Beyond Token Length: Step Pruner for Efficient and Accurate Reasoning in Large Language Models

Anonymous ACL submission

Abstract

Large Reasoning Models (LRMs) demonstrate strong performance on complex tasks but often suffer from excessive verbosity, known as "overthinking." Existing solutions via reinforcement learning (RL) typically penalize generated tokens to promote conciseness. However, these methods encounter two challenges: responses with fewer tokens do not always correspond to fewer reasoning steps, and models may develop hacking behavior in later stages of training by discarding reasoning steps to minimize token usage. In this work, we introduce **Step Pruner (SP)**, an RL framework that steers LRMs toward more efficient reasoning by favoring compact reasoning steps. Our step-aware reward function prioritizes correctness while imposing penalties for redundant steps, and withholds rewards for incorrect responses to prevent the reinforcement of erroneous reasoning. Moreover, we propose a dynamic stopping mechanism to prevent hacking behavior caused by step merging. Extensive experiments across four reasoning benchmarks demonstrate that SP achieves state-of-the-art accuracy while significantly reducing response length. For instance, on AIME24, SP reduces token usage by **69.7%**.

1 Introduction

Reasoning is a fundamental strength of large language models (LLMs), empowering them to address complex tasks that require logical deduction and multi-step inference. Since the advent of Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022; Yao et al., 2023), a wide array of techniques has emerged to further enhance the reasoning capabilities of LLMs. Recent models such as DeepSeek-R1 (Guo et al., 2025), GPT-o1 (Jaech et al., 2024), Gemini 2.5 (Comanici et al., 2025), and QwQ (Team, 2025) have ushered in a new era of LRMs, which exhibit sophisticated reasoning abilities even in the absence of explicit step-by-step prompts. These LRMs excel in challenging domains, such as mathematical problem-solving and code generation. However, recent research has highlighted a growing inefficiency in the reasoning process of LRMs.

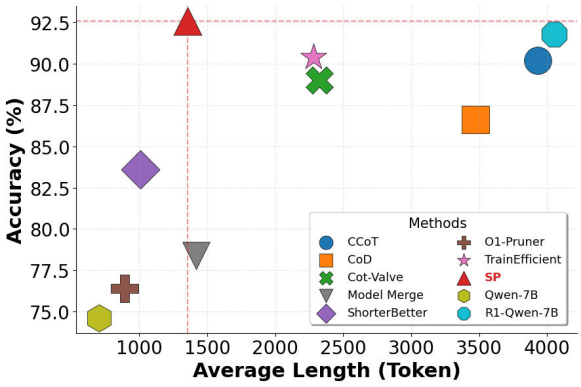

Figure 1: Comparison of Step Pruner with seven baselines on the MATH500 dataset. SP uses only 33% of the tokens compared to DeepSeek-R1-Distilled-Qwen-7B, and maintains the same level of accuracy.

For simple tasks such as "What is $2 + 3$?" (Chen et al., 2024), LRMs can sometimes generate excessively verbose responses, spanning thousands of tokens or falling into cycles of self-doubt, a phenomenon known as *overthinking* (Sui et al., 2025). This inefficiency not only leads to increased computational costs (Aytes et al., 2025), but also introduces the risk of compounding errors through unnecessary self-reflection (Cuadron et al., 2025; Su et al., 2025). In extreme cases, LRMs may even enter infinite reasoning loops, ultimately exhausting their token budgets.

To mitigate overthinking, recent studies have proposed several strategies, such as prompting large models to generate concise outputs (Renze and Guven, 2024; Xu et al., 2025) or capping token usage at a predefined limit (Han et al., 2024). However, these approaches have shown limited effectiveness for LRMs. Supervised fine-tuning (SFT) methods, which use datasets with shorter reasoning, aim to

promote brevity. In contrast, RL-based methods offer a more flexible solution by jointly optimizing for both accuracy and output brevity. State-of-the-art RL strategies (Luo et al., 2025a; Yi et al., 2025; Arora and Zanette, 2025; Hou et al., 2025) typically penalize token usage while rewarding correctness, thus encouraging models to generate shorter yet accurate answers. For example, o1-pruner (Luo et al., 2025a) introduces an additional reward term by dividing the average length of responses by the current response length, encouraging shorter responses. ShorterBetter (Yi et al., 2025) treats the shortest correct response length as optimal and penalizes answers longer than this optimal length.

Despite significant progress, token-based RL faces two main issues: First, fewer tokens do not necessarily mean fewer reasoning steps. Second, in the later stages of training, token-based penalties may incentivize the model to "hack" the reward by outputting only the final answer, omitting the reasoning steps entirely. Inspired by these findings, we propose a novel method, **Step Pruner (SP)**, which shifts the focus from reducing token count to reducing the number of steps. SP splits the response into distinct steps and penalizes redundant steps, rather than penalizing token units. Our RL-based reward function prioritizes correctness while penalizing redundant steps, thus promoting concise and clear responses. To ensure reliability, the reward for incorrect answers is suppressed, and no step reward is given if the entire response is incorrect. This approach effectively prevents the model from generating shorter but incorrect outputs. SP also reveals a clear two-phase RL dynamic: 1) redundant steps are first removed, and 2) later, logically related steps begin merging into more compact reasoning units (e.g., collapsing two steps into one paragraph). Finally, we dynamically stop the training by monitoring the model's average output length during the training process. When the model's output no longer shortens, we halt the training. This design enables concise, reliable, and structured reasoning while avoiding model hacking rewards.

Our extensive experiments demonstrate that the SP significantly enhances reasoning efficiency. On multiple benchmarks, SP utilizes only 44% of the tokens compared to the baselines, while either maintaining or improving accuracy. As shown in Figure 1, SP achieves the best balance between accuracy and token efficiency on the MATH-500 dataset. We also explore alternative segmentation strategies and find that paragraph-based segmentation offers the most effective trade-off between brevity and performance. Additionally, we apply LLM-as-judge for semantic analysis of model outputs before and after training, revealing that the trained model explores fewer redundant paths, resulting in a higher proportion of pivotal reasoning. Our main contributions are summarized as follows:

1. We propose **Step Pruner (SP)**, a reinforcement learning framework that directs LRMs to enhance reasoning efficiency by penalizing redundant steps, and introduce a dynamic stopping mechanism to prevent reward hacking.

2. We systematically compare various segmentation strategies and find that paragraph-based segmentation strikes the best balance between compression ratio and model performance.

3. Extensive experiments across four reasoning benchmarks demonstrate that SP achieves the best trade-off between accuracy and response length, outperforming existing baselines.

2 Related Work

2.1 Reasoning in LLMs

Large language models (LLMs) have demonstrated remarkable capabilities in solving complex reasoning tasks, particularly when guided by carefully designed prompting strategies. A key advancement in this domain is Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2022), which breaks down intricate problems into a series of intermediate reasoning steps. To further enhance these abilities, leading LRMs such as OpenAI-O1 (Jaech et al., 2024), DeepSeek-R1 (Guo et al., 2025), and QwQ (Team, 2025) integrate advanced techniques like reinforcement learning and multi-stage training. These strategies enable the generation of detailed, multi-step reasoning, achieving state-of-the-art performance in fields such as advanced mathematics and competitive programming.

2.2 Overthinking in LRMs

As LRMs advance, the issue of overthinking has become more prominent (Sui et al., 2025; Feng et al., 2025). The reasoning process in these models has grown increasingly verbose, and in some cases, has led to critical issues like infinite repetition. For instance, when answering a simple question like "What is 2 plus 3?" (Chen et al., 2024),

certain reasoning models, particularly smaller ones, may generate reasoning sequences containing thousands of tokens. This verbosity significantly increases the computational cost and latency of reasoning, limiting the practical deployment of these models in computation-sensitive real-world applications (Aytes et al., 2025). In more severe cases, excessive reasoning steps can introduce errors or obscure logical clarity, leading to incorrect answers (Fatemi et al., 2025). Beyond a certain threshold, increasing the length of reasoning does not necessarily enhance performance and can even degrade accuracy, as the model may misjudge the complexity of the question and introduce compounded errors (Su et al., 2025).

2.3 Efficient Reasoning

As reasoning in LRMs becomes more verbose, recent works have focused on shortening reasoning while preserving the quality of the generated output. Prompt-based methods, such as CCoT (Renze and Guven, 2024), modify user prompts by incorporating directives like "Be concise," thereby encouraging the model to generate shorter responses. CoD (Xu et al., 2025) and Token-budget (Han et al., 2024) introduce a token limit on the model's output, effectively capping the number of tokens generated and preventing excessively long responses. SFT-based methods, including C3ot (Kang et al., 2025), Cot-Valve (Ma et al., 2025) and TokenSkip (Xia et al., 2025), focus on creating diverse training datasets with reasoning of varying lengths, with a particular emphasis on shorter reasoning. Fine-tuning the model on these datasets helps it generate more concise reasoning. Merge-based approaches (Wu et al., 2025) reduce reasoning lengths by merging the parameters of a reasoning model with those of a pre-trained instruction model, enhancing efficiency without requiring additional training. Finally, RL-based methods like DAST (Shen et al., 2025), O1-pruner (Luo et al., 2025a), ShorterBetter (Yi et al., 2025), and Train-Efficient (Arora and Zanette, 2025) introduce a length penalty during training, rewarding models for generating responses with fewer tokens while maintaining accuracy. These techniques optimize reasoning, making them shorter and more efficient in problem-solving.

3 Method

This paper aims to enhance LRM efficiency by introducing the optimal number of steps as a principled metric, rather than directly penalizing response length. Section 3.1 formalizes the objective and notation. Section 3.2 presents our approach for computing optimal steps. Section 3.3 discusses reward configurations balancing correctness and segmentation. Section 3.4 details policy optimization via Group Relative Policy Optimization (GRPO).

3.1 Problem Setup

Let $\mathcal{X}$ denote the input space (e.g., prompts), and $\mathcal{Y}$ the output space (e.g., responses). Given $x \in \mathcal{X}$, a large reasoning model π_θ defines a conditional distribution over output sequences. For a response $\mathbf{y} = (y_1, \ldots, y_m) \in \mathcal{Y}$, generation proceeds autoregressively:

$$\pi_\theta(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{m} \pi_\theta(y_j \mid \mathbf{x}, \mathbf{y}_{<j}), \tag{1}$$

where $\mathbf{y}_{<j} = (y_1, \ldots, y_{j-1})$.

For each input x, let $\mathbf{y}^* \in \mathcal{Y}$ denote the ground-truth answer. The standard objective is to maximize expected correctness:

$$\max_\theta \ \mathbb{E}_{x\sim\mathcal{D}} \, \mathbb{E}_{\mathbf{y}\sim\pi_\theta(\cdot|x)} \left[R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) \right], \tag{2}$$

where R_{acc} measures correctness and $\mathcal{D}$ is the data distribution.

In the context of efficient reasoning, we further encourage the model to generate correct answers with minimal verbosity. To achieve this, we introduce a cost function $C(\mathbf{y})$ reflecting response inefficiency. The optimization objective becomes:

$$\max_\theta \ \mathbb{E}_{x\sim\mathcal{D}} \, \mathbb{E}_{\mathbf{y}\sim\pi_\theta(\cdot|x)} \left[R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) - \lambda \cdot C(\mathbf{y}) \right], \tag{3}$$

where $\lambda > 0$ is a hyperparameter that balances accuracy and efficiency. By designing $C(\mathbf{y})$ appropriately, the model is incentivized to produce responses that are both accurate and concise.

3.2 Rethinking Length-based Rewards

Recent RL methods for efficient reasoning often encourage conciseness by penalizing the length of model outputs, assuming that shorter responses correspond to more efficient reasoning. However, this assumption does not always hold: longer responses may contain fewer reasoning steps while presenting a more complete thought process, whereas shorter

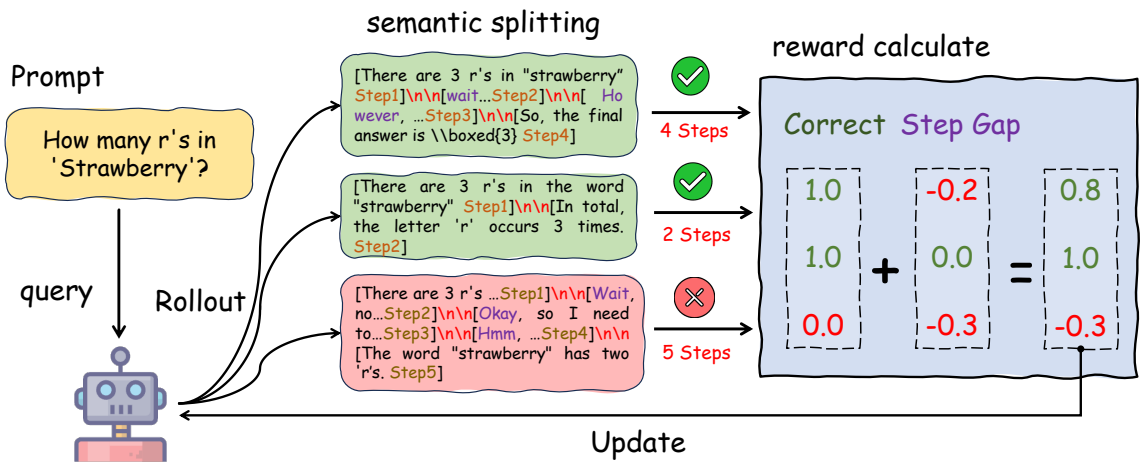

Figure 2: **Pipeline of Step Pruner**: (1) Prompt LRMs to generate multiple responses; (2) Split each response into logical steps; (3) Calculate a reward score based on both correctness and the number of steps; (4) Update LRMs using GRPO.

responses may actually involve more reasoning steps. Moreover, directly penalizing token count can lead to "hacking" phenomena, where the model entirely discards the reasoning process in the later stages of training.

To address these limitations, we propose a step-based reward, which replaces the traditional token penalty with the number of reasoning steps in the response. In practice, we approximate the number of reasoning steps using a simple **paragraph-based** approach (i.e., separated by \n\n), where each paragraph serves as a proxy for a reasoning step (Zhang et al., 2025). The advantage of step-based rewards is that they explicitly indicate the minimal reasoning unit to the model, guiding it to optimize output step by step. Furthermore, step-based training makes it straightforward to identify the appropriate stopping point during training.

3.3 Reward Function

We design a reward function that jointly accounts for both response correctness and steps efficiency. The reward consists of two main components: an *accuracy reward* R_{acc} and a *step reward* R_{seg}.

Accuracy Reward. Given an input x with reference answer $\mathbf{y}^*$, we define the accuracy reward for a generated response $\mathbf{y}$ as a binary indicator:

$$R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = \mathbb{I}[\mathbf{y} = \mathbf{y}^*], \tag{4}$$

where $\mathbb{I}[\cdot]$ is the indicator function, returning 1 for an exact match, and 0 otherwise.

Step Reward. For each input x, we consider n candidate responses $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$. Among correct responses ($R_{\text{acc}}(\mathbf{y}_i, \mathbf{y}^*) = 1$), we define the optimal number of steps as:

$$S^* = \min_{i:R_{\text{acc}}(\mathbf{y}_i,\mathbf{y}^*)=1} S(\mathbf{y}_i), \tag{5}$$

where $S(\mathbf{y})$ is the step count in $\mathbf{y}$. S^* is thus the minimal number of steps needed for correctness.

To avoid rewarding the model for incorrect answers, we mask out all step rewards for incorrect responses. We then distinguish four cases based on correctness and the number of steps:

1. **Correct, more steps:** If $R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = 1$ and $S(\mathbf{y}) > S^*$, we penalize the excess steps. The penalty for excess steps is given by:

$$R_{\text{seg}}(\mathbf{y}, S^*) = -(S(\mathbf{y}) - S^*), \tag{6}$$

2. **Correct, fewer steps:** This case is excluded by definition, as the number of steps $S(\mathbf{y})$ should always meet or exceed the optimal number S^* for correctness.
3. **Incorrect, more steps:** If $R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = 0$ and $S(\mathbf{y}) > S^*$, we penalize both the error and the excess steps. The penalty is defined as:

$$R_{\text{seg}}(\mathbf{y}, S^*) = -(S(\mathbf{y}) - S^*), \tag{7}$$

4. **Incorrect, fewer steps:** If $R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = 0$ and $S(\mathbf{y}) < S^*$, we penalize the error, but no reward is given for brevity. This case is handled as:

$$R_{\text{seg}}(\mathbf{y}, S^*) = 0, \tag{8}$$

Based on the four cases above, the step reward for a response $\mathbf{y}$ can be defined as follows:

$$R_{\text{seg}}(\mathbf{y}, S^*) = \begin{cases} -(S(\mathbf{y}) - S^*) & \text{if } R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = 1, \\ -\max(0, S(\mathbf{y}) - S^*) & \text{if } R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) = 0, \end{cases} \tag{9}$$

For correct responses, we penalize any excess in the number of steps beyond S^*. For incorrect responses, we penalize only if the number of steps exceeds S^*, but do not reward responses with fewer steps.

Final Reward. The total reward for a response $\mathbf{y}$ is defined as:

$$R(\mathbf{y}, \mathbf{y}^*, S^*) = R_{\text{acc}}(\mathbf{y}, \mathbf{y}^*) + \beta \cdot R_{\text{seg}}(\mathbf{y}, S^*), \tag{10}$$

where $\beta > 0$ is a hyperparameter that controls the penalty for exceeding the optimal number of steps.

Handling All-Incorrect Cases. To prevent the model from learning to generate incorrect answers, we skip training updates when all candidate responses for input x are incorrect.

Stopping Criterion. During training, we find that step-based reward optimization often progresses in two stages. First, the model steadily shortens its outputs as intended. Then, it begins merging multiple steps into single paragraphs in an attempt to exploit the reward. To avoid this issue, we adopt a simple stopping rule: we stop training once the model's outputs stop shortening. Further details are provided in Section 5.2.

3.4 Policy Optimization with GRPO

We optimize the model using the GRPO algorithm (Shao et al., 2024). For each input x, we sample a group of n candidate responses $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ from the current policy π_θ and compute their total rewards $R(\mathbf{y}_j, \mathbf{y}^*, S^*)$. These rewards are then normalized within the group to obtain advantage estimates $\hat{A}_j$.

We maintain a reference policy $\pi_{\theta_{\text{ref}}}$. For each response prefix $\mathbf{y}_{j,1:k}$, we compute likelihoods under both π_θ and $\pi_{\theta_{\text{ref}}}$. The GRPO objective encourages higher probability for high-advantage responses while penalizing divergence from the reference policy:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{n}\sum_{j=1}^{n}\frac{1}{t_j}\sum_{k=1}^{t_j}\Big[W\big(\hat{A}_j, r_{j,k}, r_{j,k}^{\text{ref}}, \epsilon\big) - \gamma \cdot \widetilde{\text{KL}}\big(r_{j,k} \,\|\, r_{j,k}^{\text{ref}}\big)\Big], \tag{11}$$

where t_j is the length of the j-th response, $r_{j,k} = \pi_\theta(\mathbf{y}_{j,1:k} \mid x)$, $r_{j,k}^{\text{ref}} = \pi_{\theta_{\text{ref}}}(\mathbf{y}_{j,1:k} \mid x)$, $W(\cdot)$ is a clipped policy improvement term, and $\widetilde{\text{KL}}$ is an approximate KL-divergence penalty. Hyperparameters ϵ and γ control clipping and regularization strength.

4 Experimental Setup

4.1 Datasets

We use DeepScaleR-preview (Luo et al., 2025b) as our training dataset, consisting of 40K math problems (high school to Olympiad level). We evaluate SP on four reasoning benchmarks: AIME24, MATH500, GSM8K, and GPQA. AIME24 contains 30 high-school multi-step problems from the 2024 American Invitational Mathematics Examination. MATH500 is a subset of 500 challenging problems from the MATH dataset (Lightman et al., 2023). GSM8K (Cobbe et al., 2021) tests grade-school multi-step arithmetic and algebra. GPQA (Rein et al., 2024) focuses on graduate-level reasoning across diverse fields, requiring true multi-step inference rather than factual retrieval.

4.2 Model

We use DeepSeek-R1-Distill-Qwen-2.5-7B and DeepSeek-R1-Distill-Qwen-2.5-1.5B (Guo et al., 2025) (abbreviated as R1-Qwen-7B and R1-Qwen-1.5B) as our base models. It is trained by distilling the reasoning traces of DeepSeek-R1 from Qwen-2.5-Math-7B and Qwen-2.5-Math-1.5B (Yang et al., 2024).

4.3 Baselines

We compare SP with seven recent approaches for efficient reasoning, grouped into four categories:

Prompt Engineer. Chain of Draft (CoD) (Xu et al., 2025) and Concise Chain of Thought (CCoT) (Renze and Guven, 2024) promote brevity via prompting, either by guiding models to generate shorter reasoning or by drafting concise initial reasoning with optional elaboration.

SFT. CoT-Valve (Ma et al., 2025) constructs CoT dataset ranging from short to long by learning a tunable direction in the parameter space, and trains short-thinking models using datasets with shorter CoT.

Model-merging. Long-to-Short Reasoning (Wu et al., 2025) interpolates between fast and accurate models to balance efficiency and performance.

RL. O1-Pruner (Luo et al., 2025a), ShorterBetter (Yi et al., 2025), and TrainEfficient (Arora and Zanette, 2025) use RL to optimize both answer correctness and brevity, where token length is included as a penalty term in the reward function.

4.4 Evaluations

We evaluate our models on each benchmark using three metrics: (1)**Accuracy**: The proportion of correctly answered questions, directly measuring solution correctness. (2) **Response Length**: The average number of tokens per response, indicating the conciseness and efficiency of the model's reasoning. (3) **Accuracy-Efficiency Score** (AES): A composite metric (Luo et al., 2025a) that assigns higher scores to models with greater accuracy and shorter outputs (See the Appendix A.2 for details).

4.5 Implementation Details

We utilized a batch size of 128 and generated 4 responses per input with a temperature of 0.9, truncating outputs at 8,000 tokens. The KL-divergence hyperparameter γ was set to 0.001, the learning rate was set to 0.001, and in the reward function, the hyperparameter β was set to 0.01. For evaluation, we employed the lighteval (Habib et al., 2023) toolkit, with a temperature of 0.6.

5 Experimental Result

5.1 Main Results

Table 1 summarizes the performance of SP and seven recent baselines across four benchmarks, for both 7B and 1.5B model sizes.

Accuracy. SP achieves consistently strong or best-in-class accuracy across benchmarks and model sizes while markedly shortening responses. On 7B models, SP attains the top performance on MATH500 and GSM8K and remains competitive on GPQA and AIME24. On 1.5B models, SP outperforms all baselines on GPQA: Diamond, MATH500, and GSM8K, and remains highly competitive on AIME24, showing that its advantages hold even at smaller scales.

Efficiency. SP produces much shorter outputs than the R1-Qwen baselines at both model sizes. For instance, on AIME24 with the 7B model, SP reduces average output length by 70%, and by 67% on MATH500, without compromising accuracy. Similar reductions appear for the 1.5B model. Compared with RL-based baselines such as O1-Pruner and ShorterBetter, SP attains comparable or higher accuracy with only moderately longer responses, offering a more favorable balance between brevity and correctness.

Accuracy-Efficiency Tradeoff. The AES metric summarizes the balance between correctness and conciseness. SP attains the highest AES on most benchmarks for both model sizes, reflecting its superior trade-off. Although some baselines like O1-Pruner and Model Merge yield shorter outputs, their substantial accuracy losses reduce overall AES. Prompt-based and SFT-based approaches provide moderate brevity but fall short of SP in combined performance.

5.2 When to Stop Training to Prevent Hacks

By monitoring various curves during training, we identified the point at which training can be stopped to prevent hacking. As shown in Figure 3, the average number of paragraphs consistently decreased while the average paragraph length increased, and the total token count initially decreased before stabilizing. This pattern reveals two phases in the training process: (1) the paragraph count decreases while paragraph length grows slightly, causing an overall decline in total output length, and (2) paragraph length increases more substantially, the paragraph count levels off, and the total response length stabilizes. Based on these observations, training can be halted once the total output token count ceases to decrease and reaches convergence, effectively preventing hacking.

5.3 Different Segmentations

In addition to basic paragraph-level segmentation, we explore three finer-grained approaches: (1) sentence-level segmentation using NLTK (Bird, 2006), (2) conjunction-based segmentation (e.g., "wait", "alternative"; full list in Appendix A.3), and (3) paragraph merging based on embedding similarity, using `all-MiniLM-L6-v2` with a cosine similarity threshold of 0.5.

As shown in Figure 4, segmentation based on paragraph similarity results in the smallest output length reduction, stabilizing around 2,500 tokens. Token-level segmentation achieves the greatest reduction (below 1,000 tokens) but significantly impacts accuracy. Paragraph-based segmentation strikes a balance, moderately reducing length while preserving higher accuracy. In summary, finer-grained segmentation reduces length more effectively but typically lowers accuracy, whereas coarser segmentation better preserves accuracy with less reduction in length.

5.4 Ablation Studies

Table 2 summarizes the ablation results across three mathematical datasets. Removing the correctness

Method	AIME 24			GPQA:Diamond			MATH500			GSM8K		
	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES
7B Model												
Qwen-7B	10.0	1309	–	30.3	724	–	74.6	704	–	81.8	254	–
R1-Qwen-7B	53.3	14839	0.00	52.0	8195	0.00	91.8	4053	0.00	85.6	508	0.00
CCoT	50.0	13507	-0.22	51.0	8213	-0.10	90.2	3931	-0.06	85.2	512	-0.03
CoD	43.3	13490	-0.85	51.5	7761	0.00	86.6	3475	-0.14	84.4	463	0.02
Cot-Valve	46.7	10731	-0.34	45.0	5386	-0.33	89.0	2924	0.13	84.8	439	0.09
Model Merge	26.7	5964	-1.90	41.9	4425	-0.51	78.2	1416	-0.09	81.6	636	-0.49
O1-Pruner	26.7	2574	-1.67	33.3	2180	-1.06	76.4	993	-0.08	80.2	130	0.43
ShorterBetter	43.3	4939	-0.27	45.0	4326	-0.20	90.4	1086	0.65	81.6	201	0.37
TrainEfficient	53.3	9798	0.34	52.0	6750	0.18	91.6	2403	0.40	85.6	227	0.55
SP	50.0	4502	0.39	51.5	3925	0.47	92.0	1353	0.67	89.0	344	0.44
1.5B Model												
Qwen-1.5B	0.0	5452	–	31.8	1174	–	51.8	1134	–	64.7	296	–
R1-Qwen-1.5B	33.3	14783	0.00	33.8	9896	0.00	83.8	5794	0.00	77.6	1157	0.00
CCoT	26.7	16654	-1.12	37.9	9065	0.45	82.4	4438	0.15	70.0	419	0.15
CoD	33.3	15234	-0.03	38.9	8592	0.58	79.0	3935	0.03	70.6	457	0.15
Cot-Valve	23.3	8736	-1.09	30.8	6291	-0.08	75.8	3412	0.03	74.8	703	0.15
Model Merge	0.0	3977	-4.27	21.9	2425	-1.01	12.2	3890	-3.94	26.6	3428	-5.25
O1-Pruner	23.3	3061	-0.71	32.9	1887	0.68	62.4	1002	-0.45	64.7	112	0.07
ShorterBetter	23.3	2908	-0.70	35.4	2083	0.93	68.2	1029	-0.11	62.8	105	-0.04
TrainEfficient	30.0	10682	-0.22	37.4	7114	0.60	79.2	2279	0.33	70.2	134	0.41
SP	30.0	5179	0.15	41.9	4446	1.27	86.0	1779	0.77	80.9	563	0.64

Table 1: Accuracy (%), response lengths (Len), and AES scores on AIME, GPQA, MATH500, and GSM8K for both 7B and 1.5B models across different baselines. We highlight the best results and the second-best results.

Dataset	Metric	-CR	-COS	-WRM	-SAW
AIME	Acc.	0.0	36.7	33.3	36.7
	Len.	542	4379	5107	5518
MATH	Acc.	8.8	83.4	88.0	78.4
	Len.	348	1134	1139	1224
GSM8K	Acc.	22.5	85.8	86.5	84.2
	Len.	105	319	344	361

Table 2: Ablation study on mathematical tasks. "-" denotes removal of the component. **CR**: Correct Reward; **COS**: Correct Optimal Step; **WRM**: Wrong Response MASK; **SAW**: Skip All Wrong.

reward (-CR) causes a clear drop in accuracy, showing that explicit correctness signals are essential for effective policy learning. Prioritizing only shorter answers (-COS) also harms accuracy, highlighting the need to balance brevity with correctness. Without masking wrong answers (-WRM), the model tends to generate shorter yet incorrect outputs, reducing both length quality and accuracy. Eliminating the skip-all-wrong mechanism (-SAW) further degrades performance, indicating the importance of filtering uninformative trajectories. Overall, each component contributes meaningfully to accuracy and stability, and their combination is key to achieving a sound trade-off between correctness and conciseness.

5.5 Reasoning Analysis

In this section, we analyze how SP reshapes the stylistic profile of model-generated reasonings. We prompted Gemini 2.5 to label each sentence in the R1-Qwen-7B and SP-7B outputs on the MATH500 and AIME24 according to five reasoning categories:

- ➢ **Pivotal Reasoning:** Core steps that directly advance the solution.
- ➢ **Productive Elaboration & Calculation:** Sup-

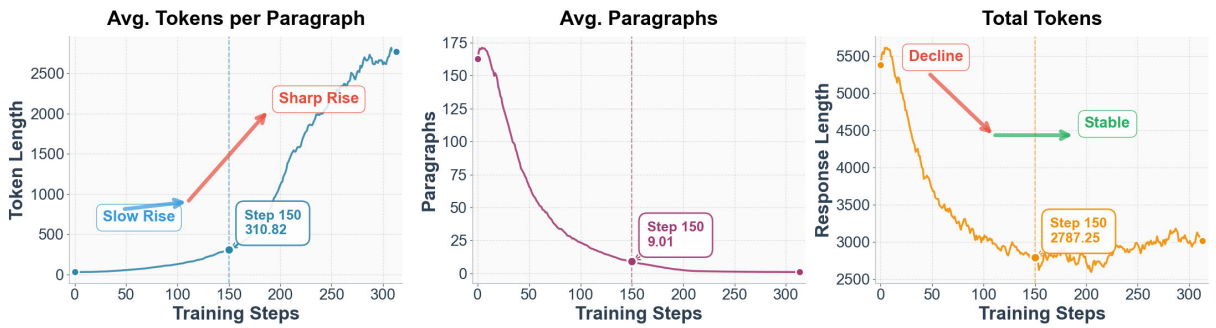

Figure 3: Training curves showing paragraph length, paragraph count, and total response length over training steps.

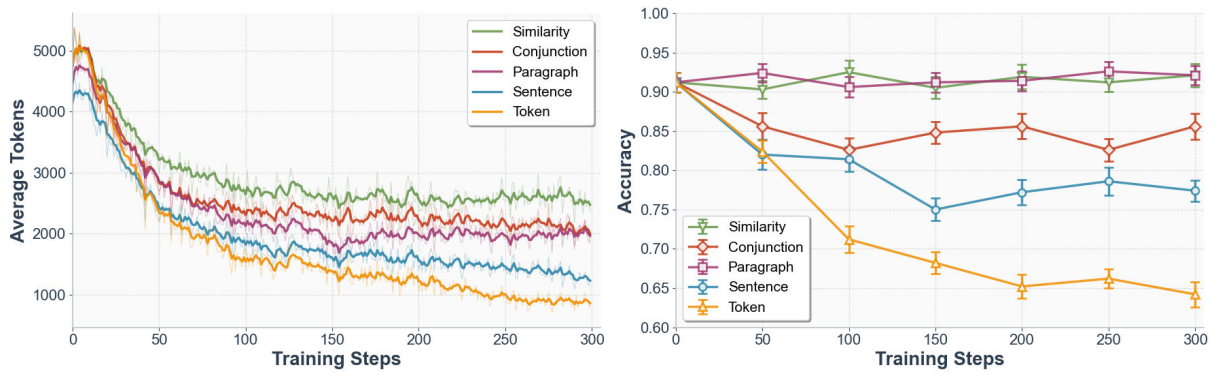

Figure 4: Comparison of different segmentation methods during training: output length (left) and accuracy change (right) on MATH500. The paragraph-based segmentation achieved the best trade-off between accuracy and output length.

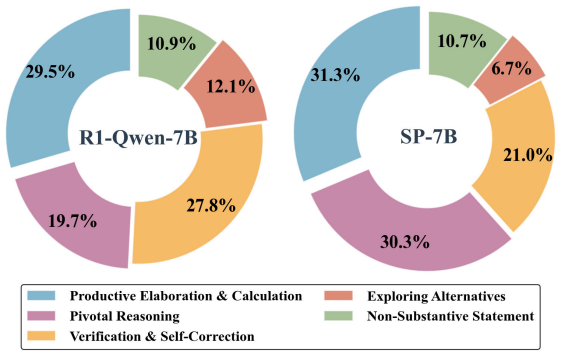

Figure 5: The Semantic Analysis of Reasoning in R1-Qwen-7B and SP-7B.

porting explanations or detailed arithmetic.

- ➢ **Exploring Alternatives:** Consideration of different methods or perspectives.
- ➢ **Verification & Self-Correction:** In-line checks or corrections of prior steps.
- ➢ **Non-Substantive Statements:** Remarks that do not contribute to the logical solution path.

Figure 5 shows clear shifts in category frequencies. Compared with R1-Qwen-7B, SP-7B produces substantially more Productive Elaboration & Calculation and Pivotal Reasoning, indicating a stronger focus on detailed computation and essential logical steps. In contrast, Exploring Alternatives and Verification & Self-Correction decline, suggesting fewer digressions and less frequent self-checking. The share of Non-Substantive Statements remains largely unchanged. Overall, SP-7B encourages more concentrated, substantive, and goal-oriented reasoning.

6 Conclusion

This paper introduces Step Pruner (SP), a reinforcement learning framework that enhances the efficiency of LRMs by reducing unnecessary reasoning steps. Unlike methods that directly penalize output length, SP employs a step-based reward mechanism to ensure concise and accurate answers. Experiments on four benchmark tasks show that SP significantly shortens output length while maintaining logical coherence and achieving state-of-the-art accuracy. Further analysis confirms the effectiveness of the step-based reward and each reward component. SP thus provides a simple and robust solution for efficient reasoning in LRMs, offering a practical approach to reduce computational cost while preserving accuracy and logical consistency.

Limitations

While Step Pruner effectively reduces redundant reasoning steps and overall response length, it may inadvertently encourage the model to merge distinct logical steps into overly long paragraphs, potentially compromising interpretability and readability. Additionally, the reliance on paragraph-based segmentation may not capture fine-grained reasoning boundaries in all cases, especially for tasks requiring nuanced step differentiation. Finally, SP's effectiveness is contingent on accurate evaluation of correctness; in domains with ambiguous or subjective answers, reward assignment may be less reliable, limiting generalizability.

References

Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *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. *arXiv preprint arXiv:2503.05179*.

Steven Bird. 2006. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 interactive presentation sessions*, pages 69–72.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, and 1 others. 2025. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *arXiv preprint arXiv:2502.08235*.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. 2025. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*.

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

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

Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. 2023. Lighteval: A lightweight framework for llm evaluation.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2025. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24312–24320.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

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. *arXiv preprint arXiv:2501.12570*.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, and 1 others. 2025b. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*.

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

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

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

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

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

Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. 2025. Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms. *arXiv preprint arXiv:2505.00127*.

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

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

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*.

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.

Han Wu, Yuxuan Yao, Shuqi Liu, Zehua Liu, Xiaojin Fu, Xiongwei Han, Xing Li, Hui-Ling Zhen, Tao Zhong, and Mingxuan Yuan. 2025. Unlocking efficient long-to-short llm reasoning with model merging. *arXiv preprint arXiv:2503.20641*.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, 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. *arXiv preprint arXiv:2502.18600*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

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.

Jingyang Yi, Jiazheng Wang, and Sida Li. 2025. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. *arXiv preprint arXiv:2504.21370*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

A Appendix

A.1 Experimental environment

We trained on the NVIDIA 8*H200 node, with CUDA version 12.7 and CPU model Intel(R) Xeon(R) Platinum 8468V.

A.2 Accuracy-Efficiency (AE) Score: A Comprehensive Definition

AES quantifies the trade-off between a model's output length and its accuracy. It evaluates whether a model can reduce the length of its responses without sacrificing accuracy. The AE Score is defined as:

$$\text{AE Score} = \begin{cases} \varphi \cdot \Delta\text{Length} + \eta \cdot |\Delta\text{Acc}|, & \text{if } \Delta\text{Acc} \geq 0 \\ \varphi \cdot \Delta\text{Length} - \theta \cdot |\Delta\text{Acc}|, & \text{if } \Delta\text{Acc} < 0 \end{cases} \tag{12}$$

In this equation, ΔLength and ΔAcc represent the percentage changes in output length and accuracy, respectively, between the evaluated model and its baseline. They are defined as follows:

$$\Delta\text{Length} = \frac{\text{Length}_{\text{baseline}} - \text{Length}_{\text{model}}}{\text{Length}_{\text{baseline}}} \tag{13}$$

$$\Delta\text{Acc} = \frac{\text{Acc}_{\text{model}} - \text{Acc}_{\text{baseline}}}{\text{Acc}_{\text{baseline}}} \tag{14}$$

Here, $\text{Length}_{\text{baseline}}$ and $\text{Acc}_{\text{baseline}}$ denote the output length and accuracy of the baseline model, while $\text{Length}_{\text{model}}$ and $\text{Acc}_{\text{model}}$ refer to those of the evaluated model. A positive ΔLength indicates a reduction in output length, and a positive ΔAcc indicates an improvement in accuracy.

For our experiments, we set the parameters as in previous work: $\varphi = 1$ (weight for length reduction), $\eta = 3$ (bonus for accuracy improvements), and $\theta = 5$ (penalty for accuracy losses). The asymmetry in the AE Score, with $\theta > \eta$, reflects a practical preference for avoiding accuracy degradation, making the penalty for accuracy drops more significant than the bonus for accuracy gains.

A positive AE Score indicates that the model produces shorter outputs while maintaining or improving accuracy. Conversely, a negative score penalizes any decrease in accuracy.

A.3 Conjunctions

The table below lists the conjunctions we use:

Conjunctions
"wait", "alternatively", "but", "however", "alternative", "check", "double-check", "hmm", "okay", "maybe"

A.4 Prompt

The prompt used in the Prompt method is shown below.

CoD

Think step by step, but only keep a minimum draft for each thinking step, with 5 words at most.

CCoT

be concise

A.5 The Hacking Phenomenon in Token-based RL

The token-based chain-of-thought compression RL method exhibits a hacking reward phenomenon in the later stages of training. Token-based RL methods require continuous supervision, frequent saving of model checkpoints, and regular evaluation to monitor output length and performance. As shown in Figure 6, ShorterBetter (Yi et al., 2025) causes the model to continuously shorten its response length during training. Even after 600 steps, the 7B model can further compress its response, eventually reducing the thinking part to 0 tokens, outputting only `<think>\n\n</think>`. The figure on the right shows that while the model's performance stabilizes in the first 200 steps, as the output becomes increasingly shorter, the model's performance begins to deteriorate.

A.6 Why Paragraph Pruning Works

Although the number of paragraphs is not an exact measure of the number of steps, it serves as a good approximation. To validate this, we sampled 40 questions from three datasets: GPQA, MATH500, and LivecodeBench. For each question, we generated 50 responses, resulting in a total of 2000 answers generated by the DeepSeek-R1-Distill-Qwen-7B model. We then used Gemini 2.5 to decompose the responses into steps, leveraging prompt engineering to estimate the number of steps in each response. Simultaneously, we computed

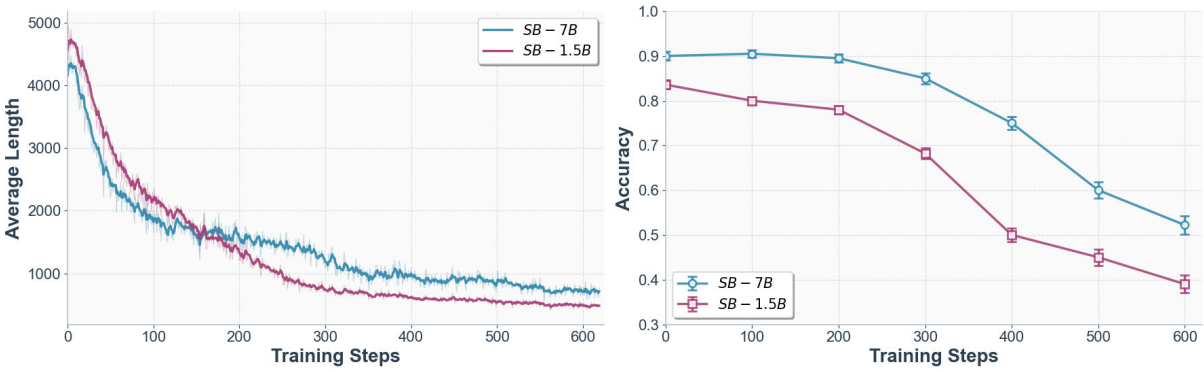

Figure 6: The average output length and accuracy of ShorterBetter during training on the MATH-500 dataset.

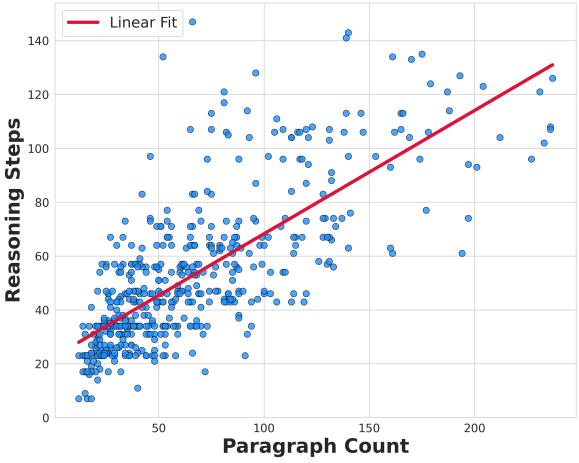

Figure 7: Scatter plot showing the relationship between the number of paragraphs and reasoning steps.

the number of paragraphs for each response by splitting the text at the "\n\n" delimiter.

Prompt

Please analyze the following solution and count the number of reasoning steps. A reasoning step is defined as a distinct logical operation or calculation that moves the solution forward. Paragraphs are split by \n\n. Even if some paragraphs are short or just verification steps, they should still be counted.
Solution:
{response}
Please provide ONLY a number representing the total count of reasoning steps. Put the number in \\boxed{} format.

For each question, we calculated the Pearson correlation coefficient between the number of steps and the number of paragraphs across its 50 responses. The resulting correlation coefficient was **0.661**, demonstrating that the number of paragraphs is a reliable approximation of the number of steps. As shown in Figure 7, the scatter plot displays the relationship between the number of paragraphs and reasoning steps across all responses.

A.7 Segments or Tokens

We conducted experiments to assess the effect of adding a token-level length penalty to our training objective. Specifically, we augmented the semantic segmentation loss with an explicit length loss, setting the token length and segment count penalty coefficients to 0.001 and 0.01, respectively. As shown in Table 3, incorporating a length penalty led to a substantial reduction in average output length across all benchmarks. However, this brevity came at the cost of a marked decrease in accuracy. These results suggest that directly penalizing output length can harm model performance, likely due to the generation of excessively brief or incomplete solutions.

	AIME24	GPQA	MATH500	GSM8K
	Acc.	Acc.	Acc.	Acc.
P	50.0	51.5	92.0	89.0
T	40.0	48.0	64.2	66.6
P+T	40.0	42.4	72.0	61.8
	Len.	Len.	Len.	Len.
P	4502	3925	1353	344
T	3165	3021	598	92
P+T	3002	2535	502	109

Table 3: Impact of different penalties on model response length (Len.) and accuracy (Acc.%) during training. **P**: Paragraph-based penalty, **T**: Token-based penalty.

A.8 Ablation experiment on the dataset

To eliminate the influence of the dataset, Table 4 presents the results of standard GRPO training using our dataset. After training with DeepScalar dataset, the model's performance on several datasets indeed improved. At the same time, SP was able to shorten the response length as much as possible while maintaining accuracy.

A.9 Ablation Study on Hyperparameters

As shown in Figure 8, we experimented with multiple settings of the hyperparameter β. The experimental results indicate that when β ranges from 0.01 to 0.0001, the training process remains relatively stable, and the final average output lengths are similar. However, when β is set to 0.1, the final output length becomes higher. This is because the increased step reward encourages the model to quickly merge different steps into the same paragraph, resulting in the model learning to exploit the reward early on rather than progressively removing redundant steps during training. Nevertheless, the accuracy on MATH500 remains high across all tested values of β.

A.10 Out of Domain Experiment

In addition to GPQA, we also evaluated the Livecodebench. These two datasets belong to the logic and code domains, respectively, which are different from the mathematical domain of the training dataset. The results are presented in Table 5. Experimental results show that although the shortening effect of SP on GPQA and livecodebench is less significant than that on mathematical datasets, a considerable compression rate is still achieved.

A.11 Experiments on the Llama Model

The results for the Llama-3.1-8B model, as shown in Table 6, demonstrate that the SP method performs exceptionally well across a variety of datasets. For instance, on the MATH500 dataset, SP achieves an accuracy of 84.8% with a relatively shorter response length compared to other methods, further highlighting its efficiency in balancing accuracy and response length.

These findings emphasize the robustness of the SP method, confirming its effectiveness not only on the Qwen model but also on the Llama model. The consistent high performance across different base models and datasets underscores the reliability of SP in achieving superior results in a range of settings, especially in terms of accuracy and response length trade-offs.

A.12 Reasoning Analysis with Different Methods

The SP-7B model demonstrates superior accuracy compared to SB-7B, largely attributable to differences in response structure. We compare our method with ShorterBetter (Yi et al., 2025). SP-7B allocates a higher proportion to "Verification & Self-Correction" (20.96% vs. 12.79%), emphasizing critical evaluation and error-checking, which likely reduces inaccuracies in reasoning. In contrast, SB-7B prioritizes "Productive Elaboration & Calculation" (35.17% vs. 31.33%), focusing on detailed computation but potentially overlooking verification steps. This structural shift in SP-7B promotes more balanced and reliable outputs, as self-correction mitigates over-elaboration risks. While SB-7B excels in exploratory aspects, SP-7B's emphasis on validation enhances overall precision, making it more effective for tasks requiring rigorous accuracy.

A.13 Answer from Different Methods

We also present the responses from Shorterbetter and Step Pruner, with the questions taken from AIME24.

Method	AIME 24			GPQA:Diamond			MATH500			GSM8K		
	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES
7B Model												
Qwen-7B	10.0	1309	–	30.3	724	–	74.6	704	–	81.8	254	–
R1-Qwen-7B	53.3	14839	0.00	52.0	8195	0.00	91.8	4053	0.00	85.6	508	0.00
GRPO	53.3	9222	0.38	50.0	6456	0.02	93.0	3009	0.30	88.8	919	-0.70
SP	50.0	4502	0.39	51.5	3925	0.47	92.0	1353	0.67	89.0	344	0.44
1.5B Model												
Qwen-1.5B	0.0	5452	–	31.8	1174	–	51.8	1134	–	64.7	296	–
R1-Qwen-1.5B	33.3	14783	0.00	33.8	9896	0.00	83.8	5794	0.00	77.6	1157	0.00
GRPO	30.0	7126	0.15	37.8	5612	1.27	87.4	2706	0.77	81.8	952	0.64
SP	30.0	5179	0.15	41.9	4446	1.27	86.0	1779	0.77	80.9	563	0.64

Table 4: Accuracy (%), response lengths (Len), and AES scores on AIME, GPQA, MATH500, and GSM8K for both 7B and 1.5B models.

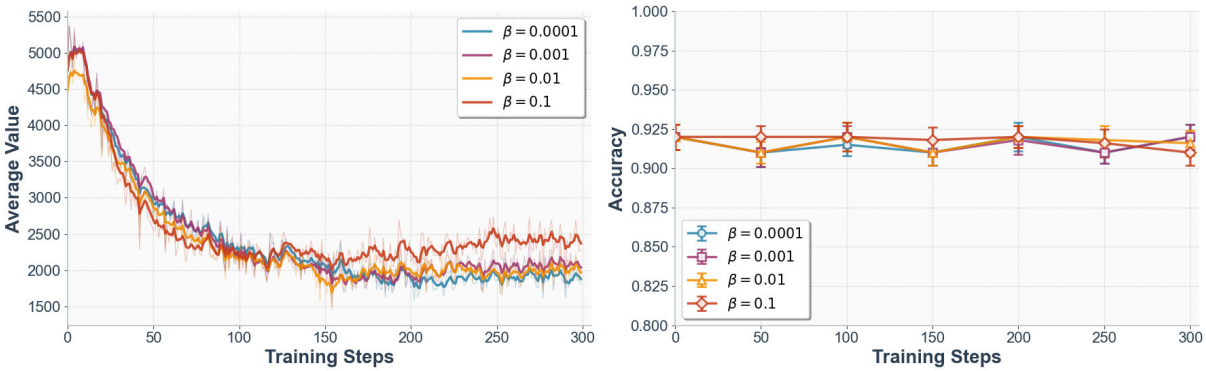

Figure 8: The average output length and accuracy during training for different beta values in the ablation study.

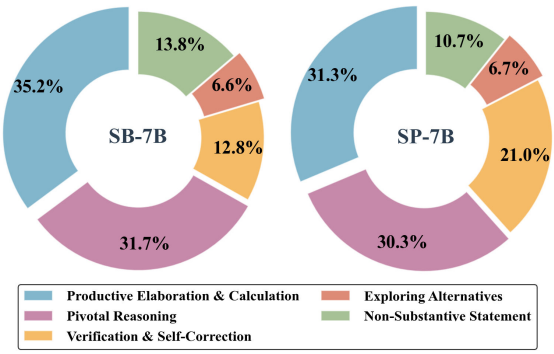

Figure 9: The Semantic Analysis of Reasoning in SB-7B and SP-7B.

Method	GPQA:Diamond			Livecodebench		
	ACC.	Len.	AES	ACC.	Len.	AES
7B Model						
Qwen-7B	30.3	724	–	13.8	376	–
R1-Qwen-7B	52.0	8195	0.00	34.7	13350	0.00
CCoT	51.0	8213	-0.10	32.5	12521	-0.25
CoD	51.5	7761	0.00	33.2	11982	-0.11
Cot-Valve	45.0	5386	-0.33	29.1	13918	-0.85
Model Merge	41.9	4425	-0.51	20.5	5517	-1.46
O1-Pruner	33.3	2180	-1.06	12.1	3591	-2.53
ShorterBetter	45.0	4326	-0.20	33.9	7596	0.32
TrainEfficient	52.0	6750	0.18	35.0	11224	0.19
SP	51.5	3925	0.47	34.3	7209	0.40
1.5B Model						
Qwen-1.5B	31.8	1174	–	4.5	842	–
R1-Qwen-1.5B	33.8	9896	0.00	14.6	16973	0.00
CCoT	37.9	9065	0.45	14.2	15723	0.15
CoD	38.9	8592	0.58	14.5	14710	0.15
ShorterBetter	35.4	2083	0.93	12.3	7623	-0.04
TrainEfficient	37.4	7114	0.60	16.1	11358	0.41
SP	41.9	4446	1.27	16.0	9251	0.64

Table 5: Accuracy (%), response lengths (Len), and AES scores on GPQA and Livecodebench for both 7B and 1.5B models across different baselines.

Method	AIME 24			GPQA:Diamond			MATH500			GSM8K		
	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES	ACC.	Len.	AES
7B Model												
Llama-3.1-8B	6.7	5535	–	33.8	3074	–	46.2	2131	–	77.6	232	–
R1-Llama-8B	50.0	10628	0.0	48.9	8919	0.0	87.8	3560	0.0	78.7	972	0.0
CCoT	43.3	11095	-0.71	49.4	8230	0.11	86.4	3163	0.03	70.2	516	-0.07
CoD	43.3	10429	-0.65	48.9	7514	0.16	86.6	2854	0.13	77.4	539	0.36
Model Merge	13.3	18495	-4.41	43.4	15882	-1.34	68.8	17441	-4.98	85.0	19796	-19.13
ShorterBetter	36.7	4291	-0.73	45.2	3812	0.19	78.2	1282	0.09	82.4	402	0.73
TrainEfficient	46.6	5812	0.11	48.9	6283	0.30	85.2	2245	0.22	84.8	591	0.62
SP	46.6	4917	0.20	48.9	4419	0.50	84.8	1639	0.37	87.2	497	0.81

Table 6: Accuracy (%), response lengths (Len), and AES scores on AIME, GPQA, MATH500, and GSM8K for 8B Llama-3.1 models across different baselines. We highlight the best results.

ShorterBetter (2615 tokens)

<think>

To solve this problem, I need to find the remainder when the product $\prod_{k=0}^{13}(2-2\omega^k+\omega^{2k})$ is divided by 1000, where $\omega \neq 1$ is a 13th root of unity.

First, I recognize that ω is a primitive 13th root of unity, so $\omega^{13}=1$. The product is over all 13th roots of unity, including $\omega^0=1$.

I notice that the expression inside the product can be rewritten as $2-2\omega^k+\omega^{2k}$. Maybe I can factor this expression or relate it to a polynomial whose roots are the ω^k.

Let me consider the polynomial $P(x)=x^{13}-1$. Its roots are the 13th roots of unity, including ω^k for $k=0,1,...,12$.

I want to relate $2-2\omega^k+\omega^{2k}$ to $P(x)$. Maybe I can express it as $\omega^{2k}-2\omega^k+2$, which resembles a quadratic in ω^k.

Let me set $y=\omega^k$. Then the expression becomes y^2-2y+2. I can write this as $(y-1)^2+1$.

So, the product becomes $\prod_{k=0}^{12}((\omega^k-1)^2+1)$.

Now, I need to compute this product. I recall that for roots of unity, products involving ω^k-1 can often be related to cyclotomic polynomials or other symmetric products.

Let me consider the polynomial $Q(x)=\prod_{k=0}^{12}(x-\omega^k)=x^{13}-1$.

But I need to compute $\prod_{k=0}^{12}((\omega^k-1)^2+1)$. Let me set $z=\omega^k$, then the term becomes $(z-1)^2+1=z^2-2z+2$.

So, the product is $\prod_{k=0}^{12}(z^2-2z+2)$ where z runs over all 13th roots of unity.

This is equivalent to evaluating the product $\prod_{k=0}^{12}(z^2-2z+2)$ where z are the roots of $z^{13}-1=0$.

I can think of this as evaluating the polynomial $R(z)=z^2-2z+2$ at all the 13th roots of unity and taking the product.

There is a formula for such products: $\prod_{k=0}^{n-1}R(\omega^k)=\prod_{k=0}^{n-1}R(\omega^k)$, which can be related to the resultant or the value of another polynomial.

Alternatively, I can consider that $\prod_{k=0}^{12}(z-\omega^k)=z^{13}-1$. So, if I set $z=\sqrt{2}e^{i\theta}$, but that might complicate things.

Wait, another approach: The product $\prod_{k=0}^{12}(z^2-2z+2)$ can be written as $\prod_{k=0}^{12}((z-(1+i))(z-(1-i)))$.

So, the product becomes $\prod_{k=0}^{12}(z-(1+i))(z-(1-i))$ evaluated at $z=\omega^k$.

Therefore, the entire product is $\prod_{k=0}^{12}(1+i-\omega^k)(1-i-\omega^k)$.

This can be written as $\prod_{k=0}^{12}(1+i-\omega^k)\times\prod_{k=0}^{12}(1-i-\omega^k)$.

Each of these products is the value of the polynomial $P(x)=x^{13}-1$ evaluated at $x=1+i$ and $x=1-i$, respectively.

So, $\prod_{k=0}^{12}(1+i-\omega^k)=(1+i)^{13}-1$ and similarly $\prod_{k=0}^{12}(1-i-\omega^k)=(1-i)^{13}-1$.

Therefore, the original product is $[(1+i)^{13}-1][(1-i)^{13}-1]$.

Now, I need to compute $(1+i)^{13}$ and $(1-i)^{13}$.

First, note that $1+i=\sqrt{2}e^{i\pi/4}$, so $(1+i)^{13}=(\sqrt{2})^{13}e^{i13\pi/4}$.

Similarly, $(1-i)^{13} = (\sqrt{2})^{13} e^{-i13\pi/4}$.

Compute $(\sqrt{2})^{13} = (2)^{13/2} = 2^6 \times 2^{1/2} = 64\sqrt{2}$.

Now, $e^{i13\pi/4} = e^{i(3\pi+\pi/4)} = e^{i\pi/4} \times e^{i3\pi} = e^{i\pi/4} \times (-1) = -e^{i\pi/4}$.

Similarly, $e^{-i13\pi/4} = -e^{-i\pi/4}$.

Therefore, $(1+i)^{13} = 64\sqrt{2} \times (-e^{i\pi/4}) = -64\sqrt{2}e^{i\pi/4}$.

Similarly, $(1-i)^{13} = -64\sqrt{2}e^{-i\pi/4}$.

Now, compute $(1+i)^{13} - 1 = -64\sqrt{2}e^{i\pi/4} - 1$.

Similarly, $(1-i)^{13} - 1 = -64\sqrt{2}e^{-i\pi/4} - 1$.

Multiply these two:

$$[-64\sqrt{2}e^{i\pi/4} - 1][-64\sqrt{2}e^{-i\pi/4} - 1]$$

Let me compute this product step by step.

First, expand the product:

$= [(-64\sqrt{2}e^{i\pi/4})(-64\sqrt{2}e^{-i\pi/4})] + [(-64\sqrt{2}e^{i\pi/4})(-1)] + [(-1)(-64\sqrt{2}e^{-i\pi/4})] + [(-1)(-1)]$

Compute each term:

1. First term: $(-64\sqrt{2})(-64\sqrt{2}) \cdot e^{i\pi/4}e^{-i\pi/4} = (64\sqrt{2})^2 \cdot e^0 = (64^2)(2) \cdot 1 = 4096 \cdot 2 = 8192$
2. Second term: $(-64\sqrt{2})(-1)e^{i\pi/4} = 64\sqrt{2}e^{i\pi/4}$
3. Third term: $(-1)(-64\sqrt{2})e^{-i\pi/4} = 64\sqrt{2}e^{-i\pi/4}$
4. Fourth term: $(-1)(-1) = 1$

So, adding all terms together:

$8192 + 64\sqrt{2}e^{i\pi/4} + 64\sqrt{2}e^{-i\pi/4} + 1$

Combine the exponential terms:

$64\sqrt{2}(e^{i\pi/4} + e^{-i\pi/4}) = 64\sqrt{2} \cdot 2\cos(\pi/4) = 64\sqrt{2} \cdot 2 \cdot (\sqrt{2}/2) = 64\sqrt{2} \cdot \sqrt{2} = 64 \cdot 2 = 128$

So, the total becomes:

$8192 + 128 + 1 = 8321$

Therefore, the product is 8321.

Now, we need the remainder when 8321 is divided by 1000.

8321 divided by 1000 is 8 with a remainder of 321.

So, the remainder is 321.

Final Answer

$$\boxed{321}$$

</think>

Given $\omega \neq 1$ is a 13th root of unity, we need to find the remainder when the product

$$\prod_{k=0}^{12}(2-2\omega^k+\omega^{2k})$$

is divided by 1000.

First, we rewrite the expression inside the product as $2-2\omega^k+\omega^{2k}$, which can be expressed as $(\omega^k-1)^2+1$. Thus, the product becomes

$$\prod_{k=0}^{12}((\omega^k-1)^2+1).$$

We recognize that this product can be related to evaluating the polynomial $P(x)=x^{13}-1$ at specific points. We rewrite the product as

$$\prod_{k=0}^{12}(z^2-2z+2)$$

where z runs over all 13th roots of unity. This product is equivalent to evaluating the polynomial at $z=1+i$ and $z=1-i$, respectively. Therefore, we need to compute

$$[(1+i)^{13}-1][(1-i)^{13}-1].$$

We find that $(1+i)^{13}=-64\sqrt{2}e^{i\pi/4}$ and $(1-i)^{13}=-64\sqrt{2}e^{-i\pi/4}$. Thus, the product becomes

$$[-64\sqrt{2}e^{i\pi/4}-1][-64\sqrt{2}e^{-i\pi/4}-1].$$

Expanding this product, we get:

$$8192+128+1=8321.$$

Finally, the remainder when 8321 is divided by 1000 is

$$\boxed{321}.$$

SP (2037 tokens)

<think>

Okay, so I have this problem:

Let $\omega \neq 1$ be a 13th root of unity. I need to find the remainder when the product from $k = 0$ to 12 of $(2 - 2\omega^k + \omega^{2k})$ is divided by 1000.

Hmm, that seems a bit complicated, but let me try to break it down.

First, I know that ω is a 13th root of unity, so $\omega^{13} = 1$. That might be useful.

The product is from $k = 0$ to 12, which is all the roots, so that's the complete set of 13th roots of unity.

Hmm, okay. So each term in the product is $2 - 2\omega^k + \omega^{2k}$. Maybe I can simplify that expression somehow.

Let me see, $2 - 2\omega^k + \omega^{2k}$... Hmm, perhaps I can write it as a quadratic in ω^k.

Let me set $x = \omega^k$, then the expression becomes $2 - 2x + x^2$.

So, $2 - 2x + x^2$. That's a quadratic, maybe I can factor it or find its roots.

Let me try to factor it: $x^2 - 2x + 2$. Hmm, the discriminant is $4 - 8 = -4$, so the roots are

$$\frac{2 \pm \sqrt{-4}}{2} = 1 \pm i.$$

So, it factors as $(x - (1 + i))(x - (1 - i))$. So,

$$2 - 2x + x^2 = (x - (1 + i))(x - (1 - i)).$$

So, each term in the product is $(\omega^k - (1 + i))(\omega^k - (1 - i))$.

So, the entire product becomes the product from $k = 0$ to 12 of $(\omega^k - (1 + i))(\omega^k - (1 - i))$. That's the same as the product of $(\omega^k - (1 + i))$ times the product of $(\omega^k - (1 - i))$.

So, I can write it as

$$\left[\prod_{k=0}^{12}(\omega^k - (1 + i))\right] \cdot \left[\prod_{k=0}^{12}(\omega^k - (1 - i))\right].$$

Now, I remember that for a root of unity, the product over $(x - \omega^k)$ is equal to $x^{13} - 1$, because the minimal polynomial for ω is $x^{13} - 1$. So,

$$\prod_{k=0}^{12}(x - \omega^k) = x^{13} - 1.$$

So, if I set $x = 1 + i$, then

$$\prod_{k=0}^{12}(1 + i - \omega^k) = (1 + i)^{13} - 1.$$

Similarly,

$$\prod_{k=0}^{12}(1 - i - \omega^k) = (1 - i)^{13} - 1.$$

So, my product is

$$[(1+i)^{13}-1][(1-i)^{13}-1].$$

So, I need to compute $(1+i)^{13}$ and $(1-i)^{13}$.

Hmm, okay, let's compute $(1+i)^{13}$. I know that $1+i$ can be written in polar form. The modulus is $\sqrt{1^2+1^2}=\sqrt{2}$, and the argument is $\pi/4$.

So,

$$1+i=\sqrt{2}\cdot e^{i\pi/4}.$$

So,

$$(1+i)^{13}=(\sqrt{2})^{13}\cdot e^{i13\pi/4}.$$

Similarly,

$$1-i=\sqrt{2}\cdot e^{-i\pi/4},$$

So

$$(1-i)^{13}=(\sqrt{2})^{13}\cdot e^{-i13\pi/4}.$$

Let me compute $(\sqrt{2})^{13}$. That's $(2^{1/2})^{13}=2^{13/2}=2^6\cdot\sqrt{2}=64\sqrt{2}$.

Okay, so

$$(1+i)^{13}=64\sqrt{2}\cdot e^{i13\pi/4}.$$

Now, $13\pi/4$ is the same as $3\pi/4+2\pi\times 3$, so

$$e^{i13\pi/4}=e^{i(3\pi/4+2\pi\times 3)}=e^{i3\pi/4}.$$

So,

$$e^{i3\pi/4}=\cos(3\pi/4)+i\sin(3\pi/4)=-\frac{\sqrt{2}}{2}+i\frac{\sqrt{2}}{2}.$$

So,

$$(1+i)^{13}=64\sqrt{2}\left(-\frac{\sqrt{2}}{2}+i\frac{\sqrt{2}}{2}\right)=64\cdot(-1)+64i\cdot 1=-64+64i.$$

Similarly,

$$(1-i)^{13}=64\sqrt{2}\cdot e^{-i13\pi/4}=64\sqrt{2}\cdot e^{-i3\pi/4}=64\sqrt{2}\left(-\frac{\sqrt{2}}{2}-i\frac{\sqrt{2}}{2}\right)=64\cdot(-1)-64i\cdot 1=-64-64i.$$

So,

$$(1+i)^{13}-1=(-64+64i)-1=-65+64i.$$

Similarly,

$$(1-i)^{13}-1=(-64-64i)-1=-65-64i.$$

So, the product is $(-65+64i)\cdot(-65-64i)$. That's the product of a complex number and its conjugate, which is the modulus squared.

So,

$$(-65)^2+(64)^2=4225+4096=8321.$$

So, the product is 8321.

So, the original product is 8321.

Now, I need to find the remainder when 8321 is divided by 1000. So, 8321 divided by 1000 is 8 with a remainder of 321. So, the remainder is 321.

So, the answer is $\boxed{321}$.

Wait, let me double-check my steps. I converted $1 + i$ to polar form, computed its 13th power, which gave me $-64 + 64i$, then subtracted 1 to get $-65 + 64i$. Similarly for the other term, I got $-65 - 64i$. Then, multiplying them together, I used the formula $(a + b)(a - b) = a^2 + b^2$, which gave me $65^2 + 64^2$. 65^2 is 4225, 64^2 is 4096, adding them gives 8321. Then, $8321 \bmod 1000$ is 321. So, yeah, that seems correct.

</think>

The remainder is $\boxed{321}$.