PACE: Prefix-Protected and Difficulty-Aware Compression for Efficient Reasoning

Anonymous ACL submission

Abstract

Language Reasoning Models (LRMs) achieve strong performance by scaling test-time computation but often suffer from "overthinking", producing excessively long reasoning traces that increase latency and memory usage. Existing LRMs typically enforce conciseness with uniform length penalties, which over-compress crucial early deduction steps at the sequence level and indiscriminately penalize all queries at the group level. To solve these limitations, we propose **PACE**, a dual-level framework for prefix-protected and difficulty-aware compression under hierarchical supervision. At the sequence level, prefix-protected optimization employs decaying mixed rollouts to maintain valid reasoning paths while promoting conciseness. At the group level, difficulty-aware penalty dynamically scales length constraints based on query complexity, maintaining exploration for harder questions while curbing redundancy on easier ones. Extensive experiments on DeepSeek-R1-Distill-Qwen (1.5B/7B) demonstrate that PACE achieves a substantial reduction in token usage (up to **55.7%**) while simultaneously improving accuracy (up to **4.1%**) on math benchmarks, with generalization ability to code, science, and general domains*.

1 Introduction

Recently, Large Language Models (LLMs) have evolved toward Language Reasoning Models (LRMs), which improve performance by scaling test-time computation (Muennighoff et al., 2025; Khatri et al., 2025). LRMs can perform extended multi-step reasoning and thus demonstrate remarkable performance improvements on logical reasoning tasks, such as mathematics (Chen et al., 2025a; Shao et al., 2024), tool using (Zhang et al., 2025b; Dong et al., 2025), and software engineering (Pan et al., 2024a; Yang et al., 2025d).

*Code and model are available at https://anonymous.4open.science/r/PACE-F6BB.

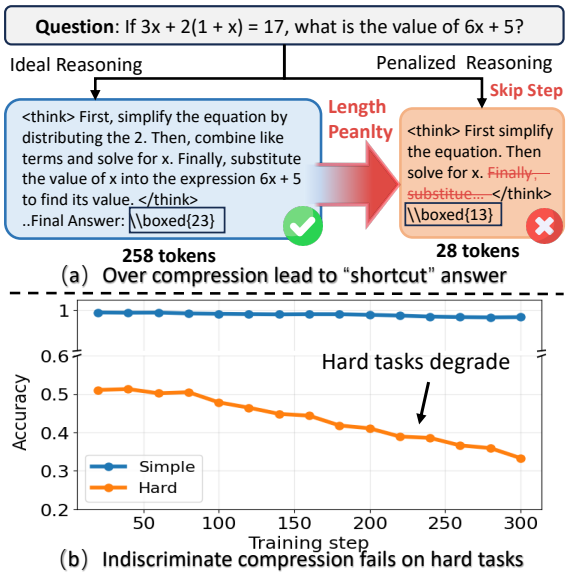


Figure 1: Two limitations of uniform length penalties. (a) **Over compression**: Using a uniform length penalty can skip step and lead to a "shortcut" answer. (b) **Indiscriminative compression**: We report avg pass@1 (32 samples) and split queries into *Simple* (> 0.75) and *Hard* (≤ 0.75) on MATH500; *Hard* accuracy drops sharply during training.

However, extended generation sequences impose a severe computational overhead. As a result, inference is increasingly memory-bound, with the Key-Value (KV) cache scaling rapidly with sequence length, leading to higher latency and increased GPU memory consumption (Kang et al., 2024a). Moreover, long generation sequences often suffer from an *overthinking* issue (Sui et al., 2025; Chen et al., 2025c), where LRMs produce excessively long and unnecessary reasoning steps even for simple problems. For example, a standard LLM solves a basic equation "$8x + 9 = 23, x =$?" in around 50 tokens, whereas an LRM (e.g., DeepSeek-R1 (Guo et al., 2025)) may use over 1,000 tokens for the same answer, incurring substantial redundancy and wasting computation.

Recent works compress long reasoning steps into shorter ones by introducing a simple length penalty during Reinforcement Learning (RL) (Arora and Zanette, 2025; Luo et al., 2025). However, we find this simple but rigid constraint introduces limitations at two distinct levels. First, at the sequence level, the uniform penalty leads to **over compression** problem, as applying the same constraint to *every token* prevents LRMs from distinguishing between redundant and essential steps, particularly in the critical early stages (Team et al., 2025; Yuan et al., 2025). Under this rigid penalty, LRMs may skip critical reasoning steps or simply avoid reasoning, leading to "shortcut" answers, reduced accuracy, and unstable training. As shown in Figure 1(a), a uniform length penalty compresses the reasoning from 258 to 28 tokens, greatly improving efficiency but skipping essential steps and leading to incorrect answers. Second, at the group level, a uniform penalty causes **indiscriminative compression** problem, as they apply the same constraint to *every query*. This ignores the fact that complex tasks inherently require more reasoning steps, disproportionately shortening reasoning for difficult tasks and limiting LRMs' ability to perform the extended exploration and reflection necessary for correct solutions. Figure 1(b) shows accuracy trends under a uniform length penalty across different difficulty levels. As training steps progress, accuracy remains high on simple tasks but drops sharply on hard ones.

To address these limitations, we propose a dual-level framework **PACE**, which incorporates **P**refix-Protected and Difficulty-**A**ware **C**ompression for **E**fficient Reasoning under hierarchical supervision. To tackle the over compression issue at the sequence level, we introduce *prefix-protected optimization*. Specifically, we utilize a prefix rollout strategy that samples initial reasoning steps from a frozen reference policy to explicitly anchor valid solution paths. By gradually removing this safeguard through a linear decay schedule, we prevent the reasoning collapse often caused by premature length penalization, ensuring the model preserves logical coherence while progressively learning conciseness. For indiscriminative compression issue at the group level, we propose a *difficulty-aware penalty*. Unlike uniform length constraints, this method first performs difficulty quantification by computing the empirical pass rate across group rollouts. Subsequently, it applies adaptive penalty modulation to penalty: constraints are effectively relaxed for difficult (low-pass-rate) tasks to accommodate necessary exploration, while strictly enforced for simple tasks to maximize efficiency.

Extensive experiments show that PACE reduces token usage by an average of 55% across multiple benchmarks and also maintains the reasoning performance without any degradation. Out-of-domain tasks further confirm PACE generalization with shorter generations and better performance.
Our contributions are summarized as follows:

- We propose a dual-level framework PACE to tackle the over compression and indiscriminative compression issue of efficient reasoning methods.
- We introduce prefix-protected optimization, using prefix rollout to anchor reasoning paths and prevent sequence-level over compression.
- We propose difficulty-aware penalty, scaling length penalties with task difficulty to prevent group-level indiscriminate compression.
- Experiments show that PACE achieves the best efficiency without accuracy loss compared to other methods. It also generalizes well to OOD tasks, shows shorter generations and better performance.

2 Related Works

Reinforcement Learning for Reasoning Capabilities. Recent studies highlight the necessity of reinforcement learning (RL) to unlock complex reasoning capabilities in large language models beyond supervised fine-tuning (SFT) (Shao et al., 2024; Hou et al., 2025; Yang et al., 2025b). Guo et al. (2025) and Wang et al. (2025b) show that pure RL can elicit emergent reasoning behaviors without human-annotated reasoning traces, significantly improving performance on math and STEM abilities. Subsequent studies further improve the effectiveness and practicality of RL for reasoning by expanding RL to real-world software engineering (Pan et al., 2024b; Wei et al., 2025) and search (Qi et al., 2024; Sun et al., 2025), pushing the boundaries of RL-driven reasoning in real-world scenarios.

Overthinking in Reasoning Models. Despite the advanced reasoning capabilities of current reasoning models, recent studies have identified a prevalent "overthinking" issue: models often think too much on easy queries (Chen et al., 2025c; Han et al., 2025; Yang et al., 2025e). To mitigate this, existing approaches can be generally categorized into two paradigms: training-free and training-based methods. Training-free methods typ-

ically intervene at the inference stage by truncating the model's thinking process to induce a direct answer (Yang et al., 2025c; Wang et al., 2025a). Conversely, training-based methods optimize efficiency by incorporating concise reasoning data or leveraging reinforcement learning. Kang et al. (2024b) compresses long rationales into shorter, information-preserving CoTs, then jointly trains an LLM with conditioned long/short CoT formats to get a more efficient model. Another line of research explicitly incorporate a length penalty into the reinforcement learning process to incentivize shorter sequences (Arora and Zanette, 2025; Team et al., 2025). However, such methods often lead to model or length collapse, reducing output length at the expense of model performance.

3 PACE Method

In this section, we detail the PACE: **P**refix-Protected and Difficulty-**A**ware **C**ompression for **E**fficient Reasoning. An overview of PACE is shown in Figure 2.

3.1 Prefix-Protected Optimization

Existing uniform length penalty methods often force LRMs to over-compress essential initial deduction steps. For example, when applying the uniform length penalty to the arithmetic problem "$8x + 9 = 2, x =$?", the compression will start from the first token. However, these early reasoning tokens serve as the logical anchor for the entire solution, and compressing them prematurely disrupts the coherence of the reasoning step.

To mitigate this, we propose the *prefix-protected optimization* that employs a prefix rollout strategy by a frozen prefix policy π_p to anchor the early generation phase using defined reasoning steps. For the same example "$8x + 9 = 2, x =$?", we provide the initial prefix "To calculate the x, we first need to subtract 9..." as the anchor. To construct this prefix tokens, we perform rejection sampling (Touvron et al., 2023) for the initial model on the training set and choose the shortest correct sample as the logical anchor. If no correct sample is available, we use the shortest one. After obtain the prefix, we can generate the entire rollout sequence, take the Deepseek-R1 template as an example: `<|im_start|>user\n 8x+9=23,x=? <|im_end|>\n <|im_start|>assistant\n <think> To calculate the x, we first need to subtract 9...then...` By locking these initial tokens, we ensure the reasoning step is firmly established before optimization begins. Specifically, at training step k, we construct a hybrid rollout distribution $\pi^{\text{pref}}_{\theta,k}$ that transitions from this frozen policy π_p (initial model) to the current training policy π_θ:

$$\pi^{\text{pref}}_{\theta,k}(y_t \mid y_{<t}, q) = \begin{cases} \pi_p(y_t \mid y_{<t}, q) \text{ if } t \le \mathcal{S}(k), \\ \pi_\theta(y_t \mid y_{<t}, q) \text{ if } t > \mathcal{S}(k), \end{cases} \tag{1}$$

where $\mathcal{S}(k)$ denotes the prefix length, π_θ is the training policy. By enforcing the prefix $\mathcal{S}(k)$ to follow π_p, we ensure the optimization starts with a valid logical foundation.

Since the LRM ultimately needs to solve the task without relying on the frozen policy π_p, we utilize a linear decay schedule to gradually reduce reliance on the protected policy by progressively shortening the prefix length $\mathcal{S}(k)$ as follows:

$$\mathcal{S}(k) = \begin{cases} \left\lfloor L_{\text{init}} \cdot (1 - \frac{k}{K}) \right\rfloor & \text{if } k < K, \\ 0 & \text{if } k \ge K, \end{cases} \tag{2}$$

where L_{init} denotes the initial prefix length, and K is the number of training steps for which prefix protection is applied. Initially, the $\mathcal{S}(k)$ strictly constrains the reasoning step. As training progresses ($k \to K$), this constraint is gradually relaxed, allowing the policy π_θ to generate the entire sequence on its own while avoiding over compression.

3.2 Difficulty-Aware Penalty

In real-world reasoning scenarios, the complexity of queries varies significantly, ranging from simple arithmetic to intricate logical deductions. However, applying a uniform length penalty ignores this critical variance, causing indiscriminative compression at the group level. It penalizes necessary exploration for difficult tasks while failing to curb overthinking on easy ones. Therefore, to adaptively align length constraints with task difficulty, we propose the *difficulty-aware penalty*.

First, we quantify task difficulties by leveraging the performance statistics of the generated rollouts. Specifically, we sample a group of G outputs $\tilde{\mathcal{Y}}^{\text{prefix}}_q = \{\tilde{y}_1, \ldots, \tilde{y}_G\}$ from prefix rollout distribution $\pi^{\text{pref}}_{\theta,k}$. Based on these rollouts, we measure the task difficulty through the proxy of the empirical pass rate $\rho(q)$ as follows:

$$\rho(q) = \frac{1}{G} \sum_{i=1}^{G} R_{\text{acc}}(\tilde{y}_i, q), \tag{3}$$

where $R_{\text{acc}}(\tilde{y}_i, q) = \mathbb{I}\left[\tilde{y}_i = y^*_q\right]$ is the correctness reward computed by the indicator function $\mathbb{I}[\cdot]$, and

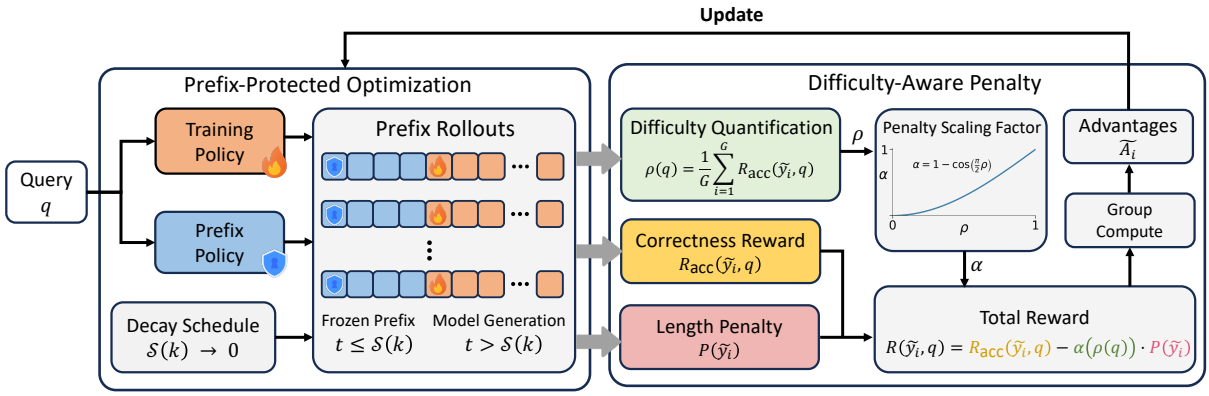


Figure 2: Overview of PACE. PACE consists of two stages: (1) Prefix-Protected Optimization, which anchors early reasoning by generating a short protected prefix with a frozen prefix policy and gradually decays the prefix length; and (2) Difficulty-Aware Penalty, which scales the length penalty by estimated task difficulty.

y_q^* denotes the ground truth answer. Here, $\rho(q)$ serves as an inverse indicator: a lower pass rate implies higher intrinsic difficulty, suggesting that the model struggles to reach the correct solution even with a prefix.

Then, we formulate the total reward $R(\tilde{y}_i, q)$ using adaptive penalty modulation. We couple the correctness reward with a dynamic length penalty, establishing a dependency between the penalty intensity and the pass rate $\rho(q)$:

$$R(\tilde{y}_i, q) = R_{\text{acc}}(\tilde{y}_i, q) - \alpha\big(\rho(q)\big) \cdot P(\tilde{y}_i), \quad (4)$$

where $P(\tilde{y}_i) = \frac{L(\tilde{y}_i) - L_{min}}{L_{max} - L_{min}}$ is the min–max length penalty (Team et al., 2025). We observe similar trends on the other length penalties shown in Section 5.6. $\alpha\big(\rho(q)\big)$ is the scaling factor and defined as follows:

$$\alpha\big(\rho(q)\big) = 1 - \cos\left(\frac{\pi}{2}\rho(q)\right). \quad (5)$$

Here, for simple problems where $\rho(q) \rightarrow 1$, the scaling factor α approaches its maximum, strictly enforcing conciseness to eliminate redundancy. Conversely, for highly complex problems where $\rho(q) \rightarrow 0$, thus α decays towards zero. This effectively shields the model from length penalties on complex tasks, granting it the necessary computational budget to perform deep reasoning without incurring negative rewards.

Finally, we integrate the above two components into the GRPO framework (Shao et al., 2024). We compute the advantages by standardizing the difficulty-aware rewards within the generated group $\tilde{\mathcal{Y}}_q^{\text{prefix}}$ as follows:

$$\tilde{A}_i = \frac{\tilde{R}(\tilde{y}_i, q) - \tilde{\mu}_{\text{q}}}{\tilde{\sigma}_{\text{q}} + \epsilon}, \quad (6)$$

where $\tilde{\mu}_{\text{q}}$ and $\tilde{\sigma}_{\text{q}}$ denote the mean and standard deviation of the total reward, ϵ is a small constant introduced to prevent division by zero.

Thus, the policy parameters θ are updated by maximizing the following objective:

$$\mathcal{J}(\theta) = \mathbb{E}\Big[\min\left(\gamma_i \tilde{A}_i,\ \text{clip}(\gamma_i, 1-\varepsilon, 1+\varepsilon)\,\tilde{A}_i\right) - \beta\,\mathbb{D}_{\text{KL}}\left(\pi_{\theta,k}^{\text{pref}}(\cdot\mid, q) \parallel \pi_{\text{ref}}(\cdot\mid, q)\right)\Big], \quad (7)$$

where γ_i is the importance sampling ratio and β is the coefficient that weights the KL-divergence penalty. This objective effectively combines sequence-level prefix protection with group-level adaptive penalty, driving the model to achieve high efficiency by eliminating redundancy without compromising reasoning accuracy.

4 Experimental Setup

Datasets and Metrics We initialize our models from DeepSeek-R1-Distill-Qwen-7B and DeepSeek-Distill-Qwen-1.5B. We choose the widely used reinforcement learning dataset Skywork-o1 (He et al., 2025) as our training set. We evaluate the performance of our proposed method on in-domain and out-of-domain tasks. For in-domain tasks, we choose four challenging mathematical benchmarks: AIME24, AIME25 (Veeraboina, 2023), MATH500 (Hendrycks et al., 2021), and GSM8k (Pan et al., 2024a). For out-of-domain tasks, we choose three representative benchmarks on different domains: science (GPQA-D (Rein et al., 2024)), code (LiveCodeBench-v6 (Jain et al., 2024)), and instruction following (IF Eval (Zhou et al., 2023)) We report three primary metrics: (1) Average Accuracy (**ACC**, %); (2) Average Token Usage (**LEN**); and (3) Token Reduction Rate (**TR**),

Method	AIME24			AIME25			MATH500			GSM8k			AVG		
	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑
DeepSeek-R1-Distill-Qwen-7B															
Vanilla	52.7	12228	-	38.6	12977	-	92.8	3832	-	91.5	687	-	68.9	7431	-
Thinkswitcher	48.3 $_{-4.4}$	7936	35.1%	37.5 $_{-1.1}$	6955	46.4%	91.3 $_{-1.5}$	3495	8.8%	92.5 $_{+1.0}$	1389	-102.2%	67.4 $_{-1.5}$	4944	33.5%
DEER	49.2 $_{-3.5}$	9839	19.5%	36.7 $_{-1.9}$	7257	44.1%	89.8 $_{-3.0}$	2143	44.1%	90.6 $_{-0.9}$	917	-33.5%	66.6 $_{-2.3}$	5039	32.2%
Dynasor-CoT	46.7 $_{-6.0}$	12695	-3.8%	33.3 $_{-5.3}$	11069	14.7%	89.0 $_{-3.8}$	2971	22.5%	89.6 $_{-1.9}$	1285	-87.0%	64.7 $_{-4.2}$	7005	5.7%
O1-Pruner	49.2 $_{-3.5}$	9719	20.5%	35.0 $_{-3.6}$	8236	36.5%	91.5 $_{-1.3}$	2268	40.8%	91.1 $_{-0.4}$	1012	-47.3%	66.7 $_{-2.2}$	5309	28.6%
DAST	45.6 $_{-7.1}$	7578	38.0%	37.0 $_{-1.6}$	10729	17.3%	89.6 $_{-3.2}$	2162	43.6%	86.7 $_{-4.8}$	489	28.8%	64.7 $_{-4.2}$	5240	29.5%
AdaptThink	**52.9** $_{+0.2}$	9596	21.5%	38.3 $_{-0.3}$	10670	17.8%	92.0 $_{-0.8}$	1875	51.1%	90.3 $_{-1.2}$	408	40.6%	69.1 $_{+0.2}$	5316	28.5%
DR.SAF	50.6 $_{-2.1}$	6288	48.6%	38.2 $_{-0.4}$	6747	48.0%	88.3 $_{-4.5}$	**1061**	**72.3%**	88.1 $_{-3.4}$	**162**	**76.4%**	66.3 $_{-2.6}$	3565	52.0%
Efficient-R	51.9 $_{-0.8}$	7464	39.0%	33.3 $_{-5.3}$	8902	31.4%	88.5 $_{-4.3}$	1306	65.9%	87.2 $_{-4.3}$	263	61.7%	65.2 $_{-3.7}$	4484	39.7%
PACE	52.7 $_{+0.0}$	**5525**	**54.8%**	**39.4** $_{+0.8}$	**5849**	**54.9%**	**93.1** $_{+0.3}$	1490	61.1%	**92.6** $_{+1.1}$	309	55.0%	**69.5** $_{+0.6}$	**3293**	**55.7%**
DeepSeek-R1-Distill-Qwen-1.5B															
Vanilla	28.3	12094	-	25.0	11860	-	84.0	4627	-	75.5	987	-	53.2	7392	-
Thinkswitcher	23.3 $_{-5.0}$	8192	32.3%	28.3 $_{+3.3}$	6689	43.6%	82.4 $_{-1.6}$	4544	1.8%	**84.7** $_{+9.2}$	2114	-114.2%	54.7 $_{+1.5}$	5385	27.2%
DEER	23.3 $_{-5.0}$	9557	21.0%	10.0 $_{-15.0}$	9281	21.7%	67.8 $_{-16.2}$	2497	46.0%	74.7 $_{-0.8}$	984	0.3%	44.0 $_{-9.2}$	5580	24.5%
O1-Pruner	28.9 $_{+0.6}$	10361	14.3%	23.2 $_{-1.8}$	8731	26.4%	82.2 $_{-1.8}$	3212	30.6%	74.8 $_{-0.7}$	458	53.6%	52.3 $_{-0.9}$	5691	23.0%
AdaptThink	30.8 $_{+2.5}$	7616	37.0%	25.8 $_{+0.8}$	7456	37.1%	82.0 $_{-2.0}$	**1685**	**63.6%**	81.2 $_{+5.7}$	439	55.5%	55.0 $_{+1.8}$	4299	41.8%
Efficient-R	24.6 $_{-3.7}$	6945	42.6%	-	-	-	79.0 $_{-5.0}$	2105	54.5%	78.4 $_{+2.9}$	**300**	**69.6%**	-	-	-
PACE	**33.3** $_{+5.0}$	**6840**	**43.4%**	**29.6** $_{+4.6}$	**6592**	**44.4%**	**85.8** $_{+1.8}$	1987	57.1%	80.3 $_{+4.8}$	357	63.8%	**57.3** $_{+4.1}$	**3944**	**46.6%**

Table 1: Performance comparison on AIME24, AIME25, MATH500, and GSM8K datasets. Rows shaded in lavender indicate training-free (inference-time) methods, while rows shaded in light purple indicate training-based methods. Best results are highlighted in **bold**, second best results are highlighted in underline.

defined as the percentage reduction in average output length relative to the Vanilla model, which quantifies the efficiency gain in terms of computational cost. Without specifying, we report Pass@1 accuracy averaged over 16 independent runs per prompt, using a context window of 32k and a sampling temperature of 0.6. All evaluation details can be found in Appendix A.

Baselines We compare our PACE against two categories of representative baselines:

- **Training-free Methods:** These methods typically employ inference-time intervention. DEER (Yang et al., 2025c), Dynasor-CoT (Fu et al., 2025) implement dynamic early exit by monitoring intermediate confidence/certainty signals during generation and stopping CoT once a reliable answer is detected. ThinkSwitcher (Liang et al., 2025) utilizes a trained switcher module to dynamically toggle the generation of reasoning chains, deciding whether to "think" before answering.
- **Training-based Methods:** These methods integrate efficiency constraints during training. Efficient-R (Arora and Zanette, 2025) applies a uniform length penalty as a reward signal to all outputs. O1-Pruner (Luo et al., 2025) harmonizes output length by pre-sampling to estimate a reference/baseline behavior, and then applies RL-style optimization to penalize redundant long reasoning while constraining accuracy. AdaptThink (Zhang et al., 2025a) trains a reasoning model with RL to adaptively switch between Thinking and NoThinking modes based on input difficulty. DAST (Shen et al., 2025) quantifies difficulty via a token-length budget and applies budget-aware reward shaping with preference optimization to regulate per-query CoT length, reducing overthinking without sacrificing accuracy. Dr.SAF (Chen et al., 2025b) facilitates dynamic termination of the chain-of-thought process via boundary-awareness alignment followed by reinforcement learning.

Implementation Details We utilize verl (Sheng et al., 2024) as the reinforcement learning framework on 32 NVIDIA A100 GPUs and set the batch size as 64 with a group size as 8. We choose L_{init} to be 512 and apply prefix protection for K set to 100 training steps. We also set the learning rate to 1×10^{-6} with 10 warm-up steps. More training details can be found in Appendix E.

5 Experimental Results

5.1 Main Results

PACE breaks the efficiency-performance trade-off. As shown in Table 1, PACE shows superior performance in terms of both token efficiency and accuracy across benchmarks and model sizes. Notably, the most significant observation is that PACE is the only method that achieves a substantial reduction in token usage ($> 45\%$) while simultaneously *improving* average accuracy over the Vanilla baseline. In contrast, existing methods struggle to balance this trade-off: aggressive pruners like DR.SAF and DEER achieve high token reduction

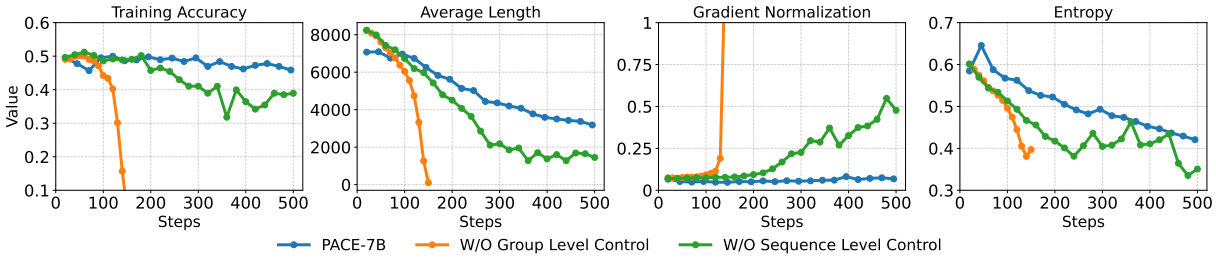


Figure 3: Training dynamics of the PACE-7B and its ablations over RL steps, reporting training accuracy, average response length, gradient normalization, and policy entropy.

Method	GPQA-D			LiveCodeBench			IF Eval		
	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑	Acc↑	Len↓	TR↑
DeepSeek-R1-Distill-Qwen-7B									
Vanilla	49.3	8011	-	35.0	11723	-	60.2	1570	-
PACE	**49.9**	**5346**	**33.3%**	**36.8**	**8110**	**30.8%**	**60.8**	**1167**	**25.7%**
DeepSeek-R1-Distill-Qwen-1.5B									
Vanilla	36.3	9342	-	17.2	13991	-	40.7	5249	-
PACE	**39.4**	**5889**	**37.0%**	**18.3**	**10286**	**26.5%**	**45.3**	**4487**	**14.5%**

Table 2: Results on out of domain (OOD) evaluation across science (GPQA-D), code (LiveCodeBench-v6), and instruction following (IF Eval).

Method	AIME24		AIME25		MATH500	
	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓
PACE	**52.7**	**5525**	**39.4**	**5849**	**93.1**	**1390**
w/o group-level control	42.5	5396	31.6	6538	80.2	1414
w/o sequence-level control	45.6	4952	32.5	4887	90.2	1351

Table 3: Ablation study on mathematical benchmarks.

rates (TR) but suffer from evident performance degradation (e.g., DR.SAF drops accuracy by 2.7% and DEER drops accuracy by 2.3% on 7B model). Conversely, conservative methods like AdaptThink preserve accuracy but yield limited efficiency gains (only 28.5% TR on 7B). However, PACE dominates the Pareto frontier. On the 7B model, we achieve a **55.7%** token reduction, which is nearly double the efficiency of the second-best performer, AdaptThink, while surpassing the vanilla model's accuracy by **0.6%**. On the smaller 1.5B model, the gain is even more pronounced: PACE cuts the average length with 46% token reduction while boosts average accuracy from **53.2** to **57.3** (+4.1), indicating that PACE pushes the Pareto frontier of reasoning compression.

PACE also shows great generalization to other domains Table 2 presents the performance of our PACE on out-of-domain (OOD) tasks. Although trained exclusively on mathematical datasets, our PACE exhibits consistent gains in both accuracy and token efficiency across science (GPQA-D), code (LiveCodeBench-v6), and instruction following (IF Eval). Specifically, the 1.5B model surpasses the vanilla baseline with 1.1%–4.6% accuracy improvements while reducing generation length by 14.5%–37.0%. A similar trend is observed in the 7B model, which achieves up to 1.8% accuracy gain with significant token reductions ranging from 25.7% to 33.3%. These results yield two key insights: (1) Beyond the mathematical domain, other tasks (*e.g.,* code) also have the potential to shorten their reasoning chains while maintaining or improving the performance. (2) Gains in both performance and token reduction are consistently enjoyed across domains, further showcasing the effectiveness and robustness of our proposed dual-level control method.

5.2 Ablation Study

Dual-level control is crucial for performance and token efficiency To investigate the efficacy of our proposed dual-level control mechanism, we conducted an ablation study by training a 7B model without sequence-level and group-level controls under identical settings. As shown in Table 3, removing group-level control significantly harms the model's internal reasoning ability, causing 9.8% performance drop on AIME24 and 12.9% on MATH500. We attribute this deterioration to indiscriminative compression of questions, causing the model to suffer from reasoning collapse during training. Removing sequence-level control also degrades performance sharply, as compression starts from the initial tokens, leading the model to skip essential reasoning steps. In contrast, sequence-level control effectively guides the policy model by validating solution entry points, ensuring that core reasoning steps are preserved during compression.

Dual-level control stabilizes training process We also present the 7B model ablation training dynamics across four metrics: average length, gra-

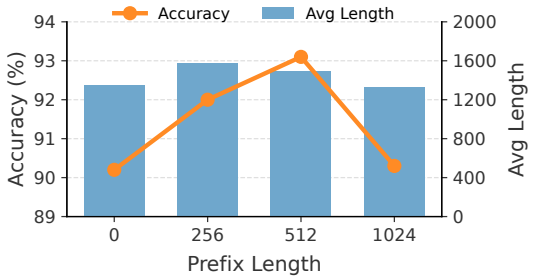


Figure 4: Ablation study of different prefix length on MATH500 dataset.

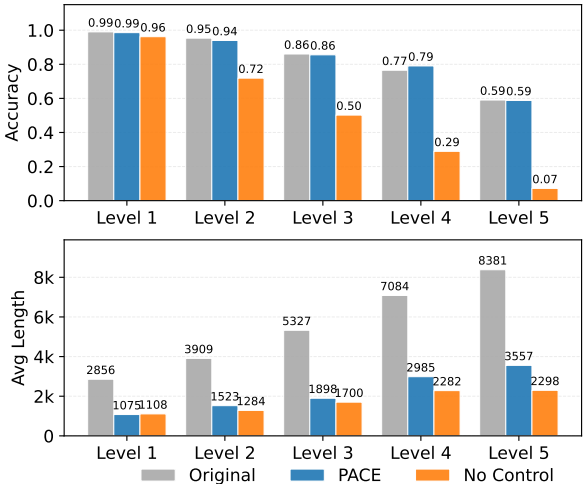


Figure 5: Accuracy and average length change across different difficulty levels on MATH500 dataset.

dient norm, training accuracy, and entropy. As shown in Figure 3, dual-level control sustains training accuracy while the response length decreases. Without group-level or sequence-level control, the model often sacrifices training accuracy for length reduction. Moreover, dual-level control can stabilize the training process, exhibiting consistent gradient norms and entropy. Our gradient norm remains stable throughout, unlike the explosion in the other two ablations. We also observe benefits in entropy evolution, especially at the later stages of training. Our method significantly prevents entropy oscillation, indicating that the model retains steady reasoning capabilities.

5.3 Prefix Length Study

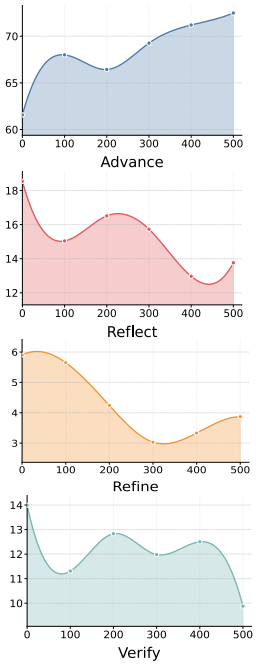
To study the effect of sequence-level prefix length, we trained the 7B model with varying prefix length L_{init} (0, 256, 512, and 1024 tokens) under identical settings and evaluated on the MATH500 dataset. As shown in Figure 4, sequence-level prefix protection clearly improves performance. Even with a shorter prefix (256 tokens), accuracy rises from 90.2% to 92.0% by preserving valuable reasoning steps. This improvement continues at 512 tokens, where the model achieves a peak accuracy of 93.1%. However, an excessively long prefix (1024 tokens) harms performance, dropping accuracy to 90.3%. This is because during the early training stages, shorter samples are fully encapsulated within the 1024-token prefix, exposing the ground truth answer immediately and enabling the model to bypass reasoning steps. Consequently, the model exhibits "shortcut learning" and degrades in reasoning capability.

5.4 Across Difficulty Levels Study

The core objective of PACE is to compress without sacrificing the model's underlying reasoning capability. We evaluate it against the original model and a "No Control" baseline across difficulty levels on MATH500 dataset. As shown in Figure 5, in the more complex categories (Levels 3-5), where multi-step reasoning is paramount, PACE's performance remains comparable to the original model while require significant fewer tokens. At Level 4, PACE even slightly outperforms the original model (0.79 vs 0.77) with a 58% token reduction, indicating that targeted compression can help reduce redundant noise in complex reasoning steps. In contrast, the "No Control" variant suffers catastrophic performance collapse as difficulty increases, with accuracy dropping from 0.96 (Level 1) to just 0.07 (Level 5), reflecting degraded reasoning ability due to excessive compression on critical reasoning steps. This significant gap highlight that without dual-level control, the model tends to "over-compress" essential logic, whereas PACE can preserve core reasoning while achieving conciseness.

5.5 Reasoning Behavior Shift During Training

Although PACE significantly reduces token usage, a natural concern arises: *Does the model simply learn to skip crucial reasoning steps rather than pruning unnecessary ones?* To investigate which part of PACE reduces token usage and how each part changes during training, we define four reasoning behaviors for reasoning models: Advance, Reflect, Refine and Verify, as color-coded in Figure 6. We test checkpoints during training with 100 step interval on AIME2024&25 across 16 independent runs. Using Qwen3-32B (Yang et al., 2025a) to annotate the generated chains of thought, we calculated the proportion of each behavior.



Question: Alice chooses a set A of positive integers. Then Bob lists all finite nonempty sets B of positive integers with the property that the maximum element of B belongs to A. Bob's list has 2024 sets. Find the sum of the elements of A.

Answer: <think>\n Okay, so I have this problem where set A of positive integers … Therefore, if A is a set containing elements a1, a2, ..., an, then the total number of sets B is the sum over all ai in A of 2^{ai - 1}. … Wait, actually, no—because B can be any nonempty set, we may consider it more carefully… But actually, above is not correct. The number of sets B is 2^{ai - 1} … Let me check. Adding these together: 1024+512=1536; 1536+256=1792. Yes, correct…</think> …So the final answer is \\boxed{49}.

Figure 6: Left: The distributional shift of four reasoning behaviors (Advance, Reflect, Refine, Verify) across training steps. Right: a sample AIME 2024 solution, with text color-coded to demonstrate each reasoning behavior in context.

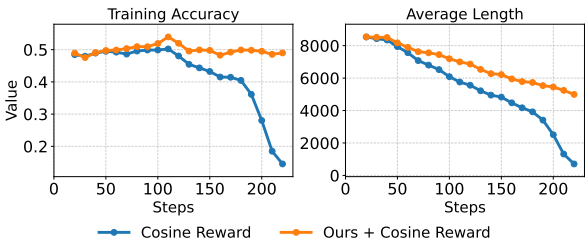


Figure 7: Training dynamics of a cosine reward compared to our PACE.

As shown in Figure 6, a clear distributional shift emerges: the Advance behavior steadily rise from 61.5% to 72.5%, while Reflect and Verify decline from 18.5% to 13.6% and 14.0% to 9.8%, respectively. This indicates that token reduction is primarily comes from suppressing redundant self-correction and excessive hesitation, rather than omitting constructive logic. During training, the model is encouraged to "think forward" more confidently, resulting in a higher ratio of Advance steps. Consequently, PACE promotes more efficient reasoning by emphasizing decisive problem-solving steps over repetitive verification loops.

5.6 Generalization to Other Length Rewards

To demonstrate the generalization ability and robustness of PACE, we extended our evaluation to include alternative reward mechanisms. Following prior work(Arora and Zanette, 2025), we adopted a static cosine reward function to penalize generation length, which is formulated as:

$$P(\tilde{y}_i) = \sigma\left(\frac{L(\tilde{y}_i) - L_{mean}}{L_{std}}\right), \tag{8}$$

where $L(\tilde{y}_i)$ is the length of the i-th rollout $\tilde{y}_i$, while L_{mean} and L_{std} are the mean and standard deviation of lengths for all rollouts for query q and σ is the sigmoid activation function.

Figure 7 shows the training dynamics of the baseline static cosine reward (blue) compared to our method (orange). The baseline exhibits a critical instability: as it only optimizes for the length reward, the training accuracy (Left) suffers a catastrophic collapse after approximately 150 steps. The length plot (Right) confirms that the baseline aggressively shortens outputs, compromising reasoning capabilities and yielding an unacceptable efficiency-performance trade-off.

Table 4 further shows that these superior training dynamics directly contribute to stronger final performance. PACE consistently outperforms the Cosine Reward baseline across all benchmarks, demonstrating robust generalization to different reward formulations and a more stable, superior Pareto frontier between inference cost and reasoning performance.

Method	AIME24		AIME25		MATH500	
	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓
Cosine Reward	47.0	6714	34.1	7817	88.8	2364
PACE	**51.7**	**6873**	**38.8**	**7517**	**91.4**	**2035**

Table 4: Performance comparison between other length reward (*e.g.,* consine reward) and our PACE.

6 Conclusion

We propose PACE, a dual-level framework for length compression of LRMs. PACE consists of a prefix-protected optimization to preserve essential early reasoning and a difficulty-aware length penalty to avoid disproportionately compressing on complex tasks. Experimental results show that PACE significantly reduces inference costs and further improves accuracy across multiple domains. Moreover, PACE generalizes to alternative length-reward designs, underscoring the effectiveness of dual-level protection in pushing the efficiency–accuracy Pareto frontier.

Limitations

Beyond the strong accuracy–efficiency trade-off achieved by PACE, there remains substantial headroom to further improve efficiency, as roughly 40% of tokens are still devoted to reasoning. In addition, while we observe promising out-of-domain performance on several benchmarks, a broader and more systematic study of generalization is needed, covering more diverse domains, prompt and format variations, and a wider range of distribution shifts to better understand when and why adaptive compression transfers.

References

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

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, et al. 2025a. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv:2507.23726*.

Qiguang Chen, Dengyun Peng, Jinhao Liu, HuiKang Su, Jiannan Guan, Libo Qin, and Wanxiang Che. 2025b. Aware first, think less: Dynamic boundary self-awareness drives extreme reasoning efficiency in large language models. *arXiv:2508.11582*.

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

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Toolstar: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv:2505.16410*.

Yichao Fu, Junda Chen, Yonghao Zhuang, Zheyu Fu, Ion Stoica, and Hao Zhang. 2025. Reasoning without self-doubt: More efficient chain-of-thought through certainty probing. In *ICLR Workshop*.

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

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware LLM reasoning. In *Findings of the ACL*, pages 24842–24855.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, et al. 2025. Skywork open reasoner 1 technical report. *arXiv:2505.22312*.

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

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. 2025. T1: Advancing language model reasoning through reinforcement learning and inference scaling. In *ICML*, pages 23976–24003.

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:2403.07974*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024a. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv:2403.05527*.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2024b. C3ot: Generating shorter chain-of-thought without compromising effectiveness. *arXiv:2412.11664*.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S Dhillon, David Brandfonbrener, and Rishabh Agarwal. 2025. The art of scaling reinforcement learning compute for llms. *arXiv:2510.13786*.

Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. 2025. Thinkswitcher: When to think hard, when to think fast. *arXiv:2505.14183*.

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:2501.12570*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *EMNLP*, pages 20286–20332.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024a. Training software engineering agents and verifiers with swe-gym. *arXiv:2412.21139*.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024b. Training software engineering agents and verifiers with swe-gym. *arXiv:2412.21139*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. 2024. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv:2411.02337*.

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*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *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:2503.04472*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv: 2409.19256*.

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

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Fei Huang, and Yan Zhang. 2025. Zerosearch: Incentivize the search capability of llms without searching. *arXiv:2505.04588*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, et al. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv:2501.12599*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Hemish Veeraboina. 2023. Aime problem set 1983-2024.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. 2025a. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. In *Findings of the EMNLP*, pages 7459–7482.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Lucas Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. 2025b. Reinforcement learning for reasoning in large language models with one training example. *arXiv:2504.20571*.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv:2502.18449*.

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

Chen Yang, Guangyue Peng, Jiaying Zhu, Ran Le, Ruixiang Feng, Tao Zhang, Wei Ruan, Xiaoqi Liu, Xiaoxue Cheng, Xiyun Xu, Yang Song, Yanzipeng Gao, Yiming Jia, Yun Xing, Yuntao Wen, Zekai Wang, Zhenwei An, Zhicong Sun, and Zongchao Chen. 2025b. Nanbeige4-3b technical report: Exploring the frontier of small language models. *arXiv:2512.06266*.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025c. Dynamic early exit in reasoning models. *arXiv:2504.15895*.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025d. Swe-smith: Scaling data for software engineering agents. *arXiv:2504.21798*.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. 2025e. Towards thinking-optimal scaling of test-time compute for llm reasoning. *arXiv:2502.18080*.

Danlong Yuan, Tian Xie, Shaohan Huang, Zhuocheng Gong, Huishuai Zhang, Chong Luo, Furu Wei, and Dongyan Zhao. 2025. Efficient rl training for reasoning models via length-aware optimization. *arXiv:2505.12284*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025a. Adaptthink: Reasoning models can learn when to think. *arXiv:2505.13417*.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025b. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv:2505.00024*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv:2311.07911*.

A Evaluation Tasks Details

We report pass@1 scores based on 16 independent runs for AIME24, AIME25, MATH500, GSM8K, and GPQA-D, and 4 independent runs for LiveCodeBench-v6. For IFEval, we evaluate using pass@1 directly due to its extensive size. All experiments were conducted using a 32k context window. Following is the details and resources for each tasks.

In-domain mathematical reasoning benchmarks.

- **AIME24** We use the 2024 American Invitational Mathematics Examination problems (AIME I & II; 30 problems in total), where each question requires multi-step contest-level reasoning and the final answer is a short integer. We report accuracy via exact-match of the final answer. https://huggingface.co/datasets/HuggingFaceH4/aime_2024
- **AIME25** Similar to AIME24, this benchmark contains the 2025 AIME I & II problems (30 problems total) and tests olympiad-style mathematical reasoning with exact-match final-answer evaluation. https://huggingface.co/datasets/math-ai/aime25
- **MATH500** MATH500 is a curated 500-problem subset derived from the MATH competition dataset, covering diverse topics (e.g., algebra, geometry, number theory) and emphasizing solution correctness on challenging problems. https://huggingface.co/datasets/HuggingFaceH4/MATH-500
- **GSM8K** GSM8K is a grade-school math word-problem benchmark designed for multi-step arithmetic reasoning, with reference solutions and final numeric answers. We use its test set (1319 samples) for our evaluation. https://huggingface.co/datasets/openai/gsm8k

Out-of-domain (OOD) generalization benchmarks.

- **GPQA-D (Science)** GPQA is a graduate-level, "Google-proof" multiple-choice QA benchmark written by domain experts across biology, chemistry, and physics. We use its Diamond (GPQA-D, 198 questions) split as a challenging science OOD evaluation and report multiple-choice accuracy. https://huggingface.co/datasets/Idavidrein/gpqa
- **LiveCodeBench-v6 (Code)** LiveCodeBench is a contamination-aware, continuously updated coding benchmark built from newly released contest problems, and supports programmatic grading via unit tests. We use the latest v6 release for our evaluation. https://github.com/LiveCodeBench/LiveCodeBench
- **IFEval (Instruction Following).** IFEval evaluates instruction-following using programmatically verifiable constraints (e.g., required keywords, length constraints, formatting rules), enabling reproducible automatic checking. It contains 25 instruction types and 541 prompts; we report instruction-following accuracy under the benchmark's verifier. https://github.com/google-research/google-research/tree/master/instruction_following_eval

B Effectiveness of Prefix

In this section, we demonstrate the effectiveness of prefix protection. Concretely, we construct the prefix via rejection sampling on AIME 2024 and AIME 2025 with DeepSeek-R1-Distill-Qwen-7B, selecting the shortest correct sample for each problem. When no correct sample is available, we fall back to the shortest sampled response. We then vary the protected prefix length and report pass@1 accuracy averaged over 8 independent runs. As shown in Table 5, prefix protection yields a clear accuracy–efficiency improvement. Compared to using no prefix (0), a 600-token prefix improves AIME24 accuracy from 52.7 to 57.8 ($+5.1$ points) while reducing generation from 12,228 to 8,568 tokens ($-3,660$). On AIME25, longer anchors are particularly beneficial: with a 1,000-token prefix, accuracy increases from 38.6 to 44.0 ($+5.4$ points) and tokens drop from 12,977 to 8,675 ($-4,302$). Overall, across the tested prefix lengths, we consistently obtain 3k–4k+ token savings while maintaining or improving accuracy, confirming our motivation that a strong early logical scaffold is crucial for downstream reasoning. Therefore, protecting this anchor from over-compression stabilizes the reasoning trajectory and leads to a substantially better trade-off between correctness and computational cost.

Prefix	AIME24		AIME25	
	Acc↑	Tokens↓	Acc↑	Tokens↓
0	52.7	12228	38.6	12977
200	52.0	8675	38.0	9484
400	53.7	8442	41.7	9726
600	57.8	8568	42.8	9568
800	55.5	9116	44.6	9116
1000	55.7	7859	44.0	8675

Table 5: Results under different prefix lengths on AIME2024/2025 using Deepseek-R1-Distill-Qwen-7B.

C Justification of Prefix Rollout

We analyze the benefit of prefix rollout by comparing the expected returns under different prefix distributions. We show that provided the prefix policy π_p induces a higher-quality state distribution than the current policy π_θ, the overall training objective is strictly improved.

Value Decomposition Formalizing generation as an MDP with query q and output y, we define the reward $R(y, q) = \mathbb{I}[y = y_q^*] - \lambda P(y)$. Consider the *conditional value* of a prefix state $s_S = (q, y_{<S})$ under the training policy π_θ:

$$V_\theta(s_S) \triangleq \mathbb{E}_{y_{>S} \sim \pi_\theta(\cdot|s_S)}\big[R(y, q)\big]. \tag{9}$$

This value $V_\theta(s_S)$ measures the expected final quality (correctness and efficiency) given a specific prefix.

The total expected return depends on how prefixes are sampled. Let d_π^S be the distribution of prefixes generated by policy π. The objectives for Prefix Rollout (J_{prefix}) and Standard Rollout (J_{base}) can be written as expected values over their respective prefix distributions:

$$J_{\text{prefix}}(\theta) = \mathbb{E}_{s_S \sim d_{\pi_p}^S}\big[V_\theta(s_S)\big], \tag{10}$$

$$J_{\text{base}}(\theta) = \mathbb{E}_{s_S \sim d_{\pi_\theta}^S}\big[V_\theta(s_S)\big]. \tag{11}$$

With Higher Quality Prefix The core premise of our method is that the frozen policy π_p (e.g., constructed via search or Best-of-N) is strictly stronger than the exploring policy π_θ. Mathematically, we assume π_p shifts probability mass towards a set of "good prefixes" $\mathcal{G}_S$, where continuation values are higher (i.e., prefixes that are correct or more concise). This implies a stochastic dominance relation:

$$\forall s \in \mathcal{G}_S, \quad d_{\pi_p}^S(s) > d_{\pi_\theta}^S(s) \quad \text{and} \quad V_\theta(s) \text{ is high.} \tag{12}$$

Prefix Rollout is better Under this assumption, the expected return of prefix rollout dominates the baseline. Specifically, since Best-of-N or similar search strategies explicitly select for correctness and shorter lengths, they ensure the prefix distribution aligns with high-value regions of V_θ. Thus, we obtain the following performance guarantee:

$$\boxed{\begin{aligned} \Delta d^S(s) &\triangleq d_{\pi_p}^S(s) - d_{\pi_\theta}^S(s), \\ J_{\text{prefix}}(\theta) - J_{\text{base}}(\theta) &= \int \Delta d^S(s)\, V_\theta(s)\, \mathrm{d}s \geq 0. \end{aligned}} \tag{13}$$

This proves that training on rollouts starting from higher-quality prefixes provides a mathematically superior optimization objective.

D Analysis of Difficulty-Aware Penalty

Why is an instance-dependent penalty necessary? We answer this by analyzing the reward dynamics of length regularization. We derive a critical *solvability bound* for the penalty coefficient. We show that a static coefficient faces a fundamental dilemma: it inevitably either causes model collapse on hard tasks or under-regularizes easy ones. Our difficulty-aware formulation is shown to be the necessary solution to resolve this conflict.

Solvability Condition Let L_q^* be the minimum sequence length required to correctly solve query q. For the model to be incentivized to generate this reasoning path rather than collapsing to a trivial failure (incorrect output with length ≈ 0, yielding reward 0), the total reward for the correct solution must be positive:

$$\begin{aligned} R_{\text{total}} &= 1 - \alpha L_q^*, \qquad R_{\text{total}} > 0 \\ \Rightarrow\ \alpha &< \frac{1}{L_q^*}. \end{aligned} \tag{14}$$

This inequality imposes a strict upper bound on the penalty coefficient α, which is inversely proportional to the intrinsic task complexity (length L_q^*).

The Dilemma of Uniform Penalty A fixed coefficient α_{uni} faces an impossible trade-off given the variance in L_q^*:

- **Under Hard Tasks** For complex queries where L_q^* is large, the bound $1/L_q^*$ is very small. If α_{uni} exceeds this threshold, the reward becomes negative, causing the model to give up and output abnormal responses.

- **Under Easy Tasks** To prevent collapse on hard tasks, α_{uni} must be set conservatively low. However, for easy queries (small L_q^*), this small α_{uni} exerts negligible pressure, failing to penalize redundancy effectively.

Resolution via Difficulty-Aware Penalty Our method resolves this conflict by modulating $\alpha(q)$ based on the empirical pass rate $\rho(q)$. Since easy tasks generally require shorter reasoning ($\rho \uparrow \Longrightarrow L^* \downarrow$), our schedule aligns with the theoretical bound:

$$\alpha(\rho) \propto \rho(q) \quad \text{Satisfies} \quad \alpha(q) < \frac{1}{L_q^*}. \tag{15}$$

This ensures α remains small enough to preserve solvability for hard tasks (low ρ) while becoming large enough to enforce strict conciseness on easy tasks (high ρ).

E Training Parameters

Parameters	Value
BF16	True
Train batch size	64
Mini batch size	64
Train epochs	10
Learning rate	1e-6
Max prompt length	2048
Max response length	16384
n	8
$\epsilon_{\min}$	0.2
$\epsilon_{\max}$	0.2
Temperature	1.0
warm up step	10
Rollout backend	VLLM

Table 6: RL training configurations for 7B and 1.5B Model

F Prompts

We compute the proportion of each reasoning behavior by aggregating its occurrences across the AIME2024 dataset and normalizing by the total number of reasoning-behavior occurrences. Formally,

$$p(b) = \frac{\sum_{i=1}^{N} c_i(b)}{\sum_{b' \in \mathcal{B}} \sum_{i=1}^{N} c_i(b')}, \tag{16}$$

where N is the number of samples in dataset, $\mathcal{B}$ denotes the set of reasoning behaviors, and $c_i(b)$ is the number of occurrences of behavior b in sample i. Detailed prompts are shown in Figure 8.

G Case study

In this section, we present a comprehensive case study of DeepSeek-R1-Distill-Qwen (7B/1.5B) on both in-domain and out-of-domain tasks (MATH500, AIME2024, GPQA-D, and LiveCodeBench-v6). As shown in Tables 9–13, PACE consistently achieves a token reduction rate of over 50% while preserving strong reasoning performance. Specifically, on the in-domain task, PACE markedly reduces *reflection* and *verification* behaviors yet still arrives at correct answers. A similar trend is observed on out-of-domain tasks, suggesting that PACE can reduce redundant reasoning steps while maintaining robust generalization.

Prompts for identifying Reasoning behavior

You are an expert analyst of Chain of Thought (CoT) reasoning processes. Your task is to analyze the provided reasoning text, break it down into distinct thought steps, and count the occurrences of four specific behaviors.

Behavior Definitions & Examples

1. **Advancing**

   Definition: Direct progress in solving the problem. This includes mathematical derivations, applying formulas, state transitions, or logical deductions that move the solution forward.

   Example: "Since $n = ab + c$, squaring both sides gives us $(a + c)^2 = ab + c$."

2. **Reflecting**

   Definition: Pausing the logical flow to evaluate the current path. This involves questioning assumptions, expressing uncertainty, or planning the next move. Look for markers like "Wait," "Hmm," "Let me think," or "Is this valid?"

   Example: "But b depends on a, so this approach might be circular. Let me try a different angle."

3. **Verifying**

   Definition: Testing a hypothesis or formula by plugging in specific, concrete numbers (often small integers or edge cases) to see if the theory holds up.

   Example: "Let's check for $d = 2$. If $s = 2$, does the formula work? $2^2 - 1 = 3 \ldots$"

4. **Refining**

   Definition: Correcting a previous formula, parameter, or conclusion based on the results of reflection or verification. This involves fixing errors or adding necessary constraints.

   Example: "So the formula isn't just 2^n, it must be $2^n - 1$ to account for the zero case."

Task Instructions

1. Read the **Input Text** below carefully.
2. Segment the text into logical units (sentences or short phases).
3. Classify each unit into one of the four behaviors defined above.
4. Count the total number of occurrences for each behavior.
5. **Output Requirement:** Return ONLY a valid JSON object containing the counts. Do not output any other text or explanation.

Input Text

```
{response}
```

Output Format

```
{
   "advancing_count": <integer>,
   "reflecting_count": <integer>,
   "verifying_count": <integer>,
   "refining_count": <integer>,
}
```

Figure 8: Prompts for identifying four reasoning behaviors and their corresponding explanations

Examples on MATH500

Question (MATH500 of 7B)

For what values of x is it true that $x^2 - 5x - 4 \leq 10$? Express your answer in interval notation.

DeepSeek-R1-Distill-Qwen-7B (2781 tokens)

<think>
I have this inequality: $x^2 - 5x - 4 \leq 10$. I need to find all the values of x and express the answer in interval notation. First, bring all terms to one side: $x^2 - 5x - 4 - 10 \leq 0$, which simplifies to $x^2 - 5x - 14 \leq 0$.
Next step is to find the roots of the corresponding equation $x^2 - 5x - 14 = 0$. Using the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$: here $a = 1, b = -5, c = -14$. So $x = \frac{5 \pm \sqrt{(-5)^2 - 4(1)(-14)}}{2(1)} = \frac{5 \pm \sqrt{25+56}}{2} = \frac{5 \pm \sqrt{81}}{2} = \frac{5 \pm 9}{2}$. The roots are $x_1 = \frac{14}{2} = 7$ and $x_2 = \frac{-4}{2} = -2$.
These roots divide the number line into three intervals: $(-\infty, -2)$, $(-2, 7)$, and $(7, \infty)$. Testing $x = -3$: $(-3)^2 - 5(-3) - 14 = 9 + 15 - 14 = 10 > 0$. Testing $x = 0$: $0^2 - 0 - 14 = -14 \leq 0$... [**omit 1982 redundant tokens of repeated verification**] ...Since the parabola opens upwards, it is negative between its roots.
</think>
To solve $x^2 - 5x - 4 \leq 10$, we rearrange it to $x^2 - 5x - 14 \leq 0$. Factoring the quadratic expression, we get $(x - 7)(x + 2) \leq 0$. The roots of the quadratic are $x = 7$ and $x = -2$. Because the coefficient of x^2 is positive, the parabola opens upward, and the expression is non-positive for values of x between and including the roots.
Final Answer: $\boxed{[-2, 7]}$

PACE (860 tokens)

<think>
I need to solve the inequality $x^2 - 5x - 4 \leq 10$. First, I will move all terms to one side to set the inequality to zero: $x^2 - 5x - 14 \leq 0$. Next, I solve for the roots of the quadratic $x^2 - 5x - 14 = 0$ by factoring. The factors of -14 that add up to -5 are -7 and 2, leading to $(x - 7)(x + 2) = 0$. The critical points are $x = 7$ and $x = -2$. Since the parabola opens upward ($a > 0$), the function is less than or equal to zero between the roots. Thus, the solution is the closed interval $[-2, 7]$.
</think>
To determine the values of x for which $x^2 - 5x - 4 \leq 10$, we start by rearranging the inequality so that the right-hand side is zero. Subtracting 10 from both sides gives:

$$x^2 - 5x - 14 \leq 0$$

Next, we factor the quadratic expression on the left-hand side. We look for factors of -14 that sum to -5. These are -7 and 2, so we can write:

$$(x - 7)(x + 2) \leq 0$$

The roots of the corresponding equation are $x = 7$ and $x = -2$. Since the leading coefficient of the quadratic is positive, the parabola opens upwards. Therefore, the expression is less than or equal to zero in the interval between the two roots, inclusive.
Final Answer: $\boxed{[-2, 7]}$

Figure 9: Case study on MATH500 dataset of 7B Model

Examples on MATH500 of 1.5B

Question (MATH500)

Compute $\arcsin\left(-\frac{1}{2}\right)$. Express your answer in radians.

DeepSeek-R1-Distill-Qwen-1.5B (4605 tokens)

<think>
Alright, so I have this problem: I need to compute the arcsine of $-\frac{1}{2}$, which is written as $\arcsin(-1/2)$. Hmm, okay. Let me recall what I know about $\arcsin$.
Arcsin is the inverse function of sine, right? So that means if $\arcsin(y) = x$, then $\sin(x) = y$. So in this case, I need to find an angle x such that $\sin(x) = -1/2$. Okay, got it. But wait, before jumping into that, I should consider the range of $\arcsin$.
I remember that the $\arcsin$ function has a range from $-\pi/2$ to $\pi/2$, or in degrees, from -90° to 90°. Hmm, actually, no. Wait, isn't the principal value of $\arcsin$ between $-\pi/2$ and $\pi/2$ in radians? So, it's defined to return the angle whose sine is the given value, but it only returns values between $-\pi/2$ and $\pi/2$. That makes sense because sine is positive in the first and second quadrants and negative in the fourth and third, but since $\arcsin$ only returns angles in the range of $-\pi/2$ to $\pi/2$, which is the fourth and first quadrants, the $\arcsin$ will only give negative or zero angles, right?
Wait, let's think about how sine behaves. I remember that sine is negative in the third and fourth quadrants. So, if $\sin x = -1/2$, one possible angle in the unit circle where sine is negative is considering the angle in the fourth quadrant. So, maybe 360° minus 30°, which is 330°, but we also have a special negative angle equivalent to that.
Alternatively, perhaps it's better to convert the angle to radians initially. So, π radians is 180°, right? So, if 30° is $\pi/6$ radians, then 330° would be $2\pi - \pi/6$. Because 360° is 2π radians, so subtracting $\pi/6$ gives $360 - 30 = 330$ degrees. So, that would be $11\pi/6$ radians. Wait, is that right?
Let me double-check that. So, in radians, 30° is $\pi/6$. So, yes, 360° is 2π. So, going clockwise from π radians (180°) would be $2\pi - \pi/6$, which is $11\pi/6$, but I'm getting a bit confused here.
Omit 1905 tokens
</think>

To compute $\arcsin\left(-\frac{1}{2}\right)$, we need to find an angle x such that $\sin(x) = -\frac{1}{2}$ and x lies within the range $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ (the principal value range of the arcsin function).

1. The arcsin function returns an angle whose sine is the given value and is within the range $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
2. The sine of $-\frac{\pi}{6}$ is $-\frac{1}{2}$ because $\sin(-\theta) = -\sin(\theta)$ and $\sin\left(\frac{\pi}{6}\right) = \frac{1}{2}$.
3. The angle $-\frac{\pi}{6}$ is within the range $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
4. Therefore, $\arcsin\left(-\frac{1}{2}\right) = -\frac{\pi}{6}$.

Thus, the value of $\arcsin\left(-\frac{1}{2}\right)$ is $\boxed{-\frac{\pi}{6}}$.

PACE (608 tokens)

<think>
Okay, so I need to compute the arcsin of $-1/2$. Hmm, arcsin is the inverse sine function, right? So, $\arcsin(-1/2)$ is the angle whose sine is $-1/2$. I remember that the sine function is negative in which quadrants? Well, sine is positive in the first and second quadrants and negative in the third and fourth quadrants. Since we're dealing with a negative value here, the angle must be in either the third or fourth quadrant.
Omit 182 tokens
Wait, just to make sure I'm not making a mistake here. So, $\arcsin(-1/2)$ is the angle whose sine is $-1/2$, and the angle must be between $-\pi/2$ and $\pi/2$. So, yes, $-\pi/6$ is within that range. So, the answer should be $-\pi/6$ radians.
</think>
To compute $\arcsin\left(-\frac{1}{2}\right)$, we need to find the angle whose sine is $-1/2$ and lies between $-\pi/2$ and $\pi/2$.
We know that $\sin(\pi/6) = 1/2$, so $\sin(-\pi/6) = -1/2$. Therefore, the angle is $-\pi/6$.

Thus, the value of $\arcsin\left(-\frac{1}{2}\right)$ is $\boxed{-\frac{\pi}{6}}$.

Final Answer

The value of $\arcsin\left(-\frac{1}{2}\right)$ is $\boxed{-\frac{\pi}{6}}$.

Figure 10: Case study on MATH500 dataset of 1.5B Model

Examples on AIME of 7B

Question (MATH500)

Let $\omega \neq 1$ be a 13th root of unity. Find the remainder when

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

is divided by 1000.

DeepSeek-R1-Distill-Qwen-7B (10451 tokens)

<think>
Okay, so I need to find the remainder when the product $\prod_{k=0}^{12}(2-2\omega^k+\omega^{2k})$ is divided by 1000, where ω is a primitive 13th root of unity, different from 1. Hmm, that sounds a bit complicated, but let me try to break it down.
First, I know that roots of unity have some useful properties. Specifically, for any n-th root of unity ω, we have $\omega^n = 1$ and the sum $\sum_{k=0}^{n-1}\omega^k = 0$. So, for 13th roots of unity, $\omega^{13} = 1$ and the sum from $k = 0$ to 12 of ω^k is zero.
Now, the expression I need to evaluate is a product from $k = 0$ to 12 of $(2-2\omega^k+\omega^{2k})$. That looks a bit like a quadratic in terms of ω^k. Let me write each term as a quadratic equation:

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

So, each term is $x^2 - 2x + 2$ where $x = \omega^k$. Therefore, the product is equal to $\prod_{k=0}^{12}(x^2-2x+2)$ evaluated at each root $x = \omega^k$ and then multiplied together.
Wait, but ω^k for $k = 0$ to 12 are all the 13th roots of unity. So, maybe I can consider the polynomial whose roots are ω^k, which is the cyclotomic polynomial. **Omit 8502 tokens**
</think>
...
After detailed calculations, we find that the product:

$$\prod_{k=0}^{12}(2-2\omega^k+\omega^{2k}) = 8321.$$

Finally, we find the remainder when 8321 is divided by 1000:

$$8321 \bmod 1000 = 321.$$

Thus, the remainder when the given product is divided by 1000 is:

$$\boxed{321}$$

PACE (2555 tokens)

<think>
Alright, so I have this problem here: Let ω be a 13th root of unity, not equal to 1. 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, okay. So, first off, ω is a primitive 13th root of unity because it's a 13th root and not equal to 1. So, $\omega^{13} = 1$, and the minimal polynomial for ω is the 13th cyclotomic polynomial, which is $\Phi_{13}(x) = x^{12} + x^{11} + \cdots + x + 1$. That might come in handy later.
The product we need is $\prod_{k=0}^{12}(2-2\omega^k+\omega^{2k})$. Let's denote this product as P. So, $P = \prod_{k=0}^{12}(2-2\omega^k+\omega^{2k})$. I need to compute this product and find $P \bmod 1000$.
Alternatively, maybe consider that the product over $k = 0$ to 12 of (x^2-2x+2) evaluated at $x = \omega^k$ is equal to the resultant of the two polynomials $x^{13}-1$ and x^2-2x+2.
But I'm not too sure about that.
Alternatively, maybe consider that since $\omega^{13} = 1$, so ω^{2k} is just another root, so perhaps we can pair terms or something.
</think>
..
Final Answer: $\boxed{321}$

Figure 11: Case study on AIME2024 dataset of 7B Model

Examples on GPQA-D of 7B

Question (GPQA-D)

Answer the following multiple choice question. The last line of your response should be of the following format: 'Answer: $LETTER' (without quotes) where LETTER is one of ABCD. Think step by step before answering.

Consider this annihilation process:

$$p + \bar{p} \rightarrow 2A^+ + 2A^-$$

The antiproton is slowly moving and $m_A c^2 = 300\,\text{MeV}$.
What is the velocity of particle A?

A) $0.96c$

B) $0.86c$

C) $0.91c$

D) $0.77c$

DeepSeek-R1-Distill-Qwen-7B (2781 tokens)

<*think*> Okay, so I've got this physics problem here, and I'm a bit rusty, but I'll try to work through it. Let me read the question again.
The problem says: $p + \bar{p} \rightarrow 2A^+ + 2A^-$. The antiproton is moving slowly, and the mass of each A particle is $300\,\text{MeV}/c^2$. We need to find the velocity of particle A. The options are A to D with various fractions of c.
Hmm, wait, wait. So, the reaction is proton and antiproton annihilating into two A particles, each with a rest mass of $300\,\text{MeV}/c^2$. But wait, protons and antiprotons are matter and antimatter, so their annihilation would produce energy. But in this case, the products are two A^+ and two A^-. I'm a bit confused about the charges here. Let me think: a proton has $+1$ charge, antiproton has -1. So the products must balance the charge. $2A^+$ is $+2$, and $2A^-$ is -2. So the total charge is zero, same as the initial. That makes sense.
But I'm not sure how that affects the problem. The question is about the velocity of A. So perhaps it's a matter of conservation of energy and momentum.
The antiproton is moving slowly, so its velocity is much less than c. Hmm, but what does "slowly moving" mean here? Maybe it's non-relativistic, so that kinetic energy is much less than rest mass energy. But wait, the rest mass of a proton is about $938\,\text{MeV}/c^2$. So if the antiproton is moving slowly, perhaps its kinetic energy is negligible compared to its rest mass.
Omit 9812 tokens
</*think*>
Since the discriminant is negative, there is no real solution, indicating an error in the assumptions or calculations. The calculations suggest inconsistencies, likely due to approximations or assumptions about the distribution of energy and momentum. However, based on the options provided and the expected range for v, the most reasonable answer is $0.77c$. Final Answer: $\boxed{D}$

PACE (4576 tokens)

<*think*>
Okay, so I've got this physics problem here about particle annihilation. Hmm, let's see. The reaction is $p + \bar{p} \rightarrow 2A^+ + 2A^-$. Wait, p is proton and $\bar{p}$ is antiproton. Oh right, protons and antiprotons annihilate each other, producing other particles.
The mass of each A is given as $300\,\text{MeV}/c^2$. Wait, but wait, the question is about the velocity of particle A. So each A has mass $m_A c^2 = 300\,\text{MeV}$. So each A is a particle with mass $300\,\text{MeV}/c^2$.
The antiproton is moving slowly, so its kinetic energy is probably negligible compared to the proton's. But wait, the proton is at rest? Or is the antiproton moving slowly? The question says the antiproton is slowly moving, so maybe its kinetic energy is small compared to the proton's rest energy. But wait, protons are about $938\,\text{MeV}/c^2$ in rest mass. If the antiproton is moving slowly, its kinetic energy would be much smaller than the proton's. Hmm, but wait, maybe I should think about conservation of energy and momentum.
Omit 3509 tokens
</*think*>
..
The velocity of particle A is approximately $0.77c$.
Final Answer: $\boxed{D}$

Figure 12: Case study on GPQA-D dataset of 7B Model

Examples on LiveCodeBench-v6 of 7B

Question (LiveCodeBench-v6)

There is an infinite 2D plane. You are given a positive integer k. You are also given a 2D array `queries`, where the queries are defined as follows:

- `queries[i] = [x, y]`: Build an obstacle at coordinate (x, y) in the plane. It is guaranteed that there is no obstacle at this coordinate when this query is made.

After each query, you need to find the distance of the k-th nearest obstacle from the origin $(0, 0)$.
Return an integer array `results` where `results[i]` denotes the distance of the k-th nearest obstacle after query i, or `results[i] == -1` if there are less than k obstacles.

Note:

- Initially, there are no obstacles anywhere.
- The distance of an obstacle at coordinate (x, y) from the origin is given by the Manhattan distance: $|x| + |y|$.

Omit 402 tokens

DeepSeek-R1-Distill-Qwen-7B (12099 tokens)

<think>
Okay, I need to solve this problem where I have to process a series of queries and for each, determine the k-th nearest obstacle from the origin. Hmm. So, the problem says that each query adds an obstacle, and after each addition, I have to find the k-th nearest obstacle. The distance is the Manhattan distance, which is $|x| + |y|$ for a point (x, y). Let's think about the approach. For each query, I add a new obstacle, and then I need to keep track of all the obstacles so far. After each addition, I need to find the k-th smallest distance. But wait, if I have a lot of queries, say up to 2e5, then for each query, if I have to sort all the obstacles each time, that could be $O(n \log n)$ per query, which would be $O(n^2 \log n)$ total time. That's probably too slow for 2e5 queries. So I need a more efficient way to manage the distances and quickly find the k-th smallest each time. Hmm. What's the maximum number of obstacles we can have? Well, each query adds one obstacle, so up to 2e5 obstacles.
Omit 10345 tokens
</think>

```
import heapq\ def resultsArray(queries, k):\ heap = []\ res = []\ for i, (x, y) in
enumerate(queries):\ d = abs(x) + abs(y)\ m = i + 1\ if m < k:\ heapq.heappush(heap,
-d)\ res.append(-1)\ else:\ current_max = -heap[0] if heap else 0\ if d < current_max:\
heapq.heappush(heap, -d)\ heapq.heappop(heap)\ res.append(-heap[0] if heap else -1)\
return res
```

PACE (4895 tokens)

<think>
Okay, I need to solve this problem where I have to process a series of queries and for each, determine the k-th nearest obstacle from the origin. Hmm. So, the problem says that each query adds an obstacle, and after each addition, I have to find the k-th nearest obstacle. The distance is the Manhattan distance, which is $|x| + |y|$ for a point (x, y). Let's think about the approach. For each query, I add a new obstacle, and then I need to keep track of all the obstacles so far. After each addition, I need to find the k-th smallest distance. But wait, if I have a lot of queries, say up to 2e5, then for each query, if I have to sort all the obstacles each time, that could be $O(n \log n)$ per query, which would be $O(n^2 \log n)$ total time. That's probably too slow for 2e5 queries. So I need a more efficient way to manage the distances and quickly find the k-th smallest each time. Hmm. What's the maximum number of obstacles we can have? Well, each query adds one obstacle, so up to 2e5 obstacles.
Omit 3034 tokens
</think>

```
import heapq\ class Solution:\ def resultsArray(self, queries: List[List[int]], k: int)
-> List[int]:\ heap = []\ count = 0\ res = []\ for q in queries:\ x, y = q\ d = abs(x)
+ abs(y)\ count += 1\ if count <= k:\ heapq.heappush(heap, -d)\ else:\ current_max =
-heap[0]\ if d < current_max:\ heapq.heappush(heap, -d)\ heapq.heappop(heap)\ if count >=
k:\ res.append(-heap[0])\ else:\ res.append(-1)\ return res
```

Figure 13: Case study on LiveCodeBench-v6 dataset of 7B Model