Mitigating Overthinking through Reasoning Shaping

Anonymous ACL submission

Abstract

Large reasoning models (LRMs) boosted by Reinforcement Learning from Verifier Reward (RLVR) have shown great power in problem solving, yet they often cause overthinking: excessive, meandering reasoning that inflates computational cost. Prior designs of penalization in RLVR manage to reduce token consumption while often harming model performance, which arises from the over-simplicity of token-level supervision. In this paper, we argue that the granularity of supervision plays a crucial role in balancing efficiency and accuracy, and propose **G**roup **R**elative **S**egment **P**enalization (**GRSP**), a step-level method to regularize reasoning. Since preliminary analyses show that reasoning segments are strongly correlated with token consumption and model performance, we design a length-aware weighting mechanism across segment clusters. Extensive experiments demonstrate that GRSP achieves superior token efficiency without heavily compromising accuracy, especially the advantages with harder problems. Moreover, GRSP stabilizes RL training and scales effectively across model sizes.

1 Introduction

Test-time Scaling with RLVR has greatly accelerated the development and adoption of Large Reasoning Models (LRMs) (Team et al., 2025; Guo et al., 2025a; Yu et al., 2025; Zheng et al., 2025). During inference, LRMs typically exhibit a distinct behavior from normal post-trained LLMs: they first produce a reasoning trajectory before generating the final response. Unlike conventional Chain-of-Thought (Wei et al., 2022), such trajectories often involve exploration and self-reflection over multiple possible solution paths, and gradually reach the final answer. However, this pattern can also lead LRMs to overthink, or repeatedly explore and revise their paths, resulting in excessively long decoding sequences and huge computational costs.

Recent works treat this issue from the algorithmic perspective, introducing additional supervision on token efficiency within RLVR. For example, Aggarwal and Welleck (2025) penalizes samples whose output length exceeds that of the reference responses. However, while such methods effectively reduce token consumption, they also compromise the benefits of Test-time Scaling, leading to a significant degradation in LRM performance.

Compressing the reasoning content essentially means removing redundant parts of the thought process. For humans, it first requires the ability to be aware of the redundant content before making a decision. When each token is considered as the candidate to remove, as illustrated in Figure 1(a), it is difficult to identify which specific tokens are unnecessary, since most tokens cannot be directly associated with the sparse verifier reward and recognized as high-value tokens. On the other hand, arbitrarily removing tokens is not feasible either, as they are still essential to preserve the readability and coherence of the reasoning. This motivates us to reconsider the granularity of supervision for balancing computational cost and task performance, and we find intermediate steps/segments can be a more natural unit. As shown in Figure 1(b), humans can more easily identify a redundant step than an individual token, since each step typically carries a semantically coherent piece of thought. Building on this insight, we propose to supervise the reasoning process at the segment level, enabling stable control of LRM reasoning behavior.

In this work, we introduce **G**roup **R**elative **S**egment **P**enalization (**GRSP**), a novel method that balances computational efficiency and task performance by operating at the reasoning-step granularity. As a foundation, our observations on open-source LRMs reveal that the quantity of segments is positively correlated with token consumption, while they are easier to assess for redundancy, making them a more reasonable target for penalization.

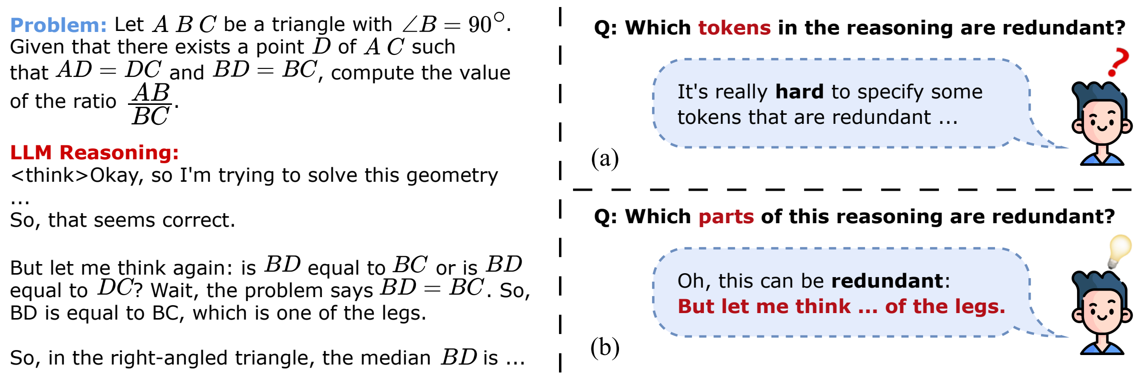


Figure 1: Illustration of redundancy detection. (a) Identifying redundant tokens is challenging, as most of them are weakly correlated with the golden answer; (b) Identifying redundant steps is much easier for its clearer meaning.

Further preliminary analyses indicate a statistical relationship between the performance of LRMs and the distribution of segment lengths: stronger models tend to exhibit more balanced segment-length distributions. It suggests a chance to mitigate performance degradation by applying length-aware penalties: penalizing segment counts within each length cluster and assigning decreasing weights for longer segments.

We conduct extensive experiments comparing GRSP with baselines and demonstrate its advantages in both token efficiency and accuracy. Notably, as task difficulty increases, together with longer reasoning, the advantages of GRSP become even clearer. We also analyze the impact of the weighting mechanism. Although it appears to contradict the intuition of encouraging shorter segments to minimize token count, our results show that it ultimately saves the cost and stabilizes RL training. Moreover, we investigate the effect of different segmentation strategies and evaluate the scalability of our approach across model sizes.

Our work can be summarized into three aspects:
(1) We first address the granularity of supervision in mitigating overthinking, proposing step-level supervision, and conducting preliminary analyses that uncover correlated features.
(2) We propose **GRSP**, which employs length-aware weighting across segment clusters to control behavior in reasoning, mitigating performance degradation.
(3) We perform comprehensive experiments to verify the broad effectiveness of segment-level penalization and weighting, and analyze its scalability on model size, offering insights for future research.

2 Preliminary

Large reasoning models (LRMs) improve downstream accuracy by prefixing a thinking block to the final answer. This pattern has been further scaled by recent RL work, such as Kimi-k1.5 (Team et al., 2025) and DeepSeek-R1 (Guo et al., 2025a). Given the prompt $x \sim D$ and corresponding responses $\{y|y = y_t \sim \pi_{\theta_{\text{old}}}\}$, the goal $\mathcal{T}$ is to maximizes the expected verifiable reward,

$$\mathcal{T} = \arg\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim \pi_{\theta_{\text{old}}}(x)} R(x, y) \quad (1)$$

where π is the policy and R is the accuracy signal obtained from a deterministic verifier. Prevalent algorithms include Reinforce

$$L = \mathbb{E}_{x\sim D, Y\sim \pi_{\theta_{\text{old}}}} \frac{1}{|Y|} \sum_{y^i \in Y} \frac{1}{|y^i|} \sum_{y_t^i} \left[A_{i,t} * \pi_\theta\left(y_t^i | x, y_{<t}^i\right)\right] \quad (2)$$

and GRPO (Shao et al., 2024), which adds an importance-sampling ratio

$$r(x, y_t, y_{<t}) = \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\theta_{\text{old}}}(y_t|x, y_{<t})} \quad (3)$$

and a clip operator inherited from PPO (Schulman et al., 2017). It also replaces the token-level advantage $A_{i,t}$ with a sequence-level score A_i

$$A_i = \frac{R(x, y^i) - \mathbb{E}_{y^j \in Y}[R(x, y^j)]}{\text{std}[R(x, y^j)]} \quad (4)$$

Despite its effectiveness, RLVR produces unnecessarily long thinking content. A common remedy is to augment R with a token-length penalty

$$R' = R \odot P(\{y_t\}) \quad (5)$$

where P penalizes the total number of generated tokens. It shortens the output but also degrades accuracy.

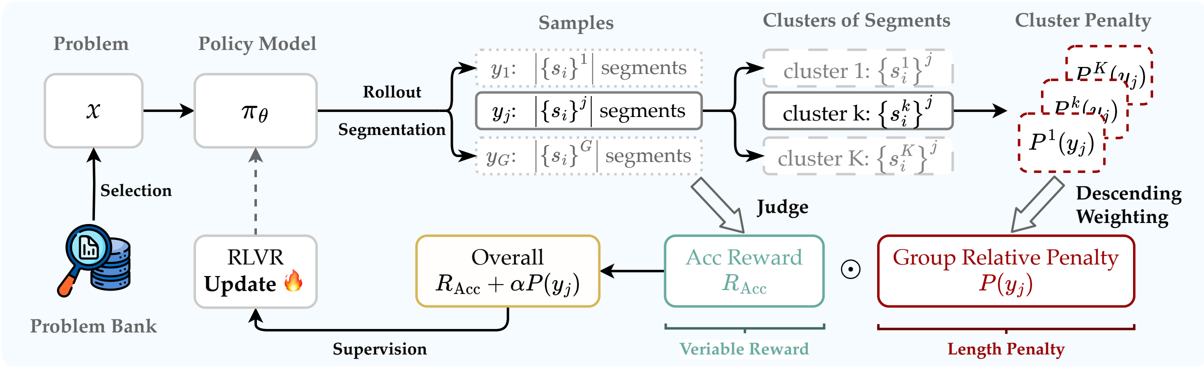


Figure 2: The overall workflow of GRSP.

Model	# Tokens	# Segments
DeepSeek-R1 (Guo et al., 2025a)	8544.70	135.75
QwQ-32B (Yang et al., 2024)	12003.31	216.52
DS-Qwen-Distill-32B	7464.53	105.51
DS-Qwen-Distill-14B	6539.29	94.73

Table 1: Statistics of open-source LRMs on token consumption and reasoning segment production.

3 Methodology

In this section, we present GRSP, a drop-in replacement for the token-level penalty in Eq 5 in RL. The overall workflow is illustrated in Figure 2.

3.1 Penalization on Segments

Mitigating overthinking essentially means compressing the thinking content. As shown in Figure 1, identifying redundant tokens is often ambiguous. However, distinct steps, i.e., trajectory segments within the reasoning process, can serve as a natural higher-level unit, as also adopted by Guo et al. (2025b). Moreover, our investigation of several open-source LRMs reveals a clear positive correlation between the quantity of segments and total token consumption (see Table 1). Therefore, by decomposing the thinking content into segments $\{s_i\}$, we can indirectly reduce token usage by discouraging unnecessary steps.

To be specific, for a single prompt x and a corresponding response group $Y = \{y^j\}$ sampled from $\pi_{\theta_{\text{old}}}$, let ${s_i}^j$ denote the trajectory segments in response y^j. We then compute the z-score of the segment count inside the group Y for each $y^j \in Y$, and treat its negative value as the penalty, which requires no task-specific threshold:

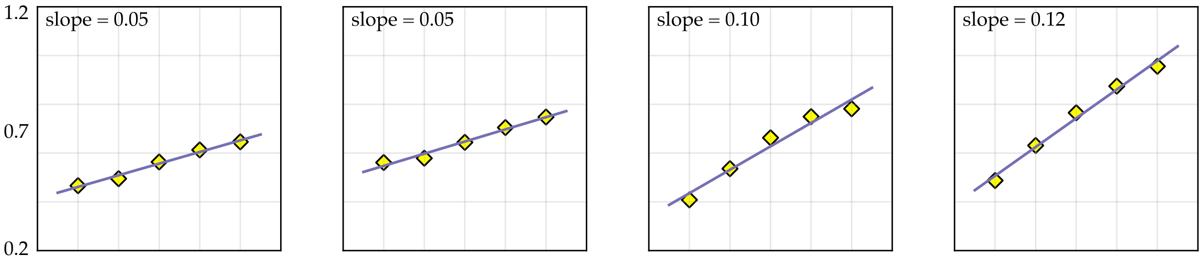
$$P(y^j) = -\frac{|\{s_i\}^j| - \mathbb{E}_{y\in Y}[|\{s_i\}|]}{\text{std}[|\{s_i\}|]} \qquad (6)$$

3.2 Group Relative Penalization

Simply changing the granularity of penalty does not completely solve the problem in §2, because the performance gain of an LRM over an ordinary post-trained model comes from **test-time scaling**, and any mechanism that shortens the reasoning consequently risks a drop in performance.

To be specific in the RL setting, continued improvement on downstream performance is usually obtained by stable training or more optimization steps (Yu et al., 2025; Zheng et al., 2025). However, introducing a length penalty can destabilize it: there is a phenomenon that once the response length exceeds a certain point, the penalty dominates and gradually the training collapses with task accuracy drifting downward. Hence, how to discourage over-length thinking while keeping the training process stable becomes the central question.

We investigate the above LRMs again for statistical signatures that correlate with these two requirements. We further hypothesize that model performance is correlated with the extent of training, i.e., stronger models tend to undergo more training, which in turn suggests greater training stability. Specifically, using the RL problems in § 4.1, we roll out a batch of trajectories for each model. We segment all reasoning content and group the resulting segments into K length-based clusters ($K = 5$ here for presentation clarity, while trends under other choices of K are consistent, as in Appendix E). Segments longer than 300 tokens are excluded, as they occur rarely. Since we focus on model performance, we apply the above procedure separately to passed and failed cases, and compute, for each cluster, the average number of segments on the two sides. Intuitively, we find that failed cases tend to contain more segments across most



(a) DeepSeek-R1 (b) QwQ-32B (c) DS-Qwen-Distill-32B (d) DS-Qwen-Distill-14B

Figure 3: The ratio of segment counts across each cluster (correct vs. wrong). Longer segments are generally more prevalent in correct cases, and stronger models (a, b) exhibit flatter slopes compared to weaker ones (c, d).

clusters. We then calculate, for each length cluster, the ratio of the average number of segments from the passed side to that from the failed side. The results are shown in Figure 3.

This experiment leads to several findings:
(1) A consistent trend is that failed cases contain more segments than passed ones.
(2) Wrong answer can be associated with the presence of relatively shorter segments in the reasoning content. In detail, the difference between passed and failed cases can be larger in shorter-length clusters (with lower ratios in Figure 3) while diminishing for longer-length clusters (with higher ratios). It is a common pattern across all examined LRMs.
(3) Stronger models show smaller variation in the passed-to-failed ratio across clusters, reflected in a smaller slope in the linear fit.

They suggest a potential link between model performance (and training stability) and a balanced distribution of reasoning segments. Since naively reducing token consumption tends to degrade accuracy, explicitly shaping segment-level reasoning patterns towards strong LRMs, while simultaneously compressing reasoning, may help strike a better balance between performance and efficiency.

Motivated by this hypothesis, we proposed GRSP. Concretely, we split each sampled reasoning into segments and cluster them by length. For each cluster k, we compute a relative penalty using z-score normalization:

$$P^k(y^j) = -\frac{|s_i^{k^j}| - \mathbb{E}_{y\in Y}[|s_i^k|]}{\text{std}[|s_i^k|]} \quad (7)$$

The overall penalty is then obtained by weighting across clusters:

$$P(y^j) = \sum_k w^k P^k(y^j) \quad (8)$$

We penalize short segments more heavily, while applying weaker penalties to longer ones. To achieve this, we assign **descending** weights from shorter to longer clusters. The final reward is then given by:

$$R' = R + \alpha P(y^j) \quad (9)$$

3.3 Segmentation

We design two mechanisms for segmentation. The first is keyword-based matching, similar to Guo et al. (2025b). We collect a list of keywords from typical reasoning cases to identify potential segment boundaries. It is used by default for conciseness and high computational efficiency, while its applicability is limited to specific languages. Cases illustrating its effect are shown in Appendix G.

The second mechanism is token log-probability matching. Following observations in Li et al. (2024); Song et al. (2025); Wang et al. (2025a), segment boundaries often correspond to local minima in token log-probabilities. This is because segment transitions usually admit more candidate continuations, leading to lower confidence at the beginning of a new segment. Based on it, we locate boundaries by identifying these local minima. We implement and test it in § 4.6.

4 Experiment

4.1 Setup

Following prevalent practice, we initialize models with SFT to acquire the reasoning pattern, then RLVR is applied for performance optimization. Problems for SFT are from NuminaMATH (LI et al., 2024), while completions of reasoning and response are O1-mini patterned examples, following the prompt-engineering procedure of Gao et al. (2025b). For the RL stage, we use more challenging problems sampled from Omni-MATH and AIME, which tend to induce longer reasoning trajectories than NuminaMATH, and therefore better expose test-time scaling effects. The data volume

Model / Method	MATH 500		AIMO Prize 1		Omni-MATH 500		Overall	
	Acc.	Avg Len.	Acc.	Avg Len.	Acc.	Avg Len.	Acc.	Avg Len.
Open-source Models								
DeepSeek-R1	96.60	2428	91.25	4704	70.80	7456	84.26	4924
QwQ-32B	98.00	4260	95.00	9222	71.20	12125	85.37	8268
DS-Qwen-Distill-32B	96.40	2594	82.50	6650	61.80	8997	79.35	5859
DS-Qwen-Distill-14B	93.80	2538	80.00	6504	63.60	9091	78.80	6075
Qwen-2.5-14B-it	54.40	488	10.00	1091	16.60	918	33.61	811
Qwen-2.5-14B-it*	84.80	1460	53.75	2579	39.80	2159	61.67	2116
Training Qwen-2.5-14B-it*								
Reinforce	87.20	1887	53.75	3568	44.20	5131	64.81	3513
+ LCPO	**86.40**	2222	57.50	3771	42.40	7994	63.89	5009
+ O1-Pruner	85.20	**1738**	**60.00**	3416	40.40	5226	62.59	**3477**
+ GRSP	85.40	2128	55.00	**3215**	**45.60**	**4866**	**64.72**	**3477**
GRPO	85.20	2131	60.00	2648	45.40	5315	64.91	3643
+ LCPO	86.00	1919	52.50	3597	43.40	5855	63.80	3866
+ O1-Pruner	85.00	**1706**	**61.25**	3299	40.60	5497	62.69	3579
+ GRSP	**86.20**	2054	60.00	**2826**	**43.80**	**4897**	**64.63**	**3427**

Table 2: Results of different models/methods across different benchmarks. The **higher** accuracy (Acc.) and **lower** average length of responses (Avg Len.) are expected. Models labeled by * have been SFT-tuned.

and implementation detail are summarized in Appendix A and B. Moreover, we set Qwen-2.5-14B-it as the starting checkpoint, and mark it with * after SFT. During RL, models are trained with Reinforce and GRPO, with the addition of a length penalty.

4.2 Evaluation

We construct three test sets with increasing difficulty: 500 problems from MATH 500, 10 challenging problems from AIMO Prize-1 (each evaluated 8 times to reduce variance), and 500 problems from Omni-MATH 500, enabling an overall conclusion and fine-grained analyses, like how task difficulty influences model behavior and token efficiency. We report two metrics: task performance via **accuracy** and token efficiency via the average length of completions. We compare GRSP with two baselines: (1) LCPO (Aggarwal and Welleck, 2025) uses the ratio of generated vs. reference length as a weighting factor for the verifiable reward, reducing the sparsity of reward distribution and introducing a direct correlation between reward and token usage. (2) O1-Pruner (Luo et al., 2025) computes a ratio factor similar to LCPO, while adding it as an auxiliary reward term to the verifiable reward.

4.3 Main Results

In this section, we report results on the above mathematical benchmarks to demonstrate the effectiveness of GRSP, which in turn substantiate the significance of analyses in § 3.2. As a supplementary reference, we include additional results on code generation in Appendix C.

We first evaluate four open-source LRMs introduced in § 3.1. Although all of them exhibit strong performance, there remains a clear gap between DeepSeek-R1/QwQ-32B and the two distill models. Notably, with 671B parameters, DeepSeek-R1 demonstrates remarkable token efficiency, suggesting that stronger base capability can contribute to more concise reasoning. In addition, the fine-tuned Qwen-2.5-14B-it* achieves a substantial improvement over Qwen-2.5-14B-it, highlighting the effectiveness of test-time scaling.

Overall, GRSP achieves the **best** scores on both task performance and token efficiency under both GRPO and REINFORCE frameworks. Notably, on the most challenging Omni-MATH 500, GRSP achieves the most significant reduction in token usage while maintaining the highest accuracy among all baselines, even matching or surpassing the naive RL version (Reinforce + GRSP). Although LCPO also performs competitively, it fails to deliver noticeable gains in token efficiency.

We further analyze how token length correlates with task difficulty. For all methods, token overhead increases as problems become more challenging, indicating that models rely on *length scaling*

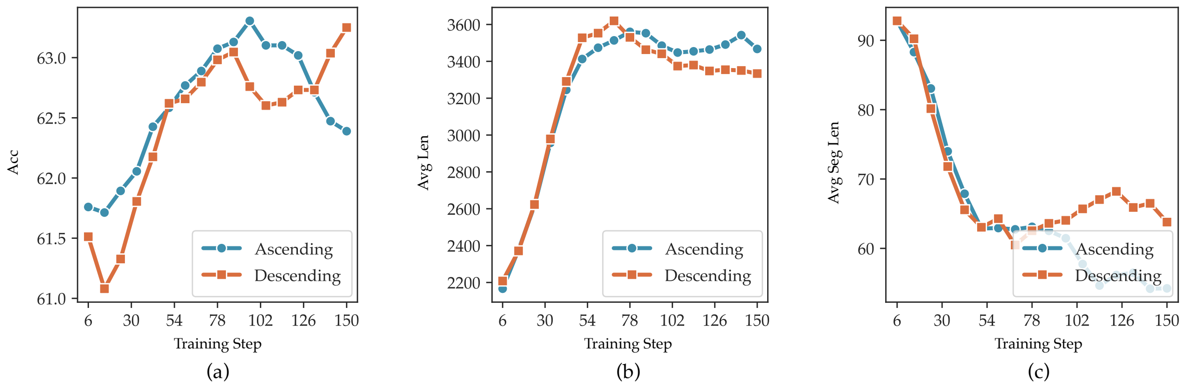


Figure 4: Comparison of two weighting strategies. (a) Accuracy over training steps; (b) Average response length over training steps; (c) Average segment length over training steps.

to tackle complex reasoning tasks. RL accentuates this trend, as the token growth on Omni-MATH is much larger than on easier benchmarks, and GRSP primarily reduces token usage on such problems, where overthinking is most likely to occur, while preserving task performance.

We also observe distinct segmentation patterns across different methods, which may strongly correlate with their results. Using the keyword-based segmentation on all models trained with Reinforce, we find that GRSP produces an average of 21.07 segments, notably fewer than 26.66 from the model trained without additional penalty. This confirms that GRSP effectively regulates the number of reasoning segments and thus reduces token overhead. In contrast, the two baselines yield substantially more segments of 51.97 and 142.12, respectively, and the largest part is short segments, as the ratio of segments from the cluster of $k = 1$ is 79.17% and 91.36% for O1-pruner and LCPO, while that for GRSP is 62.61%. We attribute this to the ambiguity in supervising token length: it disturbs the optimization on verifiable reward, making the model resort to rapidly iterating short reasoning steps to maintain performance. Such settings not only lead to lower accuracy but also have a larger likelihood of overthinking, as extremely long outputs are observed in LCPO. We also present case study on this aspect in Appendix G.

4.4 Ablation on the Weighting Mechanism

In this section, we investigate the effect of the proposed weighting mechanism across segment clusters. By default, the penalty weight decreases with the cluster index k, that is, longer segments receive smaller penalties (**Descending**), which encourages potentially deeper reasoning within each segment while discouraging the overproduction of short segments. As a contrastive setting, we reverse the order to make the weights increase with k, defined as $w^k = (k-1)\times 0.05+1$ (**Ascending**). We train both configurations under REINFORCE on Qwen-2.5-14B-it* and track the evolution of accuracy (Acc), average token quantity of responses (Avg Len), and average segment length (Avg Seg Len), smoothed and visualized in Figure 4.

The results reveal that test-time scaling emerges under both settings: as training progresses, the average token quantity grows alongside task performance. However, the Ascending configuration exhibits a much steeper rise in both metrics, reaching a peak earlier but soon suffering a sharp accuracy collapse, indicating training instability. In contrast, the Descending configuration shows steadier improvement, with accuracy increasing more smoothly over time. However, after reaching the peak length, the Ascending model fails to compress thinking content effectively. Considering its high weight on the short segment clusters, it resembles the degenerate feature observed in LCPO, where the model excessively expands reasoning without meaningful gains.

The segment-level patterns in Figure 4 (c) further support this observation. At the beginning, RLVR generally drives models to produce shorter segments to activate length scaling, and both configurations behave similarly. However, as training continues, the Ascending weights push the model to rapidly over-optimize for shorter segments, which is a turning point that coincides with its accuracy collapse. Conversely, the Descending configuration gradually shifts toward generating longer segments, i.e., increasing the proportion of long segments. Interestingly, this adjustment leads to an overall

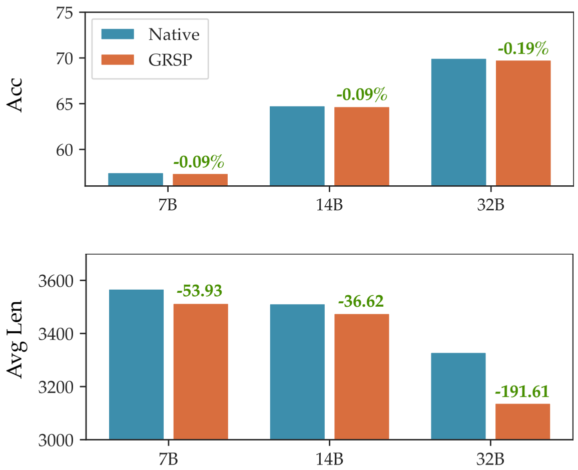


Figure 5: Effect of GRSP across models of varying capacities on all test data, comparing changes in accuracy and average response length.

shorter average response length, suggesting that the model learns a more efficient reasoning strategy: thinking more thoroughly within each segment, thus requiring fewer total steps to reach the correct answer.

These findings also shed light on the dynamics between the verifiable reward and the length penalty. During the early stage, the verifiable reward dominates, and both configurations exhibit similar trends across all three metrics. In the mid-stage, both encounter a transition point where accuracy begins to drop a bit. In the near time, length decreases, indicating a shift in optimization focus from purely correctness to brevity, which in turn risks destabilizing training. The Descending configuration, however, escapes this collapse by increasing the proportion of long segments, allowing accuracy and efficiency to improve jointly. This confirms that accuracy and length optimization are not a zero-sum game; with a proper pattern, GRSP achieves a stable balance between concise reasoning and task effectiveness. Hence, we conclude that descending weighting can be a more stable and interpretable optimization signal.

4.5 Scaling of Model Capacity

Results on open-source LRMs from Table 2 suggest that the capacity of base models plays a critical role in task performance and token efficiency. For example, DeepSeek-R1, the largest model in our evaluation, achieves near the top accuracy and token efficiency, while DS-Qwen-Distill-32B also requires fewer tokens than its 14B counterpart. This naturally raises the question of how model capacity interacts with GRSP during RL training.

To investigate it, we conduct experiments on three models from the Qwen-2.5 family: 7B-it, 14B-it, and 32B-it. Each model is first warmed up with the same SFT dataset, followed by RL training under two settings: standard Reinforce and Reinforce + GRSP, with all other hyperparameters held constant. We still track accuracy and average response length to test how the effect of GRSP varies across model scales (Figure 5). Our results reveal two clear trends:

(1) Larger models are inherently more efficient and accurate even in RL, where the accuracy improves steadily with model size, while token consumption decreases under the same training setup.

(2) GRSP consistently improves efficiency across all scales, with minimal impact on accuracy. While accuracy remains nearly unchanged, the reduction in token usage can grow. In particular, the 32B model can complete tasks with significantly fewer tokens compared to smaller models, reflecting its greater capacity to leverage reasoning compression.

4.6 Confidence-based Segmentation

In § 3.3, we introduce another segmentation strategy based on the model's confidence distribution. Unlike keyword matching, it does not rely on manually collected keywords yet achieves similarly strong performance.

Transitions between reasoning segments often correspond to local minima in the model's log probabilities. However, low logprob values can also occur in other cases, such as generating digits or single characters. Hence, after identifying positions where the smoothed logprob falls below a threshold γ, we further filter them based on token length and whether they correspond to local minima. Figure 6 (a) illustrates an example where a local minimum coincides with the start of a new reasoning segment.

We conduct experiments using the confidence-based segmentation under the REINFORCE+GRSP framework on Qwen-2.5-14B*, and compare it with the proposed keyword-based segmentation in terms of accuracy and response length. As shown in Figure 6 (b, c), the two curves exhibit similar trends. Notably, the confidence-based segmentation shows a rise–fall–rise pattern in both accuracy and length, whereas the keyword-based segmentation displays a steady decline in length without recovery. These patterns are consistent with our findings in § 4.4: the verifiable signal and the length penalty occa-

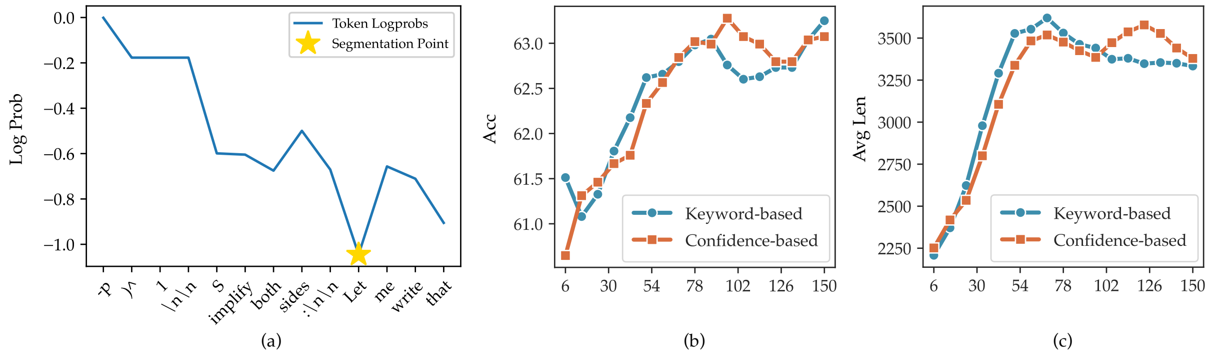


Figure 6: (a) Illustration of the log-probability trend around a segmentation point; (b) Comparison of accuracy between keyword-based and confidence-based segmentation; (c) Comparison of average response length.

sionally compete, leading to temporary degradation in one metric that later recovers, instead of collapsing entirely. Ultimately, the confidence-based segmentation achieves a higher accuracy–length pair (64.91, 3415) than the keyword-based segmentation (64.72, 3477), confirming the effectiveness of this design.

5 Related Work

5.1 Test-time Scaling of LLMs

Test-time scaling has proved effective at boosting LLM performance on complex question answering, mathematical problem solving, and code generation (Wu et al., 2024; Wang et al., 2022; Wei et al., 2022; Guo et al., 2025a). One line of such work is Monte Carlo Tree Search or tree-structured prompting (Wu et al., 2024; Yao et al., 2023). However, it involves trivial human-crafted engineering that hinders scaling up. Another line is Reinforcement Learning with Verifiable Reward (RLVR) (Team et al., 2025; Guo et al., 2025a; Muennighoff et al., 2025; Ye et al., 2025) which encourages exploration of long and complex reasoning paths via simple binary signals provided by verifiers.

5.2 Efficient Long Chain-of-Thought LLMs

Despite the remarkable effectiveness of the long reasoning patterns, they suffers from substantial computational overhead, especially for challenging user inputs where the model tends to repeatedly deliberate, named **overthinking**. To mitigate such phenomena, recent work aims at shortening reasoning trajectories to reduce computation, while preserving task performance, e.g., accuracy (Sui et al., 2025). Lightweight approaches include designing control prompt, such as indicating a token budget or specifying reasoning granularity (Han et al., 2025; Xu et al., 2025; Aytes et al., 2025), or intervening in the decoding process (Liu et al., 2025; Liao et al., 2025; Ding et al., 2025; Fu et al., 2024; Huang et al., 2025; Taubenfeld et al., 2025; Wang et al., 2025b). However, they do not essentially modify the model's reasoning patterns, which may limit their robustness and ultimate performance.

A fundamental solution is to introduce supervision on reasoning efficiency during fine-tuning (Xia et al., 2025; Zhang et al., 2025; Arora and Zanette, 2025; Aggarwal and Welleck, 2025; Luo et al., 2025; Team et al., 2025), especially in RLVR settings where both positive and negative rewards can effectively guide model behavior. Nevertheless, existing studies mostly impose penalties on tokens, while the correlation between token-level penalties and the final verifiable rewards remains weak, leading to limited compression effectiveness or even degradation in accuracy.

6 Conclusion

To mitigate the overthinking of Large Reasoning Models (LRMs), we propose Group Relative Segment Penalization (GRSP), a step-level method that regularizes reasoning in RLVR. By supervising at the granularity of reasoning segments and applying length-aware weighting, GRSP effectively mitigates performance degradation while still increasing efficiency. We conduct extensive experiments to verify that GRSP has the advantages in the above two aspects, while also improving training stability and scaling across model sizes. We hope our study could inspire future research on the granularity selection of supervision during the design of RLVR algorithms for LRMs.

Limitations

We also experimented with warm-up data constructed from other patterns, such as those derived from DeepSeek-R1. However, we found that such data tends to make the model generate overly long responses even before reinforcement learning begins. As a result, it becomes difficult to observe clear scaling effects on response length.

To further validate the compression of token usage, it would be ideal to continue training from our current checkpoint using reinforcement learning. Nevertheless, this requires substantial computational resources beyond our current budget. Despite these limitations, our experiments demonstrate the robustness of the proposed method and its potential for continued efficient scaling. We hope future work will further explore these directions.

References

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

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

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching. *CoRR*, abs/2503.05179.

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zengmao Wang, Ziwei Liu, Bo Du, and 1 others. 2025. Dynamic parallel tree search for efficient llm reasoning. *arXiv preprint arXiv:2502.16235*.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and 1 others. 2024. Efficiently scaling llm reasoning with certaindex. *arXiv preprint arXiv:2412.20993*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2025a. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations*.

Bofei Gao, Yejie Wang, Yibo Miao, Ruoyu Wu, Feifan Song, Longhui Yu, Tianyu Liu, and Baobao Chang. 2025b. Towards a better initial policy model for scalable long-cot reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7652–7665.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 24842–24855. Association for Computational Linguistics.

Chengsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. 2025. Efficient test-time scaling via self-calibration. *arXiv preprint arXiv:2503.00031*.

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*.

Bolian Li, Yifan Wang, Anamika Lochab, Ananth Grama, and Ruqi Zhang. 2024. Cascade reward sampling for efficient decoding-time alignment. *arXiv preprint arXiv:2406.16306*.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. Reward-guided speculative decoding for efficient llm reasoning. *arXiv preprint arXiv:2501.19324*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Yuliang Liu, Junjie Lu, Zhaoling Chen, Chaofeng Qu, Jason Klein Liu, Chonghan Liu, Zefan Cai, Yunhui Xia, Li Zhao, Jiang Bian, and 1 others.

2025. Adaptivestep: Automatically dividing reasoning step through model confidence. *arXiv preprint arXiv:2502.13943*.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

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

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

Feifan Song, Shaohang Wei, Wen Luo, Yuxuan Fan, Tianyu Liu, Guoyin Wang, and Houfeng Wang. 2025. Well begun is half done: Low-resource preference alignment by weak-to-strong decoding. *arXiv preprint arXiv:2506.07434*.

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

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. 2025. Confidence improves self-consistency in llms. *arXiv preprint arXiv:2502.06233*.

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

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, and 1 others. 2025a. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025b. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *arXiv preprint arXiv:2503.01422*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

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

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *CoRR*, abs/2502.18600.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

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

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. Lightthinker: Thinking step-by-step compression. *arXiv preprint arXiv:2502.15589*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

A Statistics of Data

Please refer to Table 3.

Stage	Data Source	#
SFT	NuminaMath-1.5 (LI et al., 2024)	27621
RL	AIME	800
	Omni-MATH (Gao et al., 2025a)	2400
Eval	MATH 500 (Lightman et al., 2024)	500
	AIMO Prize 1	10 * 8
	Omni-MATH 500 (Gao et al., 2025a)	500

Table 3: Statistics of data utilized for SFT, RL, and evaluation, respectively.

B Implementation Details

B.1 General Settings

We set the maximum sequence length to 30K to accommodate long reasoning trajectories. Each RL training run consists of 150 steps, with a rollout performed every 3 steps. The learning rate is set to 2e-6. For each rollout iteration, we randomly sample 1024 problems, and for each problem, we perform 10 rollouts with a temperature of 1.0. During evaluation, the temperature is set to 0.6. Dynamic sampling (Yu et al., 2025) is applied, resulting in a varying batch size across iterations.

For GRSP, the descending weights assigned to different length clusters, denoted as w^k, are defined as

$$w^k = (K - k) \times 0.05 + 1$$

where k is the cluster index and $K = 5$. This linear weighting is empirically motivated by the approximately linear trend observed in Figure 3, as well as the slope of around 0.05 exhibited by stronger models. The balancing coefficient α is set to 5e-3 for keyword-based segmentation and 2.5e-3 for confidence-based segmentation. We report the results with the highest task performance.

B.2 Analysis of Additional Overhead

GRSP does not bring significant extra cost. The additional operations it introduces, such as segmentation, clustering, and z-score computation, are all completed via CPU computation rather than GPU. They take just a few seconds and are negligible compared with rollout time. Even the confidence-based segmentation requires only logprobs, which are already available after decoding, introducing no extra overhead.

Despite such subtle extra cost, the improvements are substantial. GRSP reduces reasoning token usage, especially on the difficult Omni-MATH 500 where overthinking can be easily triggered, while maintaining accuracy, and in some cases even slightly improving it.

B.3 Why Qwen 2.5 series?

It should be acknowledged that our Megatron-based training backend currently supports only certain internal models and the Qwen 2.5 series, which becomes the main practical reason why we use only the Qwen 2.5 series in this work. Accordingly, we first perform SFT on the instruct versions of Qwen 2.5 models to explicitly equip them with reasoning patterns, allowing the subsequent RL stage to focus on exploring better policies. This setup has also been widely deployed by many popular LLMs like DeepSeek-R1 (Guo et al., 2025a), QwQ-32B (Yang et al., 2024) and Kimi-K1.5 (Team et al., 2025).

Moreover, it has been widely discussed that only the Qwen 2.5 series seem capable of acquiring reasoning ability via RL without SFT, raising concerns regarding data contamination. However, this would not impact any conclusions from our experiments, for the the following reasons:
(1) This work does not aim to pursue peak performance via Qwen 2.5 against other models, which could be affected by data contamination. Instead, we focus on the effect of mitigating over-thinking under the

same experimental settings, and the Qwen 2.5 setup is applied fairly across all compared methods.
(2) GRSP exhibits clear and systematic effects, in contrast to random rewards, which are ineffective and would rely on potential data contamination. For example, as shown in § 4.4, reversing the weighting direction leads to significant changes across multiple metrics.
(3) We further conduct an experiment on LiveCodeBench (12/1/2025–2/1/2025), which was released later than Qwen 2.5, and the results are consistent with those in the main content (See Appendix C).

C Results on Code Generation

Apart from mathematical reasoning, we conduct experiments on code generation to emphasize the significance of GRSP in balancing performance and efficiency for long reasoning. In detail, we train Qwen-2.5-14B-it* with Reinforce on 1,024 problems from AtCoder and LeetCode, as well as evaluate each model 10 times on 64 problems released in 12/1/2025–2/1/2025 from LiveCodeBench (Jain et al., 2024). We implement GRSP with keyword-based segmentation. The results in Table 4 confirm that GRSP mitigates overthinking while maintaining (even raising) performance.

Method	Pass	Avg Len.
Reinforce	21.88	2652
+ LCPO	21.48	2873
+ O1-Pruner	21.09	1831
+ GRSP	22.85 **(+0.97)**	2402 **(-250)**

Table 4: Results of different methods on coding generation, all tuned from Qwen-2.5-14B-it* with Reinforce.

D GRPO Ablation on the Weighting Mechanism

Figure 7 presents the ablation results of the weighting mechanism on GRPO. A similar trend to Reinforce in Figure 4 can be observed, where the Ascending weighting improves accuracy more rapidly while the Descending weighting achieves stronger compression and maintains high accuracy. In addition, we find that GRPO training is generally more stable than Reinforce across all methods, so the Ascending setting does not collapse in the middle stage and excessively encourages short segments as in Reinforce training. Nevertheless, the Descending weighting still produces longer reasoning segments as expected.

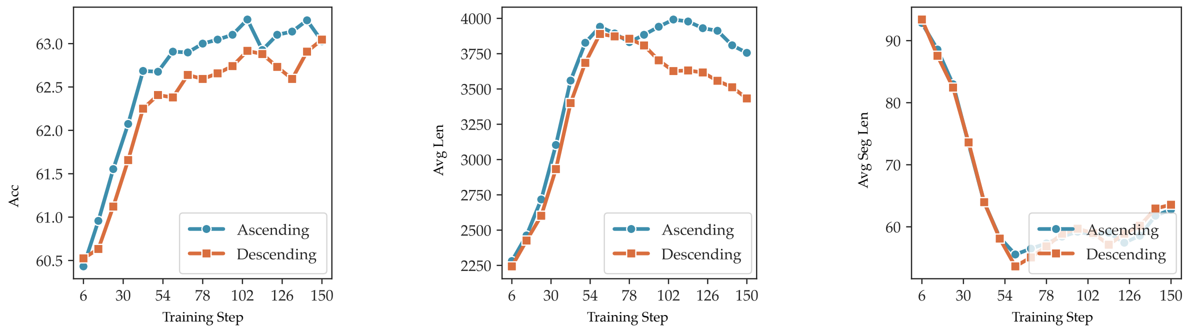


Figure 7: Comparison of two weighting strategies for GRPO. (a) Accuracy over training steps; (b) Average response length over training steps; (c) Average segment length over training steps.

E Findings on Different K

In this section, we estimate the effect of K for conclusions in § 3.2. We set different value 3 and 10 for K, the number of clusters, and plot each distribution of ratios between correct and incorrect cases across all clusters, as shown in Figure 8 and Figure 9.

Obviously, they demonstrate extremely similar trends as in Figure 9 with diffrent choices of K, and the conclusions in § 3.2 are robust and generally applicable. In detail, the ratio consistently increases with

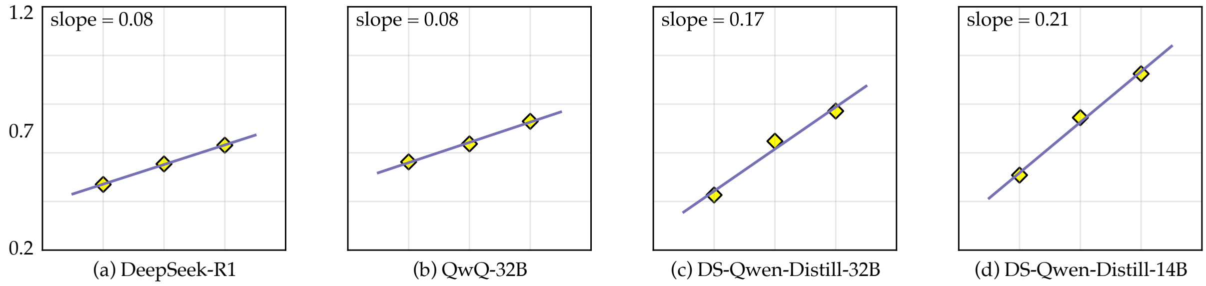


Figure 8: The ratio of segment counts across K(=3) clusters (correct vs. wrong).

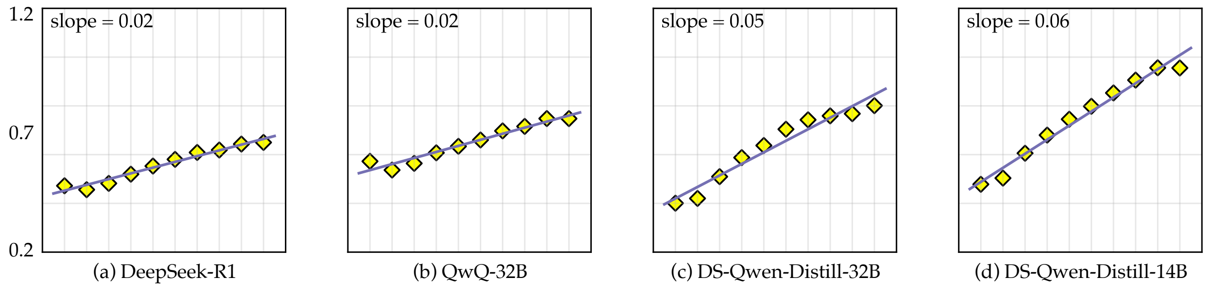


Figure 9: The ratio of segment counts across K(=10) clusters (correct vs. wrong).

clusters of higher length ranges, and strong LRMs like DeepSeek-R1 and QwQ-32B still exhibit smaller variance of ratios than other weak LRMS, as well as a more uniform total distribution and a smaller slope under linear fitting.

Moreover, the slopes for the same model vary with different settings of K. This is because the denominator in slope computation is $K-1$, while the maximum ratio difference (almost the ratio gap between cluster K and cluster 1) remains nearly constant. Despite this variation, the ratio points align well with a linear regression. The empirical design of the descending weighting of GRSP come from this observation, that is, since we use $K=5$ in GRSP by default, we set the weighting slope to 0.5, corresponding to the ratio slope of strong models, without searching over alternative weighting strategies.

F Confidence-based Segmentation on GRPO

Figure 10 illustrates the effect of confidence-based segmentation on GRPO, compared to the default keyword-based segmentation. It acquires higher accuracy in Figure 10 (a). In fact, it reaches a more surprising performance of (64.81, 3405) for accuracy and average response length than (64.63, 3427) of GRPO + GRSP + keyword-based segmentation. Note that around the last 20 steps actually witness a significant drop in response length, enabling the model to be both powerful and efficient in reasoning, which cannot be shown in Figure 10 (b) due to the curve smoothness.

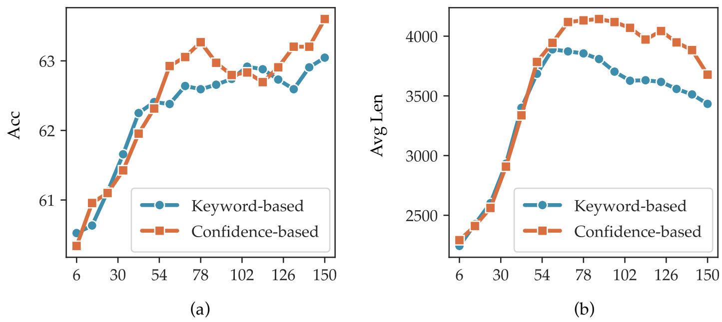


Figure 10: GRPO results. (a) Comparison of accuracy between keyword-based and confidence-based segmentation; (b) Comparison of average response length.

G Case Study

In this section, we illustrate the effect of GRSP on reasoning patterns via RL training, with two trajectories from a same question in Omni-MATH 500. They are sampled from two 14B-sized Qwen 2.5 models tuned by pure REINFORCE (Normal) and REINFORCE+GRSP (GRSP), respectively. As shown in Table 5, the **Normal** trajectory tends to explore multiple branches, often triggered by short segments like

Wait, that can't be right

Such behavior is a pattern encouraged by RLVR which increases the likelihood of solving the problem, but it also leads to the so-called overthinking.

After introducing **GRSP** supervision, each reasoning step becomes more substantial, and the overall reasoning process is direct and concise. The model successfully avoids too many unnecessary steps in **Normal**, like its final part starting with

Did I make a mistake

while arriving at the correct answer and consuming 1018 tokens, which is notably less than 1539 tokens of **Normal**. Thought switches still occur, but much less frequently.

It indicates that GRSP strikes a balance between direct reasoning and exploratory search. This balance is gradually established during RL training, corresponding to the illustrations in Figure 4: the score slightly drops at intermediate stages and then recovers as training continues, while the reasoning content can be short and efficient.

We further label each step split by the proposed Keyword-based Segmentation with alternating red and blue fonts, to clearly show the result of segmentation.

H Prompt Template

<|im_start|>system
You are a helpful assistant. You should first try a long-text process of thinking and reflection to handle the problem in the mind before each responding to the user. The thinking process are enclosed within <think>\n and </think>\n\n tags, respectively, i.e., <think>
[thinking process here]</think>
[final answer here].
<|im_end|>
<|im_start|>user
The problem text
Please reason carefully step by step, reflect thoroughly, and put the final answer in \boxed{{}}.
<|im_end|>

Figure 11: The prompt template used for SFT and RL.

Preset	**Question:** A rectangle has width x and length y. The rectangle is cut along the horizontal and vertical dotted lines to produce four smaller rectangles. The sum of the perimeters of these four rectangles is 24. What is the value of $x + y$? **Answer:** 6

GRSP
#Token 1018
#Seg 5

<think>
Alright, let's tackle this rectangle problem. So, we have a big rectangle with width x and length y. It gets cut into four smaller rectangles using some dotted lines—presumably two cuts, one horizontal and one vertical. The total perimeter of these four new rectangles adds up to 24. We need to find $x + y$.
Okay, first things first, I need to visualize this. Imagine the original rectangle. If we make one horizontal and one vertical cut, we'd end up with four smaller rectangles arranged like a grid: top-left, top-right, bottom-left, and bottom-right.
Now, each small rectangle would have its own width and length based on where those cuts are made. Let me assume the vertical cut divides the original width x into two parts: say, a and b, so $a + b = x$. Similarly, the horizontal cut divides the length y into two parts: c and d, meaning $c + d = y$.
So, the four smaller rectangles would have dimensions:
1. Top-left: width a, length c
2. Top-right: width b, length c
3. Bottom-left: width a, length d
4. Bottom-right: width b, length d
Next, I'll calculate the perimeter for each of these rectangles:
- Perimeter of top-left: $2(a + c)$
- Perimeter of top-right: $2(b + c)$
- Perimeter of bottom-left: $2(a + d)$
- Perimeter of bottom-right: $2(b + d)$
Adding them all up gives the total perimeter:

$$2(a + c) + 2(b + c) + 2(a + d) + 2(b + d)$$

Let's simplify that:

$$2a + 2c + 2b + 2c + 2a + 2d + 2b + 2d$$

Combine like terms:

$$4a + 4b + 4c + 4d$$

Factor out the 4:

$$4(a + b + c + d)$$

But wait, from earlier, $a + b = x$ and $c + d = y$. Substitute those in:

$$4(x + y)$$

According to the problem, this total perimeter equals 24:

$$4(x + y) = 24$$

Divide both sides by 4:

$$x + y = 6$$

GRSP
#Token 1018
#Seg 5

So, after working through the relationships and simplifying, it turns out that $x + y$ must be 6.

$$\boxed{6}$$

</think>
To solve the problem, consider the original rectangle with width x and length y. When cut by one horizontal and one vertical line, the rectangle is divided into four smaller rectangles. Let the vertical cut divide x into a and b such that $a + b = x$, and the horizontal cut divide y into c and d such that $c + d = y$.
The dimensions of the four smaller rectangles are:
1. Top-left: width a, length c
2. Top-right: width b, length c
3. Bottom-left: width a, length d
4. Bottom-right: width b, length d
Calculate the perimeter of each smaller rectangle:
- Perimeter of top-left: $2(a + c)$
- Perimeter of top-right: $2(b + c)$
- Perimeter of bottom-left: $2(a + d)$
- Perimeter of bottom-right: $2(b + d)$
Sum the perimeters:

$$2(a + c) + 2(b + c) + 2(a + d) + 2(b + d)$$

Simplify the expression:

$$4a + 4b + 4c + 4d = 4(a + b + c + d)$$

Substitute $a + b = x$ and $c + d = y$:

$$4(x + y)$$

Given that the total perimeter is 24:

$$4(x + y) = 24$$

Solve for $x + y$:

$$x + y = \frac{24}{4} = 6$$

Thus, the value of $x + y$ is:

$$\boxed{6}$$

Normal #Token 1539 #Seg 8	<think>

Alright, let's tackle this problem step by step. So, we have a big rectangle with width x and length y. It gets cut into four smaller rectangles using some dotted lines—presumably two cuts, one horizontal and one vertical. The total perimeter of these four smaller rectangles adds up to 24, and we need to find $x + y$.

First, I need to visualize how the big rectangle is divided. If we make one vertical cut and one horizontal cut, we'd end up with four smaller rectangles. Let me assume that the vertical cut divides the length y into two parts: y_1 and $y - y_1$. Similarly, the horizontal cut divides the width x into two parts: x_1 and $x - x_1$.

So, the four smaller rectangles would have dimensions:

1. Width x_1, length y_1
2. Width $x - x_1$, length y_1
3. Width x_1, length $y - y_1$
4. Width $x - x_1$, length $y - y_1$

Now, let's calculate the perimeter for each smaller rectangle:

1. Perimeter of first rectangle: $2(x_1 + y_1)$
2. Perimeter of second rectangle: $2((x - x_1) + y_1)$
3. Perimeter of third rectangle: $2(x_1 + (y - y_1))$
4. Perimeter of fourth rectangle: $2((x - x_1) + (y - y_1))$

Adding all these up gives the total perimeter:

$$2(x_1 + y_1) + 2((x - x_1) + y_1) + 2(x_1 + (y - y_1)) + 2((x - x_1) + (y - y_1)) = 24$$

Let me simplify this expression step by step.

First, distribute the 2s:

$$2x_1 + 2y_1 + 2x - 2x_1 + 2y_1 + 2x_1 + 2y - 2y_1 + 2x - 2x_1 + 2y - 2y_1 = 24$$

Now, combine like terms:

- For x_1: $2x_1 - 2x_1 + 2x_1 - 2x_1 = 0$
- For y_1: $2y_1 + 2y_1 - 2y_1 - 2y_1 = 0$
- For x: $2x + 2x = 4x$
- For y: $2y + 2y = 4y$

Wait, that can't be right because everything cancels out except the constants, which doesn't help us find $x + y$. Did I make a mistake?

Let me go back and check my distribution:

Original equation after distributing:

$$2x_1 + 2y_1 + 2x - 2x_1 + 2y_1 + 2x_1 + 2y - 2y_1 + 2x - 2x_1 + 2y - 2y_1 = 24$$

Hmm, maybe I missed something. Let's group similar terms differently:

$$(2x_1 - 2x_1 + 2x_1 - 2x_1) + (2y_1 + 2y_1 - 2y_1 - 2y_1) + (2x + 2x) + (2y + 2y) = 24$$

Which simplifies to:

$$0 + 0 + 4x + 4y = 24$$

So,

$$4x + 4y = 24$$

Normal #Token 1539 #Seg 8	Divide both sides by 4: $$x + y = 6$$ Oh! So, despite the intermediate confusion, it turns out that the sum $x + y$ is indeed 6. That makes sense because the way the rectangle is cut doesn't affect the total perimeter addition when simplified correctly. Therefore, the value of $x + y$ is $\boxed{6}$. </think> To solve the problem, we start by visualizing the division of the rectangle. The original rectangle with width x and length y is cut into four smaller rectangles using one vertical and one horizontal cut. Let the vertical cut divide the length y into y_1 and $y - y_1$, and the horizontal cut divide the width x into x_1 and $x - x_1$. The dimensions of the four smaller rectangles are: 1. $x_1 \times y_1$ 2. $(x - x_1) \times y_1$ 3. $x_1 \times (y - y_1)$ 4. $(x - x_1) \times (y - y_1)$ The perimeter of each smaller rectangle is: 1. $2(x_1 + y_1)$ 2. $2((x - x_1) + y_1)$ 3. $2(x_1 + (y - y_1))$ 4. $2((x - x_1) + (y - y_1))$ Summing these perimeters: $$2(x_1 + y_1) + 2((x - x_1) + y_1) + 2(x_1 + (y - y_1)) + 2((x - x_1) + (y - y_1)) = 24$$ Simplify the expression: $$2x_1 + 2y_1 + 2x - 2x_1 + 2y_1 + 2x_1 + 2y - 2y_1 + 2x - 2x_1 + 2y - 2y_1 = 24$$ Combine like terms: $$(2x_1 - 2x_1 + 2x_1 - 2x_1) + (2y_1 + 2y_1 - 2y_1 - 2y_1) + (2x + 2x) + (2y + 2y) = 24$$ This simplifies to: $$0 + 0 + 4x + 4y = 24$$ Divide both sides by 4: $$x + y = 6$$ Thus, the value of $x + y$ is $\boxed{6}$.

Table 5: Case demonstrations. The alternating red and blue fonts label seperate reasoning steps.