SIRI: Scaling Iterative Reinforcement Learning with Interleaved Compression

Anonymous ACL submission

Abstract

We introduce SIRI, **S**caling **I**terative **R**einforcement Learning with **I**nterleaved Compression, a simple yet effective RL approach for Large Reasoning Models (LRMs) that enables more efficient and accurate reasoning. Existing studies have observed repetitive thinking patterns in LRMs, and attempts to reduce them often come at the cost of performance. In this paper, we show that this trade-off can be overcome through a training regime that iteratively alternates between compressing and expanding the reasoning budget. The *compression phase* cuts the rollout length, forcing the model to make precise and valuable decisions within a limited context. The *expansion phase* then relaxes the length limit, providing space for the model to explore and plan in long-horizon settings. Remarkably, we find that after each compression-expansion cycle, the model's performance improves even as its output length decreases, steadily pushing it closer to the Pareto frontier in the performance-efficiency trade-off. Training on DeepSeek-R1-Distill-Qwen-1.5B, **SIRI-low** improves performance on AIME24 by 43.2% while reducing token usage by 46.9% after three iterations, and **SIRI-high** achieves the highest accuracy compared to all other methods (Figure 1). Our findings shed light on the potential of periodically oscillating the LRM's output truncation length during training to dynamically balance exploration and efficiency in reasoning, converging towards an optimal "sweet spot" between the two.

1 Introduction

Large Language Models (LLMs) have become the frontier of AI research, demonstrating impressive performance across a wide range of domains. In particular, Large Reasoning Models (LRMs) (OpenAI, 2024; Guo et al., 2025; Zeng et al., 2025), a branch of LLMs tailored for reasoning tasks such as math and coding, have witnessed a great leap recently, empowered by large-scale reinforcement learning (RL) algorithms (Schulman et al., 2017; Shao et al., 2024). These models are trained using Test-Time Scaling strategy (Snell et al., 2024; Muennighoff et al., 2025) that appears as long Chain-of-Thoughts (Wei et al., 2022), utilizing the model's verification, exploration and iterative refinement abilities at test time to obtain superior reasoning capability.

However, although RL can boost the model's performance on reasoning tasks, it also inevitably causes a rapid increase in the model's output length. The amount of overthinking rises significantly (Chen et al., 2024a; Qu et al., 2025), leading to a notable increase in both training and inference costs. To overcome this issue, prior works have attempted approaches such as fine-tuning with precise reasoning traces (Kang et al., 2025; Ma et al., 2025), introducing length penalties or length truncation during RL (Team et al., 2025; Aggarwal and Welleck, 2025; Luo et al., 2025a), and adopting hybrid reasoning strategies to automatically switch between thinking and non-thinking modes (Fang et al., 2025; Zhang et al., 2025; Lou et al., 2025) to improve token efficiency. However, without exception, these methods degrade or stagnate the model's performance, preventing it from fully unlocking its capabilities. To support this, in Figure 1, all other length-compression approaches perform worse than models trained with standard RL (DeepScaleR-DAPO-16k) by a large margin, placing them inside the Pareto frontier.

In this paper, we propose SIRI, **S**caling **I**terative **R**einforcement Learning with **I**nterleaved Compression, a simple yet effective framework that pushes the Pareto frontier by simultaneously reducing token usage and improving reasoning accuracy. The key idea of SIRI is to **periodically alternate between compressing and expanding reasoning budget during training** by dynamically adjust-

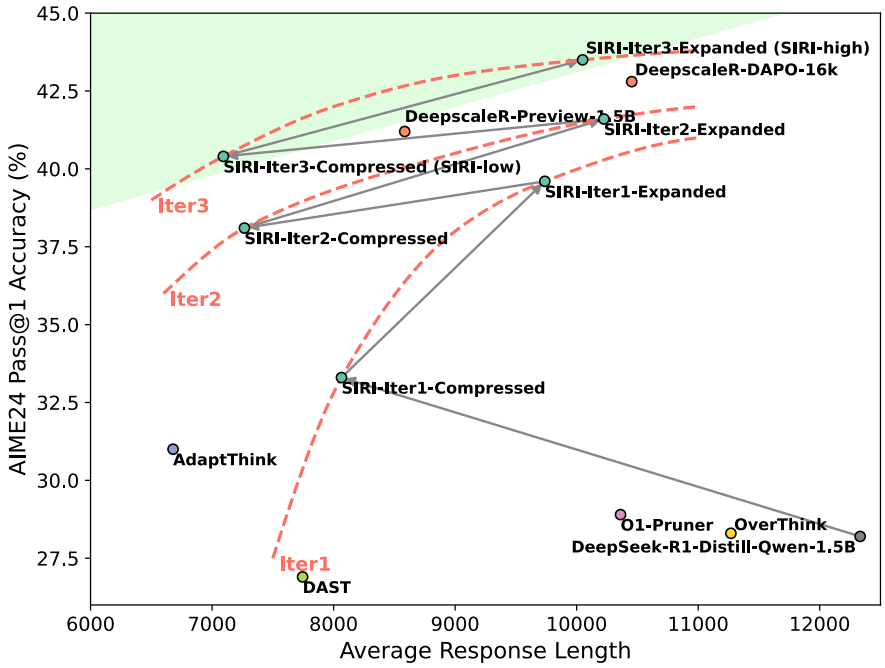


Figure 1: Performance-efficiency comparison between different training methods applied to DeepSeek-R1-Distill-Qwen-1.5B. SIRI continually pushes the model to the Pareto frontier.

ing the maximum rollout length according to a cosine scheduler. The compression phase forces the model to think concisely by reducing overthinking, while the expansion phase encourages the model to further explore based on mature reasoning traces. Unlike prior approaches that suffer from a strict efficiency-performance trade-off, SIRI leverages the compression–expansion cycle to achieve steady gains in accuracy despite shorter outputs. As shown in SIRI's evolution trace in Figure 1, with each iteration the model spirals upward by using fewer tokens and achieving higher performance.

Empirically, our method demonstrates superior performance against state-of-the-art models trained under the same 16K output limit. Training based on DeepSeek-R1-Distill-Qwen-1.5B model, the expanded-length variant (SIRI-high) achieves a **43.6%** pass@1 on AIME24 with an average of **10K** tokens, while the compressed-length variant (SIRI-low) achieves **40.4%** pass@1 using only **7K** tokens on average. This nearly halves the token usage while still delivering a **43%** improvement over the initial pre-RL model. We contribute the core factor in SIRI's success to compressing length in each iteration while not letting accuracy fall off the cliff. Furthermore, we find that SIRI generalizes effectively across different model sizes and out-of-distribution tasks. In summary, these findings suggest that iterative reinforcement learning further unlocks the model's reasoning potential under limited training context.

2 Related works

2.1 Length compression for LRMs

Large reasoning models leverage test-time scaling to boost performance, but they frequently expend unnecessary tokens on repeated backtracking, unnecessary exploration, and non-reasoning filler (Hou et al., 2025). To mitigate such inefficiency, previous studies mostly leverage reward shaping for online RL training (Team et al., 2025; Aggarwal and Welleck, 2025; Luo et al., 2025b; Hou et al., 2025; Zhang et al., 2025) or precise reasoning traces for offline fine-tuning (Ma et al., 2025; Yang et al., 2025). The most commonly used reward shaping strategy is adaptive length penalty (Team et al., 2025), which is further augmented by prompt engineering (Aggarwal and Welleck, 2025). Another line of work apply budget forcing on models by either setting the reward to zero or forcing the model to end thinking and generate the final solution once the output length surpasses the token budget (Luo et al., 2025b; Hou et al., 2025; Muennighoff et al., 2025). Recent studies also try to abandon the thinking process of relatively simple problems by training the model to generate end-of-thinking tokens (Yang et al., 2025; Zhang et al., 2025; Fang et al., 2025; Lou et al., 2025). However, all methods above inevitably compromise model performance compared to full-

length RL training, while our work shows that we can close this gap by an iterative RL framework.

2.2 Iterative training

Iterative training methods have been widely used in preference alignment and simple reasoning tasks because of their ability to leverage additional data generated by the model over iterations. One line of work uses maximum likelihood iterative training to enhance the model's reasoning abilities (Gulcehre et al., 2023; Singh et al., 2024). During each iteration, a dataset is first generated by the current model and labeled by a reward model. Then, this dataset is used to fine-tune the base model, yielding a stronger version for the next iteration. Another line utilizes positive and negative samples with Direct Preference Optimization (DPO) algorithms (Rafailov et al., 2023; Xiong et al., 2023; Chen et al., 2024b). In GSHF (Xiong et al., 2023), new chosen/rejected responses are generated in each iteration and added to the dataset for wider dataset coverage. In SPIN (Chen et al., 2024b), the chosen responses are generated by humans and fixed, while the model iteratively generates rejected responses and aligns with the human dataset. The most recent work unifies these two directions (Pang et al., 2024), applying both NLL and DPO loss for arithmetic tasks, witnessing modest gains in the GSM8K dataset. However, all methods above demonstrate only how iterative training can be applied in off-policy training scenarios for simple reasoning problems. In this work, we further show that iterative on-policy RL can be used to effectively advance the performance-efficiency Pareto frontier of LRMs.

3 Preliminaries

3.1 Problem formulation

We consider a Large Language Model (LLM) parameterized by θ, denoted by π_θ and a math dataset $\mathcal{D}$ with question-answer pairs, each denoted by $(\boldsymbol{x}, a^*)$. The model samples a problem $\boldsymbol{x}$ and generates a response $\boldsymbol{y} = [y_1, ..., y_m]$ sampled from the conditional distribution $\pi_\theta(\cdot|\boldsymbol{x})$. In the LLM's setting, each element in $\boldsymbol{y}$ is known as an output token. Specifically, for a Large Reasoning Model (LRM) trained on mathematical tasks with a fixed answer, the last output token, y_m, is the model's predicted answer for the problem. By defining the scoring function $R(\boldsymbol{y})$ and setting $R(\boldsymbol{y}) = 1$ if $a^* = y_m$ and $R(\boldsymbol{y}) = 0$ otherwise, we aim to find θ^* that satisfies

$$\theta^* = \arg\max_\theta \mathbb{E}_{(\boldsymbol{x},a^*)\sim\mathcal{D},\boldsymbol{y}\sim\pi_\theta(\cdot|\boldsymbol{x})} \Big[R(\boldsymbol{y}) \mid |\boldsymbol{y}| \leq L\Big],$$

where $|\boldsymbol{y}|$ is the length of the output $\boldsymbol{y}$, and L is the token budget.

3.2 Overlong reward shaping

A common approach to control the response length of LRMs during post-training is reward shaping. The length-capping reward introduced in DeepScaleR (Luo et al., 2025b) assigns a reward to each response $\boldsymbol{y}$ based on a maximum length L as follows:

$$R(\boldsymbol{y}) = \begin{cases} 1, & \text{if an answer can be extracted from} \\ & \text{clip}(\boldsymbol{y}, L) \text{ and is correct,} \\ 0, & \text{otherwise.} \end{cases}$$

In comparison, some works also proposed using adaptive length penalty for reward shaping (Team et al., 2025; Zhang et al., 2025). While this method is mathematically justified, it requires complex hyperparameter tuning for the penalty coefficient. Moreover, its training efficiency is worse than the direct capping method. For these reasons, we adopt the simpler length-capping reward in this work.

3.3 Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) (Shao et al., 2024), based on Proximal Policy Optimization (PPO) (Schulman et al., 2017), is widely used in post-training of LRMs. For each question-answer pair $(\boldsymbol{x}, a^*)$ sampled from dataset $\mathcal{D}_{\text{GRPO}}$, π_θ samples G individual responses $\{\boldsymbol{y}_i\}_{i=1}^G$ and estimates the advantage of the i-th response with group-level rewards

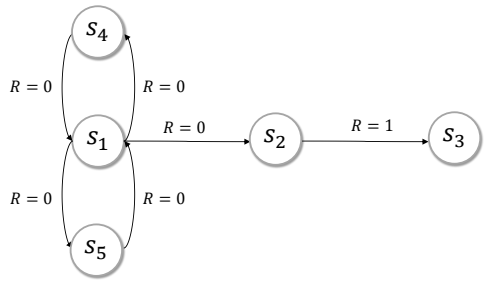
$$\hat{A}_{i,t} = \frac{R(\boldsymbol{y}_i) - \text{mean}\big(\{R(\boldsymbol{y}_i)\}_{i=1}^G\big)}{\text{std}\big(\{R(\boldsymbol{y}_i)\}_{i=1}^G\big)}.$$

Then, the loss of the policy is calculated by

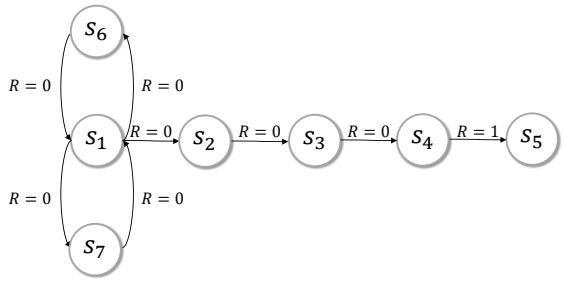
$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}_{(\boldsymbol{x},a^*)\sim\mathcal{D}_{\text{GRPO}},\{\boldsymbol{y}_i\}_{i=1}^G\sim\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{x})} \Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\boldsymbol{y}_i|}\sum_{t=1}^{|\boldsymbol{y}_i|}\Bigg(\min\Big(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\Big(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\Big)\hat{A}_{i,t}\Big) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Bigg)\Bigg],$$

where

$$r_{i,t}(\theta) = \frac{\pi_\theta(\boldsymbol{y}_{i,t} \mid \boldsymbol{x}, \boldsymbol{y}_{i,<t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{i,t} \mid \boldsymbol{x}, \boldsymbol{y}_{i,<t})}.$$



(a) An MDP illustrating redundant thinking patterns. The graph's initial state is s_1 and the goal state is s_3.



(b) An MDP illustrating a task that requires complicated reasoning. The graph's initial state is s_1 and the goal state is s_5.

Figure 2: Representative MDPs. Reward is given only at the goal state.

4 Method: SIRI

4.1 Motivation

In DeepScaleR's (Luo et al., 2025b) 8K training stage, there is an increase in the model's performance despite a sharp response length drop. This shows that *the model can compress key reasoning steps into shorter contexts, thus freeing capacity for exploration in the subsequent 16K stage*. However, the following context-expansion stage may again introduce redundant reasoning patterns, leading to suboptimal policies. Therefore, we hypothesize that interleaving compression and expansion will help the model learn more *token-efficient* policy compared to consistently training it under long-context, as well as learning more *accurate* policy compared to consistently training it under short-context.

As a behavioural explanation, consider the toy MDPs with deterministic transitions shown in Figure 2. Assume that, for iterative schedule, the rollout horizons (steps) are $H_{2k-1} = 2$, $H_{2k} = 4$ for odd and even policy update steps; While for long-context and short-context schedule, the rollout horizons are $H_k = 4$ and $H_k = 2$ for all policy update steps, respectively. When evaluating all policies under $H = 4$, we have the following conclusions:

Iterative training improves efficiency and reduces redundant reasoning. In the MDP shown in Figure 2a, the initial state is s_1, the goal state is s_3, the unique goal-reaching path is $s_1 \to s_2 \to s_3$, and all other actions from s_1 enter two-step loops, representing redundant exploration/verification behavior of the agent. Under this MDP setting, we prove that the iterative rollout schedule yields a more *efficient* policy with *shorter hitting time* compared to pure long-context rollout schedule. Formally, for policies trained by iterative or long-context schedule for t steps, denote $\mathbb{E}_I^t[T_G], \mathbb{E}_L^t[T_G]$ as their expected time to reach s_3 respectively, we have the following theorem:

Theorem 1. *Let the policy at step $k+1$ be updated by*

$$\pi_{k+1}(\cdot|s) = \arg\max_{\pi'(\cdot|s)} \mathbb{E}_{a\sim\pi'(\cdot|s)}\left[A^{\pi_k}(s,a)\right] - \beta D_{KL}(\pi'(\cdot|s)\|\pi_k(\cdot|s)), 0 < \beta \le 4,$$

then

$$E_I^t[T_G] \le E_L^t[T_G], \forall t \in \mathbb{N}.$$

Intuitively, the compression stage of the iterative schedule penalizes transitions that lead to redundant loops, while the long-context counterpart assigns them non-negligible probability mass.

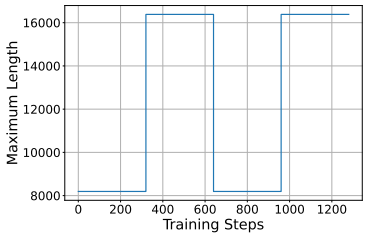
Iterative training improves capability of long-horizon reasoning. As another example, Figure 2b demonstrates a complicated multi-step task with sparse rewards. There is a path from the initial state s_1 to the goal state s_5, in which all transitions have zero reward except $s_4 \to s_5$. In this setting, we prove that the iterative rollout schedule yields a more *capable* policy with *higher expected return* compared to pure short-context training strategies. Formally, for policies trained by iterative or short-context schedule for t steps, denote $\mathbb{E}_I^t[R], \mathbb{E}_S^t[R]$ as their expected return respectively, we have the following theorem:

Theorem 2. *Let the policy at step $k+1$ be updated by*

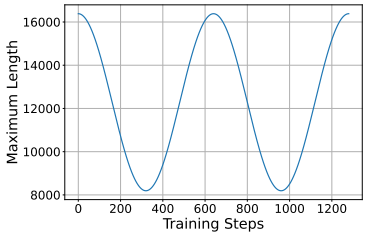
$$\pi_{k+1}(\cdot|s) = \arg\max_{\pi'(\cdot|s)} \mathbb{E}_{a\sim\pi'(\cdot|s)}\left[A^{\pi_k}(s,a)\right] - \beta D_{KL}(\pi'(\cdot|s)\|\pi_k(\cdot|s)), \beta > 0,$$

then

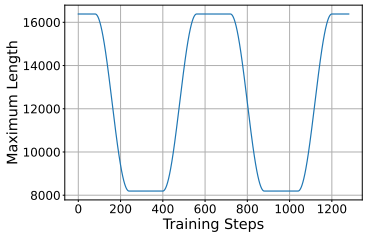
$$E_I^t[R] \ge E_S^t[R], \forall t \in \mathbb{N}.$$



(a) Stair scheduler



(b) Cosine scheduler



(c) Stair-cosine scheduler

Figure 3: Illustration of different schedulers with cycle length 640.

Intuitively, the expansion stage allows the model to explore longer trajectories and discover rewarding paths, which are difficult to observe under short-context rollouts.

Remark. Full formalizations and proofs are provided in Appendix A.

4.2 Length scheduler

In iterative training, the design of the length scheduler is important as it controls the compression and exploration behavior of the model. The scheduler should have the following properties: **1)** minimize performance degradation while maximize average token expenditure during the compression phase, and **2)** maximize exploration and performance during the expansion phase. Here, we introduce three types of schedulers. To unify notation, let T denote the cycle length (in steps), t be the current step, L_{max} and L_{min} denote the maximum and minimum capping threshold during each cycle. Figure 3 illustrates the curves of the respective schedulers.

Stair scheduler. The stair scheduler reduces the maximum generation length from the upper capping threshold L_{max} to the lower capping threshold L_{min} during the compression phase. It then switches from L_{min} to L_{max} when the model enters the expansion phase.

Cosine scheduler. To make the length reduction and recovery process smooth, we also investigate the cosine scheduler. The maximum generation length at each step t can be written as

$$\frac{L_{\text{max}} + L_{\text{min}}}{2} + \frac{L_{\text{max}} - L_{\text{min}}}{2} \cdot \cos(\frac{2\pi}{T} \cdot t).$$

Stair-cosine scheduler. The cosine scheduler doesn't maintain at L_{max} and L_{min}. However, this may hinder the model's ability to further explore at L_{max} after expansion and restore performance at L_{min} after compression. Thus, we combine the stair and cosine scheduler into a unified scheduler that ensures both smoothness of the whole process, exploration at L_{max}, and exploitation at L_{min}. Letting the current phase be $\phi = 2\pi \cdot \frac{t \bmod T}{T}$, the whole schedule can be written as

$$\begin{cases} L_{\text{max}}, & \phi < \frac{\pi}{4} \text{ or } \phi \geq \frac{7\pi}{4}, \\ \frac{L_{\text{max}}+L_{\text{min}}}{2} + \frac{L_{\text{max}}-L_{\text{min}}}{2} \cdot \cos(2(\phi - \frac{\pi}{4})), & \frac{\pi}{4} \leq \phi < \frac{3\pi}{4}, \\ L_{\text{min}}, & \frac{3\pi}{4} \leq \phi < \frac{5\pi}{4}, \\ \frac{L_{\text{max}}+L_{\text{min}}}{2} + \frac{L_{\text{max}}-L_{\text{min}}}{2} \cdot \cos(2(\phi - \frac{3\pi}{4})), & \frac{5\pi}{4} \leq \phi < \frac{7\pi}{4}. \end{cases}$$

5 Experiments

In this section, we conduct extensive experiments to validate our compression-expansion approach. Specifically, our experiments are designed to answer the following questions:

RQ1: Can the compression-expansion scheme enhance reasoning accuracy while pruning redundant tokens? What is the underlying mechanism behind this behavior?

RQ2: What is the best design of the length scheduler?

RQ3: Is the compression-expansion scheme generalize to different models and tasks?

5.1 Experiment setup

Model. For the initial pre-RL model, we select two representative open-source large reasoning models with different sizes: DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025), both fine-tuned on expert trajectories generated by DeepSeek-R1. For baseline

comparison, we evaluate several popular RL approaches, including DeepScaleR-Preview (Luo et al., 2025b) (released checkpoint from the original DeepScaleR work), DAPO-DeepScaleR-16K (trained with DeepScaleR's 8K compression followed by 16K expansion schedule, but using DAPO's clip-higher and no KL-loss strategies for a fairer comparison with our method), OverThink (Chen et al., 2024a), DAST (Shen et al., 2025), O1-Pruner (Luo et al., 2025a), and AdaptThink (Zhang et al., 2025). **All baseline models are trained on the same dataset as ours.**

Implementation detail. For RL training, we use the VeRL framework (Sheng et al., 2024). We adopt the GRPO (Shao et al., 2024) algorithm for training, but decouple the upper and lower thresholds for clipping, as well as removing the KL divergence, as proposed in DAPO (Yu et al., 2025). Specifically, we set 0.28 for clip-high and 0.2 for clip-low. For the length scheduler, we set L_{max} at 16384 and L_{min} at 8192. The models are trained with a sampling temperature of 1.0, a batch size of 128, and a learning rate of 1e-6. We use 8×H100 GPUs for training the 1.5B model and 16×H100 GPUs for the 7B model. To make our model training process more transparent, the training Wandb logs can be accessed here, while the details of dataset and evaluation can be seen in Appendix C.1.

5.2 Results

Table 1 shows the main evaluation results. Our models, SIRI-low (SIRI-Iter3-Compressed) and SIRI-high (SIRI-Iter3-Expanded), were trained with a 640-cycle cosine length scheduler over three iterations. Compared to the original 1.5B model, SIRI-low reduces response length by 43.1% and boosts performance by 27.0% on average. After expansion of SIRI-low during the third iteration, we yield SIRI-high that achieves the highest accuracy on all benchmarks, improving performance by 33.6% on average. A similar trend can also be seen on the 7B model. These results show that the interleaved compression phase enhances, instead of mitigates, the model's potential to explore and plan in long Chain-of-Thought. Regarding generation length, while SIRI-low produces longer responses than models trained with adaptive length penalties (e.g., DAST) or "no-thinking" methods (e.g., AdaptThink) on easier benchmarks (AMC23 and MATH500), its output length is comparable to them for the 1.5B model and notably shorter for the 7B model on more challenging benchmarks (AIME24 and AIME25). In addition, SIRI-low also performs similarly with DeepScalerR-Preview-1.5B (the latter is trained under 24K context). This demonstrates SIRI's robustness across tasks and its advantage on difficult problems.

We additionally report the accuracy-CR ratio that evaluates the change in the model's token efficiency after training. We find that models trained by SIRI have the optimal accuracy-CR ratio, showing that iterative compression with a length scheduler is better at pruning redundant tokens compared to other methods. We detail our findings below.

Token efficiency iteratively improves. Figure 4 shows the training dynamic of the 1.5B model trained by the 640-cycle cosine scheduler. During the iterations, we witness a stable increase in token efficiency, where the model ends each cycle with almost the same response length, but its Pass@1 accuracy consistently improves, eventually surpassing 43%. This shows that the model's reasoning is condensed through the iteration of compression and expansion. Meanwhile, as shown in Figure 1, the model keeps pushing the Pareto frontier forward after each iteration, resulting in higher accuracy as well as greater token efficiency. Specifically, the accuracy of the compressed-length variant (SIRI-Iter1/2/3-Compressed) goes up across iterations: $33.3\% \rightarrow 38.1\% \rightarrow 40.4\%$, while the average response length continually goes down: $8065 \rightarrow 7266 \rightarrow 7093$.

To validate the advantage of our iterative compression-expansion scheme over DeepScaleR's two-stage compression-then-expansion approach, we compare SIRI with DeepScaleR-DAPO-16K under similar token budget: SIRI-low, SIRI-high and DeepScaleR-DAPO-16K's token budget are approximately 20.5×10^9, 26.1×10^9 and 24.2×10^9 respectively. As shown in Figure 5, SIRI reaches comparable performance on the AIME24 benchmark while largely outperforming it on the more challenging AIME25 benchmark with substantially fewer tokens. These findings suggest that SIRI's iterative compression scheme effectively improves token efficiency and is better adapted to more demanding, reasoning-intensive tasks. A similar observation can also be drawn from the dynamics of the 7B model, as discussed in Appendix C.2.

Generalization on out-of-distribution (OOD) tasks. To test whether the interleaved training schedule can cut token usage while maintaining performance on OOD scenarios, we per-

Method	AIME24		AIME25		AMC23		MATH500		Average
	Acc	Length	Acc	Length	Acc	Length	Acc	Length	$\frac{\Delta\text{Acc}}{\text{CR}}(\uparrow)^\star$
DeepSeek-R1-Distill-Qwen-1.5B									
Original	28.2	12333	21.5	12264	61.8	8449	82.4	4745	0.00
DeepScaleR-Preview (Luo et al., 2025b)	41.1	8585	29.0	8348	73.9	5515	87.6	3054	0.39
DAPO-DeepScaleR-16K	42.8	10453	30.9	10352	74.6	7339	88.1	4223	0.36
OverThink (Chen et al., 2024a)†	28.3	11269	–	–	–	–	81.2	4131	0.00
DAST (Shen et al., 2025)†	26.9	7745	–	–	–	–	83.0	2428	0.01
O1-Pruner (Luo et al., 2025a)†	28.9	10361	–	–	–	–	82.2	3212	0.01
AdaptThink (Zhang et al., 2025)	31.0	**6679**	22.3	6800	63.3	**3498**	82.0	**1782**	0.08
SIRI-low (Ours)	40.4	7093	29.6	**6509**	74.6	4700	87.7	2881	**0.47**
SIRI-high (Ours)	**43.6**	10049	**32.2**	9739	**75.9**	7396	**88.4**	4633	0.38
DeepSeek-R1-Distill-Qwen-7B									
Original	53.5	10306	38.3	11114	79.4	6740	90.2	3674	0.00
DAPO-DeepScaleR-16K	**57.6**	9983	40.8	10705	84.5	6508	92.5	3658	0.06
OverThink (Chen et al., 2024a)†	53.1	8744	–	–	–	–	89.4	2435	0.00
DAST (Shen et al., 2025)†	45.6	7578	–	–	–	–	89.6	2162	0.00
O1-Pruner (Luo et al., 2025a)†	49.2	9719	–	–	–	–	86.6	2534	0.00
AdaptThink (Zhang et al., 2025)	55.6	8546	37.0	9556	80.1	4778	90.6	**1868**	0.02
SIRI-low (Ours)	56.1	**6122**	41.5	**6386**	85.8	**4015**	93.5	2452	0.10
SIRI-high (Ours)	57.1	8585	**45.4**	9106	**86.7**	5773	**93.7**	3378	**0.11**

* $\Delta\text{Acc} = \max(\frac{\text{current model accuracy}}{\text{initial model accuracy}} - 1, 0)$, CR (Compressed Ratio) $= \frac{\text{current model length}}{\text{initial model length}}$. Higher is better.

† For these methods, we directly use the results reported in AdaptThink (Zhang et al., 2025). Since the corresponding checkpoints were not released, we are unable to evaluate them on AIME25 and AMC23.

Table 1: Performance comparison on AIME24, AIME25, MATH500 and AMC23. Best result in **bold** and second best underlined.

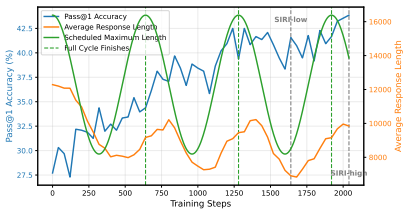
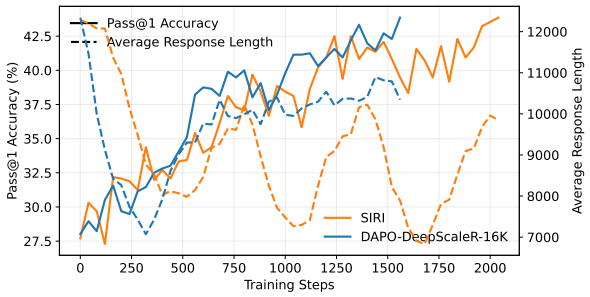


Figure 4: The 1.5B model's Pass@1 accuracy and average response length of SIRI with 640-cycle length cosine scheduler over three iterations on the AIME24 benchmark.

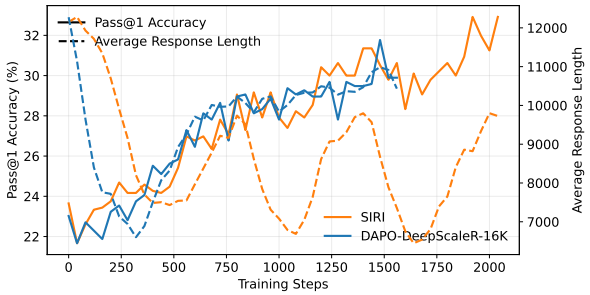
formed additional evaluations on general benchmarks, including GPQA-Diamond (Rein et al., 2024), LiveCodeBench (Jain et al., 2024) and HumanEval (Chen et al., 2021), which evaluate the model's STEM reasoning and coding abilities. The results are shown in Appendix C.4. We find that the model's performance on GPQA-Diamond is well preserved by SIRI. In addition, SIRI brings additional gains to the model's coding capabilities. Notably, SIRI-1.5B-low yields 3% performance improvement with 36.5% token reduction on GPQA-Diamond, and 19% performance improvement with 24.5% token reduction on LiveCodeBench. Together, these results show that the compression-expansion training scheme generalizes well to different task domains.

5.3 Ablations on scheduler design

Scheduler with a longer cycle performs best. The design of the scheduler is the key to iterative improvement in each cycle. Figure 6a demonstrates the 1.5B model's performance of cosine scheduler with cycle lengths of 320, 480, and 640. During the compression stage, the 320-cycle and 480-cycle scheduler suffers from sharp performance degradation, while the 640-cycle scheduler reaches its response length minima with a mild drop in performance. In addition, the longer expansion phase of the 640-cycle scheduler ensures sustained and stable accuracy gains. As a whole, the 640-cycle scheduler leads to the largest length oscillation and highest compression ratio at the response length minima. This shows that a smoother compression phase is crucial for performance maintenance, while a longer expansion phase is the key to iterative accuracy improvement. This finding is in line with earlier work (Hou et al., 2025), where the authors argue that iterative length capping preserves performance, while direct length capping leads to

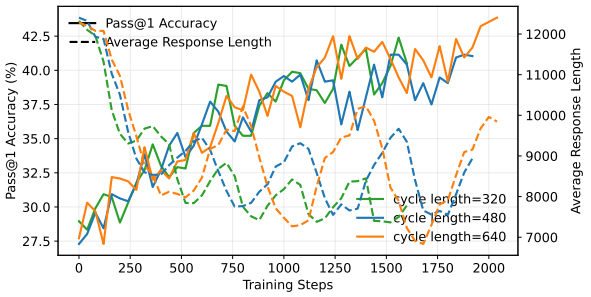


(a) Dynamics on AIME24

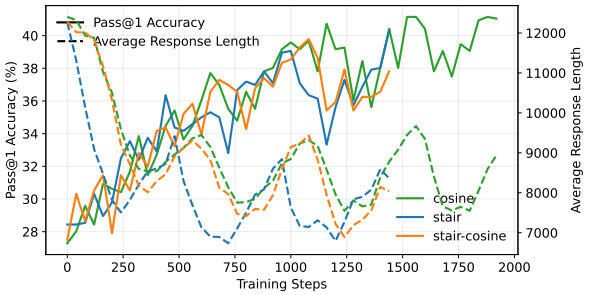


(b) Dynamics on AIME25

Figure 5: Training dynamics comparison between SIRI and DAPO-DeepScaleR-16K on R1-Distill-Qwen-1.5B. DAPO-DeepScaleR-16K transits from 8K to 16K at step 320.



(a) Dynamics of cosine scheduler with different cycle lengths.



(b) Dynamics of different-shaped schedulers with a cycle length of 480.

Figure 6: Ablation studies on scheduler design.

sharp decline in response length, causing serious performance loss.

Scheduler with different shapes has different advantages. We show in Figure 6b the 1.5B model's performance of the stair, cosine, and stair-cosine schedulers, all with a cycle length of 480. We observe that the cosine scheduler mitigates performance loss during compression, while the stair scheduler maximizes performance gain during expansion. Specifically, in Figure 6b, the cosine scheduler maintains the model's accuracy around 0.39 when its response length falls during step 960 to step 1200. However, the performance drops further while the scheduler slowly increases the maximum generation length to 16K. In comparison, the direct 8K compression phase of the stair scheduler causes a sharp drop in the model's performance, but the subsequent full 16K expansion phase significantly boosts the model's response length and accuracy. For the stair scheduler, the extended 8K compression phase also fails to improve performance, while the extended 16K expansion phase brings additional gains. In summary, we find that the compression phase should be smooth, while the expansion phase should be extended, relaxing its constraint on model's exploration behavior.

Rollout-level Analysis. We also provide evidence for SIRI's effectiveness from a rollout-level perspective, with detailed case analysis shown in Appendix C.5. We find that SIRI mainly reduces the model's backtracking and verification behavior, while preserving other reasoning skills, including alternative-seeking, general deduction and computation. This shows that SIRI can adaptively prune unnecessary reasoning processes while keeping track of intermediate steps that are important for the final correctness.

6 Conclusion

This paper introduces SIRI, a simple but effective method that enhances LRMs by pruning repetitive reasoning through alternating token budget expansion and compression. Experiments show consistent gains in performance and token efficiency across iterations. Looking forward, online RL post-training has been an ever-broadening avenue towards artificial general intelligence, and we hope this work can help to further scale up RL training.

7 Limitations

While SIRI has provided extra gains, the upper performance threshold remains to be discovered and understood (e.g., limited by dataset size, algorithm efficiency, etc). Moreover, how SIRI can be applied in other tasks that require intensive reasoning, such as code generation, is also a promising direction.

References

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

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

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

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, and 1 others. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, and 1 others. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

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*.

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. *Preprint*, arXiv:2403.07974.

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.

Chenwei Lou, Zewei Sun, Xinnian Liang, Meng Qu, Wei Shen, Wenqi Wang, Yuntao Li, Qingping Yang, and Shuangzhi Wu. 2025. Adacot: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning. *arXiv preprint arXiv:2505.11896*.

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*.

Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, and 1 others. 2024. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*.

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

OpenAI. 2024. Openai o1 system card. *arXiv preprint*, arXiv preprint arXiv:2412.16720.

Richard Yuanzhe Pang, Weizhe Yuan, He He, Kyunghyun Cho, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *Advances in Neural Information Processing Systems*, 37:116617–116637.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement finetuning. In *Forty-second International Conference on Machine Learning*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

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*.

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

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 preprint*, 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*.

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 preprint arXiv: 2409.19256*.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, and 21 others. 2024. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

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

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.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. 2023. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. *arXiv preprint arXiv:2312.11456*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

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

A Formalizations and proofs

A.1 Proof of the result in Figure 2a

MDP setup. We first detail the MDP setup in Figure 2a. Its initial state is s_1, and there are n loops: $s_1 - s_4 - s_1, s_1 - s_5 - s_1, \cdots, s_1 - s_{n+3} - s_1$. The unique path from the initial state to the goal state s_3 is $s_1 - s_2 - s_3$. We compare the two training strategies under simplified settings. For the iterative schedule, the rollout horizons are $H_{2k-1} = 2$, $H_{2k} = 4$ for odd and even update steps respectively. For the long-context schedule, the rollout horizon is $H_k = 4$ for all update steps. We evaluate both policies under $H = 4$.

Theorem 1. *Let the policy at step $k+1$ be updated by*

$$\pi_{k+1}(\cdot|s) = \arg\max_{\pi'(\cdot|s)} \mathbb{E}_{a\sim\pi'(\cdot|s)}\left[A^{\pi_k}(s,a)\right] - \beta D_{KL}(\pi'(\cdot|s)\|\pi_k(\cdot|s)), 0 < \beta \le 4,$$

Then

$$E_I^t[T_G] \le E_L^t[T_G], \forall t \in \mathbb{N}.$$

Proof. We first note that the solution to the above update step has a well-known closed form:

$$\pi_{k+1}(a|s) = \frac{\pi_k(a|s)\exp(\beta A^{\pi_k}(s,a))}{Z^{\pi_k}(s)},$$

where $Z^{\pi_k}(s)$ is the partition function.

Since the transitions are deterministic, we abuse the notation a little by expressing the probability of the actor taking the transition from x to y as $\pi(s' = y|s = x)$.

Since the policy will reach the goal only after it selects the transition from s_1 to s_2, the expected hitting time follows a geometric distribution. Formally, denote $p = \pi(s' = s_2|s = s_1)$ for any policy π, we have

$$T_G = 2K, K \sim \text{Geom}(p).$$

Therefore $\mathbb{E}[T_G] = \frac{2}{p}$.

We now use induction to show that the iterative schedule yields a better policy compared to long-horizon schedule. Denote p_I^t, p_L^t as the probability of the policy transitioning to s_2 from s_1 after trained by iterative and long-horizon schedule for t steps respectively. In particular, since the two schedules start training from the same policy, we have $p_I^0 = p_L^0$. For convenience, we also set $p_I^{-1} = p_L^{-1}$.

For the induction step, assume $p_I^{2k-1} \ge p_L^{2k-1}$ and $p_I^{2k} \ge p_L^{2k}$ both hold.

For $H = 2$ update step, denote $g(q) = g(\pi(s' = s_2|s = s_1)) = \pi'(s' = s_2|s = s_1)$ as the updated probability after this step, it can be calculated that

$$g(q) = \frac{qe^{\beta}}{qe^{\beta} + 1 - q}.$$

For $H = 4$ update step, similarly define $f(q)$ as the updated probability after this step, then

$$f(q) = \frac{qe^{\beta(1-q)}}{qe^{\beta(1-q)} + 1 - q}.$$

Now we compare $g(q)$ and $f(q)$. For $0 < y \le x \le 1$, direct calculation gives

$$g(x) = \frac{xe^{\beta}}{xe^{\beta} + 1 - x} \ge \frac{ye^{\beta(1-y)}}{ye^{\beta(1-y)} + 1 - y} = f(y).$$

In particular,

$$p_L^{2k+1} = f(p_L^{2k}) \le g(p_I^{2k}) = p_I^{2k+1}.$$

The derivative of $f(q)$ is

$$f'(q) = \frac{1 - \beta q(1-q)}{(q + (1-q)e^{\beta(q-1)})^2}.$$

Since $q(1-q) \le \frac{1}{4}$, we have $f'(q) \ge 0$ when $\beta \le 4$. Therefore $f(q)$ is monotonic on $(0, 1)$. In particular,

$$p_L^{2k+2} = f(p_L^{2k+1}) \le f(p_I^{2k+1}) = p_I^{2k+2}.$$

Which finishes the induction step. Therefore, for any step t during training, we have

$$\mathbb{E}_I^t[T_G] = \frac{2}{p_I^t} \le \frac{2}{p_L^t} = \mathbb{E}_L^t[T_G].$$

□

Remark. Here, the advantage for each step is computed with respect to the schedule-dependent objective, which slightly deviates from its original definition under infinite-horizon setting. However, this does not affect the correctness of our deduction.

A.2 Proof of the result in Figure 2b

MDP setup. For the MDP in Figure 2b, the initial state is s_1, and there are $n-1$ loops: $s_1 - s_6 - s_1, s_1 - s_7 - s_1, \cdots, s_1 - s_{n+5} - s_1$. The unique path from the initial state to the goal state is $s_1 - s_2 - s_3 - s_4 - s_5$. For the iterative schedule, the rollout horizons are $H_{2k-1} = 2$, $H_{2k} = 4$ for odd and even update steps respectively. For the short-context schedule, the rollout horizon is $H_k = 2$ for all update steps. We evaluate both policies under $H = 4$.

Theorem 2. *Let the policy at step $k+1$ be updated by*

$$\pi_{k+1}(\cdot|s) = \arg\max_{\pi'(\cdot|s)} \mathbb{E}_{a\sim\pi'(\cdot|s)}\left[A^{\pi_k}(s,a)\right] - \beta D_{KL}(\pi'(\cdot|s)\|\pi_k(\cdot|s)), \beta > 0,$$

Then

$$E_I^t[R] \geq E_S^t[R], \forall t \in \mathbb{N}.$$

Proof. Since the policy will reach the goal if and only if it transits from s_1 to s_2 at step 1, the expected return is $\mathbb{E}[R] = p$, where $p = \pi(s' = s_2|s = s_1)$.

Similar to the proof of Theorem 1, we use induction to show that iterative schedule yields a better policy compared to short-horizon schedule. Denote p_I^t, p_S^t as the probability of the policy transitioning to s_2 from s_1 after trained by iterative and short-horizon schedule for t steps respectively, and let $p_I^0 = p_S^0, p_I^{-1} = p_S^{-1}$.

For the induction step, assume $p_I^{2k-1} \geq p_S^{2k-1}$ and $p_I^{2k} \geq p_S^{2k}$ both hold.

For $H = 2$ step, denote $g(q) = g(\pi(s' = s_2|s = s_1)) = \pi'(s' = s_2|s = s_1)$ as the updated probability after this step. It's easy to see that $A^\pi(s,a) = 0$ for all actions a and any policy π when $H = 2$, so $g(q) = q$. In particular,

$$p_I^{2k+1} = g(p_I^{2k}) \geq g(p_S^{2k}) = p_S^{2k+1}.$$

For $H = 4$ step, similarly define $f(q)$ as the updated probability after this step, then

$$f(q) = \frac{qe^\beta}{qe^\beta + 1 - q} \geq q.$$

The derivative of $f(q)$ is

$$f'(q) = \frac{e^\beta}{(qe^\beta + 1 - q)^2} > 0.$$

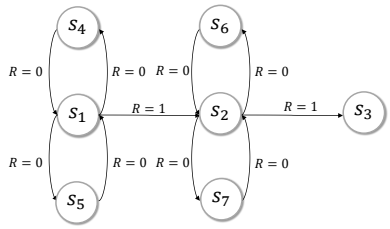


Figure 7: A compositional MDP incorporating loops and long paths. The graph's initial state is s_1 and the goal state is s_3.

Therefore $f(q)$ is monotonic on $(0,1)$. In particular,

$$\begin{aligned} p_I^{2k+2} = f(p_I^{2k+1}) &\geq f(p_S^{2k+1}) \\ &\geq p_S^{2k+1} \\ &= g(p_S^{2k+1}) \\ &= p_S^{2k+2}. \end{aligned}$$

Which finishes the induction step. Therefore, for any step t during training, we have

Therefore

$$\mathbb{E}_I^t[R] = p_I^t \geq p_S^t = \mathbb{E}_S^t[R].$$

□

B Experiments on a representative MDP

In this section, we validate the superiority of iterative schedules over non-iterative ones by running an experiment on the compositional MDP shown in Figure 7.

MDP setup. The MDP incorporates both loops and paths that represent redundant thinking and complicated reasoning respectively. The initial state is s_1 and the goal state is s_3. s_1 falls into two loops: $s_1 - s_4 - s_1$ and $s_1 - s_5 - s_1$; s_2 falls into two loops: $s_2 - s_6 - s_2$ and $s_2 - s_7 - s_2$. In practice, the training dataset often consists of both simple and hard problems, so the agent can gradually learn to assemble atomic subtasks to solve more complicated ones. Therefore in this MDP, the agent will get reward $R = 1$ as it transits from s_1 to s_2, simulating its success of solving a simple task. Another reward is given when the agent transits from s_2 to s_3 that represents the success of solving a complex, long-horizon task that is built on solving a simpler one, i.e., the transition from s_1 to s_2.

Training setup. We adopt the naive GRPO algorithm for training. The policy is updated by

$$\pi_{k+1}(a|s) = \frac{\pi_k(a|s)\exp(\beta A^{\pi_k}(s,a))}{Z^{\pi_k}(s)}.$$

and the advantage estimate is given by

$$A^{\pi_k}(s,a) = \sum_{i=1}^{G} N_{\tau_i}^{(s,a)} \cdot \frac{R(\tau_i) - \text{mean}(\{R(\tau_i)\}_{i=1}^{G})}{\text{std}(\{R(\tau_i)\}_{i=1}^{G})}.$$

where $N_{\tau_i}^{(s,a)}$ is the number of the transitions (s,a) taken in trajectory τ_i.In this experiment, we set $\beta = 1$ and $G = 2$.

During training, we initialize the policy with 100 different random seeds and run the above algorithm 256 steps for each seed.

Scheduler setup. We compare three schedules:

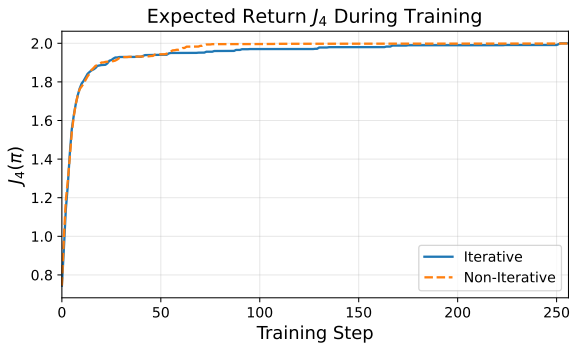
1. Iterative schedule, whose rollout horizon alternates between $H = 2$ and $H = 4$, i.e., $H_{2n-1} = 2, H_{2n} = 4$.

2. Long-context schedule, whose rollout horizon is fixed at $H = 4$.

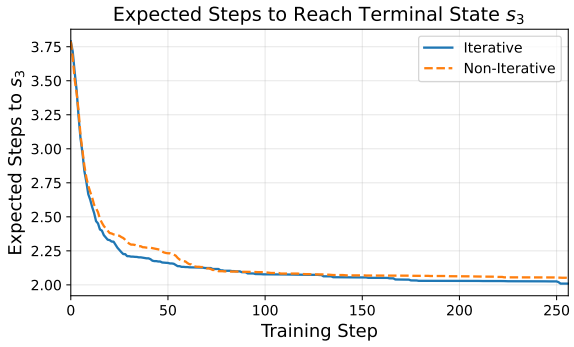
3. Short-context schedule, whose rollout horizon is fixed at $H = 2$.

Evaluation setup. Consistent with the maximum rollout horizon during training, we cap the rollout horizon during evaluation at $H = 4$. 100 trajectories are randomly sampled from the policy after each training step to calculate the expected return of the trajectory and arrival time of the final state s_3. Note that if the policy fails to arrive at s_3 within the horizon, its arrival time is capped at 4, similar to the evaluation standard of LLM's token usage.

Results. Figure 8 shows the training dynamics of the iterative schedule and long-context schedule. Compared to the long-context schedule, the iterative schedule achieves comparable return while reducing the expected arrival time, which is consistent with Theorem A.1. Figure 9 shows the training dynamics of the iterative schedule and the short-context schedule. The iterative schedule improves both the expected return and reduces the expected arrival time, validating Theorem A.2. In summary, these results demonstrate that the iterative schedule effectively reduces arrival time while maintaining the policy's performance well compared to other schedules.



(a) Dynamics of expected return



(b) Dynamics of expected arrival time

Figure 8: The expected return and arrival time comparison between iterative schedule ("Iterative") and long-context schedule ("Non-Iterative").

C Additional experiment results

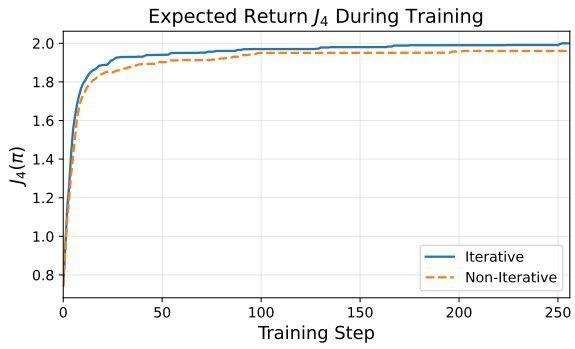
C.1 Additional experiment configurations

Training dataset. To provide a fair comparison with the strong DeepScaleR (Luo et al., 2025b) baseline, we use the same training set used in training DeepScaleR-1.5B-Preview, which comprises 40K high-quality math questions with groundtruth answers selected from AIME (-2023), AMC (-2022), Omni-Math (Gao et al., 2024), and STILL (Min et al., 2024) datasets.

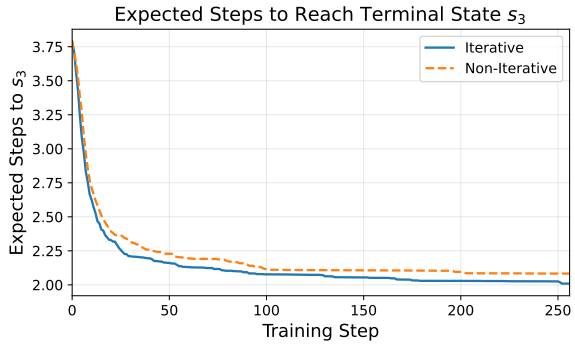
Evaluation. For all benchmarks evaluated in this paper, we set the maximum generation length (including thinking tokens and answer tokens) at 16384, aligned with $L_{\max}$ during training. We sample 32 outputs for each question during training, and sample 64 outputs for each question to obtain the final evaluation results. The sampling temperature is set to 0.6. We report both the Pass@1 accuracy and the average token number of the responses.

C.2 Results on DeepSeek-R1-Distill-Qwen-7B

Figure 10 demonstrates the training dynamics of SIRI and DAPO-DeepScaleR-16K for the

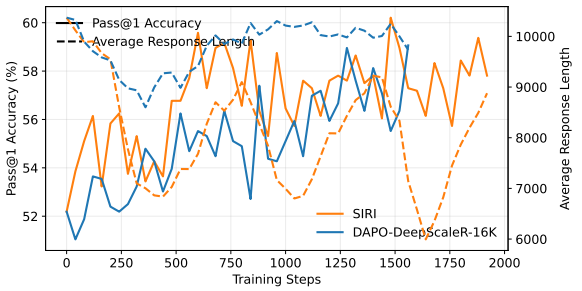


(a) Dynamics of expected return

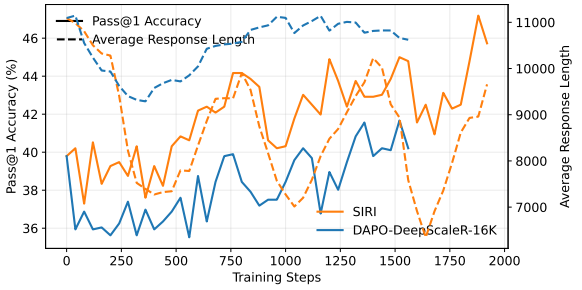


(b) Dynamics of expected arrival time

Figure 9: The expected return and arrival time comparison between iterative schedule ("Iterative") and short-context schedule ("Non-Iterative").



(a) Dynamics on AIME24



(b) Dynamics on AIME25

Figure 10: The training dynamics comparison between SIRI and DAPO-DeepScaleR-16K. DAPO-DeepScaleR-16K transits from 8K to 16K at step 360.

DeepSeek-R1-Distill-Qwen-7B model. SIRI reaches comparable performance to DAPO-DeepScaleR-16K on AIME24 and outperforms DAPO-DeepScaleR-16K on AIME25, both with less tokens.

C.3 Details on training dynamics: an entropy view

We additionally report the change of entropy during the cosine scheduler training in Figure 11. During the compression stage, the model's entropy decreases; During the expansion stage, its entropy slowly increases. However, we find that the model's entropy does not collapse. Instead, it tends to remain stable within a certain range as training proceeds. However, for non-iterative models such as DAPO-DeepScaleR-16K, we do not notice similar trends. As shown in Figure 12, the model's entropy fluctuates during 16K context training of DAPO-DeepScaleR-16K. This shows that the periodic oscillation of entropy may depend on the design of the length scheduler.

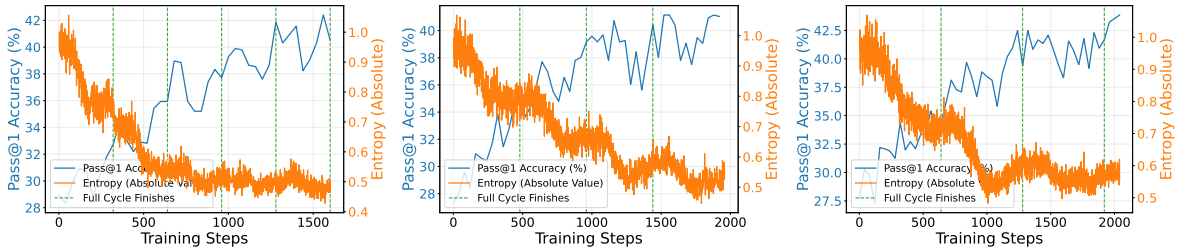
Moreover, we notice an improvement in performance when entropy increases during SIRI's expansion stage. This implies the possibility of using entropy bonus or clipping even higher to further enhance SIRI's performance.

C.4 Results on general benchmarks

We evaluated SIRI's performance on three general benchmarks: GPQA-Diamond (Rein et al., 2024), LiveCodeBench (Jain et al., 2024) and HumanEval (Chen et al., 2021). As shown in Table 2, SIRI-low consistently reduces the model's token usage while improving its accuracy on most benchmarks, while SIRI-high further enhances the model's accuracy but with modest increase in token usage. This shows that SIRI can generalize to out-of-distribution tasks, pruning the model's core overthinking patterns while preserving its base knowledge.

C.5 Rollout Analysis

We first show the model's token-level reasoning behavior before and after compression. Specifically, we choose the 1.5B model's responses for AIME24 problems at step 1280 (the finish of the second expansion stage), step 1600 (the finish of the third compression stage), and step 1920 (the finish of the third expansion stage) during the 640-cycle cosine schedule. We choose tokens that represent the model's backtrack-verification ("wait", "hold on"), alternative-seeking ("alternatively"), and gen-



(a) 320 cycle (b) 480 cycle (c) 640 cycle

Figure 11: The entropy during training for cosine scheduler with different cycle length.

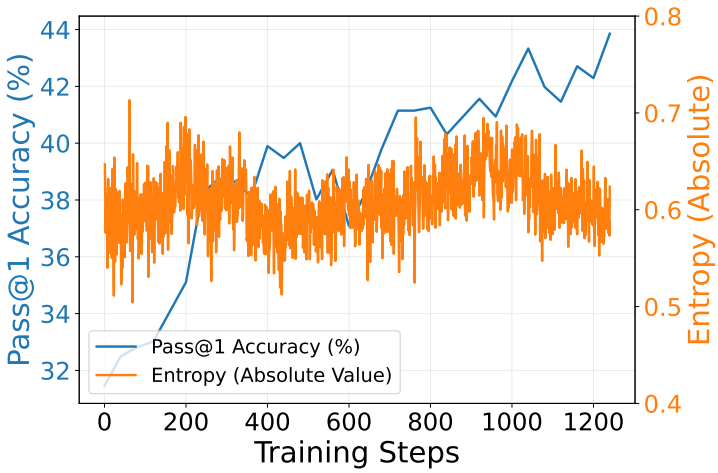


Figure 12: The entropy of DAPO-DeepScaleR-16K during 16K context training.

eral deduction behavior ("so", "compute"). As shown in Figure 13, the frequency of "wait" tokens that stand for backtracking and verification changes significantly during training, while others remain stable. In particular, the "wait" tokens are suppressed during compression and encouraged during expansion, and this trend is consistent for both correct and incorrect responses. Notably, the correct responses from the model at step 1280 and 1920 are almost identical, despite the latter having better performance. This shows that the interleaved compression phase indeed encourages the model to add more information under the same generation context.

Then, we analyze the model's rollout-level shift in reasoning behavior before and after compression. We select a difficult problem in GPQA-Diamond, an out-of-distribution benchmark, such that it captures the essential changes in reasoning. As shown in Figure 15 and 16, compared to DeepSeek-R1-Distill-Qwen-1.5B, SIRI-1.5B-low efficiently pushes reasoning forward, arriving at the first key equation '$\tau = \mu \times \mathbf{B}$' with fewer tokens. Notably, instead of repeating the backtracking behavior as its counterpart, SIRI-1.5B-low is able to find an alternative solution that leads to the answer faster. This demonstrates SIRI's ability to utilize spared tokens for seeking alternative and more efficient solutions, which probably leads to the improvement of the model's reasoning accuracy.

Method	GPQA-Diamond		LiveCodeBench (v6)		HumanEval	
	Acc	Length	Acc	Length	Acc	Length
DeepSeek-R1-Distill-Qwen-1.5B						
Original	33.8	9838	19.2	11836	70.9	3720
SIRI-low (Ours)	34.8	**6249**	22.9	**8933**	76.1	**3520**
SIRI-high (Ours)	**37.1**	8098	**23.4**	11077	**78.5**	4709
DeepSeek-R1-Distill-Qwen-7B						
Original	49.6	8275	33.8	10973	90.0	2728
SIRI-low (Ours)	48.2	**5427**	36.7	**7793**	91.8	**2421**
SIRI-high (Ours)	**49.8**	6863	**37.1**	9390	**92.4**	2923

Table 2: Performance comparison on GPQA-Diamond, LiveCodeBench(v6) and HumanEval. Best result in **bold** and second best underlined.

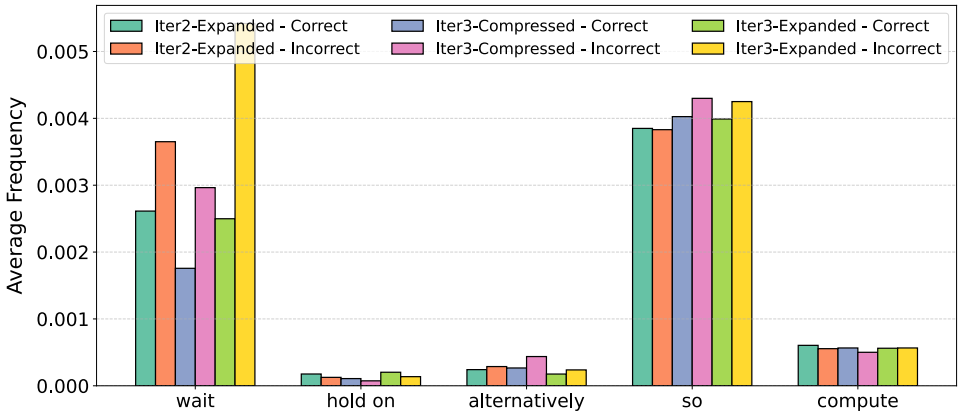


Figure 13: Representative token frequency before and after compression.

Prompt for the Problem

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 A, B, C, D. Think step by step before answering.
There is a spin-$\frac{1}{2}$ particle with magnetic moment $\mu = \gamma S_z$, aligned with a magnetic field pointing in the $+Z$ direction. Suddenly, the magnetic field is turned off, and another one is turned on along the $+Y$ direction; both fields have the same magnitude B. The average of the z-component of the particle's moment starts oscillating. What is its oscillation frequency?

A) $\sqrt{2}\gamma B$
B) $\frac{\gamma B}{\sqrt{2}}$
C) γB
D) $\frac{\gamma B}{2}$

Figure 14: Formulation of the problem in GPQA-Diamond.

Response Generated by Deepseek-R1-Distill-Qwen-1.5B

<think>
Okay, so I'm trying to figure out this physics problem. Let me read through it again.
So there's a spin-$\frac{1}{2}$ particle. Its magnetic moment is $\mu = \gamma S_z$. **Oh wait**, right, the magnetic moment is proportional to the spin component in the z-direction. So initially, the particle is aligned with a magnetic field pointing in the $+Z$ direction.
Then, the magnetic field is turned off, and another one is turned on along the $+Y$ direction. Both have the same magnitude B. So the magnetic field now has components B_x along Y and B_y along Z, but both equal. **Wait, no, wait**: the problem says both fields have the same magnitude B. So the initial field was B in $+Z$, then after switching, it's B in $+Y$. So the total magnetic field now is B in $+Y$ and B in $+Z$. So the vector is $(0, B, B)$ in some coordinate system.
The average of the z-component of the particle's moment starts oscillating. Hmm. I think that when you have a time-dependent magnetic field, the particle's magnetic moment will experience a torque, leading to precession. Since the magnetic moment is a vector, its direction will change over time due to this torque. The average of its z-component oscillates, which suggests some kind of oscillation in the z-component.
Wait, but the initial state before the switch is a spin aligned with the $+Z$ field. So the spin is along $+Z$, so S_z is maximum. After the field is turned on, the magnetic field is in $+Y$ and $+Z$ directions. So the new $\mathbf{B}$ vector is $(0, B, B)$. So the direction of $\mathbf{B}$ is along $+Y$ and $+Z$. So the direction of $\mathbf{B}$ is 45° from the Z-axis, pointing into the positive Y and Z directions.
The particle's magnetic moment is $\mu = \gamma S_z$. **Wait, no**: the magnetic moment is $\mu = \gamma S_z$. **Oh wait, no, wait**. Spin is represented by the operator S_z, so μ is γS_z. So the magnetic moment vector is in the z-direction, scaled by the spin component.
Wait, I think I'm getting this wrong. The magnetic moment is given by $\mu = \gamma S_z$, right. So the direction of μ is the same as S_z, but magnitude is γ times the spin.
Wait, when the magnetic field is applied, the particle will experience a torque. The torque on the particle is

$$\tau = -\boldsymbol{\mu} \times \mathbf{B}.$$

Because the magnetic moment's torque is opposite the direction of the magnetic field. So the torque is the cross product of the magnetic moment and the magnetic field.
But wait, the initial situation: before the field is turned on, the magnetic field is along $+Z$, so $\mathbf{B}$ is along Z. So the torque would be $\boldsymbol{\mu} \times \mathbf{B} \ldots$
Hmm, I'm getting stuck. I think I should select option B:

$$\frac{\gamma B}{\sqrt{2}}.$$

But I'm not confident. Alternatively, perhaps the correct answer is option B.
</think> (16384 tokens)

Figure 15: Response Generated by Deepseek-R1-Distill-Qwen-1.5B on the problem in GPQA-Diamond.

Response Generated by SIRI-low-1.5B

<think>
Okay, so I've got this physics problem here about a spin-$\frac{1}{2}$ particle and its magnetic moment. Hmm, let me try to think through this step by step. I'm a bit rusty on some of these concepts, but I'll try to recall.
The problem says the particle has a magnetic moment μ equal to γ times S_z. **Oh wait**, right, the magnetic moment for a spin-$\frac{1}{2}$ particle is typically something like $\mu = \gamma S_z$, where γ is the gyromagnetic ratio. So μ is the magnetic moment vector.
Initially, the magnetic field is pointing in the $+Z$ direction. Then, suddenly, the field is turned off, and another one is turned on along the $+Y$ direction. Both fields have the same magnitude B. So the magnetic field $\mathbf{B}$ is applied in the $+Y$ direction after being turned off.
The question is about the average of the z-component of the particle's magnetic moment oscillating. What's the frequency of this oscillation?
I remember that when a magnetic field is applied to a magnetic moment, it experiences a torque, causing the particle to precess. The precession frequency is related to the magnetic field and the gyromagnetic ratio.
Wait, the magnetic moment μ is given by $\mu = \gamma S_z$. The initial magnetic field is in the $+Z$ direction, so the particle would have a certain precession. Then, after the field is turned off, a new field is applied in the $+Y$ direction. So maybe the particle starts precessing around the new field direction.
Wait, but wait. Let me think about the system. Initially, the magnetic field is in $+Z$, so the magnetic moment μ is aligned with $\mathbf{B}$, so the torque would cause precession about the $+Z$ axis. But then, the field is turned off. So the particle's spin precesses around $+Z$ for some time. Then, the field is turned on in the $+Y$ direction. So the particle is subject to a new magnetic field in $+Y$.
Wait, but after the field is turned off, the magnetic moment is $\mu = \gamma S_z$. But when the field is turned on in $+Y$, the new magnetic field is B in $+Y$. So the torque would cause the particle's spin to precess around the new $\mathbf{B}$ field direction.
Hmm. So after the field is turned on, the magnetic moment μ is still given by $\mu = \gamma S_z$. But the field is B in $+Y$. So the torque τ is

$$\tau = \boldsymbol{\mu} \times \mathbf{B}.$$

Wait, but μ is γS_z. So

$$\tau = (\gamma S_z) \times \mathbf{B} \cdot I,$$

assuming $I = 1$ for spin-$\frac{1}{2}$. The torque would cause the spin to precess about the new $\mathbf{B}$ direction.
Wait, maybe it's easier to model this using the precession frequency. The precession frequency is given by the magnitude of the magnetic field times γ, but perhaps in a certain direction.
Alternatively, think about the system in the rotating frame. After the field is switched on, the particle's spin is subject to a magnetic field in $+Y$. So the precession frequency would be

$$\omega = \gamma B.$$

So if the field is B in the Y direction, the precession frequency is γB.
So the average of μ_z, which is γS_z, would oscillate with frequency γB.
Looking at the options, option C is γB. So the answer would be C.
Wait, but let me think again. The initial state is along $+z$, so the average μ_z is γ, since S_z has eigenvalues $\pm\hbar/2$, so the expectation value is $\gamma(\hbar/2)$ for $+z$, etc. But when the field is turned on in Y, the system starts precessing, so the expectation value of μ_z would oscillate.
The time evolution of μ_z would involve a cosine or sine function with frequency ω.
So the average μ_z would oscillate with frequency

$$\omega = \gamma B.$$

So the correct answer is C) γB
</think> (2502 tokens)

Figure 16: Response Generated by SIRI-low-1.5B on the problem in GPQA-Diamond.