DO NOT GO GENTLE INTO REPETITION: ESCAPING OVERTHINKING WITH REPETITION-AWARE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent reasoning models achieve strong performance through reinforcement learning, but often generate excessively long traces containing redundant verification and repetitive loops. Existing efficiency-oriented methods mainly rely on length rewards or position-agnostic truncation, which may interrupt productive reasoning and can even assign negative advantages to correct trajectories. We propose **REFRAIN**, **re**petition-aware rein**f**o**r**cement le**a**rn**in**g for efficient reasoning. REFRAIN uses repetition patterns to identify unproductive reasoning and construct shorter training rollouts. It further introduces correctness-preserving optimization to avoid distorting the learning signal of successful trajectories. Experiments across mathematical reasoning benchmarks and model scales show that REFRAIN improves both training and inference efficiency while maintaining or improving reasoning accuracy.

1 INTRODUCTION

Large reasoning models have established long-form chain-of-thought generation as an effective paradigm for solving complex problems. GRPO was introduced in DeepSeek-Math for outcome-supervised reasoning optimization (Shao et al., 2024) and gained broad attention through DeepSeek-R1, which showed that large-scale reinforcement learning can elicit advanced reasoning behaviors without annotated reasoning traces (Guo et al., 2025). This success further motivated improved methods such as DAPO (Yu et al., 2025). However, long-form reasoning also incurs substantial cost: models may spend thousands of tokens repeatedly verifying intermediate results, reformulating conclusions, or exploring alternatives after sufficient evidence has already been obtained, increasing both training rollout cost and inference latency.

Longer reasoning is not necessarily better reasoning. Additional computation is useful when it discovers a missing derivation or corrects an earlier mistake, but becomes wasteful once the model stops making meaningful progress. Long trajectories often contain redundant verification, repeated reformulations, and self-reinforcing loops, and may even drift away from an initially valid solution (Zhang et al., 2026). Outcome-reward reinforcement learning discourages such behavior only indirectly when an overlong trajectory fails to produce a verifiable answer, providing little guidance on how to recover after entering a repetitive state.

Existing approaches improve reasoning efficiency from several perspectives. Length-aware methods such as ShorterBetter (Yi et al., 2025) and SmartThinker (Hu et al., 2026) encourage shorter correct responses, while budget forcing (Muennighoff et al., 2025) explicitly controls test-time computation. Chain of Draft compresses intermediate textual reasoning (Xu et al., 2025), whereas latent reasoning and System-2 distillation reduce the need to verbalize intermediate computation (Hao et al., 2025; Yu et al., 2024). Speculative decoding instead lowers the computational cost of each generated token (Leviathan et al., 2023; Chen et al., 2023). These directions are complementary, but response length itself remains a coarse proxy for progress: a long derivation may remain productive, whereas a shorter trajectory may already be trapped in an unproductive loop.

Another line of work reduces training cost by truncating completed trajectories and regenerating from retained prefixes. JET uses predefined truncation ratios (Han et al., 2026), whereas S-GRPO samples temporal positions and inserts an early-exit instruction (Dai et al., 2025). However, their

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

intervention points are largely independent of reasoning structure and may occur before an essential derivation, inside a coherent argument, or only after substantial redundant computation. The question of *where* to intervene therefore remains underexplored.

Efficiency-oriented rewards also interact nontrivially with group-relative advantage estimation. When correctness and length signals are combined and centered within a rollout group, a correct but relatively long response may receive a negative advantage. This is especially problematic in all-correct groups, where favoring one short trajectory necessarily suppresses other correct trajectories. Conversely, uncentered efficiency bonuses may introduce trajectory-correlated optimization bias. An effective method must therefore identify genuinely unproductive computation while preserving correctness as the primary objective.

We propose REFRAIN, short for **re**petition-aware rein**f**o**r**cement le**a**rn**in**g, a method for efficient reasoning. As its name suggests, REFRAIN trains reasoning models to refrain from unproductive repetition while preserving useful deliberation. REFRAIN first identifies a recurring reasoning cycle and branches immediately before the cycle resumes. It then inserts a lightweight transition cue and samples a short recovery continuation. The original and recovery trajectories are evaluated with a correctness-preserving efficiency advantage, after which their feedback is propagated through a shared-prefix trie. In this way, REFRAIN learns whether continuing beyond a repetition boundary is useful, rather than treating response length itself as evidence of inefficiency.

2 Background

2.1 Reinforcement Learning with Verifiable Rewards

Given a prompt x and a response y, reinforcement learning with verifiable rewards (RLVR) optimizes a policy π_θ using an outcome reward $R(x, y)$ provided by a rule-based verifier:

$$\mathcal{J}_{\mathrm{RLVR}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x)}\left[R(x, y)\right]. \tag{1}$$

Group Relative Policy Optimization (GRPO), introduced in DeepSeek-Math (Shao et al., 2024), avoids training a separate critic by estimating the advantage from a group of responses sampled for the same prompt. DeepSeek-R1 later demonstrated that GRPO-style optimization with verifiable accuracy and format rewards can elicit strong reasoning behaviors in mathematics and code (Guo et al., 2025). For a group of G responses with rewards $R_{i=1}^G$, GRPO computes

$$\hat{A}_i = \frac{R_i - \mathrm{mean}(R_{i=1}^G)}{\mathrm{std}(R_{i=1}^G) + \epsilon}. \tag{2}$$

Following DAPO (Yu et al., 2025), we adopt token-level loss aggregation on original GRPO objective. We first define the clipped importance-sampling ratio as $\bar{\rho}_{i,t} = \mathrm{clip}\left(\rho_{i,t}, 1-\epsilon, 1+\epsilon\right)$, where $\rho_{i,t} = \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\mathrm{old}}}(y_{i,t}|x, y_{i,<t})}$. The policy objective is then

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{\sum_{i=1}^G T_i}\sum_{i=1}^{G}\sum_{t=1}^{T_i}\Big(\min(\rho_{i,t}\hat{A}_i, \bar{\rho}_{i,t}\hat{A}_i) - \beta D_{\mathrm{KL}}(\pi_\theta\|\pi_{\mathrm{ref}})_{i,t}\Big)\right], \tag{3}$$

where G is the number of responses sampled for each prompt, T_i is the number of valid tokens in response i, and $\hat{A}_i$ is the corresponding group-relative advantage.

2.2 Training and Inference Efficiency

RLVR directly optimizes answer correctness but does not explicitly regulate reasoning efficiency. Consequently, reasoning models may spend excessive computation on redundant verification or become trapped in repetitive loops.

At inference time, DEER (Yang et al., 2026) treats discourse-transition points as potential exits, induces trial answers, and terminates reasoning when their confidence exceeds a threshold. As a training-free method, however, it does not internalize early-exit behavior and requires additional answer probing during decoding. Training-based approaches instead encourage concise generation through length-aware rewards. ShorterBetter (Yi et al., 2025) uses the shortest correct response

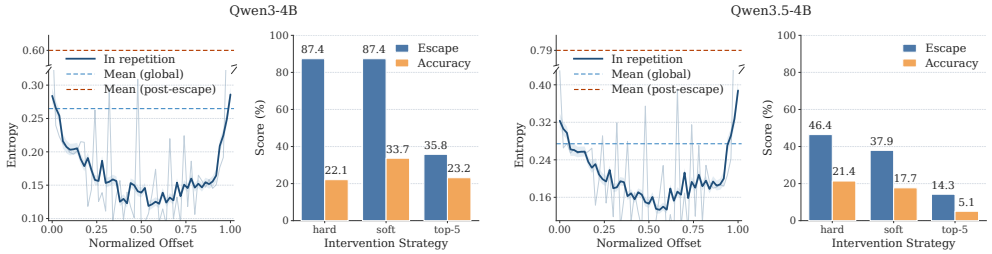


Figure 1: Entropy dynamics within escaping repetitions and the effectiveness of different interventions on Qwen3-4B and Qwen3.5-4B.

within a sampled group as a length target, while SmartThinker (Hu et al., 2026) progressively calibrates the target reasoning length and its reward strength.

Other methods construct early-exit trajectories from complete rollouts. Just-Enough Thinking (JET) (Han et al., 2026) truncates trajectories at predefined progressive positions and combines them with a quality-controlled length reward. S-GRPO (Dai et al., 2025) samples temporal positions uniformly from a single complete trajectory and assigns exponential decaying rewards to correct responses obtained from progressively later exits. Although effective, these methods select intervention points primarily according to temporal position or response length rather than whether reasoning has actually ceased to make progress. Moreover, directly combining correctness with relative length rewards may assign negative advantages to correct but longer trajectories, reversing their intended update direction. These limitations motivate a repetition-aware intervention signal and a correctness-preserving advantage design.

3 METHOD

REFRAIN is orthogonal to the underlying group-based policy optimization algorithm, as it operates on rollout construction, token-level credit assignment, and auxiliary advantage design. In this work, we instantiate REFRAIN with GRPO as the base policy optimization algorithm.

As shown in Fig. 2, REFRAIN consists of three components. First, it identifies repetitive reasoning states using the longest repeated substring and constructs targeted early-exit rollouts. Second, it distinguishes policy-generated tokens from externally inserted tokens and distributes credit over a unified trie containing both full and truncated trajectories. Third, it introduces a correctness-preserving efficiency advantage that encourages concise successful reasoning without reversing the update direction of correct trajectories.

3.1 LOCATING AND ESCAPING REPETITION IN REASONING

Repetition structure. Let $s_{1:n}$ denote a tokenized reasoning trajectory. For each prefix $s_{1:i}$, we define its *longest matched suffix* (LMS) as

$$\mathrm{LMS}_i = \max\left(1, \max_{\ell \geq 1,\ j+\ell-1<i-\ell+1} \{\ell : s_{j:j+\ell-1} = s_{i-\ell+1:i}\}\right), \tag{4}$$

where $\mathrm{LMS}_i = 1$ denotes the absence of a non-trivial repeated suffix under our convention. The *longest repeated substring* (LRS) of the complete trajectory is the longest LMS over all prefixes:

$$i^\star \in \arg\max_i \mathrm{LMS}_i,\ \mathrm{LRS}(s_{1:n}) = s_{i^\star-\mathrm{LMS}_{i^\star}+1:i^\star}. \tag{5}$$

This equivalence allows repetition to be characterized globally through the LRS or locally through the evolution of LMS_i. Ties are broken by selecting the latest occurrence.

We distinguish two outcomes after a trajectory enters a repeated region. An *escaping repetition* has an escape boundary i satisfying $\mathrm{LMS}_i > 1$ and $\mathrm{LMS}_{i+1} = 1$, where the token at position $i+1$ breaks the preceding repeated suffix $s_{i-\mathrm{LMS}_i+1:i}$. A *non-escaping repetition*, in contrast, does not reach such a boundary before generation terminates or reaches its length limit.

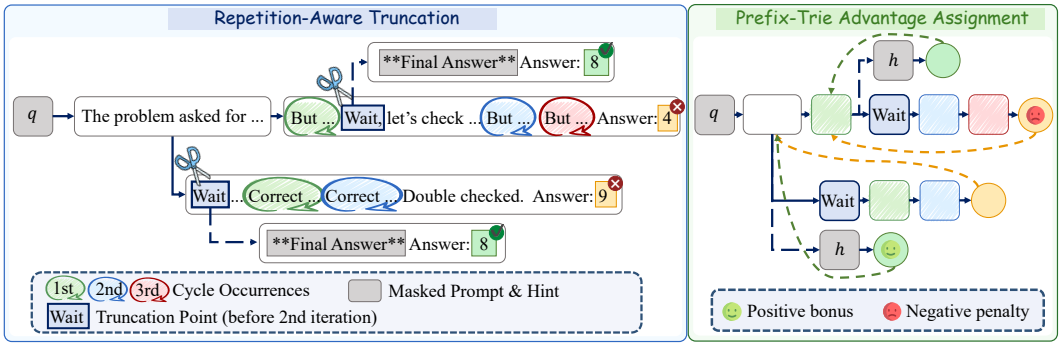


Figure 2: Overview of REFRAIN. **Left:** Repetition-aware truncation identifies repeated cycles and branches immediately before the second occurrence, where a transition hint guides a short recovery continuation. **Right**: Full and recovery trajectories are organized in a shared prefix trie, which aggregates potentially conflicting downstream feedback before assigning credit to identical shared-prefix actions.

Escape analysis protocol. To study the dynamics of repetition, we sample four responses for every AIME25 problem at each temperature $T \in \{0.1, 0.4, 0.7, 1.0\}$. For escaping repetitions, we align each repeated span from its onset to its escape boundary, normalize token positions within the span, and aggregate token-level entropy across trajectories. For non-escaping repetitions, we apply each intervention at the same detected repetition boundary and continue decoding progressively in chunks of $2{,}048$ tokens. After each chunk, we rerun repetition detection on the generated continuation. If its suffix ends with two consecutive occurrences of the same repeating unit, we classify the trajectory as non-escaping and stop generation. Otherwise, decoding continues until normal termination or a maximum total length of $32{,}768$ tokens. Trajectories that reach the maximum length without normal termination are also treated as non-escaping. Fig. 1 aggregates results across all four sampling temperatures.

Entropy dynamics and intervention. As shown in Fig. 1, naturally escaping repetitions exhibit a pronounced high–low–high entropy profile. Entropy is relatively high near the beginning of the matched suffix, decreases rapidly after the trajectory enters the repeated region, and remains low over most of the repeated span. It then rises again near the escape boundary, with the escape token at position $i + 1$ reaching an entropy level comparable to that near the repetition onset.

This pattern suggests that repetition is sustained by a prolonged low-entropy decoding regime in which tokens consistent with the current cycle dominate the sampling distribution. Natural escape becomes possible when uncertainty increases sufficiently for a token outside the repeating pattern to be sampled. The model may therefore spend substantial computation remaining in the cycle before such an escape occurs. Non-escaping repetitions represent cases in which generation does not leave this repetitive regime before termination or the length limit.

We next examine whether local resampling is sufficient to break non-escaping repetitions. At the same repetition boundary, we compare three interventions: hard closing by inserting "`\n</think>\n\n`", soft answer prompting by inserting "`\n\n**Final Answer**\n`", and top-5 branching from alternative next-token candidates. Top-5 branching produces five continuations, one from each candidate token. For each insertion-based intervention, we likewise sample five independent continuations, so that escape rate and final-answer accuracy are computed with the same denominator across all intervention strategies. Across both Qwen3-4B and Qwen3.5-4B, insertion-based interventions achieve substantially higher escape rates than top-5 branching. Soft answer prompting also improves final-answer accuracy on both models, whereas hard closing exhibits model-dependent accuracy effects. These results suggest that an explicit transition cue can redirect generation away from repetitive decoding regimes, with soft answer prompting providing a favorable balance between escape effectiveness and answer accuracy. We therefore use soft answer prompting as the default transition hint in REFRAIN.

Cycle localization and truncation. Operationally, we first locate the LRS using a suffix automaton and apply the Knuth–Morris–Pratt (KMP) algorithm to obtain a candidate repeating unit. If a

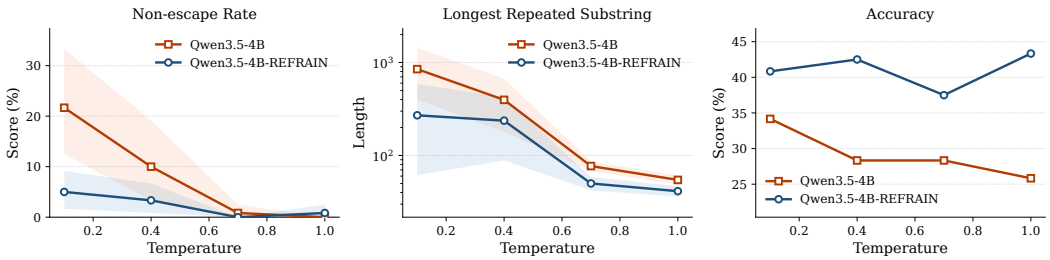


Figure 3: Repetition behavior and accuracy of Qwen3.5-4B and REFRAIN on AIME25 under different sampling temperatures. Shaded regions denote 95% confidence intervals estimated by problem-level bootstrap.

second non-overlapping occurrence is found, its beginning defines the raw truncation boundary; otherwise, the boundary falls back to the end of the reasoning span. We then align the raw boundary to a semantic boundary, searching backward first and forward only when necessary, before appending a lightweight transition hint.

Two-stage truncated rollout. For each prompt x, we first sample G_{F} complete trajectories $\mathcal{Y}^{\mathrm{F}} = \{y_i^{\mathrm{F}}\}_{i=1}^{G_{\mathrm{F}}}$. For each selected trajectory, we retain the tokens before its truncation boundary c_i, append a transition hint h, and sample a new continuation:

$$y_i^{\mathrm{T}} = y_{i,<c_i}^{\mathrm{F}} \oplus h \oplus \widetilde{y}_i, \qquad \widetilde{y}_i \sim \pi_\theta\left(\cdot \mid x, y_{i,<c_i}^{\mathrm{F}}, h\right). \tag{6}$$

The transition hint encourages the model to leave the current reasoning pattern and proceed toward a final answer. The final rollout group contains both complete and recovery trajectories:

$$\mathcal{G}(x) = \mathcal{Y}^{\mathrm{F}} \cup \mathcal{Y}^{\mathrm{T}}, \qquad G_{\mathrm{F}} + G_{\mathrm{T}} = G. \tag{7}$$

Compared with generating G complete trajectories, this procedure requires only G_{F} full responses and G_{T} short recovery continuations, thereby reducing rollout cost. Our truncation can be viewed as a form of *speculative exit*. Whereas speculative decoding proposes candidate tokens and verifies whether they can be accepted, REFRAIN proposes an early transition at a repetition boundary and uses the outcome of the recovery branch to determine whether the remaining computation is useful. The repetition-guided proposal is more targeted than position-based or random truncation, while both successful and unsuccessful recovery branches provide learning signals for subsequent policy optimization.

3.2 Correctness-Preserving Efficiency Advantage

A common approach to efficient reasoning is to add a length bonus or penalty directly to the outcome reward and then normalize the combined rewards within each rollout group. However, this can assign a negative advantage to a correct trajectory merely because another correct trajectory is shorter. The resulting update suppresses a successful reasoning path and conflicts with the primary correctness objective. We instead compute the correctness advantage first and add a separate auxiliary advantage whose sign agrees with correctness. Let R_i be the verifier reward of trajectory i. The base GRPO advantage $\hat{A}_i^{\mathrm{base}}$ is defined as Eq. 2.

Let $\mathcal{C} = \{i : R_i > 0\}$, $\mathcal{W} = \{i : R_i \le 0\}$ denote the correct and incorrect subsets. Auxiliary efficiency optimization is applied only to mixed groups satisfying $|\mathcal{C}| > 0$ and $|\mathcal{W}| > 0$. For all-correct or all-incorrect groups, we set the auxiliary advantage to zero. Let L_i denote the reasoning length of trajectory i, excluding the externally inserted hint. We normalize length within the subsets as $\widetilde{L}_i = \frac{L_i - L_{\min}}{L_{\max} - L_{\min} + \epsilon}$. The class-balanced linear efficiency advantage is defined as

$$\hat{A}_{i,\mathrm{linear}}^{\mathrm{eff}} = \begin{cases} 1 - \widetilde{L}_i, & i \in \mathcal{C}, \\ -\widetilde{L}_i, & i \in \mathcal{W}. \end{cases} \tag{8}$$

Thus, shorter correct trajectories receive larger positive bonuses, while longer incorrect trajectories receive larger negative penalties. The final trajectory advantage is

$$\hat{A}_i^{\mathrm{final}} = \hat{A}_i^{\mathrm{base}} + \alpha \hat{A}_i^{\mathrm{eff}}, \tag{9}$$

where α controls the strength of efficiency optimization. Crucially, we do not renormalize the advantages after adding the auxiliary term. For a mixed group with binary rewards, $\hat{A}_i^{\text{base}} > 0$ for correct trajectories and $\hat{A}_i^{\text{base}} \leq 0$ for incorrect trajectories. Since $\hat{A}_i^{\text{eff}} \geq 0$ for correct trajectories and $\hat{A}_i^{\text{eff}} \leq 0$ for incorrect trajectories, Eq. 9 preserves their intended update directions.

3.3 Branch-Aware Credit Assignment

A truncated trajectory contains three token regions with distinct causal roles: the *shared prefix*, which was sampled by the policy during the original rollout; the *inserted hint*, which is supplied externally; the *continuation*, which is newly sampled after intervention. Applying the same trajectory-level advantage to all three regions ignores these distinctions.

Masking externally inserted tokens. The inserted hint is not an action sampled from π_θ and can have low probability under the original generation distribution. Token-level analyses of RLVR show that entropy change depends jointly on the token probability and its advantage: penalizing a low-probability token further suppresses that token and contributes to entropy decrease (Hao et al., 2026). This issue is particularly pronounced in our setting because the same static hint is repeatedly inserted across many trajectories. We therefore treat the hint solely as conditioning context. Let $\mathcal{H}_i$ denote the hint-token positions in trajectory i. The policy-loss mask is

$$m_{i,t} = \mathbb{I}[t \notin \mathcal{H}_i], \tag{10}$$

such that externally inserted tokens receive neither a policy-gradient update nor a KL-loss contribution.

Unified prefix trie. The shared prefix differs from the hint because it was originally sampled by the policy. Completely masking it discards feedback from the downstream recovery outcome, while directly assigning branch-specific trajectory advantages can give conflicting credit to identical shared-prefix actions. Inspired by the process-set interpretation of shared prefixes in λ-GRPO (Sullivan & Koller, 2026), we organize *all* full and truncated trajectories for the same prompt into a unified prefix trie. Thus, an original trajectory and each of its derived trajectories share the same nodes before the truncation boundary and diverge only after the intervention. Let $\mathcal{D}(v)$ denote the set of full and truncated trajectories whose paths traverse trie node v. Given the final trajectory-level advantage $\hat{A}_i^{\text{final}}$, we assign node v the trie advantage

$$\hat{A}_v^{\text{trie}} = \frac{1}{|\mathcal{D}(v)|} \sum_{i \in \mathcal{D}(v)} \hat{A}_i^{\text{final}}. \tag{11}$$

Eq. 11 has several useful properties. First, all identical prefix tokens receive a consistent update derived from every downstream full and recovery branch. Second, neither a single successful recovery nor a single unsuccessful branch can independently determine the update of a heavily shared prefix. Third, a node unique to one trajectory satisfies $|\mathcal{D}(v)| = 1$, so its trie advantage naturally reduces to the original trajectory advantage. The same formulation therefore covers shared prefixes and branch-specific continuations without separate heuristics. The sign-preserving property applies to the trajectory-level advantage before branch aggregation; shared-prefix trie nodes can aggregate downstream advantages of different signs.

Using the clipped ratio $\bar{\rho}_{i,t}$ defined in Section 2, the resulting policy objective is

$$\mathcal{J}_{\text{REFRAIN}}(\theta) = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{G}\sum_{t=1}^{T_i} m_{i,t}\left(\min\left(\rho_{i,t}\hat{A}_{v(i,t)}^{\text{trie}}, \bar{\rho}_{i,t}\hat{A}_{v(i,t)}^{\text{trie}}\right) - \beta D_{\text{KL}}\left(\pi_\theta \| \pi_{\text{ref}}\right)_{i,t}\right)\right]. \tag{12}$$

where $N = \sum_{i=1}^{G}\sum_{t=1}^{T_i} m_{i,t}$ is the total number of trainable token occurrences.

Consequently, both correctness and efficiency feedback are jointly averaged over every full and truncated branch sharing a prefix. This connects the three components of REFRAIN: LRS determines where to intervene, the correctness-preserving auxiliary advantage determines how efficiency is optimized without unlearning successful reasoning, and branch-aware trie credit determines how downstream signals are aggregated for shared generated tokens.

Table 1: Comparison with baselines on mathematical reasoning benchmarks. For each problem, we independently sample four responses and average correctness over the resulting generations (mean@4); token counts are averaged over the same responses. The best accuracy in each model block is shown in bold.

Method	HMMT		AIME25		AIME26		AMC23		Average	
	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓
DeepSeek-R1-Distill-Qwen-1.5B										
Vanilla	12.12	13,354	23.33	11,307	18.33	11,913	65.63	6,879	29.85	10,863
ShorterBetter	0.75	2,522	19.17	3,440	16.67	3,743	63.13	1,729	24.93	2,859
LC-R1	12.12	8,179	22.50	6,580	16.67	8,296	**70.63**	3,372	30.48	6,607
SmartThinker	**15.15**	8,939	23.33	7,457	17.50	7,781	69.38	3,607	31.34	6,946
REFRAIN	14.39	11,906	**24.17**	8,661	**20.83**	8,652	**70.63**	4,194	**32.51**	8,353
Qwen3-1.7B										
Vanilla	20.45	13,939	33.33	12,977	31.67	13,006	76.88	7,988	40.58	11,978
ShorterBetter	23.48	11,692	27.50	9,074	30.00	8,346	77.50	4,256	39.62	8,342
SmartThinker	22.73	11,933	28.33	9,229	27.50	9,867	73.75	4,805	38.08	8,959
JET	21.21	13,299	**35.83**	10,581	28.33	10,632	73.75	6,075	39.78	10,147
GRPO	**25.76**	12,872	31.67	10,090	**33.33**	9,656	77.50	4,706	42.07	9,331
REFRAIN	24.24	12,269	33.33	9,567	31.67	9,090	**80.63**	4,451	**42.47**	8,844
Qwen3.5-4B										
Vanilla	12.89	15,777	30.83	14,702	30.83	15,182	84.38	9,799	39.73	13,865
ShorterBetter	19.70	8,384	18.33	1,080	11.67	969	61.88	1,031	27.90	2,866
SmartThinker	29.55	10,402	30.00	2,480	30.00	2,539	82.50	2,284	43.01	4,426
GRPO	32.58	13,751	38.33	6,899	50.00	7,207	**92.50**	3,633	53.35	7,873
REFRAIN	**39.39**	13,038	**45.83**	5,902	**52.50**	5,758	**92.50**	2,623	**57.56**	6,830

4 EXPERIMENT

4.1 EXPERIMENTAL SETUP

Datasets and models. We adopt DAPO-MATH-17K (Yu et al., 2025) as the training dataset and use GSM8K (Cobbe et al., 2021) and AIME24 (Zhang & Math-AI, 2024) as validation sets for model selection. We evaluate on five reasoning benchmarks: four mathematical benchmarks, including AIME25, AIME26 (Zhang & Math-AI, 2025; 2026), AMC23 (math-ai, 2025), and HMMT-Feb-2026 (HMMT; Dekoninck et al., 2026), as well as GPQA-Diamond (Rein et al., 2024) for scientific reasoning. The four mathematical benchmarks cover different levels of contest difficulty, while GPQA-Diamond provides an out-of-domain evaluation on scientific reasoning. For each problem, we perform four independent generation runs with different random seeds and report metrics averaged over all four runs. We conduct experiments on three reasoning models across different scales and architectures: DeepSeek-R1-Distill-Qwen-1.5B (Distill-1.5B), Qwen3-1.7B, and Qwen3.5-4B.

Baselines. We compare REFRAIN with publicly released checkpoints from ShorterBetter (Yi et al., 2025), LC-R1 (Cheng et al., 2026), and SmartThinker (Hu et al., 2026). GRPO (Shao et al., 2024) serves as the standard outcome-reward RL baseline. ShorterBetter uses the shortest correct rollout as a sample-based target length, whereas SmartThinker estimates a distribution-aware optimal length and dynamically controls the length-reward coefficient to prevent correct trajectories from receiving negative advantages. LC-R1 extracts the effective reasoning prefix and combines a global length reward with a token-level compression reward to remove reasoning after the answer has been derived. JET (Han et al., 2026) is the closest truncation-based baseline: it constructs short rollouts from progressive truncation positions and applies an accuracy-conditioned length reward, but does not explicitly constrain the signs of normalized advantages.

For model configurations without compatible released checkpoints, we train GRPO, ShorterBetter, and SmartThinker under the same training setup as REFRAIN, using the official implementations for ShorterBetter and SmartThinker. For JET, we use its official implementation and original training configuration.

Implementation details. We use a training batch size of 32 and maximum response lengths of 8,192 and 16,384 tokens for training and inference, respectively. For each prompt, we sample $G = 8$ rollouts, evenly split into $G_F = 4$ full rollouts and $G_T = 4$ truncated rollouts. We set the coefficient of the additional advantage term to $\alpha = 0.1$ and the KL regularization coefficient to $\beta = 0.001$. At inference time, we use temperature 0.6 and top-p sampling with $p = 0.95$. All answers are evaluated using Math-Verify. We train the 1.5B and 1.7B models for 120 steps and Qwen3.5-4B for 40 steps. For JET, we select the checkpoint immediately before performance plateaus, applying the same model-selection criterion to REFRAIN. Experiments are conducted on two NVIDIA A800 GPUs, except for Qwen3.5-4B, which is trained on four A800 GPUs. For more details, please refer to Appendix D.

4.2 Main Results

Accuracy–efficiency trade-off.

Table 1 shows that REFRAIN achieves the highest average accuracy for all three model families while reducing generation relative to the corresponding base models. On Distill-1.5B, REFRAIN improves average accuracy from 29.85% to 32.51% and reduces response length by 23.1%. On Qwen3-1.7B, it slightly improves over GRPO in accuracy, from 42.07% to 42.47%, while using 5.2% fewer tokens. The gain is larger on Qwen3.5-4B, where REFRAIN improves over GRPO by 4.21 accuracy points and reduces generation by 13.2%. Thus, the improvement cannot be attributed solely to applying reinforcement learning. Rather than replacing GRPO-style optimization, REFRAIN complements it with repetition-aware rollout construction and fine-grained credit assignment. As shown in the training-efficiency analysis below, these stronger results are also obtained with lower rollout cost per training step.

Table 2: Out-of-domain evaluation on GPQA-Diamond with a maximum response length of 8,192 tokens.

Method	Acc. (%) ↑	Tokens ↓
Distill-1.5B		
Vanilla	35.10	6,934
ShorterBetter	32.83 (−2.27)	**3,869** (−44.2%)
LC-R1	32.95 (−2.15)	5,226 (−24.6%)
SmartThinker	34.09 (−1.01)	5,849 (−15.6%)
REFRAIN	**35.86** (+0.76)	6,183 (−10.8%)
Qwen3-1.7B		
Vanilla	**37.25**	5,897
GRPO	**37.25** (+0.00)	4,721 (−19.9%)
REFRAIN	**37.25** (+0.00)	**4,538** (−23.0%)
Qwen3.5-4B		
Vanilla	59.34	7,094
ShorterBetter	65.28 (+5.94)	**1,417** (−80.0%)
SmartThinker	72.98 (+13.64)	3,197 (−54.9%)
GRPO	73.36 (+14.02)	4,999 (−29.5%)
REFRAIN	**73.49** (+14.15)	4,919 (−30.7%)

Methods that optimize length more aggressively often produce shorter responses but lower accuracy. For example, ShorterBetter reduces Distill-1.5B generation to 2,859 tokens but falls 7.58 accuracy points below REFRAIN. On Qwen3-1.7B, ShorterBetter is 502 tokens shorter than REFRAIN but 2.85 points less accurate, while JET is both longer and 2.69 points less accurate. These comparisons indicate that REFRAIN primarily improves the accuracy–efficiency trade-off rather than minimizing response length in isolation.

Out-of-domain evaluation. Table 2 evaluates whether the efficiency improvements transfer beyond the mathematical training domain. On Distill-1.5B, REFRAIN achieves the highest accuracy and is the only efficiency-oriented method that improves upon Vanilla accuracy, while reducing generation by 10.8%. On Qwen3-1.7B, REFRAIN matches GRPO accuracy and further reduces token usage by 3.9%. On Qwen3.5-4B, REFRAIN slightly improves both accuracy and efficiency relative to GRPO, reaching 73.49% accuracy with 4,919 tokens. These results indicate that REFRAIN does not introduce an observable OOD accuracy penalty and can provide additional generation savings beyond standard outcome-reward optimization.

Robustness to sampling temperature. Fig. 3 compares Qwen3.5-4B and REFRAIN on AIME25 across different sampling temperatures. REFRAIN consistently achieves higher accuracy and shorter LRS values, while substantially reducing non-escaping repetitions, particularly in the low-temperature regime where the base model is most prone to deterministic loops. Although increasing the temperature also facilitates natural escape, it degrades the accuracy of the base model. In contrast, REFRAIN remains effective across the full temperature range. These results suggest that repetition-aware training reduces the exact repetitive patterns measured by LRS while maintain-

ing a favorable accuracy–length trade-off. They support LRS as a useful diagnostic for recurrent token-level behavior, rather than a complete measure of stalled reasoning.

Training efficiency. Fig. 4 compares the wall-clock step time and policy entropy of REFRAIN and GRPO on Qwen3-1.7B. By replacing a subset of full trajectories with short recovery continuations, REFRAIN consistently reduces the time required for each training step by approximately $1/3$ to $2/5$ throughout training. Importantly, this efficiency gain is not achieved through policy collapse. REFRAIN's entropy remains close to GRPO during early training, while the later difference is mainly caused by increasing GRPO entropy rather than REFRAIN collapse. Thus, the rollout savings do not exhibit the persistent entropy decrease associated with reduced exploration.

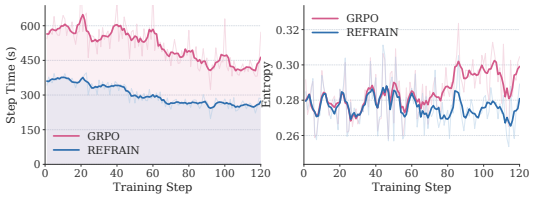


Figure 4: Training efficiency and policy entropy of REFRAIN and GRPO on Qwen3-1.7B.

4.3 Ablation Study

Table 3 isolates the contributions of the linear efficiency advantage, prefix-trie credit assignment, recovery construction, and repetition-aware localization. Adding the linear advantage to standard GRPO yields a numerically similar accuracy (42.18% versus 42.07%) but increases the average response length from 9,331 to 9,587 tokens. GRPO-Trie likewise does not improve over GRPO, obtaining 41.96% accuracy with 9,857 tokens. Thus, neither component alone explains the gain of the complete method.

Table 3: Core ablations on Qwen3-1.7B, averaged over HMMT, AIME25, AIME26, and AMC23. Values in parentheses denote absolute accuracy changes and relative token changes with respect to GRPO.

Method	Avg. Acc. (%) ↑	Avg. Tokens ↓
GRPO	42.07	9,331
GRPO + Linear Adv.	42.18 (+0.11)	9,587 (+2.7%)
GRPO-Trie ($\alpha = 0$)	41.96 (−0.11)	9,857 (+5.6%)
REFRAIN-Trie	39.28 (−2.79)	9,603 (+2.9%)
REFRAIN-Random	41.48 (−0.59)	9,449 (+1.3%)
REFRAIN-Resample	39.11 (−2.96)	**7,934** (−15.0%)
REFRAIN	**42.47** (+0.40)	8,844 (−5.2%)

Adding repetition-guided recovery branches without the auxiliary advantage reduces generation relative to GRPO-Trie, from 9,857 to 9,603 tokens, but lowers accuracy from 41.96% to 39.28%. With the linear advantage, REFRAIN reaches 42.47% accuracy with 8,844 tokens. Relative to GRPO + Linear Adv., this corresponds to a 0.29-point accuracy improvement and a 7.8% token reduction. These results suggest that the auxiliary signal complements recovery construction rather than independently shortening standard GRPO generations.

Replacing the repetition-aware boundary with a uniformly random valid boundary reduces accuracy to 41.48% and increases generation to 9,449 tokens. Relative to random truncation, repetition-aware localization improves accuracy by 0.99 points while reducing token usage by 6.4%, supporting repetition structure as a useful intervention signal. REFRAIN-Resample instead keeps the LRS boundary but replaces the transition hint with local next-token resampling, yielding shorter but substantially less accurate responses (7,934 tokens, 39.11%). Detailed variant definitions and additional ablations are provided in Appendix H.

5 Conclusion

We introduced REFRAIN, a repetition-aware reinforcement learning method for efficient long-form reasoning. By locating recurrent token patterns through the longest repeated substring, REFRAIN performs a speculative exit at the recurrence boundary, and learns from both full and recovery trajectories with structured credit assignment. Across multiple models and benchmarks, it improves the accuracy–efficiency trade-off. Our analyses further show that repetition-aware intervention is more effective than local next-token rerouting and that the observed efficiency gains are not explained by entropy collapse. These results support repetition as a useful structural signal for training more efficient reasoning policies.

AI USE STATEMENT

In this work, we used generative AI tools to assist with experimental design, method implementation, and interpretation of experimental results. We did not use generative AI tools for the other required-disclosure tasks, where applicable. We also used generative AI tools for literature search and summarization, brainstorming, code editing, and improving the readability of the manuscript. All AI-assisted outputs were reviewed and verified by the authors, including checking relevant literature against original sources and validating code and experimental interpretations against the underlying implementations and results. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed implementation and experimental settings to facilitate reproducibility. Prompt templates are documented in Appendix A, training configurations and hyperparameters in Appendix B, baseline implementation details in Appendix D, and algorithmic procedures in Appendix I. These materials specify the key components required to reproduce the reported results. Code will be made publicly available upon acceptance.

REFERENCES

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Zhengxiang Cheng, Dongping Chen, Mingyang Fu, and Tianyi Zhou. Optimizing length compression in large reasoning models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3232–3250, 2026.

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

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *Advances in Neural Information Processing Systems*, 38:48178–48204, 2025.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

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

Jinyi Han, Ying Huang, Ying Liao, Haiquan Zhao, Zishang Jiang, Xinyi Wang, Xikun Lu, Guanghao Zhou, Sihang Jiang, Jiaqing Liang, Weikang Zhou, Zeye Sun, Fei Yu, and Yanghua Xiao. Your models have thought enough: Training large reasoning models to stop overthinking. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2u5ZRzDyS0.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Zhezheng Hao, Hong Wang, Haoyang Liu, Jian Luo, Jiarui Yu, Hande Dong, Qiang Lin, Can Wang, and Jiawei Chen. Rethinking entropy interventions in RLVR: An entropy change perspective. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31105–31133, San Diego, California, United States, July 2026. Association for Computational

Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1436. URL https://aclanthology.org/2026.acl-long.1436/.

Chenzhi Hu, Qinzhe Hu, Yuhang Xu, Junyi Chen, Ruijie Wang, Shengzhong Liu, Jianxin Li, Fan Wu, and Guihai Chen. Smartthinker: Progressive chain-of-thought length calibration for efficient large language model reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=RciRcJ9LHw.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

math-ai. AMC23. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/math-ai/amc23.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

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

Michael Sullivan and Alexander Koller. GRPO is secretly a process reward model. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=nMGaOCVlDW.

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

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=NpU7ZXafRi.

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. *Advances in Neural Information Processing Systems*, 38: 39011–39043, 2025.

Ping Yu, Jing Xu, Jason E Weston, and Ilia Kulikov. Distilling system 2 into system 1. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*, 2024. URL https://openreview.net/forum?id=WUoC4BpJBC.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2025.

Xinliang Frederick Zhang, Anhad Mohananey, Alexandra Chronopoulou, Pinelopi Papalampidi, Somit Gupta, Tsendsuren Munkhdalai, Lu Wang, and Shyam Upadhyay. Do LLMs really need 10+ thoughts for "find the time 1000 days later"? towards structural understanding of LLM overthinking. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17005–17030, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.773. URL https://aclanthology.org/2026.acl-long.773/.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2026, 2026.

A Evaluation Prompt Templates

All evaluation prompts are single-turn `user` messages of the form `prefix + problem + suffix`. We deliberately vary the answer-format instruction across benchmark families: AIME25 / AIME26 / AMC23 follow the DAPO-Math-17K style used in training-adjacent math RL settings, while HMMT and GPQA use different templates so that reported gains are not tied to one surface form of the answer cue.

A.1 AIME25-26 / AMC23

These three contest-math sets share the default math prompt (`MATH_PREFIX` / `MATH_SUFFIX` in our data preparation), which matches the instruction wrapping used by DAPO-Math-17K: step-by-step reasoning, with the final line of the form `Answer: $Answer`.

AIME25 / AIME26 / AMC23

[prefix] Solve the following math problem step by step. The last line of your response should be of the form `Answer: $Answer` (without quotes) where `$Answer` is the answer to the problem.

[body] *<problem statement>*

[suffix] Remember to put your answer on its own line after "`Answer:`".

A.2 HMMT

HMMT uses a distinct default: empty prefix and a short `\boxed{}` CoT suffix. There is no `Answer:` line instruction; the model is asked only to think step by step and place the final answer in a box.

HMMT

[prefix] *(empty)*

[body] *<problem statement>*

[suffix] Let's think step by step and output the final answer within `\boxed{}`.

A.3 GPQA Diamond

GPQA is multiple-choice rather than free-form contest math. We use an OpenAI simple-evals-style MCQ template: options `A)`–`D)`, step-by-step reasoning, and a final line `Answer: $LETTER` with `LETTER` $\in \{$`A, B, C, D`$\}$. Choices are shuffled with a fixed seed at data-preparation time. Together with the HMMT boxed template, this checks generalization beyond the DAPO `Answer:` math format.

GPQA Diamond

[prefix] 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.

[body] *<question stem>*

`A) ... B) ... C) ... D) ...`

[suffix] *(empty)*

B Detailed Training Setup

B.1 Optimization Configuration

Table 4 summarizes the optimization settings used in our experiments. Unless otherwise specified, all methods trained under our unified setup use the same dataset, prompt batch size, rollout-group size, maximum response length, and optimization budget.

Hyperparameter	Value
Training dataset	DAPO-MATH-17K
Global prompt batch size	32
Rollouts per prompt, G	8
Full rollouts, G_{F}	4
Truncated rollouts, G_{T}	4
Maximum training response length	8,192 tokens
Learning rate	10^{-6}
Optimizer	AdamW
Learning-rate schedule	constant
Gradient clipping	1
GRPO clipping range	$[0.8, 1.2]$
KL coefficient, β	0.001
Additional advantage coefficient, α	0.1
Training temperature	1
Training top-p	1
Gradient checkpointing	enabled
Random seed	42

Table 4: Optimization hyperparameters used in training.

DeepSeek-R1-Distill-Qwen-1.5B and Qwen3-1.7B are trained for 120 optimization steps, while Qwen3.5-4B is trained for 40 steps. We use the correctness reward

$$R_i^{\mathrm{acc}} \in \{-1, 1\}, \tag{13}$$

following DAPO-MATH-17K. For methods evaluated with their original configurations, we retain the reward definition and method-specific hyperparameters from the corresponding official implementation.

B.2 Software Environment

DeepSeek-R1-Distill-Qwen-1.5B and Qwen3-1.7B are trained on two NVIDIA A800 GPUs, while Qwen3.5-4B is trained on four NVIDIA A800 GPUs. The principal hardware and software configurations are listed in Table 5.

Component	Configuration
Operating system	Ubuntu 22.04 LTS
CPUs	64
GPU	NVIDIA A800
GPU memory	80GB
GPUs for 1.5B/1.7B models	2
GPUs for 4B model	4
Python	3.12
CUDA	13.0
PyTorch	2.8.0/2.10.0(Qwen3.5)
Transformers	4.50.2/5.14.1(Qwen3.5)
Training framework	VeRL
Rollout engine	vLLM

Table 5: Hardware and software environment.

C Case Study

We illustrate the proposed truncation heuristic on two AIME25 trajectories of Qwen3.5-4B (temperature 0.6). In both runs the model has already derived the correct answer, then *repeats* a previously completed block of reasoning (a detected cycle). The intervention is cycle-centric: once a cycle

appears for the *second* time, we search *backward* from that second onset for a natural cut entry (preferring a nearby hesitation string such as `Wait`, else a paragraph/line break), truncate there, append a `</think>` hint, and continue for at most 1024 tokens. For visualization, these case studies use the hard-cue variant; the main results use the soft cue. Thus `Wait` is only a local boundary cue; the substantive requirement is merely that the cut land *before the second onset*, so that generation exits before the repetition resumes. Depending on how far back the nearest clean entry lies, this places the cut either in the gap between the two occurrences or already before the first one; both keep the productive prefix and discard the impending replay.

Setup. Model: Qwen3.5-4B. Benchmark: AIME25. Both source trajectories are originally correct.

C.1 CASE A: AIME25 #0; ANSWER 70

Problem (paraphrased). Find the sum of all integer bases $b > 9$ such that 17_b divides 97_b.

Kept prefix (step summary).

1. Translate: $17_b = b + 7$, $97_b = 9b + 7$.
2. Require $(b + 7) \mid (9b + 7)$.
3. Rewrite $9b + 7 = 9(b + 7) - 56$, hence $(b + 7) \mid 56$.
4. With $b > 9$, only divisors $d = b + 7 \in \{28, 56\}$ survive, so $b \in \{21, 49\}$.
5. Verify both bases; conclude $\text{sum} = 21 + 49 = 70$, and state "Looks solid."

Cycle structure and cut. A short cycle that re-lists factorizations of 56 (cycle length ≈ 28) first appears near position 1986 and recurs at 2041. Searching backward from the second onset yields a cut at 1862 (boundary cue: `Wait`). Here the nearest clean entry lies before *both* onsets, so the cut falls just after the answer is locked (`Looks solid.`) and the divisor-enumeration loop is never entered at all. Original length: 2960 tokens; the discarded suffix restates content already present in the prefix ($\approx$ 1.1k tokens).

Regeneration after the hint. The model emits a short, structured solution that reuses the same algebra and terminates normally:

$$b + 7 \in \{28, 56\} \;\Rightarrow\; b \in \{21, 49\} \;\Rightarrow\; 21 + 49 = 70,$$

ending with `Answer:` `70` (642 generated tokens, finish reason `stop`; kept+regen $\approx$ 2504 tokens vs. original 2960).

AIME25 #0

[1. Translate] **Translate the Problem into Algebra:** $\langle \dots \rangle$ $17_b = 1 \cdot b^1 + 7 \cdot b^0 = b + 7.$ $\langle \dots \rangle$ $97_b = 9 \cdot b^1 + 7 \cdot b^0 = 9b + 7.$

[2. Require] $\langle \dots \rangle$ this means $(b + 7) \mid (9b + 7).$

[3. Rewrite] $\langle \dots \rangle$ $9b + 7 = 9(b + 7) - 56$ $\langle \dots \rangle$ So, $k = 9 - \frac{56}{b+7}.$ $\langle \dots \rangle$ $b + 7$ must be a divisor of 56.

[4. Filter] $\langle \dots \rangle$ $d = b + 7 > 16.$ $\langle \dots \rangle$ Divisors > 16: 28, 56. $\langle \dots \rangle$ $b + 7 = 28 \implies b = 21.$ $b + 7 = 56 \implies b = 49.$

[5. Verify / lock] $\langle \dots \rangle$ $196/28 = 7.$ Yes. $\langle \dots \rangle$ $448/56 = 8.$ Yes. $\langle \dots \rangle$ Sum $= 21 + 49 = 70.$ Looks solid.

■ **[cut @ token pos. below ⇒ insert hint]**

[cut @ 1862; marker `Wait`] Wait, check if there are any subtle constraints I missed.

$\langle \dots \rangle$ "integer bases $b > 9$". $\langle \dots \rangle$ Everything seems correct.
Final check: $21 + 49 = 70.$ Divisors of 56:
[1st cycle @ 1986] $56 = 2 \times 28$ $56 = 4 \times 14$
$\langle \dots \rangle$ $56 = 7 \times 8$ $56 = 1 \times 56$
[2nd cycle @ 2041] $56 = 2 \times 28$ $56 = 4 \times 14$
$\langle \dots \rangle$ Sorted: $1, 2, 4, 7, 8, 14, 28, 56.$ Candidates: $28, 56.$ $b = 21, 49.$ Sum $= 70.$

Takeaway. The useful derivation ends before the first cycle; neither occurrence adds new information. Cutting ahead of the impending replay converts an unbounded restatement loop into a finite answer-writing act.

C.2 Case B: AIME25 #15; answer 468

Problem (paraphrased). Points A, B, C, D, E, F are collinear in that order; G is off the line. Given $AC = 26$, $BD = 22$, $CE = 31$, $DF = 33$, $AF = 73$, $CG = 40$, $DG = 30$, find the area of $\triangle BGE$.

Kept prefix (step summary).

1. Let the consecutive segment lengths be a, b, c, d, e; summing the four given two-segment equations and subtracting $AF = 73$ gives $BE = b + c + d = 39$.
2. Recover $c = CD = 14$ and use $CG = 40$, $DG = 30$ to obtain the perpendicular height $h = 24$.
3. Compute $\text{Area}(\triangle BGE) = \frac{1}{2} \cdot 39 \cdot 24 = 468$ and verify all segment lengths.

Cycle structure and cut. A segment-recovery cycle (length ≈ 50) first appears at 3746 and recurs at 6161. Crucially, the selected cut at 5781 (boundary cue: `\n\nWait`) lies *between* these two occurrences. By the cut point the trajectory has already derived $BE = 39$, $h = 24$, and area 468; the second occurrence would only recompute $c = 14$, $b = 8$, $a = 18$, and $e = 16$. Original length: 11379 tokens.

Regeneration after the hint. Continuation independently recovers $BE = 39$, $CD = 14$, and $h = 24$, preserving the correct path to $\frac{1}{2} \cdot 39 \cdot 24 = 468$. Generation reaches the 1024-token cap while completing the geometric derivation, but all sufficient intermediate quantities remain correct.

AIME25 #15

[1. Solve for BE] $\langle \dots \rangle$ $(a+b)+(b+c)+(c+d)+(d+e)=112$.
Subtract $a+b+c+d+e=73$. $b+c+d=39$.
We need $BE=BC+CD+DE=b+c+d=39$.
[1st cycle @ 3746] $c=31-17=14$.
$b=22-14=8$. $a=26-8=18$. $e=33-17=16$.
$\langle \dots \rangle$ Sum $a+b+c+d+e=18+8+14+17+16=73$. Correct.
[2. Height and area] $\langle \dots \rangle$ C is at 26. D is at 40. $CG=40$, $DG=30$.
Solution $x=58$, $h=24$. Area $=0.5\times 39\times 24=468$.
$\langle \dots \rangle$ $\cos\alpha=4/5$. $h=CG\sin\alpha=40\times 0.6=24$.
$\langle \dots \rangle$ Area of BGE is $0.5\times 39\times 24=468$.
■ **[cut @ token pos. below ⇒ insert hint]**
[cut @ 5781; marker `\n\nWait`] Wait, I should check if B, E are on the line.
Yes, "Six points … lie in a straight line". So B, C, D, E are collinear. The height of G from the line is constant.
$\langle \dots \rangle$ Let's re-verify the equations.
[2nd cycle @ 6161] $c=31-17=14$.
$b=22-14=8$. $a=26-8=18$. $e=33-17=16$.
$\langle \dots \rangle$ Check AF: $18+8+14+17+16=73$. OK. Check $BE=39$. $\langle \dots \rangle$

Takeaway. Productive content—base length and height—lives in the pre-cycle prefix. Truncating between the two cycle occurrences avoids replaying the same algebra; regeneration stays on the correct numerical path because those quantities remain in context.

C.3 Cross-case summary

In both examples the failure mode is not incorrect reasoning but *repetition after success*: a completed derivation is replayed as a detectable cycle. The heuristic does not target `Wait` per se; it uses the second cycle onset as an *anchor* and backs up to the nearest clean entry, so the only requirement is that the cut precede that second onset—generation must exit before the repetition resumes. The two cases illustrate the two admissible placements this induces: in #0 the backward search reaches past the first onset and cuts *before any* repetition has begun ($1862 < 1986$), whereas in #15 it lands *in the gap between* the two occurrences ($3746 < 5781 < 6161$), so the first pass is kept and only the second is discarded. Either way the pre-repetition prefix retains the productive content while the redundant replay is dropped.

A natural alternative is to cut as soon as a candidate final answer appears in the stream (e.g., after the first boxed value or an `Answer:` span). That rule is tightly coupled to tasks with an extractable, fixed answer format: it needs a reliable answer detector, and it has no well-defined analogue when the output is an open-ended proof, plan, or free-form response. Worse, early surface forms of an answer are often *guesses* floated mid-reasoning; truncating at their first occurrence can amputate subsequent verification or correction and reward premature commitment.

By contrast, our rule never inspects answer tokens. It only pattern-matches for a repeated reasoning block—a symptom of *stagnation* rather than of answer availability. The cut is therefore (i) independent of whether a ground-truth or canonical answer exists, (ii) potentially transferable to open-ended settings where "the answer" is not a short extractable string, and (iii) less likely to interrupt an exploratory or guessing phase, because a single speculative claim does not by itself constitute a second cycle. Only once a completed block is detectably *replayed* do we treat the trajectory as stalled and cut ahead of that replay—the signal we wish post-truncation reinforcement learning to reinforce: exit when reasoning stagnates, not merely when an answer-shaped token sequence first appears.

D BASELINE DETAILS

This section provides further details on the reinforcement-learning and efficient-reasoning baselines considered in our experiments. For a prompt x, let $\{y_i\}_{i=1}^{G}$ denote a group of sampled responses, ℓ_i the length of y_i, and r_i^{acc} its correctness reward. To ensure a controlled comparison, all baselines reimplemented under our training setup use the same correctness reward $R_i^{\text{acc}} \in \{0, 1\}$, consistent with their original implementations. For methods run with their official original configurations, including JET, we preserve their original reward formulations and other method-specific settings.

ShorterBetter. ShorterBetter (Yi et al., 2025) approximates a prompt-dependent optimal reasoning length using the Sample Optimal Length (SOL). When at least one response in the group is correct, SOL is defined as the length of the shortest correct response:

$$\ell^{\text{SOL}} = \min_{i:\, R_i^{\text{acc}}=1} \ell_i. \tag{14}$$

If all responses are incorrect, it instead uses the mean group length. Each response receives a correctness reward together with a penalty for deviating from SOL:

$$R_i = \alpha_{\text{SB}} R_i^{\text{acc}} - \beta_{\text{SB}} \left| \ell_i - \ell^{\text{SOL}} \right|. \tag{15}$$

This provides a dynamic target that adapts to the current prompt and policy. However, the shortest correct sample may be an outlier rather than the length with the highest probability of success. Moreover, after group normalization, a correct response that is substantially longer than SOL may receive a negative advantage.

SmartThinker. SmartThinker (Hu et al., 2026) extends the optimal-length formulation of ShorterBetter in two respects. First, rather than directly adopting the shortest correct response, it estimates the length that maximizes the conditional probability of correctness:

$$\ell^{\text{opt}} = \arg\max_{\ell} \Pr\left(R^{\text{acc}} = 1 \mid \ell, x\right). \tag{16}$$

It models the distribution of all response lengths and that of correct response lengths, and clips the resulting estimate to the observed group range. Only correct responses longer than the estimated optimum receive a length penalty:

$$R_i^{\text{len}} = \begin{cases} -\operatorname{ReLU}\left(\ell_i - \ell^{\text{opt}}\right), & R_i^{\text{acc}} = 1, \\ 0, & R_i^{\text{acc}} = 0. \end{cases} \tag{17}$$

Second, SmartThinker dynamically selects the coefficient of the length reward so that, after group normalization, correct trajectories retain non-negative advantages and incorrect trajectories retain non-positive advantages. This prevents a long but correct trajectory from being optimized in the same direction as an incorrect trajectory.

LC-R1. LC-R1 (Cheng et al., 2026) targets the redundant verification that occurs after a model has already derived the correct answer. It uses an auxiliary LC-Extractor to identify the effective reasoning prefix, defined as the portion of the reasoning trace ending at the first occurrence of the correct answer. The remaining reasoning is treated as an invalid or redundant suffix, and the compressed trajectory concatenates the effective prefix with the original final answer.

LC-R1 combines two reward components. Its Length Reward encourages global conciseness among correct compressed responses:

$$R_i^{\text{len}} = \begin{cases} 1 - \dfrac{|y_i'|}{\max_{j \in \mathcal{C}} |y_j'|}, & i \in \mathcal{C}, \\ 0, & i \notin \mathcal{C}, \end{cases} \tag{18}$$

where y_i' is the extracted compressed response and $\mathcal{C}$ is the set of correct responses. Its Compress Reward is applied specifically to the `</think>` token. It rewards terminating after sufficient reasoning has been completed and assigns a negative reward when the reasoning block is closed before the correct answer has been derived. Thus, LC-R1 explicitly distinguishes the valid reasoning prefix from post-solution redundancy, but requires an external extractor to locate the boundary.

JET. Just-Enough Thinking (JET) (Han et al., 2026) is the closest baseline to REFRAIN in terms of rollout construction. It first samples full reasoning trajectories and then constructs shortened variants using Progressive Early-Stopping. The truncation positions follow a predefined progression,

$$t_k = t_0 + k\Delta t, \tag{19}$$

and an explicit stop-thinking cue is inserted at each selected position before the model generates its final answer.

JET further introduces an accuracy-conditioned length reward. Only correct responses receive this reward, and shorter correct responses are favored relative to the shortest and longest correct responses in the same group:

$$R_i^{\text{len}} = \begin{cases} \left(\dfrac{\ell_{\max} - \ell_i}{\ell_{\max} - \ell_{\min} + \epsilon} \right) \alpha_{\text{JET}}(1-\delta) + \delta, & i \in \mathcal{C}, \\ 0, & i \notin \mathcal{C}. \end{cases} \tag{20}$$

This design makes correctness a prerequisite for receiving an efficiency reward. However, the total reward is subsequently normalized across the rollout group, and JET does not impose an explicit constraint ensuring that every correct trajectory has a non-negative normalized advantage. In particular, in a group containing multiple correct responses, a relatively long correct trajectory can still fall below the group mean.

The main difference from REFRAIN lies in the truncation criterion: JET uses progressively predefined temporal positions, whereas REFRAIN selects an endogenous recurrence boundary identified by repetition structure.

E Interpretation of Prefix-Trie Credit Assignment

This section provides a node-level interpretation of the prefix-trie credit assignment introduced in Section 3.3. The implementation retains the standard trajectory-token GRPO objective: every token occurrence is optimized once, while occurrences mapped to the same trie node receive the same branch-averaged advantage. We show that this construction preserves the aggregate local policy-gradient contribution in the unclipped case. Under PPO clipping, it also remains equivalent to trajectory-wise credit when all downstream advantages have the same sign; the two assignments can differ only when positive and negative downstream advantages coexist and the common importance ratio lies outside the clipping interval.

Node-level interpretation. Consider a rollout group $\mathcal{G}(x)$ containing G full and recovery trajectories. Let R_i be the verifier reward of trajectory i, and define

$$\mu_{\mathcal{G}} = \frac{1}{G}\sum_{i=1}^{G} R_i, \qquad \sigma_{\mathcal{G}} = \sqrt{\frac{1}{G}\sum_{i=1}^{G}(R_i - \mu_{\mathcal{G}})^2}. \tag{21}$$

The base group-normalized advantage is

$$\hat{A}_i^{\text{base}} = \frac{R_i - \mu_{\mathcal{G}}}{\sigma_{\mathcal{G}} + \epsilon}. \tag{22}$$

For a trie node v, let

$$\mathcal{D}(v) = \{i \in \{1, \dots, G\} : \text{trajectory } i \text{ traverses } v\}, \tag{23}$$

and

$$D_v = |\mathcal{D}(v)|. \tag{24}$$

Averaging the base advantages of the downstream branches gives

$$\hat{A}_{v,\text{base}}^{\text{trie}} = \frac{1}{D_v}\sum_{i \in \mathcal{D}(v)} \hat{A}_i^{\text{base}} \tag{25}$$

$$= \frac{\overline{R}_v - \mu_{\mathcal{G}}}{\sigma_{\mathcal{G}} + \epsilon}, \tag{26}$$

where

$$\overline{R}_v = \frac{1}{D_v} \sum_{i \in \mathcal{D}(v)} R_i \tag{27}$$

is the mean reward of the branches descending from v.

Thus, up to the positive group-normalization factor, the base advantage of a shared node is its mean downstream reward relative to the rollout-group baseline. The node receives positive base credit when its downstream branches perform better on average than the rollout group, and negative base credit when they perform worse.

For the final trajectory advantage $\hat{A}_i = \hat{A}_i^{\text{base}} + \alpha \hat{A}_i^{\text{eff}}$, linearity gives

$$\hat{A}_v^{\text{trie}} = \hat{A}_{v,\text{base}}^{\text{trie}} + \frac{\alpha}{D_v} \sum_{i \in \mathcal{D}(v)} \hat{A}_i^{\text{eff}}. \tag{28}$$

Correctness and efficiency feedback are therefore averaged over the same downstream branches before being assigned to the shared node. For each token occurrence (i, t) mapped to node $v(i, t)$, the objective uses

$$\hat{A}_{i,t}^{\text{trie}} = \hat{A}_{v(i,t)}^{\text{trie}}. \tag{29}$$

Relation to Monte Carlo value backup. This node-level interpretation resembles a batch Monte Carlo value backup. A Monte Carlo node value is estimated from the mean return of simulations passing through the node:

$$Q(v) = \frac{W(v)}{N(v)} = \frac{1}{N(v)} \sum_{j=1}^{N(v)} R^{(j)}. \tag{30}$$

Similarly, $\overline{R}_v$ averages the outcomes of all rollout branches traversing v. REFRAIN additionally centers this estimate using the rollout-group mean, converting the downstream value estimate into a group-relative advantage for policy optimization.

Equivalence under the unclipped objective. Let

$$g_v(\theta) = \nabla_\theta \log \pi_\theta(y_v \mid x, y_{<v}) \tag{31}$$

denote the score-function gradient of the action represented by node v. Since all trajectories in $\mathcal{D}(v)$ share the same prompt, prefix, and next token at this node, their local score-function gradients are identical.

Under standard trajectory-wise credit assignment, each occurrence receives its trajectory advantage. The aggregate local contribution is

$$G_v^{\text{traj}} = \sum_{i \in \mathcal{D}(v)} \hat{A}_i g_v \tag{32}$$

$$= \left(\sum_{i \in \mathcal{D}(v)} \hat{A}_i \right) g_v. \tag{33}$$

Under trie credit assignment, every occurrence mapped to v receives the same averaged advantage

$$\hat{A}_v^{\text{trie}} = \frac{1}{D_v} \sum_{j \in \mathcal{D}(v)} \hat{A}_j. \tag{34}$$

Because the shared node occurs once in each of the D_v trajectories, its aggregate contribution is

$$G_v^{\text{trie}} = \sum_{i \in \mathcal{D}(v)} \hat{A}_v^{\text{trie}} g_v \tag{35}$$

$$= D_v \left(\frac{1}{D_v} \sum_{j \in \mathcal{D}(v)} \hat{A}_j \right) g_v \tag{36}$$

$$= \left(\sum_{j \in \mathcal{D}(v)} \hat{A}_j \right) g_v \tag{37}$$

$$= G_v^{\text{traj}}. \tag{38}$$

Therefore, trie averaging does not change the aggregate first-order policy-gradient contribution of a shared action under the unclipped objective. Instead, it changes how this credit is distributed across trajectory occurrences: trajectory-wise assignment applies different branch-specific advantages to identical shared actions, whereas trie assignment applies the same downstream-consensus advantage to every occurrence.

Effect under PPO clipping. Let

$$\phi(\rho, A) = \min\left(\rho A,\ \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A\right) \tag{39}$$

denote the clipped surrogate for one token occurrence. Because the shared prefix and action are identical, all occurrences mapped to node v have the same importance ratio ρ_v.

Trajectory-wise credit produces

$$L_v^{\text{traj}} = \sum_{i \in \mathcal{D}(v)} \phi(\rho_v, \hat{A}_i), \tag{40}$$

whereas trie credit produces

$$L_v^{\text{trie}} = \sum_{i \in \mathcal{D}(v)} \phi(\rho_v, \hat{A}_v^{\text{trie}}) \tag{41}$$

$$= D_v \phi \left(\rho_v, \frac{1}{D_v} \sum_{i \in \mathcal{D}(v)} \hat{A}_i \right). \tag{42}$$

When ρ_v lies inside the clipping interval, $\phi(\rho_v, A) = \rho_v A$, and hence

$$L_v^{\text{trie}} = \rho_v \sum_{i \in \mathcal{D}(v)} \hat{A}_i = L_v^{\text{traj}}. \tag{43}$$

Thus, trie and trajectory-wise credit are exactly equivalent whenever clipping is inactive, regardless of the signs of the downstream advantages.

The equivalence also holds when clipping is active but all downstream advantages have the same sign. For a fixed importance ratio, define

$$c_+(\rho) = \min(\rho, 1+\epsilon), \qquad c_-(\rho) = \max(\rho, 1-\epsilon). \tag{44}$$

Then

$$\phi(\rho, A) = \begin{cases} c_+(\rho)A, & A \geq 0, \\ c_-(\rho)A, & A < 0. \end{cases} \tag{45}$$

If all $\hat{A}_i$ are nonnegative, both assignments equal

$$c_+(\rho_v) \sum_{i \in \mathcal{D}(v)} \hat{A}_i. \tag{46}$$

If all $\hat{A}_i$ are nonpositive, both equal

$$c_-(\rho_v) \sum_{i \in \mathcal{D}(v)} \hat{A}_i. \tag{47}$$

The two assignments can therefore differ only when positive and negative downstream advantages coexist and clipping is active. Let

$$\mathcal{P}_v = \{i \in \mathcal{D}(v) : \hat{A}_i \geq 0\},\ \mathcal{N}_v = \{i \in \mathcal{D}(v) : \hat{A}_i < 0\}. \tag{48}$$

Trajectory-wise credit then gives

$$L_v^{\text{traj}} = c_+(\rho_v) \sum_{i \in \mathcal{P}_v} \hat{A}_i + c_-(\rho_v) \sum_{i \in \mathcal{N}_v} \hat{A}_i, \tag{49}$$

while trie credit first forms

$$\overline{A}_v = \frac{1}{D_v} \sum_{i \in \mathcal{D}(v)} \hat{A}_i \tag{50}$$

and applies one common clipping direction:

$$L_v^{\text{trie}} = D_v \begin{cases} c_+(\rho_v)\overline{A}_v, & \overline{A}_v \geq 0, \\ c_-(\rho_v)\overline{A}_v, & \overline{A}_v < 0. \end{cases} \tag{51}$$

Thus, trie aggregation does not alter shared-prefix updates when all downstream branches support the same update direction. Its effect arises when branches sharing an identical prefix lead to conflicting outcomes. In this case, trajectory-wise credit clips the positive and negative signals separately, whereas trie credit first resolves them into a consensus advantage and then applies a common clipped update.

Implications across trie depth. Nodes near the root are shared by many trajectories and therefore summarize a broad set of downstream outcomes. Trie aggregation assigns all occurrences of such a node the same consensus advantage. As trajectories diverge, D_v decreases and the credit becomes increasingly branch-specific. At a trajectory-unique node, $D_v = 1$, so

$$\hat{A}_v^{\text{trie}} = \hat{A}_i. \tag{52}$$

The trie therefore provides a continuous transition from consensus credit on shared prefixes to trajectory-specific credit on unique continuations. Importantly, it does not change updates merely because a prefix is shared by many branches. Sharing a prefix alone does not change its aggregate update. Trie aggregation changes the optimization only when positive and negative downstream advantages coexist and the common importance ratio lies outside the clipping interval.

F Implementation Details of Length-Aware Baselines

The publicly available implementations of ShorterBetter and SmartThinker are based on an earlier version of VeRL and cannot be directly applied to the newer stack used for our Qwen3.5 experiments. We therefore port both methods to the same Qwen3.5-compatible VeRL framework and train them under the same model, data, rollout budget, context length, batch size, and optimization budget as our Qwen3.5-4B experiments. The original reward definitions and method-specific hyperparameters are retained without modification.

Both ShorterBetter and SmartThinker require prompt-level group statistics. ShorterBetter computes the Sample Optimal Length from the correctness labels and lengths of all responses in a rollout group, while SmartThinker estimates an optimal length from the distributions of all responses and correct responses.

The newer VeRL stack does not expose the complete rollout group during batched reward computation. We therefore move only the group-dependent length computation from the reward stage to the advantage-computation stage. Correctness rewards and response lengths are first collected for individual trajectories. After responses are grouped by prompt, we compute the original optimal-length statistic, construct the original combined reward, and apply the corresponding group-relative advantage computation.

For ShorterBetter, this procedure preserves the original Sample Optimal Length, length deviation penalty, and normalized advantage. For SmartThinker, it preserves the estimated optimal length, the

penalty applied to correct responses longer than the estimated optimum, and the dynamically calibrated length-reward coefficient. No REFRAIN-specific advantage or sign constraint is introduced into either baseline.

We validate the ports using unit tests constructed from fixed rollout groups. Given identical correctness labels and response lengths, the reference and ported implementations produce the same optimal-length statistics, per-response length terms, and final advantages up to floating-point precision. This verifies that moving the group-dependent computation to the advantage stage does not change either baseline algorithm.

Tables 1 and 2 show that the effect of length-aware reinforcement learning depends on both the strength of the length objective and the reasoning requirements of the evaluation task.

ShorterBetter applies the strongest length pressure and produces the shortest responses on both mathematical reasoning and GPQA-Diamond. On the mathematical benchmarks, this compression reduces average accuracy below both the base model and GRPO. On GPQA-Diamond, it improves over the base model, but remains substantially below GRPO. These results indicate that selecting the shortest correct response as the group-level length target can encourage excessive compression, particularly when problems require longer derivations.

SmartThinker provides a more favorable trade-off by dynamically calibrating its length-reward coefficient so that correct trajectories retain non-negative advantages. On the mathematical benchmarks, it improves over both the base model and ShorterBetter, but remains below GRPO. On GPQA-Diamond, however, SmartThinker nearly matches GRPO accuracy while using 36.0% fewer tokens. Sign-preserving calibration therefore substantially reduces the interference between correctness and length optimization, and can provide an effective accuracy–efficiency trade-off when shorter trajectories remain sufficient for the task.

The difference between the mathematical and GPQA-Diamond results also shows that preserving the sign of a correct trajectory's advantage does not guarantee a uniform outcome across tasks. A correct response can retain a non-negative advantage while still receiving a smaller relative update than under correctness-only GRPO. Whether this change harms performance depends on how strongly successful reasoning relies on longer trajectories. In our experiments, the remaining length pressure has a larger effect on contest mathematics than on GPQA-Diamond.

Unlike trajectory-level length objectives that determine the efficiency signal solely from total response length, REFRAIN uses detected repetition boundaries to construct targeted recovery rollouts. Both the original and recovery trajectories then receive a correctness-preserving length-based auxiliary advantage, rather than applying a penalty only to the detected repetition. The resulting correctness and efficiency signals are further aggregated through a shared-prefix trie, which assigns consistent credit to identical actions shared by the original and recovery branches.

On the mathematical benchmarks, REFRAIN improves average accuracy from 53.35% to 57.56% relative to GRPO while reducing average generation from 7,873 to 6,830 tokens. It also outperforms both length-aware baselines in mathematical accuracy. On GPQA-Diamond, REFRAIN maintains GRPO-level accuracy with a small reduction in generated tokens. The GPQA-Diamond results present a complementary trade-off. SmartThinker produces substantially shorter responses than REFRAIN and reaches accuracy within 0.51 points of REFRAIN. REFRAIN, in contrast, remains closer to the GRPO response length and preserves its accuracy. Thus, REFRAIN does not uniformly dominate SmartThinker in token reduction. Its main advantage in these experiments is a more consistent correctness result across the mathematical and scientific-reasoning benchmarks, while SmartThinker achieves a stronger compression ratio on GPQA-Diamond.

These experiments also verify that the comparison is not caused by incompatibility between the original baseline implementations and the newer Qwen3.5 training stack. The ported implementations preserve the original optimal-length and advantage computations, as confirmed by unit tests, while using the same model, data, context length, rollout size, and optimization budget as the corresponding Qwen3.5-4B experiments.

G Controlled Comparison with JET and Recovery Dynamics

JET (Han et al., 2026) is the closest baseline to REFRAIN in terms of rollout construction: both methods retain a prefix of a complete reasoning trajectory, explicitly encourage an early transition toward the answer, and optimize jointly over complete and shortened trajectories. Their intervention criteria, however, are fundamentally different. JET applies Progressive Early-Stopping (PES) at predefined temporal positions, whereas REFRAIN determines the intervention point from the repetition structure of each trajectory.

The JET results in the main experiments follow its original training recipe as closely as possible. For the analysis in this section, we additionally port JET to our training framework and construct a fully controlled setting in which the surrounding training configuration is matched to REFRAIN. This allows us to compare not only final performance, but also how the quality of full and truncated trajectories evolves during training.

G.1 Porting JET to Our Training Framework

We port the two components of JET that directly affect its optimization: the quality-controlled length reward and Progressive Early-Stopping (PES). The original implementation is built on EasyR1, where the length reward is implemented inside a batch-level reward function and PES is integrated into the rollout engine. Our implementation instead follows the current VeRL training stack used by REFRAIN, while preserving the algorithmic formulation of JET.

Progressive Early-Stopping. We follow the PES schedule specified in the JET paper. Given a group of complete trajectories, JET constructs truncated trajectories at 25%, 50%, and 75% of the character-level response. The original paper uses a rollout group size of $G = 12$, consisting of three complete trajectories and nine truncated trajectories. To match the $G = 8$ rollout groups used in our experiments, we preserve the same 1:3 ratio between complete and truncated trajectories:

$$\mathcal{G}_{\text{JET}} = \{y_1^F, y_2^F, y_{1,0.25}^T, y_{1,0.50}^T, y_{1,0.75}^T, y_{2,0.25}^T, y_{2,0.50}^T, y_{2,0.75}^T\}.$$

Thus, two complete trajectories are generated first, and each produces three PES branches. The character-level truncation point follows the original rule

$$c = \max(1, \lfloor r|y^F| \rfloor), \qquad r \in \{0.25, 0.50, 0.75\}.$$

After truncation, a stop-thinking cue is appended and a short answer continuation of at most 50 tokens is generated.

The released EasyR1 code contains a different PES configuration, using two complete trajectories and five truncation ratios when $G = 12$. Since this differs from the schedule stated in the JET paper, we follow the paper formulation rather than this implementation-specific configuration. Our $G = 8$ setting is therefore a proportional scaling of the paper's $3+9$ construction, rather than a reproduction of the alternative EasyR1 rollout split.

Quality-controlled length reward. We also reproduce the quality-controlled length reward used by JET. For trajectory i, the adjusted reward is

$$r_i = w_{\text{acc}} a_i + w_f f_i + w_\ell r_{\ell,i}, \tag{53}$$

where a_i and f_i denote correctness and format rewards, respectively, and

$$w_{\text{acc}} = 0.9, \qquad w_f = 0.1, \qquad w_\ell = 1.$$

For a correct response,

$$r_{\ell,i} = \frac{\ell_{\max} - \ell_i}{\ell_{\max} - \ell_{\min} + \epsilon} \alpha(1 - \delta) + \delta, \tag{54}$$

with $\alpha = 1.2$ and $\delta = 0.05$. Incorrect responses receive no length bonus. The resulting rewards are then normalized using the standard group-relative estimator.

Following the mathematical definition in JET, $\ell_{\min}$ and $\ell_{\max}$ are computed over correct responses to the same question. The released EasyR1 reward implementation instead computes these extrema over all correct responses passed to one batch-level reward call, which can contain multiple

prompts. We follow the paper definition and perform the computation independently within each rollout group.

To verify the port, we compare our implementation against the original EasyR1 reward function and GRPO estimator on fixed single-prompt rollout groups. Given identical correctness labels, format indicators, and response lengths, the adjusted rewards and normalized advantages agree up to floating-point precision. We separately test the character-level cut positions, PES grouping, and two-stage continuation construction.

Answer-format adaptation. The original JET implementation uses a `\boxed{}`-based stop-thinking cue and answer extractor. Our DAPO-Math training pipeline instead evaluates answers using the `Answer:` format. For the controlled experiment, we therefore replace only the surface form of the JET stop-thinking cue with `Answer:`, while retaining the same PES truncation procedure and continuation budget. This adaptation is necessary to avoid systematically treating otherwise valid PES continuations as invalid outputs under our verifier. It does not alter the temporal truncation ratios or the quality-controlled reward.

G.2 Native-Recipe and Controlled JET Comparisons

JET and REFRAIN were originally developed under substantially different training configurations. In particular, the native JET recipe uses a larger training batch, a longer maximum response length, a different training-data mixture, and its original `\boxed{}`-based answer pipeline. In contrast, our REFRAIN experiments use DAPO-Math-17K, a smaller training batch, an 8,192-token maximum training response length, and the `Answer:`-based evaluation pipeline.

We therefore use two complementary JET settings.

Native-recipe JET for the main comparison. The JET result reported in Table 1 follows its original training recipe as closely as possible. We use this version in the main table because JET performs substantially better under its native training recipe than after being transferred to the fully matched setting described below. Reporting only the matched result would therefore understate the baseline by evaluating it outside the configuration in which it was originally designed and tuned. It is intended to provide a strong baseline comparison rather than a single-variable causal comparison between repetition-based and position-based intervention.

Matched JET for controlled dynamics analysis. For the analysis below, we instead retrain JET under the same surrounding environment as REFRAIN. Specifically, we match the training dataset, training batch size, rollout group size, maximum response length, sampling configuration, answer-extraction pipeline, and number of optimization steps. This controlled setting is necessary for comparing training dynamics. For example, with the original batch sizes, one JET optimization step and one REFRAIN step process different numbers of prompts and therefore represent different amounts of optimization and rollout computation; their step-wise trajectories would not be directly aligned.

JET performs worse under this matched setting than under its native recipe. Because several components of the training environment change simultaneously, we do not attribute this degradation to any single hyperparameter. Instead, the difference indicates that JET is sensitive to the surrounding training configuration. The controlled run is therefore used specifically to study the behavior of the two training procedures under a common environment, while the stronger native-recipe result remains the appropriate JET baseline for the main performance table.

G.3 Recovery Dynamics under the Controlled Setting

The matched setting allows us to study a question that cannot be answered from final benchmark accuracy alone: when a full trajectory is truncated and redirected toward an answer, does the intervention preserve useful reasoning or recover unsuccessful reasoning?

Let

$$R_F \in \{0, 1\}$$

denote the correctness of a source full trajectory, and let

$$R_T \in \{0, 1\}$$

Table 6: JET configurations used for the main comparison and the controlled training-dynamics analysis.

Setting	Native JET	Controlled JET
Training data	Original JET mixture	DAPO-Math-17K
Training batch size	128	32
Rollout group size	12	8
Max. response length	10,000	8,192
PES ratios	10/30/50/70/90%	25/50/75%
Answer format	`\boxed{}`	`Answer:`
Usage	Main Table	Dynamics Analysis

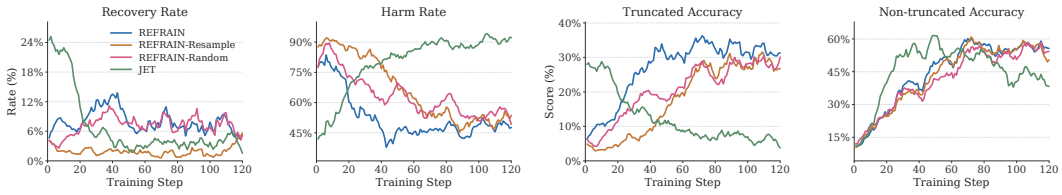


Figure 5: Recovery dynamics under the fully controlled training setting.

denote the correctness of the truncated trajectory constructed from it. For JET, one full trajectory produces multiple PES branches; each truncated branch is paired with its corresponding source trajectory. REFRAIN and its variants are analyzed analogously using their full–recovery pairs.

We define two conditional transition metrics. The *recovery rate* is

$$\text{RecoveryRate} = P(R_T = 1 \mid R_F = 0), \tag{55}$$

which measures how often an intervention converts an incorrect full trajectory into a correct truncated trajectory. The *harm rate* is

$$\text{HarmRate} = P(R_T = 0 \mid R_F = 1), \tag{56}$$

which measures how often truncation destroys the correctness of an originally successful trajectory.

We additionally report

$$\text{Acc}_T = P(R_T = 1), \qquad \text{Acc}_F = P(R_F = 1), \tag{57}$$

for truncated and full trajectories, respectively. These quantities satisfy

$$P(R_T = 1) = (1 - \text{HarmRate}) \cdot P(R_F = 1) + \text{RecoveryRate} \cdot P(R_F = 0). \tag{58}$$

Therefore, truncated accuracy alone conflates two factors: the quality of the underlying full trajectories and the quality of the intervention applied to them. Equations 55–58 separate these effects.

Full-trajectory quality alone does not explain the difference. We first examine the full trajectories from which truncated branches are constructed. As shown in Fig. 5, REFRAIN, REFRAIN-Random, and REFRAIN-Resample exhibit very similar full-trajectory accuracy throughout training. Their curves remain close despite substantial differences in the quality of the corresponding truncated trajectories.

JET also develops a competitive full-trajectory policy during much of training. Its full accuracy rises rapidly in the early stage and reaches a level comparable to the REFRAIN variants before declining later in training. Therefore, the poor quality of JET's truncated branches cannot be explained simply by a uniformly weaker source policy. This distinction is important because the truncated trajectories are generated conditionally on these full trajectories: similar $P(R_F = 1)$ together with different $P(R_T = 1)$ implies that the intervention process itself contributes substantially to the observed difference.

Repetition-aware intervention produces higher-quality recovery branches. REFRAIN exhibits a markedly different trajectory for its truncated rollouts. Its truncated accuracy improves rapidly during the early stages of training and reaches roughly 30–35%, remaining substantially above REFRAIN-Random and REFRAIN-Resample for most of the optimization process.

The conditional transition metrics further clarify where this improvement comes from. Given an incorrect source trajectory, REFRAIN maintains a non-negligible probability of recovering a correct answer throughout training. REFRAIN-Resample, despite having comparable full-trajectory accuracy, exhibits a much lower recovery rate for most steps. This shows that simply generating an alternative continuation does not by itself produce equally useful supervision.

Conditioned on a correct source trajectory, REFRAIN also generally yields a lower harm rate than the two internal variants during the middle and later stages of training. The absolute harm rate remains non-trivial, meaning that recovery construction is not guaranteed to preserve an already correct trajectory. The relevant observation is comparative: repetition-aware intervention is less likely to destroy correctness than less targeted recovery constructions while simultaneously being more capable of recovering some incorrect trajectories.

Together, these trends explain why REFRAIN attains higher overall truncated accuracy even though the marginal quality of its full trajectories is similar to that of the internal controls.

JET's truncated trajectories progressively decouple from its full policy. JET exhibits a qualitatively different training dynamic. At the beginning of optimization, its PES branches are relatively effective: the recovery rate is initially high and truncated accuracy is substantially higher than the accuracy of the still-weak full policy. However, this behavior reverses quickly as training proceeds.

While JET's full-trajectory accuracy improves rapidly, the quality of its PES trajectories continuously deteriorates. Its conditional recovery rate falls from a high initial value to only a few percent, whereas its harm rate rises steadily and approaches approximately 90% in the later stage. As a consequence, truncated accuracy decreases from the high-20% range early in training to only a few percent near the end, even though the full policy has meanwhile learned to solve a substantial fraction of the problems.

This divergence reveals a progressive decoupling between complete reasoning and position-based early exits. A JET full trajectory may be correct, yet its prefixes at fixed temporal ratios increasingly fail to contain sufficient information for a successful early answer. Equivalently, improvements in the complete reasoning policy do not automatically translate into improvements in the PES branches constructed from fixed relative positions.

The result is particularly visible in the conditional harm rate. As training progresses, an increasing fraction of correct full trajectories become incorrect after PES intervention. The simultaneous increase in full accuracy and decrease in truncated accuracy rules out the simple explanation that the PES degradation is merely caused by a deteriorating underlying policy. Instead, under the controlled configuration, the shortened trajectories themselves become a progressively poorer proxy for the successful complete reasoning paths from which they are derived.

Interpretation. These dynamics are consistent with the central motivation of REFRAIN: the usefulness of an early-exit trajectory depends critically on *where* the intervention is applied.

A fixed temporal ratio does not directly indicate whether the reasoning process has already completed the derivation necessary for a correct answer. As the policy changes during RL, the amount and location of useful computation may also shift, so a fixed 25%, 50%, or 75% prefix need not preserve the same semantic content across training stages. The JET dynamics observed above illustrate this mismatch: full reasoning becomes more successful, while temporally truncated branches become progressively less aligned with that success.

REFRAIN instead conditions its intervention point on trajectory structure. It branches when recurrent token-level behavior indicates that reasoning has begun to revisit an existing pattern, rather than at a predetermined fraction of the response. The lower conditional harm and higher truncated accuracy observed for REFRAIN are consistent with the hypothesis that such structure-aware boundaries are more likely to preserve useful reasoning before redirecting generation toward the answer.

Table 7: Configurations of the core and extended ablations. All REFRAIN variants use the same rollout and training setup unless explicitly modified below.

Method	Recovery	Boundary	Intervention	Efficiency Adv.	Prefix Credit
Core ablations in Table 3					
GRPO	No	–	–	–	–
GRPO + Linear Adv.	No	–	–	Linear	–
GRPO-Trie ($\alpha = 0$)	No	–	–	–	Trie
REFRAIN-Trie	Yes	LRS	Soft hint	–	Trie
REFRAIN-Random	Yes	Random	Soft hint	Linear	Trie
REFRAIN-Resample	Yes	LRS	Top-20 resample	Linear	Trie
REFRAIN	Yes	LRS	Soft hint	Linear	Trie
Extended ablations in Table 8					
Sparse	Yes	LRS	Soft hint	Sparse	Trie
Linear-Mask-All	Yes	LRS	Soft hint	Linear	Mask shared prefix
Linear-Hard	Yes	LRS	Hard closing	Linear	Trie
Linear	Yes	LRS	Soft hint	Linear	Trie

Table 8: Extended ablations on Qwen3-1.7B at temperature $T = 0.6$ with a maximum response length of 16,384 tokens. Accuracy is averaged over four independent generations per problem. The Average columns report the unweighted macro-average over HMMT, AIME25, AIME26, and AMC23. The best value in each column is bolded; all tied best accuracy values are bolded.

	HMMT		AIME25		AIME26		AMC23		Average	
Method	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓	Acc. ↑	Tokens ↓
Vanilla	20.45	13,939	**33.33**	12,977	**31.67**	13,006	76.88	7,988	40.58	11,978
Sparse	21.97	12,636	30.83	**8,990**	**31.67**	**8,810**	78.75	**4,132**	40.81	**8,642**
Linear-Mask-All	23.48	13,291	28.33	10,449	27.50	10,093	77.50	5,145	39.20	9,745
Linear-Hard	**24.24**	12,705	32.50	9,235	**31.67**	8,836	76.88	4,527	41.32	8,826
Linear	**24.24**	**12,269**	**33.33**	9,567	**31.67**	9,090	**80.63**	4,451	**42.47**	8,844

We emphasize that this controlled experiment compares the complete JET and REFRAIN training procedures, not only their boundary-selection rules. The two methods additionally differ in their reward construction and credit-assignment mechanisms. Therefore, Fig. 5 does not by itself establish a causal advantage of LRS localization over every positional boundary rule. Instead, it shows that under the same training environment, repetition-aware recovery produces substantially healthier full-to-truncated outcome dynamics than JET's PES procedure and the less-targeted REFRAIN variants.

H Extended Ablation Study

Table 7 summarizes the variants used in both the main-text and extended ablations. REFRAIN-Random preserves the recovery and optimization pipeline but replaces the LRS-selected boundary with a random boundary. REFRAIN-Resample instead keeps the LRS boundary and replaces the transition hint with local next-token resampling: the next token is sampled from the renormalized top-20 distribution, after which generation continues without an inserted hint. Thus, the two variants probe complementary questions of *where* and *how* to intervene.

Resampling intervention. REFRAIN-Resample uses the same LRS-selected boundary as REFRAIN, but replaces the inserted transition hint with a locally resampled next token. Specifically, the original next token is excluded when possible, and a replacement is sampled from the renormalized top-20 next-token distribution before generation continues normally. We did not mask the resampled token in the ablation study. Unlike the explicit transition cue, this local intervention does not directly encourage the rollout to terminate reasoning, and we observe that it produces substantially shorter but less accurate responses, together with lower policy entropy.

Sparse efficiency advantage. In addition to the default linear efficiency advantage, we evaluate a sparse variant that places efficiency feedback only on the shortest correct and longest incorrect trajectories. Let $\mathcal{C}_{\min}$ denote all tied shortest correct trajectories and $\mathcal{W}_{\max}$ all tied longest incorrect trajectories. The sparse auxiliary advantage is defined as

$$\widehat{A}^{\text{eff}}_{i,\text{sparse}} = \begin{cases} 1, & i \in \mathcal{C}_{\min}, \\ -1, & i \in \mathcal{W}_{\max}, \\ 0, & \text{otherwise.} \end{cases} \tag{59}$$

As in the linear variant, the auxiliary term is disabled for all-correct and all-wrong rollout groups.

Masking shared prefixes. The Mask-All variant removes prefix-trie credit assignment. For every full/recovery branch pair, all policy-generated tokens in their shared prefix are masked from the optimization objective. Only branch-specific tokens after the divergence point remain trainable, while externally inserted transition tokens remain masked as in the default method. This control tests whether discarding shared-prefix feedback can replace the proposed branch-aware trie aggregation.

Hard transition cue. The default soft variant appends `\n\n**Final Answer**\n` at the selected truncation boundary. The hard variant instead inserts `\n</think>\n\n`, explicitly closing the reasoning block before regeneration. All other rollout and optimization settings are unchanged.

Overall accuracy–efficiency trade-off. Table 8 shows that the default linear configuration achieves the highest macro-average accuracy, reaching 42.47% while reducing average generation from 11,978 to 8,844 tokens relative to Vanilla. This corresponds to a 1.89-point increase in reported mean accuracy and a 26.2% reduction in response length. The improvement is also consistent across the four benchmarks: REFRAIN-Linear matches or exceeds Vanilla accuracy on every task while producing shorter responses in all cases.

Linear versus sparse efficiency feedback. The sparse variant produces the shortest responses on average, reducing generation to 8,642 tokens. However, this additional reduction is modest: it uses only 202 fewer tokens, or 2.3% fewer tokens relative to the linear variant, while its macro-average accuracy is 1.66 points lower (40.81% versus 42.47%). The difference is most visible on AIME25, where the sparse variant decreases accuracy from 33.33% to 30.83%. These results indicate that concentrating the efficiency signal only on the shortest-correct and longest-incorrect trajectories yields more aggressive compression but a less favorable accuracy–length trade-off than dense linear feedback.

Shared-prefix credit assignment. Masking all shared-prefix tokens substantially degrades both metrics relative to trie-based credit assignment. REFRAIN-Linear-Mask-All reaches only 39.20% average accuracy with 9,745 tokens, compared with 42.47% and 8,844 tokens for the default linear configuration. Thus, masking shared prefixes lowers accuracy by 3.27 points and increases average response length by 10.2%. The degradation occurs on every benchmark relative to REFRAIN-Linear, with particularly large accuracy drops on AIME25 and AIME26. These results show that simply discarding learning signals from shared prefixes loses useful downstream feedback; retaining and aggregating branch outcomes through the prefix trie gives a substantially better trade-off. This comparison establishes the importance of preserving shared-prefix feedback, although it does not by itself show that trie averaging is superior to every possible non-masking credit-assignment strategy.

Soft versus hard transition cues. The hard cue provides essentially no additional efficiency benefit over the default soft cue. Its average response length is 8,826 tokens, only 18 tokens shorter than the 8,844-token average of the soft variant, while its accuracy is 1.15 points lower. The two cues obtain the same accuracy on HMMT and AIME26, but the hard cue performs worse on AIME25 and especially AMC23. This suggests that explicitly closing the reasoning block can terminate generation slightly more aggressively, but may also interrupt useful answer construction. We therefore use the soft transition cue as the default.

Ablation summary. These results motivate the default configuration used in the main experiments: dense linear efficiency feedback, trie-aggregated credit, and a soft transition cue.

Algorithm 1 Latest Longest Repeated Substring

Require: Token sequence $s_{1:n}$
Ensure: Latest longest repeated substring z

Initialize suffix automaton with root 0
$\text{len}[0] \leftarrow 0$, $\text{link}[0] \leftarrow -1$, last $\leftarrow 0$
for $i = 1$ to n **do**
  Create state q with $\text{len}[q] \leftarrow \text{len}[\text{last}] + 1$, $\text{minend}[q] \leftarrow i$, and $\text{maxend}[q] \leftarrow i$
  $p \leftarrow \text{last}$
  while $p \neq -1$ and $\text{go}[p, s_i]$ is undefined **do**
    $\text{go}[p, s_i] \leftarrow q$
    $p \leftarrow \text{link}[p]$
  end while
  if $p = -1$ **then**
    $\text{link}[q] \leftarrow 0$
  else
    $r \leftarrow \text{go}[p, s_i]$
    if $\text{len}[p] + 1 = \text{len}[r]$ **then**
      $\text{link}[q] \leftarrow r$
    else
      Clone r as r'
      $\text{len}[r'] \leftarrow \text{len}[p] + 1$
      $\text{minend}[r'] \leftarrow \text{minend}[r]$
      $\text{maxend}[r'] \leftarrow \text{maxend}[r]$
      while $p \neq -1$ and $\text{go}[p, s_i] = r$ **do**
        $\text{go}[p, s_i] \leftarrow r'$
        $p \leftarrow \text{link}[p]$
      end while
      $\text{link}[r] \leftarrow r'$
      $\text{link}[q] \leftarrow r'$
    end if
  end if
  last $\leftarrow q$
end for
for each state v in decreasing order of $\text{len}[v]$ **do**
  $p \leftarrow \text{link}[v]$
  if $p \neq -1$ **then**
    $\text{minend}[p] \leftarrow \min(\text{minend}[p], \text{minend}[v])$
    $\text{maxend}[p] \leftarrow \max(\text{maxend}[p], \text{maxend}[v])$
  end if
end for
for each state $v \neq 0$ **do**
  $\ell_v \leftarrow \min\left(\text{len}[v], \text{maxend}[v] - \text{minend}[v]\right)$
end for
$v^\star \leftarrow \arg\max_{\ell_v > \text{len}[\text{link}[v]]} \big(\ell_v, \text{maxend}[v]\big)$ in lexicographic order
$\ell \leftarrow \ell_{v^\star}$, $e \leftarrow \text{maxend}[v^\star]$
return $s_{e-\ell+1:e}$

I Algorithmic Details

This section summarizes the implementation of REFRAIN. Algorithms 1–4 locate the repetition and determine the truncation boundary, while Algorithm 5 gives the complete training step.

Latest longest repeated substring. Algorithm 1 implements the suffix-automaton procedure described in Section 3.1. Each state records the minimum and maximum end positions of its occurrences. For each state, we take the longest substring supported by two non-overlapping occurrences and break ties by the latest occurrence.

Algorithm 2 KMP Prefix Table

Require: Pattern $p_{1:m}$
Ensure: Prefix table $\text{next}_{1:m}$
$\text{next}[1] \leftarrow 0$
for $i = 2$ to m **do**
  $j \leftarrow \text{next}[i-1]$
  while $j > 0$ and $p_{j+1} \neq p_i$ **do**
    $j \leftarrow \text{next}[j]$
  end while
  if $p_{j+1} = p_i$ **then**
    $j \leftarrow j + 1$
  end if
  $\text{next}[i] \leftarrow j$
end for
return next

KMP prefix table. Algorithm 2 gives the KMP prefix-table construction used in Section 3.1. For a pattern p, $\text{next}[i]$ is the length of the longest proper prefix of $p_{1:i}$ that is also its suffix.

Cycle and truncation boundary. Algorithm 3 derives a candidate repeating unit from the LRS and searches for its second non-overlapping occurrence; we optionally rotate the unit to a semantic ending before matching. If no second occurrence is found, the raw boundary defaults to the end of the reasoning span. Algorithm 4 then aligns the raw boundary to a `Wait`, paragraph, line, or sentence boundary, searching backward first and forward only if necessary. Sentence-ending punctuation is retained in the prefix, while the other matched markers are excluded; trailing whitespace is removed after alignment. All repetition-localization operations in Algorithm 5 are applied to the reasoning span rather than the full response.

Two-stage rollout and training. Algorithm 5 summarizes the two-stage rollout in Section 3.1, the hint masking and prefix-trie credit assignment in Section 3.3, and the correctness-preserving efficiency advantage in Section 3.2. The reasoning length L_i is the number of tokens between the `<think>` and `</think>` delimiters. The externally inserted hint is used only as conditioning context and is excluded from both the policy-gradient and KL terms.

Algorithm 3 Candidate Cycle and Second Non-Overlapping Match

Require: Trajectory $s_{1:n}$ and LRS $z_{1:m}$
Ensure: Candidate cycle u and raw truncation boundary c_{raw}

$\text{next}^z \leftarrow \text{KMPNext}(z)$
$p \leftarrow m - \text{next}^z[m]$
$u \leftarrow z_{1:p}$
Optionally rotate u to a semantic ending before matching
$\text{next}^u \leftarrow \text{KMPNext}(u)$
$j \leftarrow 0$, $a_1 \leftarrow \perp$, $c_{\text{raw}} \leftarrow \perp$
for $i = 1$ to n **do**
  while $j > 0$ and $u_{j+1} \neq s_i$ **do**
    $j \leftarrow \text{next}^u[j]$
  end while
  if $u_{j+1} = s_i$ **then**
    $j \leftarrow j + 1$
  end if
  if $j = p$ **then**
    $a \leftarrow i - p + 1$
    if $a_1 = \perp$ **then**
      $a_1 \leftarrow a$
    else if $a \geq a_1 + p$ **then**
      $c_{\text{raw}} \leftarrow a$
      break
    end if
    $j \leftarrow \text{next}^u[j]$
  end if
end for
if $c_{\text{raw}} = \perp$ **then**
  $c_{\text{raw}} \leftarrow n$
end if
return (u, c_{raw})

Algorithm 4 Semantic Boundary Alignment

Require: Trajectory $s_{1:n}$ and raw boundary c_{raw}
Ensure: Aligned truncation boundary c

$\mathcal{M}_{\text{wait}} \leftarrow \{\backslash\text{n}\backslash\text{nWait}, \backslash\text{nWait}, \text{Wait}\}$
$\mathcal{M}_{\text{sep}} \leftarrow \{\backslash\text{n}\backslash\text{n}, \backslash\text{n}\}$
$\mathcal{M}_{\text{sent}} \leftarrow \texttt{PERIOD}$
$\mathcal{M} \leftarrow [\mathcal{M}_{\text{wait}}, \mathcal{M}_{\text{sep}}, \mathcal{M}_{\text{sent}}]$
for each marker set $\mathcal{Q} \in \mathcal{M}$ **do**
  $(a, \ell) \leftarrow$ latest match of any $q \in \mathcal{Q}$ fully before c_{raw}
  if a exists and $a > 0$ **then**
    return $a + \ell$ if $\mathcal{Q} = \mathcal{M}_{\text{sent}}$, otherwise a
  end if
end for
for each marker set $\mathcal{Q} \in \mathcal{M}$ **do**
  $(a, \ell) \leftarrow$ earliest match of any $q \in \mathcal{Q}$ at or after c_{raw}
  if a exists **then**
    return $a + \ell$ if $\mathcal{Q} = \mathcal{M}_{\text{sent}}$, otherwise a
  end if
end for
return c_{raw}

Algorithm 5 One REFRAIN Training Step

Require: Prompt batch $\mathcal{X}$, policy π_θ, old policy π_{old}, reference policy π_{ref}, hint h, rollout sizes $G_{\text{F}}, G_{\text{T}}$, coefficients α, β, and clipping threshold ϵ

$\mathcal{L} \leftarrow 0$, $N \leftarrow 0$
for each prompt $x \in \mathcal{X}$ **do**
  Sample full rollouts $\mathcal{Y}^{\text{F}} = \{y_i^{\text{F}}\}_{i=1}^{G_{\text{F}}}$
  Sort $\mathcal{Y}^{\text{F}}$ in decreasing order of $|y_i|^{\text{F}}$
  Let $\mathcal{I}_{\text{T}}$ contain the indices of the longest $\min(G_{\text{T}}, G_{\text{F}})$ full trajectories
  for each $i \in \mathcal{I}_{\text{T}}$ **do**
    $z_i \leftarrow \text{LatestLRS}(y_i^{\text{F}})$
    $(u_i, c_i^{\text{raw}}) \leftarrow \text{FindCycle}(y_i^{\text{F}}, z_i)$
    $c_i \leftarrow \text{AlignBoundary}(y_i^{\text{F}}, c_i^{\text{raw}})$
  end for
  Generate

$$y_j^{\text{T}} = y_{j,<c_j}^{\text{F}} \oplus h \oplus \widetilde{y}_j, \qquad \widetilde{y}_j \sim \pi_{\text{old}}(\cdot \mid x, y_{j,<c_j}^{\text{F}}, h)$$

  $\mathcal{G}(x) \leftarrow \mathcal{Y}^{\text{F}} \cup \mathcal{Y}^{\text{T}}$
  for each trajectory $y_i \in \mathcal{G}(x)$ **do**
    $R_i \leftarrow \text{Verify}(x, y_i)$
    $\mathcal{H}_i \leftarrow$ positions of inserted hint tokens in y_i
    $(b_i, e_i) \leftarrow \text{ThinkSpan}(y_i)$
    $L_i \leftarrow e_i - b_i - 1$ {tokens between <think> and </think>}
  end for
  $\widehat{A}_i^{\text{base}} \leftarrow \dfrac{R_i - \text{mean}_{j \in \mathcal{G}(x)} R_j}{\text{std}_{j \in \mathcal{G}(x)} R_j + \varepsilon}$
  $\mathcal{C} \leftarrow \{i : R_i > 0\}$, $\mathcal{W} \leftarrow \{i : R_i \le 0\}$
  if $\mathcal{C} \neq \phi$ and $\mathcal{W} \neq \phi$ **then**
    for each $i \in \mathcal{C} \cup \mathcal{W}$ **do**

$$\mathcal{S}_i \leftarrow \begin{cases} \mathcal{C}, & i \in \mathcal{C}, \\ \mathcal{W}, & i \in \mathcal{W} \end{cases}$$

$$\widetilde{L}_i \leftarrow \frac{L_i - \min_{j \in \mathcal{S}_i} L_j}{\max_{j \in \mathcal{S}_i} L_j - \min_{j \in \mathcal{S}_i} L_j + \varepsilon}$$

$$\widehat{A}_i^{\text{eff}} \leftarrow \begin{cases} 1 - \widetilde{L}_i, & i \in \mathcal{C}, \\ -\widetilde{L}_i, & i \in \mathcal{W} \end{cases}$$

    end for
  else
    $\widehat{A}_i^{\text{eff}} \leftarrow 0$ for all i
  end if
  Build a unified prefix trie from all $y_i \in \mathcal{G}(x)$
  for each trie node v **do**
    $\mathcal{D}(v) \leftarrow \{i : y_i \text{ traverses } v\}$

$$\widehat{A}_v^{\text{trie}} \leftarrow \frac{1}{|\mathcal{D}(v)|} \sum_{i \in \mathcal{D}(v)} \left(\widehat{A}_i^{\text{base}} + \alpha \widehat{A}_i^{\text{eff}} \right)$$

  end for
  for each token $y_{i,t}$ at trie node $v(i,t)$ **do**
    $m_{i,t} \leftarrow \mathbf{1}[t \notin \mathcal{H}_i]$
    $\hat{A}_{i,t} \leftarrow \widehat{A}_{v(i,t)}^{\text{trie}}$

$$\rho_{i,t} \leftarrow \frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t} \mid x, y_{i,<t})}$$

$$\mathcal{L} \leftarrow \mathcal{L} - m_{i,t} \left[\min \left(\rho_{i,t} \hat{A}_{i,t}, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) \hat{A}_{i,t} \right) - \beta D_{\text{KL}} \left(\pi_\theta \| \pi_{\text{ref}} \right)_{i,t} \right]$$

    $N \leftarrow N + m_{i,t}$
  end for
end for
$\mathcal{L}_{\text{REFRAIN}} \leftarrow \mathcal{L} / \max(N, 1)$
Update θ by minimizing $\mathcal{L}_{\text{REFRAIN}}$