$ is $\sqrt{10}$, so m is 10.

10

Response from R1-Distill-Qwen-1.5B 23454 Tokens

Okay, so I have this infinite product to evaluate... It's written as:

$$\sqrt[3]{10} \cdot \sqrt[3]{\sqrt[3]{10}} \cdot \sqrt[3]{\sqrt[3]{\sqrt[3]{10}}} \dots$$

And it's supposed to evaluate to a real number which is the square root of some number m .

First, let's try to understand the structure. Each term is a cube root of the previous term. So, if I write out the first few terms, it's: $10^{1/3} \cdot 10^{1/9} \cdot 10^{1/27} \cdot 10^{1/81} \dots$. The exponents are $1/3, 1/9, 1/27, 1/81$, etc. The sum of the exponents is the sum of the infinite geometric series with $a = 1/3$ and $r = 1/3$. $S = \frac{1/3}{1-1/3} = \frac{1/3}{2/3} = \frac{1}{2}$. Therefore, the entire infinite product is $10^{1/2} = \sqrt{10}$. So $\sqrt{m} = \sqrt{10} \implies m = 10$.

Wait, that seems too simple. Let me double-check. Let me compute the partial products numerically to see. $P_1 = 10^{1/3} \approx 2.154$. $P_2 = 10^{1/3} \cdot 10^{1/9} = 10^{4/9} \approx 2.783$. Wait, $4/9 \approx 0.444$ which is less than 0.5.

... (omitted: detailed numerical computation of P3, P4, P5 finding P5 approx 2.887)

Wait, so $P_5 \approx 2.887$, which is less than $\sqrt{10} \approx 3.162$. My partial products are approaching 2.8 something, not 3.16. Why? Let me compute more accurately.

... (omitted: re-calculating P1 to P6 with more precision, still finding discrepancy)

So the partial products are approaching approximately 2.815. This is less than $\sqrt{10}$. Maybe my exponent series is wrong? The exponents are $1/3, 1/9, 1/27, \dots$. Sum is $1/2$. So theoretically it is $10^{1/2}$. But numerically, it looks like it's converging to something else.

... (omitted: questioning the geometric series sum formula and re-deriving it)

... (omitted: calculating P7, P8... P12, finding values slowly creeping up to 2.963)

$P_{12} \approx 2.9632$. Still less than 3.162. This is confusing. The math says $\sqrt{10}$, the numbers say 2.96. Maybe I need to compute even more terms.

... (omitted: computing up to P24, P30, P41, finding values like 2.9632263)

So up to P_{41} , the product is approximately 2.963226. Wait, $10^{1/2} \approx 3.162$. Why is there such a gap? Is it possible the limit is not $\sqrt{10}$?

Let's rethink. $P = \prod_{n=1}^{\infty} 10^{1/3^n} = 10^{\sum 1/3^n}$. Sum is $1/2$. So $P = 10^{1/2}$. Why do my numbers disagree? Maybe my calculator precision is the issue? Or maybe convergence is very slow? Actually, let's look at the partial sum of exponents. $S_n = \sum_{k=1}^n \frac{1}{3^k} = \frac{1}{2} (1 - \frac{1}{3^n})$.

So $P_n = 10^{0.5(1-3^{-n})}$. As $n \rightarrow \infty$, $3^{-n} \rightarrow 0$, so $P_n \rightarrow 10^{0.5}$.

Let's check $n = 2$: $P_2 = 10^{0.5(1-1/9)} = 10^{0.5(8/9)} = 10^{4/9}$. $4/9 \approx 0.444$. $10^{0.444} \approx 2.78$. Correct. $n = 10$: $10^{0.5(1-1/59049)} \approx 10^{0.5} \approx 3.162$. Wait, earlier I calculated $P_{10} \approx 2.96$. Why? Ah, I must have been accumulating rounding errors or multiplying wrong terms in my manual simulation.

The theoretical derivation is solid. The sum of the geometric series is $1/2$. The product must be $10^{1/2}$. The numerical confusion is likely due to calculation errors or slow convergence interpretation. The problem states it evaluates to \sqrt{m} . So $\sqrt{m} = \sqrt{10} \implies m = 10$.

10

Figure 12: Case study from AMC.

Question and Ground Truth

Question: Compute 997^{-1} modulo 1000. Express your answer as an integer from 0 to 999.

Ground Truth:

333

Response from DOT 547 Tokens

Okay, so I need to find the inverse of 997 modulo 1000. That means I need to find an integer x such that $997 \cdot x \equiv 1 \pmod{1000}$. Hmm, so $997x - 1000y = 1$ for some integer y . This is a linear Diophantine equation.

Maybe I can use the Extended Euclidean Algorithm. Let's see. First, let's compute $\gcd(997, 1000)$. 1000 divided by 997 is 1 with a remainder of 3. So $1000 = 997 \cdot 1 + 3$. Then, 997 divided by 3 is 332 with a remainder of 1. $997 = 3 \cdot 332 + 1$. Then, 3 divided by 1 is 3 with remainder 0. So gcd is 1, which means inverse exists.

Now, to backtrack. $1 = 997 - 3 \cdot 332$. But $3 = 1000 - 997 \cdot 1$. Substitute:

$$\begin{aligned} 1 &= 997 - (1000 - 997 \cdot 1) \cdot 332 \\ &= 997 - 1000 \cdot 332 + 997 \cdot 332 \\ &= 997 \cdot (1 + 332) - 1000 \cdot 332 \\ &= 997 \cdot 333 - 1000 \cdot 332. \end{aligned}$$

So, $1 = 997 \cdot 333 - 1000 \cdot 332$. Therefore, $997 \cdot 333 \equiv 1 \pmod{1000}$. So inverse is 333.

333

Response from R1-Distill-Qwen-32B 18984 Tokens

Alright, so I need to find the modular inverse of 997 modulo 1000. Hmm, that means I need to find an integer x such that $997x \equiv 1 \pmod{1000}$.

First, check if 997 and 1000 are coprime. 997 is prime, and 1000 is divisible by 2 and 5. So yes, $\gcd(997, 1000) = 1$. The inverse exists.

Method 1: Extended Euclidean Algorithm.

$$\begin{aligned} 1000 &= 997 \cdot 1 + 3 \\ 997 &= 3 \cdot 332 + 1 \\ 3 &= 1 \cdot 3 + 0 \end{aligned}$$

Working backwards:

$$\begin{aligned} 1 &= 997 - 3 \cdot 332 \\ &= 997 - (1000 - 997 \cdot 1) \cdot 332 \\ &= 997 \cdot 333 - 1000 \cdot 332 \end{aligned}$$

So $x = 333$. Let's verify: $997 \cdot 333 = 332001 \equiv 1 \pmod{1000}$. Correct.

Method 2: Solving Congruence directly. $997 \equiv -3 \pmod{1000}$. So $-3x \equiv 1 \pmod{1000}$. $3x \equiv -1 \equiv 999 \pmod{1000}$. Divide by 3: $x = 333$. Same result.

Method 3: Euler's Theorem. $\phi(1000) = 1000(1 - 1/2)(1 - 1/5) = 400$. So $997^{400} \equiv 1 \pmod{1000}$. Inverse is $997^{399} \pmod{1000}$. This seems tedious to compute manually.

Let's try to compute $3^{399} \pmod{1000}$ using exponentiation by squaring. $399 = 256 + 128 + 8 + 4 + 2 + 1$. Let's calculate powers of 3:

$$\begin{aligned} 3^1 &= 3 \\ 3^2 &= 9 \\ 3^4 &= 81 \\ 3^8 &= 6561 \equiv 561 \\ 3^{16} &\equiv 561^2 \equiv 314721 \equiv 721 \\ 3^{32} &\equiv 721^2 \equiv 519841 \equiv 841 \\ 3^{64} &\equiv 841^2 \equiv 707281 \equiv 281 \\ 3^{128} &\equiv 281^2 \equiv 78961 \equiv 961 \\ 3^{256} &\equiv 961^2 \equiv 923521 \equiv 521 \end{aligned}$$

Now multiply terms for 399: $521 \cdot 961 \equiv 500681 \equiv 681$, $681 \cdot 561 \equiv 382041 \equiv 41$, $41 \cdot 81 \equiv 3321 \equiv 321$, $321 \cdot 9 \equiv 2889 \equiv 889$, $889 \cdot 3 \equiv 2667 \equiv 667$. So $3^{399} \equiv 667$. Since $997 \equiv -3$, $997^{399} \equiv (-3)^{399} \equiv -667 \equiv 333$. Matches.

... (omitted: attempting method 4: trial multiplication $997 \cdot 1$, $997 \cdot 2$... inefficient)

Method 5: Inverse of 3. We need inverse of -3. Inverse of 3 is 667 (since $3 \cdot 667 = 2001 \equiv 1$). So inverse of -3 is $-667 \equiv 333$.

All methods confirm the result is 333.

333

Figure 13: Case study from MATH-500.

Question and Examples

Prompt: Given a list of numbers, return the sum of squares of the numbers in the list that are odd. Ignore numbers that are negative or not integers.

Examples:

- '[1, 3, 2, 0]' → $1^2 + 3^2 = 10$
- '[-1, -2, 0]' → 0
- '[9, -2]' → 81
- '[0]' → 0

Response from DOT 546 Tokens

Okay, I need to write a Python function called `double_the_difference`. Hmm, wait, looking at the examples, the function's name might be a bit misleading because the docstring says it's the sum of squares of odd numbers, ignoring negatives and non-integers.

So the steps are: for each number in the list, check if it's an integer and not negative. Then, check if it's odd. If all conditions met, square it and add to sum.

First, iterate through each element in `lst`. For each element, check if it's an integer. Wait, but in Python, how to determine if it's an integer. Because for example, `3.0` is technically a float but represents integer. But the problem says to ignore numbers that are not integers. So perhaps, the element must be an instance of `int`.

So the condition is: type must be `int`, and the number must be `>= 0`, and odd. So for each `num` in `lst`: if `type(num) is int and num >= 0 and num % 2 != 0`: then square it and add to sum.

Solution Code:

```
def double_the_difference(lst):
    total = 0
    for num in lst:
        if isinstance(num, int) and num >= 0 and num % 2 == 1:
            total += num ** 2
    return total
```

This handles the examples: `[1,3,2,0]` -> 1 is odd (1), 3 is odd (9), 2 even, 0 even -> 10. Correct. `[-1, -2, 0]` -> negative/even -> 0. Correct. The logic holds.

Response from R1-Distill-Qwen-32B 5486 Tokens

Okay, I need to write a Python function called `double_the_difference`. Let me break it down. For each number in the list, I have to check two things: is it a positive integer, and is it odd.

... (omitted: checking simple cases like `[1,3,2,0]` and confirming odd/even logic)

The problem says to ignore numbers that are not integers. Question: Is a float like `3.0` considered an integer? The examples only include pure integers. If `3.0` is passed, `isinstance(3.0, int)` is False. If I use `num.is_integer()`, it is True. Which one does the problem want?

... (omitted: debating the ambiguity of "not integers" in Python context)

Let's assume strict type checking. Only `int` is allowed.

Now, what about Booleans? In Python, `True` and `False` are instances of `int`. `isinstance(True, int)` returns `True`. If the list contains `True`, it acts like 1. 1 is positive and odd. So `True` would be squared and added (1).

... (omitted: testing code snippet with `isinstance` against boolean inputs)

Is this desired behavior? The input is a "list of numbers". Booleans are technically numbers in Python, but semantically usually distinct. If I want to exclude Booleans, I must check the type explicitly.

Let's compare:

- `isinstance(x, int)`: Accepts `True, False`.
- `type(x) is int`: Rejects `True, False`.

... (omitted: considering if the problem implies strict number types only)

I think strict type checking `type(num) is int` is the safest bet to avoid edge cases like Booleans or subclasses of `int` that might behave weirdly.

So the plan is: 1. Iterate over `lst`. 2. Check `type(num) is int`. 3. Check `num > 0`. 4. Check `num % 2 != 0`.

... (omitted: verifying the negative number logic and zero handling)

Let's double check example 3: `[9, -2]`. 9: `int`, positive, odd. Add 81. -2: `int`, not positive. Skip. Result 81. Matches.

... (omitted: verifying empty list behavior)

Final check on the implementation details.

Solution Code:

```
def double_the_difference(lst):
    total = 0
    for num in lst:
        # Strict type check to exclude bools
        if type(num) is int:
            if num > 0 and num % 2 != 0:
                total += num ** 2
    return total
```

Figure 14: Case study from HumanEval.

Question and Examples

Problem: There are N beans, each with deliciousness A_i and color C_i . You choose one color, then eat one bean of that color. The deliciousness you get is the minimum among all beans of that color. Find the color that maximizes this minimum value.

Input: N pairs of (A_i, C_i) . **Output:** The maximum possible value of the minimum deliciousness.

Response from DOT

999 Tokens

Okay, so I have this problem to solve. Let's see what it's about. Hmm. The problem is about choosing a color to eat a bean. Each bean has a deliciousness A_i and color C_i . When you choose a color, you eat one bean of that color. But since multiple beans can have the same color, we have to pick the optimal color that maximizes the minimum deliciousness. So the goal is to select a color such that when you take all beans of that color, the minimum A_i among them is as large as possible. Then, the answer is the maximum of these minima.

So the approach is: 1. Group the beans by their color. 2. For each color group, find the minimum A_i in that group. 3. Find the maximum value among these minima.

That's straightforward.

Code Implementation:

```
n = int(input())
from collections import defaultdict
groups = defaultdict(list)
for _ in range(n):
    a, c = map(int, input().split())
    groups[c].append(a)
```

```
min_per_color = [min(group) for group in groups.values()]
max_min = max(min_per_color)
print(max_min)
```

Testing with Sample 1: Color 1: 100, 40 -> min 40. Color 5: 20, 30 -> min 20. Max of (40, 20) is 40. Correct.

This approach efficiently groups the beans and computes the required values in linear time.

Response from R1-Distill-Qwen-7B 12018 Tokens

Okay, so I've got this problem to solve. Let's see what it's about. The problem is about selecting a color of beans to maximize the minimum deliciousness. For each color, we look at all the beans of that color. The minimum deliciousness among those beans is what contributes to the overall maximum.

... (omitted: initial confusion about binary search and threshold values)

Maybe the optimal is somewhere in between. Perhaps we can think of this as trying to find a threshold value x such that there exists at least one color group with all beans having $A_i \geq x$.

Wait, maybe another approach. Let's collect all the colors, and for each color, compute the minimum A_i among its beans. Then, the answer is the maximum of these minima. Let's check Sample 2 with this logic. The minima are 68, 5, 2, 4, 4, 5, 2, 1, 3. The maximum is 68. But the sample output is 35. So that approach is incorrect.

... (omitted: lengthy confusion about why Sample 2 output is 35, suspecting the max-of-min strategy is wrong)

Wait, I might be misunderstanding the problem. Let me re-read Sample 2 carefully. Colors and values: 68 (3), 17 (2), 99 (2), 92 (4), 82 (4), 10 (3), 100 (2), 78 (1), 3 (1), 35 (4).

Let's group them properly: Color 3: 68, 10 -> Min is 10. Color 2: 17, 99, 100 -> Min is 17. Color 4: 92, 82, 35 -> Min is 35. Color 1: 78, 3 -> Min is 3.

Minima list: 10, 17, 35, 3. Maximum of these is 35.

Ah! My manual calculation earlier was wrong. I thought the minima were much different. Wait, previously I listed minima as "68, 5, 2..." where did those come from? I must have misread the input format or mixed up lines.

... (omitted: re-verifying the logic multiple times, still doubting the simple solution because of previous calculation errors)

So, let's confirm the logic again. The problem asks to maximize the minimum possible deliciousness. 1. Group by color. 2. Find min deliciousness for each group. 3. Find max of those minimums.

In Sample 2: - Color 3 group: 68, 10. Min = 10. - Color 2 group: 17, 99, 100. Min = 17. - Color 4 group: 92, 82, 35. Min = 35. - Color 1 group: 78, 3. Min = 3.

Max(10, 17, 35, 3) = 35. This matches the sample output perfectly.

So the initial "simple" approach was correct all along, I just failed to parse the sample input correctly in my head. The confusion came from misinterpreting the input lines or maybe grouping incorrectly in the thought process.

... (omitted: final code construction using dictionary)

Solution seems robust. **Solution Code:**

```
import sys
from collections import defaultdict

def main():
    n = int(sys.stdin.readline())
    colors = defaultdict(list)
    for _ in range(n):
        a, c = map(int, sys.stdin.readline().split())
        colors[c].append(a)

    max_min = 0
    for c in colors:
        min_a = min(colors[c])
        if min_a > max_min:
            max_min = min_a
    print(max_min)

if __name__ == "__main__":
    main()
```

Figure 15: Case study from LiveCodeBench.

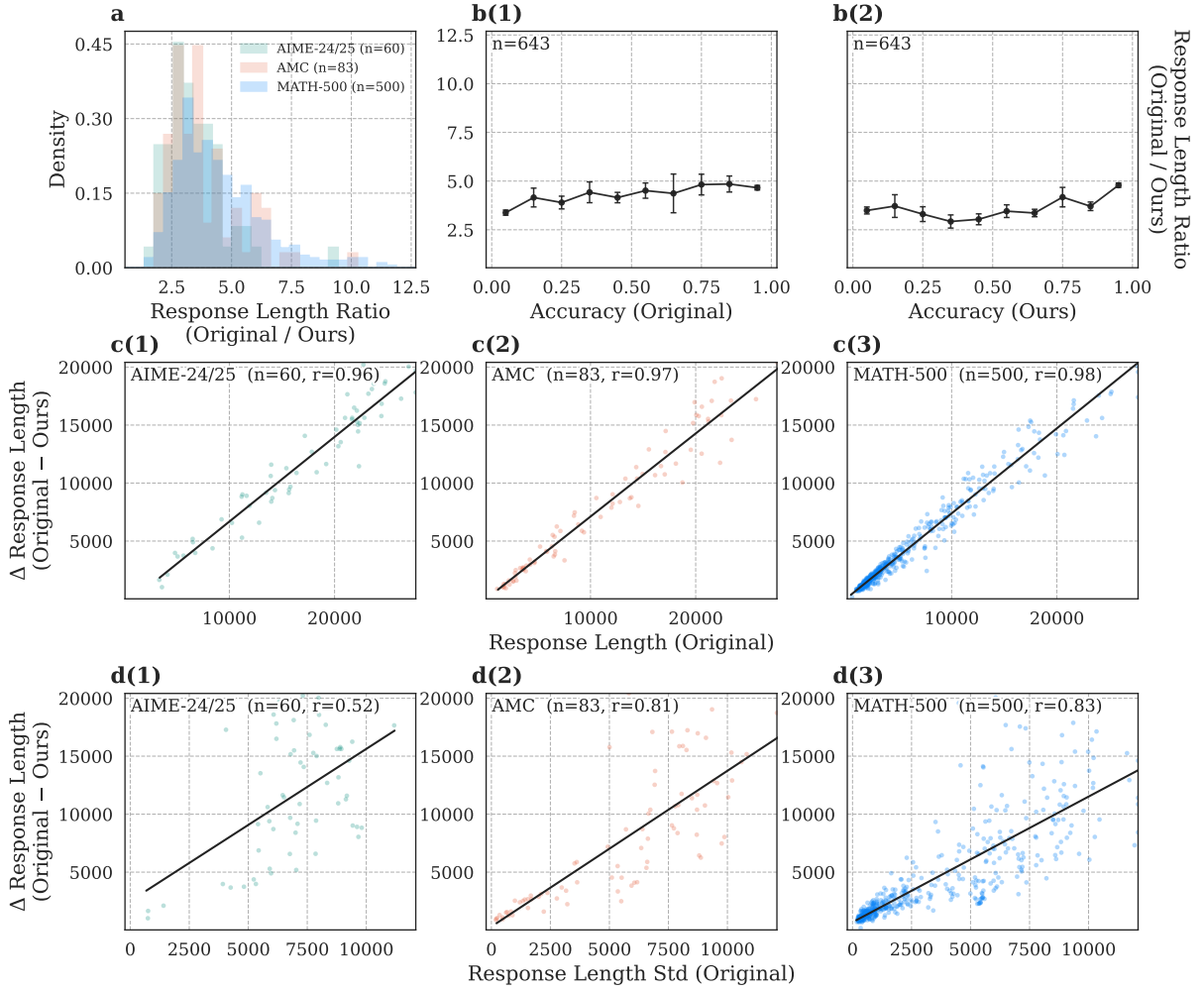


Figure 16: Quantitative analysis of reasoning efficiency for DOT-8K on DeepSeek-R1-Distill-Qwen-1.5B. (a) Density distribution of response length ratios between the original and DOT-optimized models. (b) Response length ratio across problems of varying difficulty, represented by original and our accuracy. (c) Token savings relative to the original response length. (d) Token savings relative to the standard deviation of the original response length.

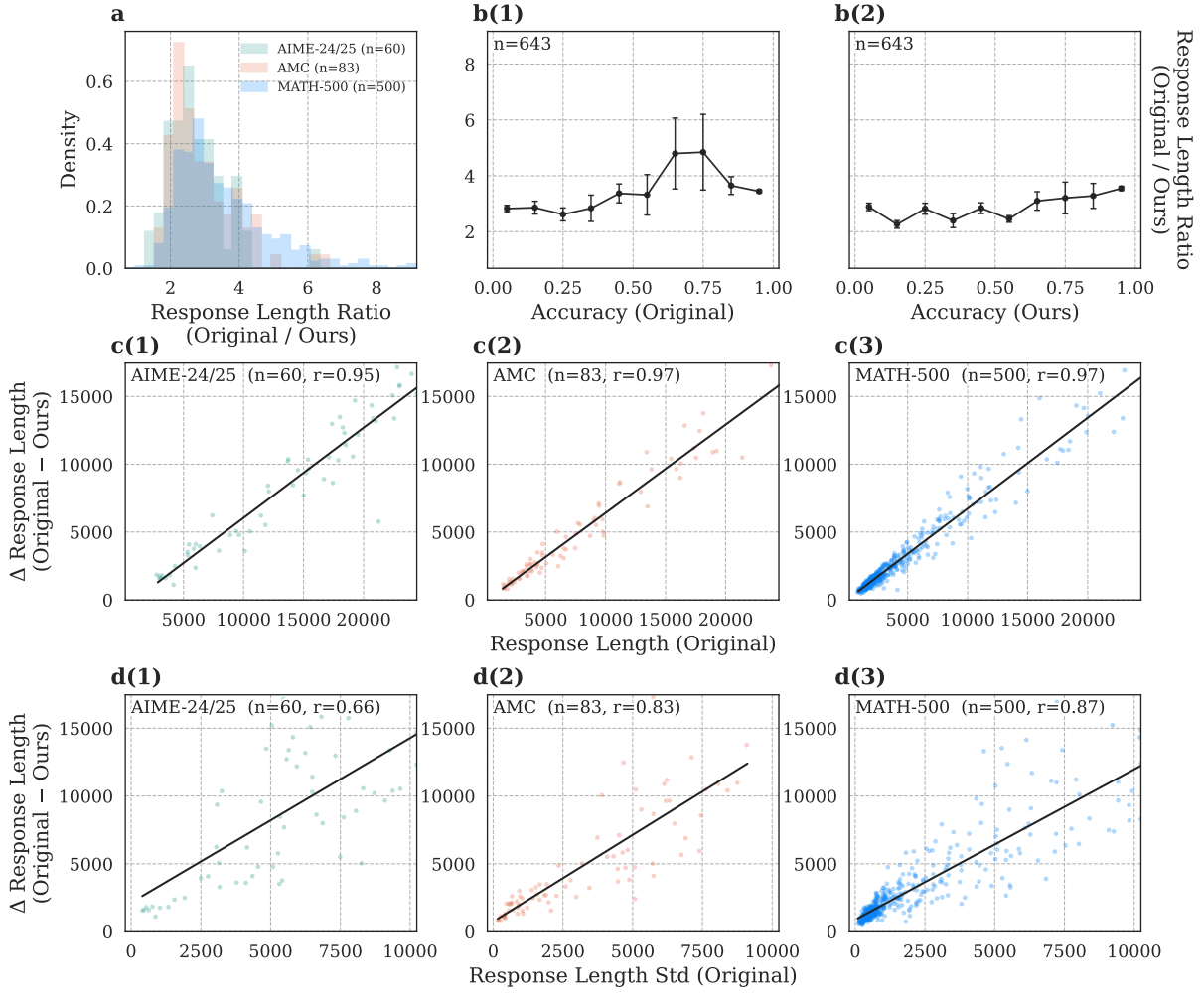


Figure 17: Quantitative analysis of reasoning efficiency for DOT-8K on DeepSeek-R1-Distill-Qwen-7B. (a) Density distribution of response length ratios between the original and DOT-optimized models. (b) Response length ratio across problems of varying difficulty, represented by original and our accuracy. (c) Token savings relative to the original response length. (d) Token savings relative to the standard deviation of the original response length.

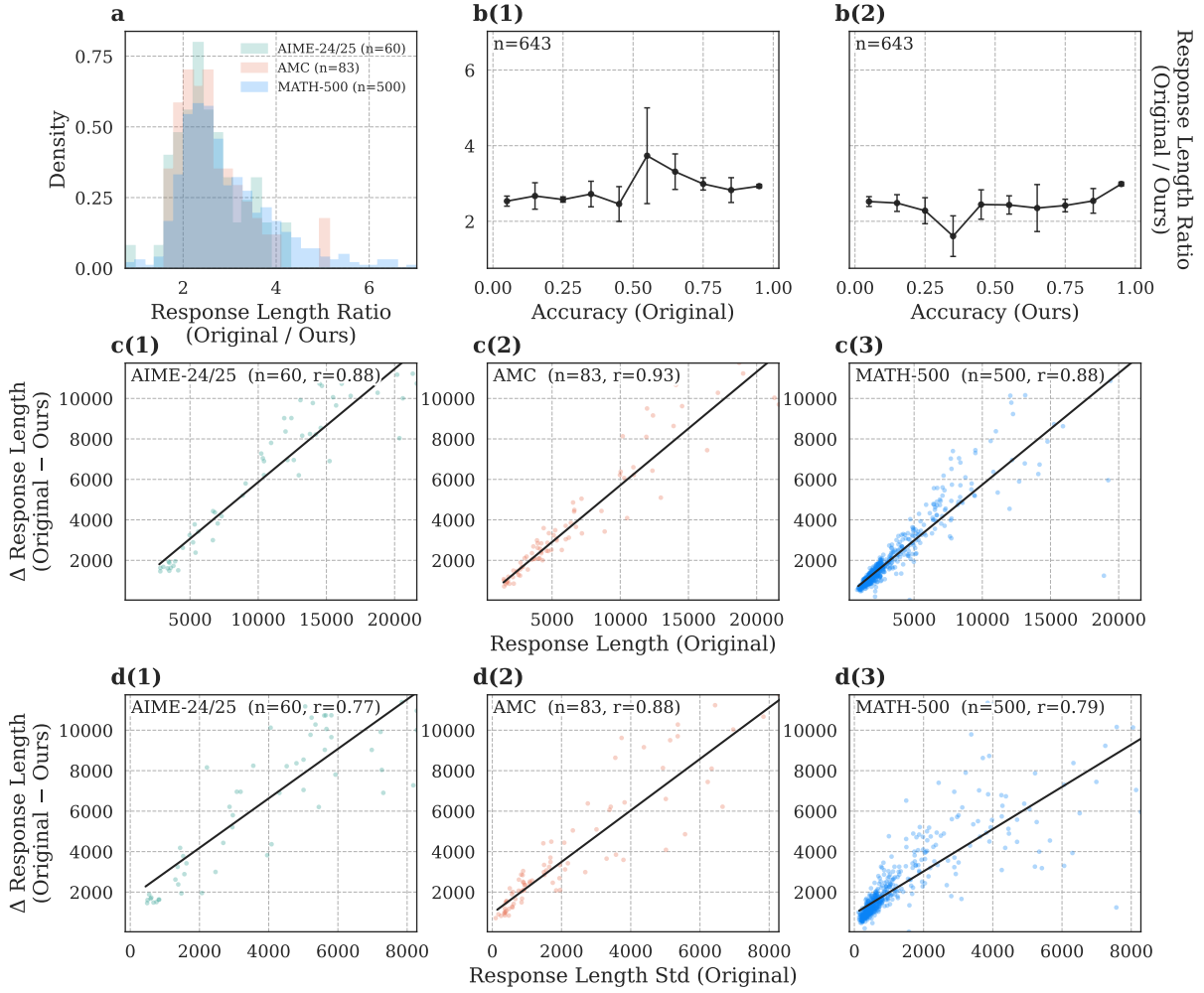


Figure 18: Quantitative analysis of reasoning efficiency for DOT-8K on DeepSeek-R1-Distill-Qwen-32B. (a) Density distribution of response length ratios between the original and DOT-optimized models. (b) Response length ratio across problems of varying difficulty, represented by original and our accuracy. (c) Token savings relative to the original response length. (d) Token savings relative to the standard deviation of the original response length.

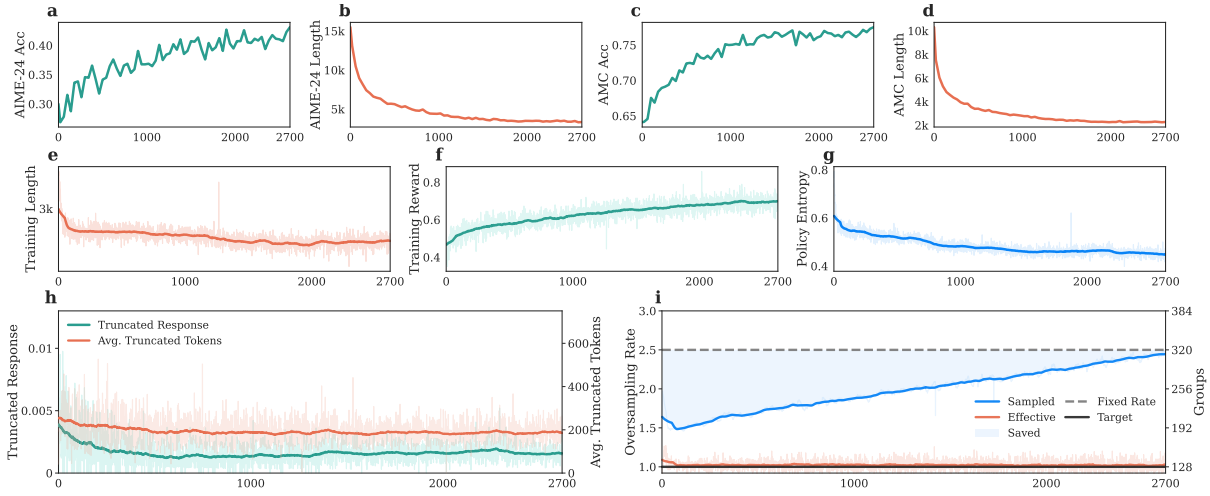


Figure 19: Metric curves monitoring the DOT-4K training process on DeepSeek-R1-Distill-Qwen-1.5B. (a, c) Evolution of pass@1 accuracy on AIME-24 and AMC. (b, d) Decrease in average response length for corresponding benchmarks. (e) Global training length reduction. (f) Progression of average training reward. (g) Controlled policy entropy decline. (h) Frequency of truncated responses and average truncated tokens per response. (i) Stability of oversampling rate and effective batch size via Predictive Dynamic Sampling

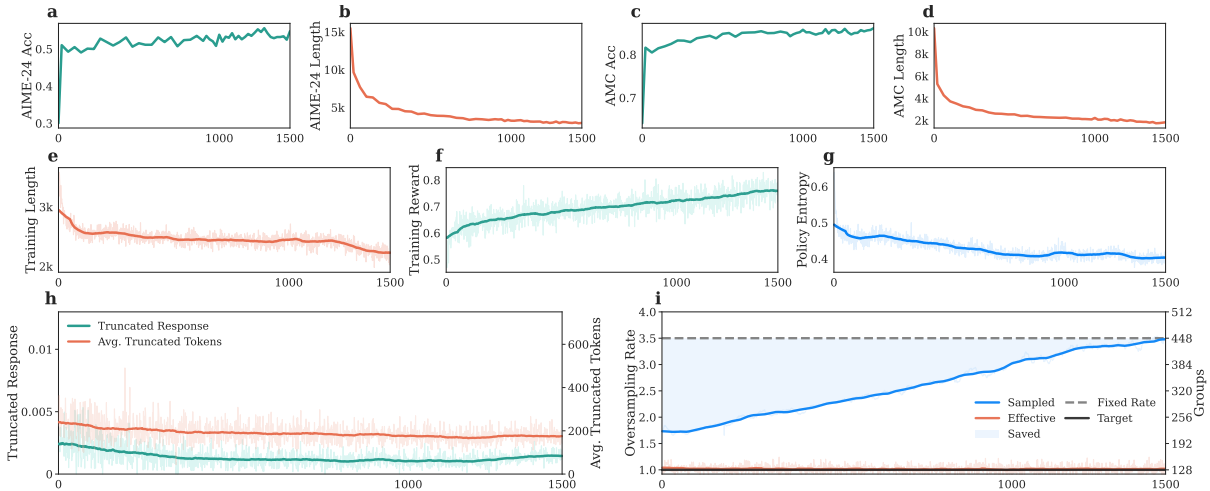


Figure 20: Metric curves monitoring the DOT-4K training process on DeepSeek-R1-Distill-Qwen-7B. (a, c) Evolution of pass@1 accuracy on AIME-24 and AMC. (b, d) Decrease in average response length for corresponding benchmarks. (e) Global training length reduction. (f) Progression of average training reward. (g) Controlled policy entropy decline. (h) Frequency of truncated responses and average truncated tokens per response. (i) Stability of oversampling rate and effective batch size via Predictive Dynamic Sampling

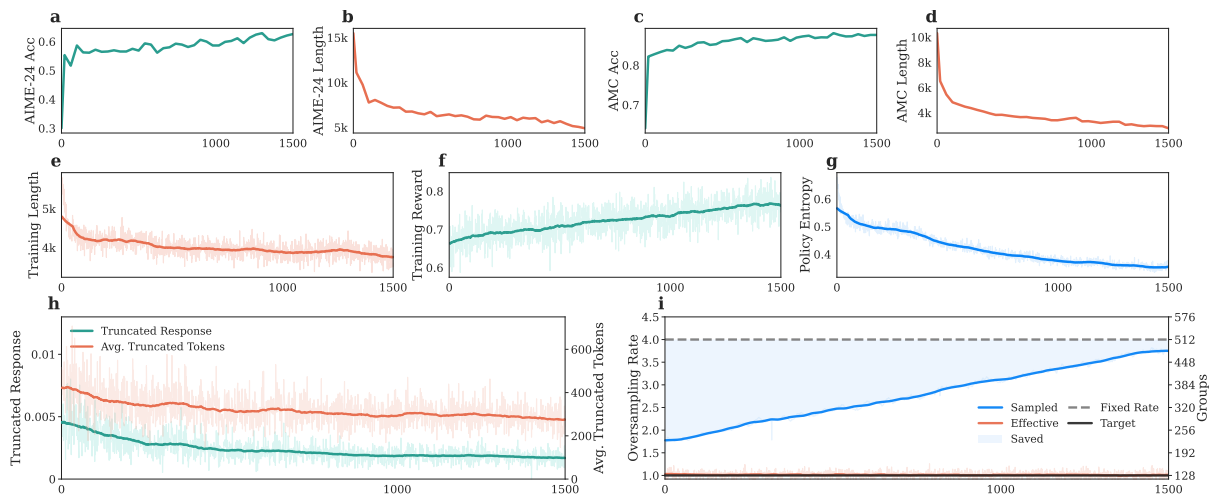


Figure 21: Metric curves monitoring the DOT-8K training process on DeepSeek-R1-Distill-Qwen-7B. (a, c) Evolution of pass@1 accuracy on AIME-24 and AMC. (b, d) Decrease in average response length for corresponding benchmarks. (e) Global training length reduction. (f) Progression of average training reward. (g) Controlled policy entropy decline. (h) Frequency of truncated responses and average truncated tokens per response. (i) Stability of oversampling rate and effective batch size via Predictive Dynamic Sampling

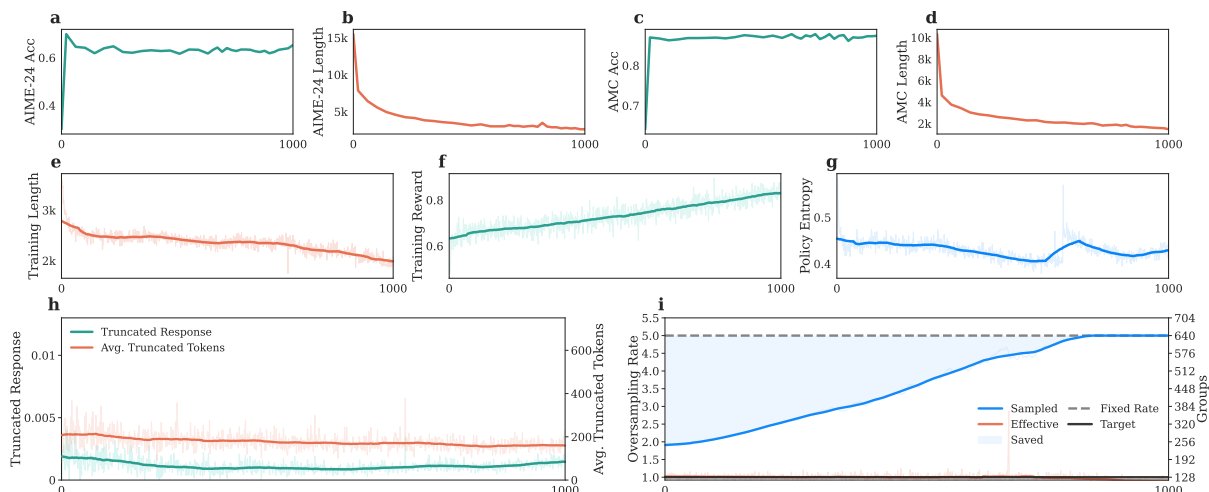


Figure 22: Metric curves monitoring the DOT-4K training process on DeepSeek-R1-Distill-Qwen-32B. (a, c) Evolution of pass@1 accuracy on AIME-24 and AMC. (b, d) Decrease in average response length for corresponding benchmarks. (e) Global training length reduction. (f) Progression of average training reward. (g) Controlled policy entropy decline. (h) Frequency of truncated responses and average truncated tokens per response. (i) Stability of oversampling rate and effective batch size via Predictive Dynamic Sampling

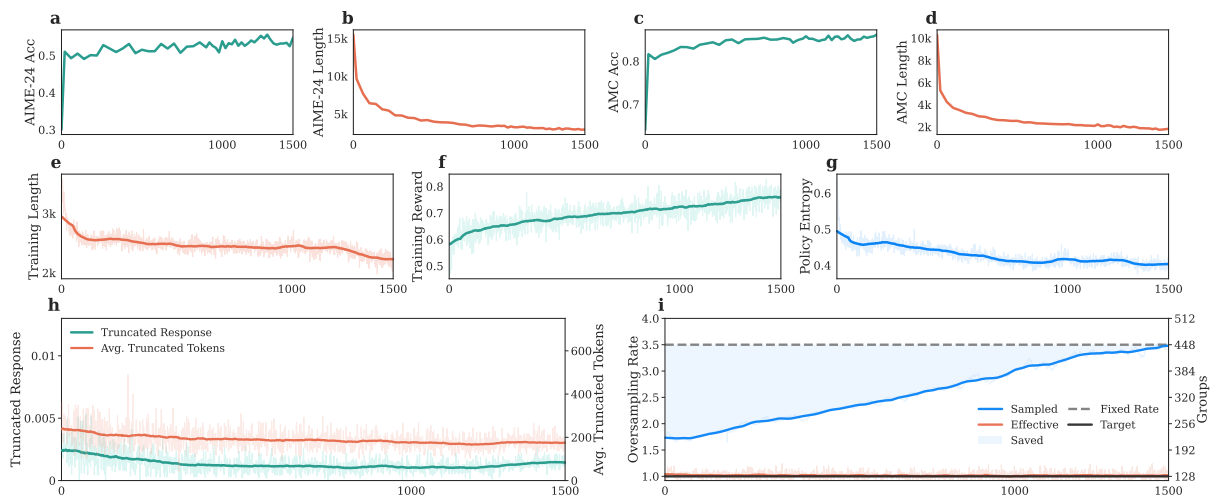


Figure 23: Metric curves monitoring the DOT-8K training process on DeepSeek-R1-Distill-Qwen-32B. (a, c) Evolution of pass@1 accuracy on AIME-24 and AMC. (b, d) Decrease in average response length for corresponding benchmarks. (e) Global training length reduction. (f) Progression of average training reward. (g) Controlled policy entropy decline. (h) Frequency of truncated responses and average truncated tokens per response. (i) Stability of oversampling rate and effective batch size via Predictive Dynamic Sampling