GIVING CREDIT WHERE IT'S DUE: REDUNDANCY-AWARE LEARNING FOR EFFICIENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large reasoning models can produce correct yet unnecessarily long reasoning traces. Existing methods improve reasoning efficiency with trajectory-level objectives or local token- and step-level signals, but rarely model inter-step semantic dependencies. This limits their ability to distinguish redundant steps from those that support later deductions, making it harder to shorten reasoning without sacrificing accuracy. We introduce **RECAP** (REdundancy-aware Credit Assignment via Propagation), which addresses this limitation by **assigning credit where it is due** based on both a step's downstream role in the reasoning structure and its contribution to solving the problem correctly. We define *structural responsibility* to capture the step's downstream role by measuring how strongly later reasoning depends on it, using credit propagated backward from the final-answer node through an outcome-independent, LLM-annotated semantic dependency graph. **However, a step can have high structural responsibility yet steer the reasoning away from the correct solution.** RECAP therefore introduces *step efficacy* to measure answer-directed progress through changes in gold-answer log-likelihood as each step is added. Together, these signals reshape rollout-level GRPO advantages into step-specific updates. RECAP requires neither a separately trained process reward model nor preconstructed concise trajectories. Across two 7B models and four mathematical reasoning benchmarks, RECAP improves the accuracy–efficiency trade-off. On Qwen2.5-Math-7B, it improves pass@1 by 2.0-3.7 percentage points while reducing reasoning tokens by 8%–31% relative to GRPO across all four benchmarks. Analysis suggests these savings reflect fewer reasoning operations and less dead-end reasoning, rather than more compact expression.

1 INTRODUCTION

Large reasoning models achieve strong performance on challenging mathematical and logical tasks by learning to generate extended reasoning traces through reinforcement learning with verifiable rewards (Guo et al., 2025; Jaech et al., 2024). Although extended reasoning improves performance, not every step within a reasoning trace contributes equally to the final answer. Even successful traces can contain repetition, unproductive exploration, and unnecessary verification that contribute little to the final solution (Chen et al., 2025; Sui et al., 2025; Yue et al., 2025). These low-contribution steps introduce unnecessary computation and, in turn, increase inference-time cost. Standard outcome-based reinforcement learning provides no direct pressure to reduce this overhead, as it rewards a trace based on final-answer correctness (Shao et al., 2024). The resulting trajectory-level advantage is applied across the generated sequence, allowing low-contribution steps to be reinforced alongside the deductions that support the answer and thereby permitting redundant reasoning to persist.

Recent work improves reasoning efficiency by assigning efficiency signals at different granularities. Trajectory-level methods encourage shorter solutions using explicit length signals (Arora & Zanette, 2025; Li et al., 2026d; Peng et al., 2026; Li et al., 2026a) or model-derived objectives such as entropy (Huang et al., 2026), but assign a single signal to the entire trace and therefore cannot identify which steps should be preserved. Finer-grained methods estimate token- or step-level utility using proxies such as attention or changes in correct-answer likelihood (Liu et al., 2026b; He et al., 2026; Li et al., 2026c). These signals provide more selective supervision, but primarily capture a step's direct relation to the final answer and may miss contributions realized only through later deductions. Pruning methods use such proxies to remove low-utility segments and train on compressed

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.

traces (Liu et al., 2026c), inheriting the same limitation. These methods derive efficiency supervision from aggregate trajectory objectives, local proxy scores, or post-hoc compressed behaviors, but **do not explicitly model how information introduced by a step propagates through later reasoning and contributes to the final answer**.

To address this limitation, we propose *REdundancy-aware Credit Assignment via Propagation (RECAP)*, which assigns step-level credit along two complementary dimensions: **whether later reasoning depends on the step**, and **whether that contribution advances the correct solution**.

To capture downstream contribution, RECAP represents each trajectory as a semantic dependency graph whose edges are typed as *support*, *context*, and *restatement*, depending on whether an earlier step is required for later deductions, provides useful context, or merely repeats what is already available. RECAP then propagates credit backward from the final answer through this graph, assigning greater **structural responsibility** to steps whose information supports downstream reasoning and less to redundant or unused steps. Structural responsibility alone, however, does not indicate whether a step advances the correct solution. *An incorrect step may still support an entire downstream reasoning.* RECAP therefore complements structural responsibility with **step efficacy**, measured by the change in correct-answer log-likelihood as each step is added. We couple efficacy with the sign of the rollout-level advantage, reinforcing answer-advancing steps on successful rollouts while protecting them from excessive penalty on unsuccessful ones.

RECAP combines step efficacy with the propagated structural responsibility into a step-specific weight, which refines the outcome-derived advantage into an *outcome-aware, dependency-sensitive* training signal for each step. It requires neither a separately trained process reward model nor preconstructed concise trajectories. In summary, our contributions are:

1. We introduce a typed semantic dependency graph that distinguishes three inter-step relations: *support*, *context*, and *restatement*. We propagate structural credit through these dependencies, assigning each step credit by its downstream role rather than its distance from the final answer.
2. We complement structural credit with a step-level efficacy signal that measures each step's progress toward the correct answer. Combining the two signals turns the rollout-level GRPO advantage into step-specific updates that reinforce useful reasoning while downweighting redundancy in successful trajectories, and preserve helpful steps while penalizing misleading ones in unsuccessful trajectories.
3. We demonstrate across models and mathematical benchmarks that RECAP maintains or improves accuracy while reducing reasoning tokens, cutting them by 8%–31% over GRPO (Shao et al., 2024) on Qwen2.5-Math-7B (Yang et al., 2024) and by 9%–20% over strong efficiency baselines on DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025). Further analysis shows that these savings are associated with reduced unnecessary computation, including dead-end reasoning.

2 Related Work

Existing approaches to token-efficient reasoning can be broadly grouped into prompt-based control, length-regularized policy optimization, pruning and distillation, and fine-grained credit assignment. Prompt-based methods encourage concise reasoning through explicit instructions, output constraints, or adaptive hints (Renze & Guven, 2024; Lee et al., 2025; Tang et al., 2025), but provide limited guidance on which reasoning steps should be preserved.

Length Regularization and Trace Compression. Length-regularized methods incorporate response length into rollout-level optimization through fixed or adaptive penalties conditioned on correctness, problem difficulty, or training dynamics (Arora & Zanette, 2025; Li et al., 2026a; Peng et al., 2026; Li et al., 2026d). These methods can effectively shorten responses, but their efficiency signal is defined at the trajectory level and does not identify which individual steps are redundant. A complementary line of work constructs concise training traces by pruning redundant content or distilling compressed trajectories (Jiang et al., 2026; Liu et al., 2026a; Ma et al., 2026; Yuan et al., 2026; Liu et al., 2026c). Such methods learn from compressed traces rather than assigning credit to steps within newly sampled rollouts.

Fine-Grained and Structure-Aware Credit Assignment. More recent methods assign token- or step-level signals using answer likelihood, confidence, conditional mutual information, or attention (Li et al., 2026b; Wang et al., 2026; He et al., 2026; Nie et al., 2026; Liu et al., 2026b; Li et al.,

2026c). These signals provide more selective supervision, but largely evaluate each unit through its local effect and do not explicitly capture how information introduced by one step contributes to later reasoning. Structure-aware methods introduce trees or graphs to organize reasoning trajectories or assign finer-grained credit (Ji et al., 2026; Dong et al., 2026; Zhan et al., 2026). However, they either target more efficient sampling and credit assignment rather than shorter reasoning traces, or improve token efficiency through shorter paths without directly identifying redundant computation.

RECAP directly models these intra-trace semantic dependencies and propagates structural responsibility through the resulting graph, while using step efficacy to incorporate answer-directed progress. This allows policy updates to distinguish steps by both their downstream role and their effect on the final solution. We provide a more detailed comparison in Appendix A.1.

3 PRELIMINARY

RECAP extends Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with step-specific credit assignment. For each prompt, GRPO samples a group of rollouts and assigns each rollout an outcome reward. Given reward R_k for rollout k, GRPO computes the group-normalized advantage $A_k^{GRPO} = \frac{R_k - mean(\mathbf{R})}{std(\mathbf{R}) + \epsilon}$, where $\epsilon > 0$ is used for numerical stability. Rollouts with above-average rewards receive positive advantages, while those below the group average receive negative advantages. Then, it assigns $A_{k,t} = A_k^{GRPO}$ for all tokens t within the rollout and optimize

$$J(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{k=1}^{G}\frac{1}{|o_k|}\sum_{t=1}^{|o_k|}\left(\min\left(r_{k,t}A_{k,t},\ \text{clip}(r_{k,t}, 1-\varepsilon, 1+\varepsilon)A_{k,t}\right) - \beta_{DL}\,\mathbb{D}_{KL}\right)\right] \tag{1}$$

The expectation is over queries q sampled from the distribution Q, and rollouts $o_k \sim \pi_{\theta_{old}}(\cdot \mid q)$. π_θ and $\pi_{\theta_{old}}$ denote the current and old policy models, respectively. G is the number of rollouts in each group. The token-level importance sampling ratio is $r_{k,t} = \frac{\pi_\theta(o_{k,t}|q,o_{k,<t})}{\pi_{\theta_{old}}(o_{k,t}|q,o_{k,<t})}$. Clipping this ratio limits drastic policy updates, while $\beta_{DL}\mathbb{D}_{KL}$ is a KL divergence term that regularizes the current policy π_θ against large deviation from the reference policy π_{ref}.

A key limitation of GRPO lies in the uniform assignment $A_{k,t} = A_k^{\text{GRPO}}$. All tokens in a rollout receive the same learning signal despite contributing differently to the final solution. Useful derivations, repeated checks, and abandoned branches are therefore reinforced or penalized together. To reduce redundant reasoning, we seek to redistribute the rollout-level signal A_k^{GRPO} across the reasoning trace according to the contribution of each step. Intuitively, a step should receive greater credit when its information is subsequently used by later reasoning and contributes to the final answer. This leads to our central question: *how should learning credit be distributed across reasoning steps so that, on the same problems, the learned policy preserves successful reasoning while suppressing redundant content and using fewer inference tokens?*

4 METHODOLOGY

The question of how to assign the advantage at the step-level cannot be judged from the step alone. A locally correct step may still be redundant if nothing depends on it, while a short transition may be essential if it supports later deductions. We therefore model dependencies among reasoning steps explicitly and use them to construct dependency-aware step credit. As shown in Figure 1, the main design of RECAP has three stages:

1. **Structural Responsibility within a trace.** We build a semantic dependency graph within each rollout to identify which steps are used by later reasoning or the final answer (Section 4.1.1), and propagate structure credit backward from the final answer (Section 4.1.2).
2. **Step Efficacy toward the solution.** Besides the structual responsibilty, we measure whether each step moves the reasoning toward or away from the correct answer (Section 4.2).
3. **Reshape the rollout-level advantage.** We combine structural responsibility and step efficacy to reshape the rollout-level GRPO advantage into step-specific advantages (Section 4.3).

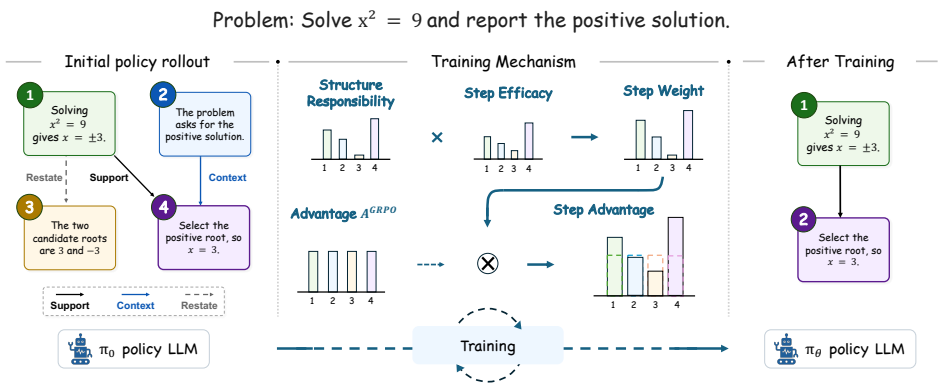


Figure 1: The overview of RECAP.

4.1 Structural Responsibility within a Trace

A model could waste tokens by restating existing information or introducing content unused by later reasoning. Therefore, improving efficiency requires distinguishing steps that carry useful information forward from those that do not. We characterize this distinction through a step's *structural responsibility*, which measures how strongly subsequent deductions depend on the information it introduces. Specifically, we (1) construct an intra-trace semantic dependency graph to represent how information is carried forward, and (2) propagate credit over the graph to quantify each step's responsibility for the downstream reasoning leading to the final answer.

4.1.1 Intra-Trace Dependency Modeling

A step's contribution depends on how later reasoning uses it, so we represent each trace as a directed acyclic graph (DAG) of semantic dependencies rather than as a plain sequence. We segment each rollout into steps, one per node, where an edge $i \to j$ means that Step j uses information from Step i. Dependencies point forward in the trace, thus the graph is acyclic. Steps with no directed path to the final answer form disconnected branches.

Not every dependency carries the same weight. A step may be strictly necessary for subsequent steps, merely helpful, or entirely useless. Collapsing these cases into a single edge type would obscure these distinctions. Therefore, we define three edge types and use LLM-as-a-Judge (Gu et al., 2026) to annotate pairwise step dependencies:

- ***Support***. Step j directly requires Step i for its derivation, verification, or answer extraction. These edges form the backbone of the solution.
- ***Context***. Step i supplies useful framing for Step j but is not logically required, which is helpful, yet removable without breaking the derivation.
- ***Restate***. Step j repeats what Step i already established without making reasoning progress, which is a marker of redundancy rather than contribution.

Figure 1 shows an example of the three edge types and how we turn a trace into a graph with them.

4.1.2 Dependency-based Structure Credit Propagation

To leverage the structural contributions captured by the graph for policy optimization, we convert it into a structure responsibility score for each step, quantifying its contribution to the final answer. We propagate responsibility backward under a local conservation rule. Specifically, we initialize the final-answer node with a responsibility $resp(F) = 1$, so that **the responsibility is outcome-independent**, and each node then distribute its responsibility back to the prerequisite steps on which it depends. For an edge $s_{k,i} \to s_{k,j}$ between step i and step j of rollout k, each child $s_{k,j}$ distributes the responsibility among its parents $s_{k,j}$ in proportion to their edge coefficients, normalized over $s_{k,j}$'s incoming edges. This gives the following update for each step $s_{k,i}$:

$$resp(s_{k,i}) = \sum_{j \in \{j' | \exists i \to j'\}} \frac{\gamma_{i \to j}}{W_j} resp(s_{k,j}), \qquad W_j = \sum_{p' \in \text{parents}(j)} \gamma_{p' \to j}. \tag{2}$$

The coefficient $\gamma_{p\to c}$ is determined by the edge type, so responsibility flows according to the role of each dependency: support edges carry the most, context edges partial, and restate edges none. Normalizing over each child's incoming edges ensures it distributes only the responsibility it holds across its parents, preventing an "in-degree hack" in which adding more dependencies could otherwise duplicate responsibility. The resulting responsibility of a step reflects how much structural responsibility reaches it through downstream dependencies. Backbone steps that support more downstream reasoning receive more, while redundant branches accumulate little. Because step responsibility follows semantic dependencies rather than positional distance, essential prerequisites can retain high responsibility regardless of how early they appear.

4.2 Step Efficacy toward the Solution

Structural responsibility measures how strongly downstream reasoning depends on a step, but not whether the step moves the model toward the correct solution. Two steps may have similar structural responsibility while having opposite effects on the correctness of the solution. We therefore introduce *step efficacy* to measure each step's immediate progress toward or away from the correct answer, complementing structural responsibility with an answer-directed signal.

Let the gold answer for the prompt q be tokenized as $a^\star = (a_1^\star, \dots a_T^\star)$. For rollout τ_k, let $\tau_{k,\le i} = (s_{k,1}, \dots, s_{k,i})$ denote the trajectory prefix ending at step $s_{k,i}$. Using teacher forcing, we evaluate the gold answer under each prefix by computing its mean token log-likelihood:

$$L_{k,i} = \frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta(a_t^\star \mid q, \tau_{k,\le i}, a_{<t}^\star), \tag{3}$$

where T is the number of the gold answer tokens and $a_{<t}^\star$ denotes the gold-answer tokens preceding $a_t^\star$. For $i = 0$, we compute $L_{k,0}$ from the prompt x without any generated steps. We define the efficacy of step $s_{k,i}$ as the change in the mean gold-answer log-likelihood caused by extending the preceding trajectory prefix with that step:

$$\Delta_{k,i} = L_{k,i} - L_{k,i-1} \tag{4}$$

A positive $\Delta_{k,i}$ indicates that the step moves the model toward the gold answer, while a negative value indicates the opposite. Step efficacy therefore captures the step's immediate answer-directed effect under the current policy.

4.3 From step scores to policy-gradient advantages

Structural responsibility captures downstream dependence, while step efficacy measures whether a step moves the model toward or away from the gold answer. We combine these complementary signals into a multiplicative step score that jointly captures downstream dependence and answer-directed effect, using an exponential transformation to convert efficacy into a positive factor on structural responsibility. Since exponentiation can amplify extreme efficacy values, we first clip the efficacy as $\widehat{\Delta}_{k,i} = \mathrm{clip}(\Delta_{k,i}, -c, c)$, where $c > 0$ is a fixed threshold. The effective contribution score of step $s_{k,i}$ is then

$$M_{k,i} = resp(s_{k,i}) * e^{\alpha * \mathrm{sign}(A_k)\widehat{\Delta}_{k,i}} \tag{5}$$

where $\alpha > 0$ controls the influence of step efficacy and $\mathrm{sign}(A_k)$ denotes the sign of the rollout-level advantage. The sign term makes the efficacy adjustment consistent with the rollout-level learning direction. Positive efficacy increases the step score in positive-advantage rollouts but decreases it in negative-advantage rollouts, while negative efficacy has the opposite effect. Thus, answer-advancing steps are reinforced more strongly in positive-advantage rollouts and protected from excessive penalty in negative-advantage rollouts. Structural responsibility independently modulates these updates according to each step's downstream use.

The scale of step scores may vary across rollouts, so we normalize them within each trace before reshaping the policy-gradient advantage. We compute the token-weighted mean $\bar{M}_k = \sum_i \frac{d_{k,i}}{D_k} M_{k,i}$, where $d_{k,i}$ is the number of tokens in step i and $D_k = \sum_i d_{k,i}$, and normalize each step score as $w_{k,i} = \frac{M_{k,i}}{\bar{M}_k + \epsilon}$, where $\epsilon > 0$ ensures numerical stability. This normalization expresses each step score relative to the token-weighted mean within the rollout, preserving the rollout-level update scale while redistributing credit across steps.

Finally, we use $w_{k,i}$ to adjust the rollout-level GRPO advantage. Specifically, instead of assigning a uniform advantage to the entire rollout, RECAP assigns all tokens within step i with the advantage

$$A_{k,i}^{RECAP} = [(1-\beta) + \beta * w_{k,i}]A_k. \tag{6}$$

where $\beta \in [0,1]$ controls the strength of step-level credit assignment. Since $w_{k,i}$ is normalized so that its expectation over tokens within each rollout is approximately one, the expected RECAP advantage over tokens t satisfies $\mathbb{E}_{t\sim\tau_k}\left[A_{k,t}^{\mathrm{RECAP}}\right] \approx A_k^{\mathrm{GRPO}}$. RECAP therefore redistributes the original GRPO advantage across the trace while approximately preserving its overall scale and update direction. Ideally, in positive-advantage rollouts, structurally important, answer-advancing steps have above-average effective contribution and thereby receive stronger updates, while redundant, weakly connected, or counterproductive steps are downweighted. In negative-advantage rollouts, answer-advancing steps are protected from excessive penalty, whereas structurally important, answer-degrading steps receive stronger penalties. The resulting RECAP advantages replace the original GRPO advantages in Eq. 1, and the complete procedure is summarized in Algorithm 1.

5 Experiments

Our evaluation addresses three research questions (RQs): RECAP's accuracy–efficiency trade-off (**RQ1**), the contributions of structural responsibility and step efficacy (**RQ2**), and the source of its token savings (**RQ3**). We first describe the experimental setup, then address these questions in turn.

Datasets. We randomly sample 2048 problems from DAPO-Math-17k (Yu et al., 2025) for training and evaluate on four mathmatical reasoning benchmarks of increasing difficulty: GSM8K (Cobbe et al., 2021) for simple grade school problems, MATH-500 (Lightman et al., 2024) for more challenging competition mathematics, and AIME 2024 (Zhang & Math-AI, 2024) and AIME 2025 (Zhang & Math-AI, 2025) for highly challenging competition problems.

Models. To test whether RECAP generalizes across different reasoning regimes, we evaluate two representative 7B models: Qwen2.5-Math-7B (Yang et al., 2024), a non-thinking model, and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025), a reasoning model.

Baselines. We restrict baselines to standard autoregressive reasoning methods for directly comparable token efficiency and exclude structure-aware methods whose primary objectives are sampling or attribution during training rather than inference efficiency, and discuss them separately in Related Work. We compare RECAP against eight baselines: the base model; a prompt-based baseline that encourages concise step-by-step reasoning (Renze & Guven, 2024); standard GRPO (Shao et al., 2024); GRPO with a simple length penalty; DDCA (Peng et al., 2026), which restricts length optimization to correct rollouts and adapts its strength by problem difficulty; LEASH (Li et al., 2026d), which dynamically adjusts a length penalty to enforce a target response length; BINGO (Liu et al., 2026b), which applies stronger efficiency pressure to less important tokens; and ThoughtFold (Liu et al., 2026c), which prunes redundant rollout segments, constructs preference pairs across compression levels, and combines preference optimization with GRPO.

Training & Evaluation Details. We use Claude Haiku 4.5 (Anthropic, 2025) for dependency annotation. The annotation is outcome-independent. The annotator sees neither the gold answer nor solution trajectories, and only labels dependencies among steps. Annotator selection and annotation quality are evaluated in Appendix A.3. For training, we sample 16 rollouts per prompt with temperature 1.0 and a maximum response length of 8192 tokens. We train on 2K prompts for four epochs (60 steps) with batch size 128, mini-batch size 1024, two policy-update epochs, and AdamW at a constant learning rate of 1×10^{-6}. RECAP modifies only advantage construction and all other training settings are shared with the baselines. We select checkpoints by greedy $pass@1$ on a held-out 512-prompt DAPO-Math split evaluated every 15 steps. Full details are provided in Appendix A.4. For evaluation, following DeepSeek-R1 (DeepSeek-AI, 2025), we sample $k = 8$ rollouts per prompt with temperature 0.6 and top-$p = 0.95$, and report $pass@1$, $pass@k$, and mean generated tokens.

5.1 Overall Accuracy–Efficiency Trade-off (RQ1)

We first examine whether RECAP improves reasoning efficiency without sacrificing task performance. Figure 2 summarizes the average accuracy–efficiency trade-off across the four benchmarks and Table 1 reports the results across the benchmarks.

Table 1: Sampled $pass@1$ (%) and mean completion tokens across four benchmarks. Grey ± values are the standard deviations over 8 runs. **Bold**/underline mark best and second best per block.

Method	GSM8K		MATH-500		AIME 2024		AIME 2025	
	pass@1↑	tokens↓	pass@1↑	tokens↓	pass@1↑	tokens↓	pass@1↑	tokens↓
Qwen2.5-Math-7B								
Base	49.1 ±1.2	1253 ±101	44.6 ±1.1	1684 ±75	11.2 ±5.6	2464 ±308	4.2 ±2.4	2206 ±736
Prompt-guided	57.0 ±0.8	746 ±62	49.3 ±1.7	1121 ±74	10.4 ±5.5	1892 ±362	2.1 ±2.5	1799 ±228
GRPO	84.4 ±0.4	354 ±6	72.2 ±1.0	641 ±24	24.2 ±3.9	1529 ±389	8.8 ±2.5	1405 ±262
Simple Length	82.2 ±0.5	308 ±3	70.2 ±1.3	534 ±40	26.2 ±5.8	**1333** ±440	6.2 ±4.9	**1037** ±251
DDCA	83.9 ±0.6	380 ±4	71.5 ±1.0	702 ±34	20.4 ±7.0	1883 ±218	7.9 ±1.7	1503 ±334
LEASH	83.4 ±0.9	373 ±4	72.1 ±1.2	654 ±28	26.2 ±7.2	1738 ±363	11.2 ±3.5	1400 ±312
Bingo	82.2 ±0.9	400 ±9	68.6 ±1.2	785 ±30	21.2 ±6.7	1789 ±513	8.3 ±4.4	1571 ±275
ThoughtFold	84.0 ±0.6	364 ±6	72.3 ±1.4	617 ±16	22.1 ±5.9	1404 ±271	11.2 ±1.7	1469 ±174
RECAP (ours)	**87.7** ±0.3	**246** ±3	**74.2** ±1.0	**524** ±16	**26.7** ±3.1	1401 ±211	**12.5** ±1.5	1098 ±182
DeepSeek-R1-Distill-Qwen-7B								
Base	87.4 ±0.6	463 ±4	84.7 ±1.0	2488 ±53	43.8 ±5.2	6628 ±226	27.9 ±3.5	6837 ±148
Prompt-guided	87.2 ±0.8	**430** ±2	79.9 ±1.2	1758 ±26	33.8 ±4.2	6204 ±153	28.7 ±4.3	6726 ±180
GRPO	90.1 ±0.5	518 ±4	82.3 ±1.6	**1347** ±23	39.2 ±8.1	**4856** ±279	25.4 ±6.4	**4432** ±365
Simple Length	87.7 ±0.6	447 ±2	80.4 ±0.8	1681 ±19	41.2 ±3.5	5828 ±233	29.6 ±3.3	6449 ±190
DDCA	91.9 ±0.9	670 ±9	88.3 ±0.9	1951 ±35	**45.8** ±7.1	5644 ±216	**32.1** ±4.3	5964 ±236
LEASH	92.0 ±0.6	666 ±12	**88.5** ±1.0	1983 ±42	45.0 ±6.9	5666 ±308	**32.1** ±4.7	5840 ±188
Bingo	87.7 ±0.4	459 ±3	85.7 ±0.5	2061 ±34	44.2 ±4.3	6208 ±255	31.3 ±1.7	6344 ±155
ThoughtFold	**92.1** ±0.5	657 ±5	87.9 ±0.7	1900 ±42	44.6 ±7.1	5802 ±274	29.6 ±4.9	5666 ±148
RECAP (ours)	**92.1** ±0.6	599 ±5	87.4 ±0.6	1513 ±29	44.2 ±6.4	4895 ±378	30.8 ±5.0	4659 ±297

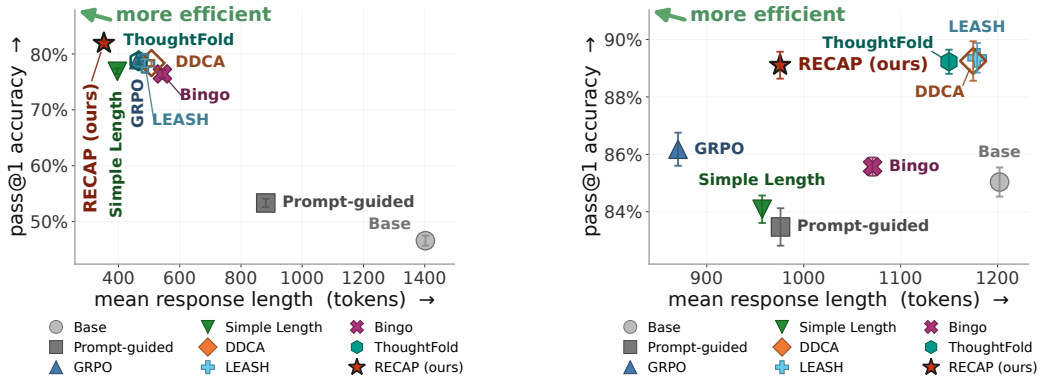
RECAP improves both accuracy and efficiency on Qwen2.5-Math-7B. As shown in Figure 2, RECAP lies on the upper-left frontier, achieving a favorable reduction in reasoning cost without sacrificing accuracy. Table 1 shows that this advantage is consistent across tasks. It achieves the highest $pass@1$ on all four benchmarks, while producing the shortest responses on GSM8K and MATH-500 and the second-shortest responses on AIME 2024 and AIME 2025. Compared with GRPO, RECAP reduces response length by 31%, 18%, 8%, and 22% on GSM8K, MATH-500, AIME 2024, and AIME 2025, respectively, while improving $pass@1$ by 3.3, 2.0, 2.5, and 3.7 percentage points. These results show a consistent shift toward higher accuracy at lower inference cost, rather than shorter reasoning obtained by sacrificing solution quality.

RECAP retains high accuracy at substantially lower cost on DeepSeek-R1-Distill-Qwen-7B. RECAP remains in the highest-accuracy region while using substantially fewer tokens than methods with comparable accuracy. With averages weighted by the number of evaluation examples in each benchmark, RECAP saves at least 170 tokens per rollout compared with each of DDCA, LEASH, and ThoughtFold. On the three harder benchmarks, it produces the second-shortest responses, behind only GRPO, while remaining close to the best $pass@1$. GRPO is shorter, but its $pass@1$ is lower by 2.0–5.4 percentage points across the four benchmarks. Overall, RECAP achieves the most favorable accuracy–efficiency trade-off among the compared methods.

5.2 Contributions of Structural Responsibility and Step Efficacy (RQ2)

Table 2: **Ablation of the two signals in RECAP's step score** on Qwen2.5-Math-7B across four benchmarks. All runs share $\beta = 0.3$, $\gamma_{context} = 0.5$, 16 rollouts per prompt and 60 steps.

	pass@1↑	pass@8↑	tokens↓
RECAP (full)	**81.90**	91.80	352
w/o structural responsibility	78.30	91.75	459
w/o step efficacy	81.32	**92.18**	370
Shuffled structural responsibility	79.10	91.86	**340**
Random structural responsibility	78.45	92.12	343



(a) Qwen2.5-Math-7B (b) DeepSeek-R1-Distill-Qwen-7B

Figure 2: Accuracy–efficiency trade-off across models.

We examine the contributions of RECAP's two signals by ablating either structural responsibility or step efficacy. To test whether structural responsibility benefits from its semantic alignment rather than merely non-uniform weighting, we additionally replace it with shuffled or random credit. Shuffling preserves the responsibility-score distribution within each rollout while breaking its correspondence to reasoning steps. Table 2 reports the results.

Structural responsibility drives the accuracy–efficiency gain. Removing structural responsibility decreases $pass@1$ from 81.90 to 78.30 while increasing the average response length from 352 to 459 tokens. The simultaneous degradation in accuracy and efficiency indicates that step efficacy alone cannot reliably identify the reasoning steps that should receive stronger updates. Structural responsibility is essential for concentrating updates on consequential steps, allowing RECAP to generate shorter reasoning traces without sacrificing accuracy.

Step efficacy further calibrates structural credit. Removing step efficacy reduces $pass@1$ by only 0.58 points but increases mean response length by $\sim 5\%$. Structural credit captures downstream dependence but not whether a step advances the correct solution. Step efficacy adds an answer-directed calibration, making credit assignment more selective and reducing unnecessary reasoning.

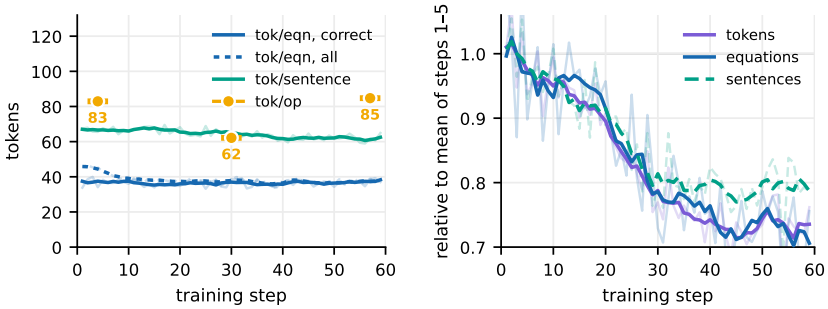
Semantic dependency enables effective credit assignment. Replacing structural responsibility with shuffled or random credit slightly shortens the average response, but reduces $pass@1$ by 2.80 and 3.45 points, respectively. The shuffled variant preserves the responsibility-score distribution while breaking its alignment with reasoning steps, showing that semantic alignment is critical for preserving accuracy rather than merely inducing non-uniform reweighting. Random credit provides complementary evidence that arbitrary non-uniform weighting is insufficient. These results demonstrate the importance of semantic dependency modeling for effective credit assignment.

Overall, structural responsibility is the main source of RECAP's accuracy–efficiency improvement, while step efficacy acts as a complementary calibration signal. Their combination yields the strongest $pass@1$ performance at nearly the shortest response length.

5.3 Where Do RECAP's Token Savings Come From? (RQ3)

We next investigate whether RECAP saves tokens by expressing the same reasoning more compactly or by removing unnecessary computation. We measure how the number and token cost of reasoning units change during training, using equations and sentences as automatic proxies and manually annotating atomic reasoning operations on a subset of rollouts. Atomic operations are minimal reasoning steps that change the solution state or introduce new information used in the derivation, including setup, theorem recall, equation derivation, substitution, numerical computation, case analysis, verification, and conclusion. Detailed annotation criteria are provided in Appendix A.4.5.

RECAP performs fewer reasoning operations rather than compressing them. Figure 3a shows that the token cost per reasoning unit remains largely stable during training. On correct rollouts, tokens per equation change only from 36.4 to 36.8, with manually annotated operations showing the same pattern. In contrast, Figure 3b shows that total tokens fall by 23.7%, alongside reductions of 24.5% in equations and 17.3% in sentences. Together, these results suggest that RECAP shortens responses primarily by performing fewer reasoning operations rather than expressing the same operations with fewer tokens.



(a) Token cost per reasoning operation. (b) Evolution of reasoning unit counts.

Figure 3: **Training dynamics of reasoning content and token usage.** (a) Tokens per reasoning unit, measured by equations, sentences, and manually annotated atomic operations. (b) Evolution of response tokens, equations, and sentences per rollout, normalized to their early-training values.

Table 3: **Dead-end reasoning in Qwen2.5-Math-7B test rollouts** (200 sampled prompts each from GSM8K and MATH500, all AIME prompts, and 8 rollouts per prompt for each method). A step is a *dead end* if it has no directed path to the final-answer node F, and *isolated* if its total degree is zero. **step %** and **tok %** are macro-averaged per-rollout fractions by step count and token count, respectively. All three annotators label the same rollouts under the same segmentation. All standard errors are at most 0.6 pp and are omitted. Best (lowest) value per column in **bold**.

	Dead end (no path to F)						Isolated (degree 0)						
	step % ↓			tok % ↓			step % ↓			tok % ↓			mean
Method	Haiku	Opus	GPT	Haiku	Opus	GPT	Haiku	Opus	GPT	Haiku	Opus	GPT	steps
Base	26.4	29.1	23.7	22.9	25.8	21.6	7.6	15.3	10.6	6.8	13.8	9.7	12.69
Prompt-guided	25.0	25.4	21.4	21.8	22.2	19.6	8.8	13.2	10.2	8.0	12.0	9.4	11.21
GRPO	25.3	19.5	20.1	17.2	11.1	12.6	10.0	11.6	13.4	4.0	4.7	7.0	9.25
Simple Length	13.6	9.6	9.7	10.4	6.4	7.3	2.8	4.0	5.4	1.4	2.2	3.5	7.49
DDCA	20.7	12.1	13.1	16.7	8.4	10.0	1.9	3.9	5.1	1.0	1.6	3.0	10.62
LEASH	19.7	12.3	12.7	15.7	8.2	9.7	**1.6**	3.4	4.9	**0.8**	1.2	2.9	10.04
Bingo	22.0	15.4	14.5	17.0	10.6	11.3	3.1	4.4	5.3	1.7	2.0	3.4	10.96
ThoughtFold	17.8	14.3	14.5	12.8	8.7	10.2	1.7	4.0	5.9	**0.8**	1.3	3.3	9.39
RECAP (ours)	**5.6**	**5.7**	**5.4**	**3.9**	**3.3**	**4.0**	**1.6**	**1.7**	**3.5**	**0.8**	**0.8**	**2.1**	**4.29**

RECAP substantially reduces dead-end reasoning. We next measure the fraction of semantic steps with no directed path to the final answer, reported in Table 3. Across three independent annotators, RECAP consistently has the lowest dead-end rate. Only 5.4–5.7% of its steps are dead ends, compared with 9.6–13.6% for Simple Length and 19.5–25.3% for GRPO. The same pattern holds when weighting by inference tokens, with only 3.3–4.0% of RECAP's tokens spent on dead-end steps, versus 6.4–10.4% for Simple Length and 11.1–17.2% for GRPO. RECAP also yields the lowest or tied-lowest incidence of isolated steps across annotators. Complementary topology metrics in Appendix Table 15 show the same pattern, with RECAP reducing both multiple-sink DAGs (ML) and isolated steps (ISO) across all three annotators.

Taken together, these results suggest that RECAP's efficiency gains reflect a structural simplification of reasoning. The model performs fewer reasoning operations, with a smaller fraction devoted to dead-end branches, rather than merely expressing the same computation with fewer tokens.

6 Conclusion

We introduced RECAP, a redundancy-aware credit assignment method that uses semantic dependencies and step-level reasoning progress to reshape GRPO updates for more efficient reasoning. Across two 7B models and four mathematical reasoning benchmarks, RECAP improves the accuracy–efficiency trade-off over trajectory-level, compression-based, and fine-grained baselines. Our analysis suggests that these gains come from reducing unnecessary reasoning computation rather than merely expressing the same reasoning more compactly, with substantially less inference budget spent on dead-end reasoning.

AI USE STATEMENT

Generative AI tools were used to assist with coding and debugging, including writing scripts for data analysis and plotting figures, literature search for related work, and refining the manuscript. All AI-assisted outputs were reviewed by the authors. The authors take the responsibility for the final content of this work.

ETHICS STATEMENT

This work studies reinforcement learning methods for improving the efficiency of large language model reasoning. We use publicly available datasets and models in accordance with their applicable licenses and terms of use, and the work does not involve human subjects, private or sensitive data, or safety-critical applications.

REPRODUCIBILITY STATEMENT

We provide the full RECAP algorithm and training objective in Section 4 and Algorithm 1. We report the model, dataset, optimization, sampling, and evaluation settings in Section 5. Additional implementation details, annotation procedures, and analysis protocols are documented in Appendix A.4, including the semantic dependency annotation scheme, hyperparameters, and prompt templates.

REFERENCES

Anthropic. Claude haiku 4.5 system card, October 2025. URL https://www-cdn.anthropic.com/7aad69bf12627d42234e01ee7c36305dc2f6a970.pdf.

Anthropic. Claude opus 4.8 system card, May 2026. URL https://www-cdn.anthropic.com/0f0c97ad20d8005706296bd92aa1c27c6b2f4f61/Claude%20Opus%204.8%20System%20Card.pdf.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 60770–60808. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/579b5b84311e122584dedab1d3b7613f-Paper-Conference.pdf.

Benteng Chen, Weida Wang, Shufei Zhang, Mingbao Lin, and Min Zhang. Step-GRPO: Internalizing dynamic early exit for efficient reasoning. 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. 21710–21724, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.990. URL https://aclanthology.org/2026.acl-long.990/.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? On the overthinking of long reasoning models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9487–9499. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/chen25bx.html.

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.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Zhichen Dong, Yang Li, Yuhan Sun, Weixun Wang, Yijia Luo, Zinian Peng, Wenbo Su, Cheng Yu, Bo Zheng, and Junchi Yan. How does reasoning flow? tracing attention-induced information flow for targeted RL in LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Ym4KYa1n76.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *The Innovation*, 7(6), 2026.

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.

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.

Yinhan He, Yaochen Zhu, Mingjia Shi, Wendy Zheng, Lin Su, Xiaoqing Wang, Qi Guo, and Jundong Li. IAPO: Information-aware policy optimization for token-efficient reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=S21HtCHa40.

Chen Huang, Wei Lu, and Wenxuan Zhang. PEAR: Phase entropy aware reward for efficient reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HLc2igXEA3.

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

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZpQwAFhU13.

Yuxuan Jiang, Dawei Li, and Francis Ferraro. DRP: Distilled reasoning pruning with mathematical skill-aware step decomposition for efficient large reasoning models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 4020–4039, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.196. URL https://aclanthology.org/2026.findings-acl.196/.

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do LLMs compress their own chain-of-thought? a token complexity approach. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. URL https://openreview.net/forum?id=uj5u4o5xjT.

Gang Li, Yan Chen, Ming Lin, and Tianbao Yang. DRPO: Efficient reasoning via decoupled reward policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=GP5RHZnEsw.

Jiawei Li, Yang Gao, Huashan Sun, and Chong Feng. Think better, not longer: Token-level marginal utility for efficient reasoning in large reasoning models. 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. 30052–30063, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1386. URL https://aclanthology.org/2026.acl-long.1386/.

Xintong Li, Sha Li, Rongmei Lin, Hongye Jin, Linwei Li, Hejie Cui, Sarah Zhang, Chia-Yuan Chang, Kewei Cheng, Besnik Fetahu, et al. Stepwise penalization for length-efficient chain-of-thought reasoning. *arXiv preprint arXiv:2603.00296*, 2026c.

Yanhao Li, Lu Ma, Jiaran Zhang, Lexiang Tang, Wentao Zhang, and Guibo Luo. LEASH: Adaptive length penalty and reward shaping for efficient large reasoning model. 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. 2846–2856, San Diego, California, United States, July 2026d. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.129. URL https://aclanthology.org/2026.acl-long.129/.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Han Liu, Shuotian Ma, Hui Li, Xiaotong Zhang, Fenglong Ma, and Hong Yu. Pru-CoT: Towards efficient reasoning distillation via pruning chain-of-thought. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 33744–33756, San Diego, California, United States, July 2026a. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1684. URL https://aclanthology.org/2026.findings-acl.1684/.

Hanbing Liu, Lang Cao, Yuanyi Ren, Mengyu Zhou, Haoyu Dong, Xiaojun Ma, Shi Han, and Dongmei Zhang. Not all tokens matter: Towards efficient LLM reasoning via token significance in reinforcement learning. 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. 15989–16016, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.726. URL https://aclanthology.org/2026.acl-long.726/.

Ziyan Liu, Xueda Shen, Yuzhe Gu, Songyang Gao, Kuikun Liu, Guangran Cheng, Chengqi Lyu, Dahua Lin, Wenwei Zhang, and Kai Chen. Thoughtfold: Folding reasoning chains via introspective preference learning. In *Forty-third International Conference on Machine Learning*, 2026c. URL https://openreview.net/forum?id=qeeD6aZLLX.

Zi-Ao Ma, Xian-Ling Mao, Tian Lan, Chen Xu, and Zhijing Wu. Your reasoning model knows what counts: Self-guided chain-of-thought pruning for efficient reasoning. 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. 586–605, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.25. URL https://aclanthology.org/2026.acl-long.25/.

Shuaiyi Nie, Dingsiyu, Wenyuan Zhang, Linhao Yu, Tianmeng Yang, Yao Chen, Weichong Yin, Yu Sun, Hua Wu, and Tingwen Liu. AttnPO: Attention-guided process supervision for efficient reasoning. 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. 39728–39748, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1845. URL https://aclanthology.org/2026.acl-long.1845/.

OpenAI. Gpt-5.5 system card, April 2026. URL https://openai.com/index/gpt-5-5-system-card/.

Keqin Peng, Yuanxin Ouyang, Xuebo Liu, Zhiliang Tian, Ruijian Han, Yancheng Yuan, and Liang Ding. Think dense, not long: Dynamic decoupled conditional advantage for efficient reasoning. *arXiv preprint arXiv:2602.02099*, 2026.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 476–483, 2024. doi: 10.1109/FLLM63129.2024.10852493.

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.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=HvoG8SxggZ.

Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. Stop when enough: Adaptive early-stopping for chain-of-thought reasoning. 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. 27250–27268, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1256. URL https://aclanthology.org/2026.acl-long.1256/.

Siao Tang, Xinyin Ma, Gongfan Fang, and Xinchao Wang. Concisehint: Boosting efficient reasoning via continuous concise hints during generation. *arXiv preprint arXiv:2506.18810*, 2025.

Han Wang, Xiaodong Yu, Jialian Wu, Jiang Liu, Ximeng Sun, Mohit Bansal, and Zicheng Liu. Stabilizing efficient reasoning with step-level advantage selection. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26754–26765, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1333. URL https://aclanthology.org/2026.findings-acl.1333/.

Yang Xiang, Yixin Ji, Ruotao Xu, Dan Qiao, Zheming Yang, Juntao Li, and Min Zhang. When is thinking enough? early exit via sufficiency assessment for efficient reasoning. 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. 23541–23556, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1080. URL https://aclanthology.org/2026.acl-long.1080/.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Hongyuan Yuan, Xinran He, Run Shao, Bolei He, Xianwei Xue, Mengke Chen, Qiutong Pan, Haiwei Wang, and Haifeng Li. Graph-based chain-of-thought pruning for reducing redundant reflections in reasoning LLMs. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5708–5723, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.281. URL https://aclanthology.org/2026.findings-acl.281/.

Linan Yue, Yichao Du, Yizhi Wang, Weibo Gao, Fangzhou Yao, Li Wang, Ye Liu, Ziyu Xu, Qi Liu, Shimin Di, et al. Don't overthink it: A survey of efficient r1-style large reasoning models. *arXiv preprint arXiv:2508.02120*, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yuliang Zhan, Xinyu Tang, Jian Li, Dandan Zheng, Weilong Chai, Jingdong Chen, Jun Zhou, Ge Wu, Wenyue Tang, and Hao Sun. Graphpo: Graph-based policy optimization for reasoning models. *arXiv preprint arXiv:2606.18954*, 2026.

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.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Appendix

A.1 Related Work

Token-efficient reasoning involves two distinct objectives: stopping once sufficient reasoning has been completed and reducing redundancy within the reasoning process itself. The former is addressed by inference-time or learned early-stopping methods (Chen et al., 2026; Sun et al., 2026; Xiang et al., 2026). We focus on the latter objective that is to produce more compact reasoning traces without compromising solution quality (Li et al., 2026b). Prompt-based methods encourage concise reasoning through explicit instructions, output constraints, or adaptive hints (Renze & Guven, 2024; Lee et al., 2025; Tang et al., 2025), but provide little guidance on which reasoning steps should be

preserved. Post-training methods can provide more targeted control. Existing post-training methods for improving reasoning efficiency can be broadly grouped into length-regularized policy optimization, pruning and distillation, fine-grained credit assignment, and structure-aware attribution.

Length-Regularized Policy Optimization. Length-regularized methods incorporate response length into rollout-level optimization, using fixed or adaptive penalties conditioned on correctness, problem difficulty, or training dynamics (Arora & Zanette, 2025; Li et al., 2026a; Peng et al., 2026; Li et al., 2026d). While effective at shortening responses, these methods assign efficiency signals at the trajectory level and therefore cannot identify which individual reasoning steps are redundant.

Pruning and Distillation of Reasoning Traces. Some methods construct concise training trajectories by pruning redundant content using teacher guidance, model-based importance estimates, or dependency analysis (Jiang et al., 2026; Liu et al., 2026a; Ma et al., 2026; Yuan et al., 2026). ThoughtFold further constructs sub-trajectories at different compression levels and uses masked preference optimization to encourage the model to bridge retained reasoning segments (Liu et al., 2026c). These methods learn from preconstructed compressed trajectories rather than assigning step-specific credit within newly sampled rollouts, potentially limiting generalization to unseen redundancy patterns. They may still retain unnecessary steps or overcompress useful reasoning.

Fine-Grained Credit Assignment. Fine-grained methods assign token- or step-level signals using answer likelihood, confidence, conditional mutual information, or attention (Li et al., 2026b; Wang et al., 2026; He et al., 2026; Nie et al., 2026). BINGO estimates token significance and applies stronger efficiency pressure to less important tokens (Liu et al., 2026b), while SwAP uses changes in correct-answer likelihood to redistribute excess-length penalties across steps (Li et al., 2026c). Although more selective than trajectory-level objectives, these methods primarily estimate the utility of each unit in isolation and do not capture whether a step supports downstream deductions, provides context needed later, or merely repeats existing information. As a result, they may undercredit structurally important steps while overcrediting locally plausible but redundant ones.

Structure-Aware Attribution. Several methods introduce tree- or graph-structured representations, primarily for efficient rollout sampling or finer-grained credit assignment rather than token efficiency. Tree-based methods organize trajectories through shared prefixes, while FlowTracer derives token-level credit from attention-induced information flow (Ji et al., 2026; Dong et al., 2026). GraphPO merges semantically equivalent states across rollouts and rewards shorter paths to shared states (Zhan et al., 2026). These methods capture shared rollout structure or model-internal information flow, rather than semantic dependencies among reasoning steps within a single derivation, and still rely on path length for token efficiency. RECAP instead directly constructs a semantic dependency graph over the reasoning steps of each rollout and uses it to attribute each step's contribution to the downstream derivation, allowing policy updates to reinforce useful steps more strongly while downweighting redundant ones.

Latent Reasoning. Latent-reasoning methods reduce explicit chain-of-thought generation by performing intermediate computation in continuous hidden states or recurrent depth (Hao et al., 2025; Zhu et al., 2025). This changes the inference interface. Fewer generated tokens can correspond to additional latent model computation, making token counts not directly comparable with standard autoregressive reasoning. Moreover, existing approaches require specialized training or architectures rather than operating as drop-in post-training methods for the same reasoning model. We focus on methods that improve token efficiency under the standard autoregressive reasoning interface, where reductions in generated reasoning tokens are directly comparable.

A.2 RECAP Algorithm

Algorithm 1 summarizes the complete RECAP training procedure, including dependency annotation, structural-responsibility propagation, step-efficacy computation, and advantage reshaping.

Algorithm 1 RECAP Step-Level Credit Assignment

Require: Prompt q; rollout group $\{\tau_k\}_{k=1}^{G}$; gold answer $a^\star = (a_1^\star, \ldots, a_T^\star)$; rollout-level advantages $\{A_k^{\mathrm{GRPO}}\}_{k=1}^{G}$; hyperparameters $\alpha, \beta, c, \epsilon$; policy model θ; annotator model ϕ
Ensure: Token-level advantages for all rollouts

for $k = 1, \ldots, G$ **do**
  Segment τ_k into semantic steps $\tau_k = (s_{k,1}, \ldots, s_{k,N_k})$
  $\mathcal{G}_k \leftarrow \textsc{BuildDependencyDAG}(\tau_k, \phi)$
  $\{\mathrm{resp}(s_{k,i})\}_{i=1}^{N_k} \leftarrow \textsc{PropagateResponsibility}(\mathcal{G}_k))$
  Compute the gold-answer log-likelihood before the first step:

$$\mathcal{L}_{k,0} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta\left(a_t^\star \mid q, a_{<t}^\star\right)$$

  for $i = 1, \ldots, N_k$ **do**
    Compute the prefix-conditioned gold-answer log-likelihood:

$$\mathcal{L}_{k,i} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta(a_t^\star \mid q, \tau_{k,\le i}, a_{<t}^\star)$$

    $\Delta_{k,i} \leftarrow \mathcal{L}_{k,i} - \mathcal{L}_{k,i-1}$
    $\widehat{\Delta}_{k,i} \leftarrow \mathrm{clip}(\Delta_{k,i}, -c, c)$
    Compute the effective contribution score:

$$M_{k,i} \leftarrow \mathrm{resp}(s_{k,i}) \exp(\alpha \ \mathrm{sign}\left(A_k^{\mathrm{GRPO}}\right) \widehat{\Delta}_{k,i})$$

  end for
  Let $d_{k,i}$ be the number of tokens in $s_{k,i}$, and compute

$$D_k \leftarrow \sum_{i=1}^{N_k} d_{k,i}, \qquad \overline{M}_k \leftarrow \frac{1}{D_k}\sum_{i=1}^{N_k} d_{k,i} M_{k,i}$$

  for $i = 1, \ldots, N_k$ **do**
    $w_{k,i} \leftarrow M_{k,i}/(\overline{M}_k + \epsilon)$
    Compute the step-level advantage:

$$A_{k,i}^{\mathrm{RECAP}} \leftarrow [(1-\beta) + \beta w_{k,i}] A_k^{\mathrm{GRPO}}$$

    Assign $A_{k,i}^{\mathrm{RECAP}}$ to every token in $s_{k,i}$
  end for
end for
return token-level advantages

A.3 Annotation Quality

RECAP relies on an annotator to construct the dependency graph that determines its structural credit. Annotation errors may therefore distort the credit assigned to individual reasoning steps and ultimately affect policy optimization. To assess whether the structural signal is sufficiently reliable, we evaluate candidate annotators along two dimensions: annotation validity, which measures compliance with the required graph format, and annotation correctness, which measures the quality of the predicted dependencies. Because exact step dependencies lack ground-truth annotations, we estimate correctness through agreement with independent judges.

Annotation validity. We re-annotate 800 fixed rollouts generated by Qwen2.5-Math-1.5B-Instruct using Qwen3 models ranging from 0.6B to 32B, under both thinking and non-thinking settings, as well as Claude Haiku 4.5. We define the annotation validity rate as the fraction of outputs that can be parsed into fully typed, acyclic dependency graphs satisfying all format constraints.

Figure 4 that validity improves non-linearly with model scale. Among the Qwen3 models, the validity rate increases from 9.8% at 0.6B to 63.8% at both 1.7B and 4B, 80.5% at 8B, and 99.9% at

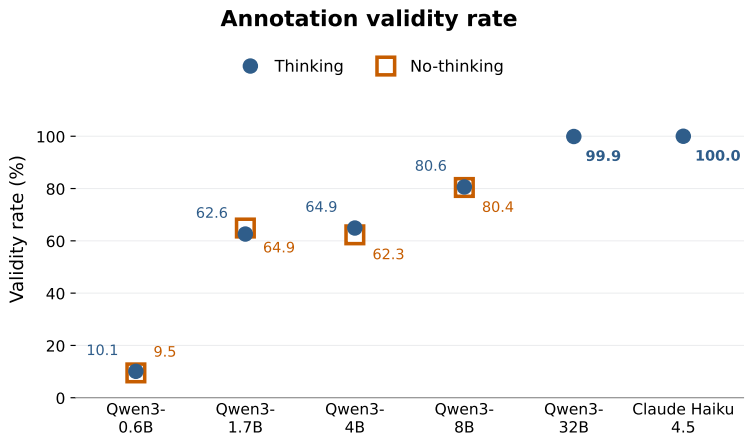


Figure 4: **Annotation validity by annotator.** Fraction of rollouts whose dependency annotations parse into a valid, fully-typed, acyclic DAG, over 800 frozen rollouts re-annotated by each of six candidate annotators. Tick marks on the Qwen3 bars show the thinking/no-thinking range ($\leq$ 2.5 points). Only Qwen3-32B and Claude Haiku 4.5 achieve annotation validity rates above 99%.

32B. Switching between thinking and non-thinking modes changes validity by at most 2.5 percentage points at any model size. Claude Haiku 4.5 achieves a validity rate of 100.0%, comparable to Qwen3-32B. Since only Qwen3-32B and Claude Haiku 4.5 produce valid graphs nearly universally, we retain these two models for the subsequent annotation-correctness evaluation.

Annotation accuracy, Qwen3-32B vs. Claude Haiku 4.5 Format compliance only verifies that an annotator produces a valid graph. It does not establish whether the predicted dependencies are correct. Therefore, we evaluate the two finalists from the validity study, Qwen3-32B and Claude Haiku 4.5, against the consensus of three independent judges (Claude Opus 4.8 (Anthropic, 2026), GPT-5.5 (OpenAI, 2026), and GLM-5 (Zeng et al., 2026)). These judges exhibit pairwise agreement rates of 75–85% on this task, on 800 rollouts sampled from the DAPO training set, comprising 16 fixed rollouts for each of 50 prompts.

We evaluate step-to-step support, context, and restatement edges, as well as dependencies from reasoning steps to the final answer. We report pooled edge accuracy and edge-type F1 scores. Pooled accuracy is the fraction of decidable edge candidates for which the annotator's binary prediction agrees with the judge consensus, aggregated across all dependency types. The aggregation weights each type by its frequency. Here, a decidable cell is one where the judges reach consensus and the test model itself parses the relevant step successfully. Here, an edge candidate is considered decidable when the judges reach consensus and the candidate annotator successfully parses the relevant steps. F1 score for each dependency type, is the harmonic mean of their precision and recall, where precision measures the fraction of predicted edges supported by the judge consensus and recall measures the fraction of consensus edges recovered by the annotator. Table 4 shows that Claude Haiku 4.5 outperforms Qwen3-32B on every metric, with a 35-point advantage in pooled accuracy.

The validity and agreement evaluations show that Claude Haiku 4.5 matches Qwen3-32B in format reliability while agreeing substantially more often with the independent judge panel. We therefore use Claude Haiku 4.5 as the annotator for RECAP.

Table 4: **Claude Haiku 4.5 agrees with the judge panel far more than Qwen3-32B, on the same rollouts.** Pooled accuracy and per-type F1 against the $\geq$ 2-of-3 judge consensus (Claude Opus 4.8, GPT-5.5, GLM-5), 800 rollouts (50 prompts $\times$ 16 rollouts).

Model	Pooled accuracy %	Support F1	Context F1	Restate F1
Qwen3-32B	41.9	62.8	13.3	12.3
Claude Haiku 4.5	**77.0**	**83.0**	**35.4**	**51.3**

Table 5: Ablation of the annotator models under fixed hyperparameter settings: $\beta = 0.3, \alpha = 1, \gamma_{context} = 0.5, \tau_{\Delta} = 2, \tau_w = 5$.

Annotator	GSM8K		MATH-500		AIME 2024		AIME 2025	
	pass@1↑	tokens↓	pass@1↑	tokens↓	pass@1↑	tokens↓	pass@1↑	tokens↓
Haiku 4.5	**84.6**	**237**	**73.2**	**509**	**25.8**	**1401**	**11.2**	**1135**
Qwen3-32B	82.2	259	69.2	512	24.6	1421	8.7	1222

Table 6: Annotator models on the same four benchmarks, scored with the identical prompt, answer extractor and grader as the trained-checkpoint tables.(p@1=pass@1, p@8=pass@8)

Model	GSM8K			MATH-500			AIME 2024			AIME 2025		
	p@1↑	p@8↑	tok↓	p@1↑	p@8↑	tok↓	p@1↑	p@8↑	tok↓	p@1↑	p@8↑	tok↓
Haiku 4.5	**95.8** ± 0.3	**98.0**	473	**92.6** ± 0.5	**96.2**	1594	**65.8** ± 3.0	**83.3**	3909	**46.7** ± 5.4	**70.0**	4096
Qwen3-32B	94.3 ± 0.4	97.7	**273**	82.1 ± 0.7	92.4	**834**	30.0 ± 4.7	46.7	**2601**	19.2 ± 6.4	40.0	**2563**

Effect of annotator choice. Beyond annotation accuracy, we further examine whether the choice of annotator affects downstream training outcomes with all other hyperparameters fixed. Note that this setting was not fully tuned and differs from the one used in Table 1. As shown in Table 5, RECAP with Haiku 4.5 achieves higher $pass@1$ and fewer tokens than Qwen3-32B on all four benchmarks. This suggests that annotation quality translates into more effective credit assignment and supports our use of Haiku 4.5 in the main experiments. Since we do not retune hyperparameters for each annotator, the comparison reflects performance under a common configuration rather than each annotator's best achievable result.

Reasoning capability of annotator models. We also evaluate the annotator models themselves on the same four reasoning benchmarks, as shown in Table 6. Haiku 4.5 consistently achieves higher $pass@1$ and $pass@8$ than Qwen3-32B, with particularly large gaps on MATH-500 and AIME. This suggests stronger mathematical reasoning capability for interpreting the target-domain traces. Since dependency annotation is answer-blind and provides neither gold answers nor solution trajectories, this capability is used to assess relations among policy-generated steps rather than to provide task-level supervision.

A.4 Training Details.

A.4.1 Setup

During training, we set the sampling temperature to 1.0 and sample $n = 16$ rollouts per prompt, with a maximum response length of 8192 tokens. We use a training batch size of 128 prompts and a mini-batch size of 1024 rollouts, with two policy-update epochs per training step, and train for four passes over the prompt pool. Filtering prompts longer than 1024 tokens removes exactly one of the 2048 training prompts, leaving the pool just below 16 full batches. With `drop_last`, this results in 15 training steps per epoch and 60 training steps in total, with the remaining 127 prompts discarded at the end of each epoch. The actor is optimized with AdamW using a constant learning rate of 1×10^{-6} without warmup, gradient clipping at 1.0, a PPO clip ratio of 0.2, and no entropy bonus. Following GRPO, the KL regularizer is applied directly to the policy loss rather than incorporated into the reward. We use the low-variance k_3 estimator with coefficient 10^{-3} against a frozen reference policy. Rollouts are generated with vLLM using dynamic batching, and a single random seed (42) is used for the dataloader, vLLM engine, and sampler.

Advantage shaping introduces no additional optimizer hyperparameters. RECAP modifies only the advantage-construction stage, so the optimizer, rollout configuration, and all other training settings are kept identical to those of the corresponding GRPO and controlled baselines. For structural responsibility, only the relative ratios among the edge-type coefficients matter during propagation. We therefore fix $\gamma_{\text{support}} = 1$ and $\gamma_{\text{restate}} = 0$, and tune only γ_{context}. Support edges form the main dependency backbone and receive full edge weight, whereas restatement edges introduce no new

Table 7: **Hyperparameter Study of** $\gamma_{context} : \gamma_{support}$ on DeepSeek-R1-Distill-Qwen-7B. We fixed $\gamma_{support} = 1$ and adjust $\gamma_{context}$ to change its ratio.

$\gamma_{context}$	**0.33**	**0.50**	**0.75**	**1.00**
Accuracy	0.6725	0.6451	0.6804	0.6804

Table 8: Offline distribution of the signed step efficacy Δ on the two backbones, measured with the production scorer over 400 rollouts per model. Δ is a per-step log-probability difference in nats per gold token, i.e. an *absolute* quantity whose scale is model-dependent. neg. denotes the fraction of steps with negative efficacy.

		Δ distribution								
		$\lvert\Delta\rvert$ quantiles					signed Δ		clip	
Backbone	steps/roll.	med	p90	p95	p99	max	mean	neg.(%)	τ_Δ	%
Qwen2.5-Math-7B	18.5	0.363	1.996	2.909	5.590	13.9	-0.022	51.5	2.0	10.0
R1-Distill-7B	36.5	0.602	2.893	3.988	6.519	14.2	$+0.023$	49.5	4.0	5.0

reasoning contribution and therefore receive zero weight. We tune γ_{context} separately for each model on the validation set. Table 7 shows the hyperparameter study for DeepSeek-R1-Distill-Qwen-7B as an example. The selected values are $\gamma_{\text{context}} = 0.5$ for Qwen2.5-Math-7B and $\gamma_{\text{context}} = 0.75$ for DeepSeek-R1-Distill-Qwen-7B.

The remaining hyperparameters control the strength and stability of step-level reshaping. We clip step efficacy at $|\Delta| \leq \tau_\Delta$, with model-specific thresholds chosen from the offline efficacy statistics in Table 8. Qwen2.5-Math-7B typically produces fewer than 20 semantic steps per trace, so we set $\tau_\Delta = 2$, approximately the 90th percentile of its offline distribution. Given the typical trace length, this clips only about two steps per trace. DeepSeek-R1-Distill-Qwen-7B produces substantially longer traces. After merging its traces to a target of at most 50 semantic steps, we set $\tau_\Delta = 4$, which is near the 95th percentile of its offline distributions, again limiting clipping to only a small number of steps per trace (~ 2 steps). We further clip the normalized step weight at $w \leq \tau_w$, using $\tau_w = 5$ for Qwen2.5-Math-7B and $\tau_w = 10$ for DeepSeek-R1-Distill-Qwen-7B. These thresholds are chosen near the 99th percentile of the corresponding offline w distributions in Table 9, preventing a small number of extreme weights from dominating the update while leaving the vast majority of steps unaffected.

The efficacy scaling coefficient α and residual-mixing coefficient β are selected through hyperparameter search on the validation set. We use $\alpha = 0.5$ for Qwen2.5-Math-7B and $\alpha = 1.0$ for DeepSeek-R1-Distill-Qwen-7B, with $\beta = 0.3$ for both models. In each case, we select the configuration with the highest validation accuracy. For brevity, Tables 10 and 11 report the sensitivity to α and β on Qwen2.5-Math-7B. DeepSeek-R1-Distill-Qwen-7B is tuned using the same validation protocol.

We evaluate the policy every 15 training steps on the held-out 512-prompt DAPO-Math validation split using greedy decoding. As with the training set, we filter out prompts longer than 1024 tokens, and this removed two of the 512 validation prompts. The reported checkpoint is selected by the highest validation $pass@1$ and is frozen before any test-set evaluation. Training rollouts are sampled at temperature 1.0 to maintain sufficient within-group diversity for informative relative advantages.

For the annotator, we set $temperature = 0$, $maxTokens = 4096$, and $enable_thinking = False$.

The test datasets and their sizes are reported in Appendix Table 12.

A.4.2 STEP SPLITTING

In our experiments, each rollout is split into steps at blank lines ($\backslash n\backslash n$). For Qwen2.5-Math-7B, this raw $\backslash n\backslash n$ split is used as is: it yields $\sim$16 steps of $\sim$77 tokens each, a granularity that matches what a step intuitively means. The two long-thinking models break this assumption. On 128 rollouts generated at training sampling parameters, DeepSeek-R1-Distill-Qwen-7B's raw split yields a me-

Table 9: Distribution of the per-step multiplier M and of the normalised weight $w = \mathrm{clip}\left(\frac{M}{M+\epsilon},\, 0,\, \tau_w\right)$. w_{raw} is the same quantity before the τ_w clip, so the gap between w and w_{raw} at p99 and max is exactly what clipping removes.

	Qwen2.5-Math-7B	R1-Distill-7B
τ_Δ / τ_w / α	2.0 / 5 / 0.5	4.0 / 10 / 1.0
β	0.3	0.3
$resp$ median	1.0000	0.3869
$resp$ zero fraction	12.9%	**27.0%**
$\lvert\Delta\rvert$ median (in training)	0.4928	0.5817
M median	0.8361	0.2462
w median	**0.7757**	0.1195
w mean	0.8955	0.6572
w p90	1.8057	1.5245
w p95	2.3394	3.1872
w p99	3.9964	9.7834
w max	5.0000	10.0000
w_{raw} p90	1.81	1.52
w_{raw} p95	2.34	3.19
w_{raw} p99	4.00	9.96
w_{raw} max	**161.8**	61.4
median per-rollout max w	1.533	6.373
token-weighted mean w	**1.0000**	0.7490
τ_w trigger rate	**0.59%**	0.99%
τ_Δ trigger rate	**16.83%**	5.25%

Table 10: **Hyperparameter Study of** α under fixed hyperparameters: $\beta = 0.3, \gamma_{context} = 0.5$.

α	**0.3**	**0.5**	**0.8**	**1.0**
Accuracy	0.4510	0.5000	0.4667	0.4824

dian of 174 fragments per rollout at a median of only 22 tokens each (65.1% are ≤ 30 tokens). The maximum number of the fragments in a rollout can reach 447. Since the RECAP annotator emits $\sim$18 output tokens of dependency declarations per step, a 174-step trace exhausts its output budget (4096), its dependency graph fails to parse ($\sim$29%), and the rollout contributes *no* structural signal at all.

We address this by introducing a per-rollout *step target*. Rollouts at or below the target keep the raw split and rollouts above it have adjacent short steps merged until they fit. We use two experiments set a propriate target: an offline screen over many targets to find where annotation stops being reliable, and training runs at a coarse grid to choose among the targets the screen leaves undecided.

Experiment 1: Offline Annotation Screen We use Claude Haiku 4.5 to annotate the 128 fixed rollouts processed with different target step. The same 128 frozen rollouts areannotated at 14 step targets by Haiku 4.5 (4096 output tokens, 3 passes each), scored over the 76 rollouts containing `\boxed` (a rollout truncated before its answer makes the annotator refuse regardless of the target). Figure 5 shows that the annotation reliability is flat at 1.000 for targets $\leq$ 100, **still** 0.980 **at** 150, then decays (0.882 at 200, 0.796 at 250, 0.711 with the raw split). We then search the target steps below 150 steps because they are $\sim$100% *ok_frac*. The screen bounds the target from above but cannot discriminate within $[50, 150]$, since annotation is reliable throughout that range. Specifically, we did a widely spaced targets search with small training (512 rollouts $\times$ 4 training steps): 150 (just past saturation), 100 (86 tok/step, closest to Qwen's 77), and 50 (the 2$\times$-coarser control).

Experiment 2: Training at the three targets Each target was used for a four-step training ($\beta = 0.3$, $\gamma_{\mathrm{context}} = 0.5$, 512 rollouts per step, validation on `dapo_val512` every two steps).

Table 11: **Hyperparameter Study of** β under fixed hyperparameters: $\alpha = 1, \gamma_{context} = 0.5$.

β	**0**	**0.2**	**0.3**	**0.5**	**1.0**
Accuracy	0.4490	0.4804	0.4824	0.4510	0.4431

Table 12: The test dataset sizes.

Dataset	GSM8K	MATH-500	AIME 2024	AIME 2025
Size	1319	500	30	30

Table 13 reports the outcome trained on DeepSeek-R1-Distill-Qwen-7B. Target 50 wins on validation accuracy and holds the highest live annotation reliability, never falling below 0.992, against 150's drop to 0.953–0.963. It also leads to a faster training speed ($461s$ – $644s$ per step). Based on the results, we adopt 50 as the step target (threshold) of for thinking models.

A.4.3 Training Time Comparison

Table 14 reports the approximate wall-clock training time of RECAP and the post-training baselines under the available hardware configurations. All runs use 8 GPUs, 60 training steps, and a maximum response length of 8192 tokens. Since not every method was run on both H100 and A100 GPUs, we report measured times only without cross-hardware normalization.

Figure 6 further decomposes RECAP's per-step training time. Dependency annotation accounts for only about 12% of the recorded step time for both models. Annotation latency occasionally spikes due to external API throttling, but these spikes do not dominate the overall training cost. On 8×H100, standard GRPO takes $\sim$ 300s per step on average, while RECAP takes $\sim$ 510s with Qwen2.5-Math-7B. The additional overhead of RECAP consists of $\sim$ 149s per step for computing step efficacy and $\sim$ 61s for annotating semantic dependencies. With DeepSeek-R1-Distill-Qwen-7B, whose rollouts are relatively longer, standard GRPO takes $\sim$ 912s per step on average, while RECAP takes $\sim$ 1320s. In this setting, RECAP incurs an additional $\sim$ 250s per step for computing step efficacy and $\sim$ 158s for semantic dependency annotation.

A.4.4 Training Dynamics Analysis.

Reward and reasoning length during training. Figure 7 compares GRPO, Simple Length, and RECAP throughout training. We focus on these three methods because they isolate the main mechanisms of interest: standard outcome-based RL without an explicit efficiency objective, trajectory-level length regularization, and RECAP's step-level redundancy-aware credit assignment.

Across both models, RECAP progressively reduces response length and the number of reasoning steps while maintaining strong training reward. Relative to GRPO, RECAP ends training with substantially fewer tokens and steps while achieving higher reward. Simple Length often reduces token count more aggressively, but at substantially lower reward, especially on DeepSeek-R1-Distill-Qwen-7B. These trajectories shows that reducing length alone is easy, whereas RECAP aims to remove reasoning while preserving the computation needed for successful solutions.

Structural redundancy decreases during training. Figure 8 tracks the fraction of reasoning steps that receive zero structural responsibility under RECAP. This fraction decreases throughout training for both models. Thus, as training progresses, a larger fraction of generated steps participates in the credited downstream derivation. Together with the simultaneous reduction in total step count in Figure 7, this trend is consistent with RECAP progressively suppressing structurally unused reasoning rather than merely shortening individual steps. It also provides training-time evidence complementary to the test-time dead-end analysis in Table 15 and Section 5.3.

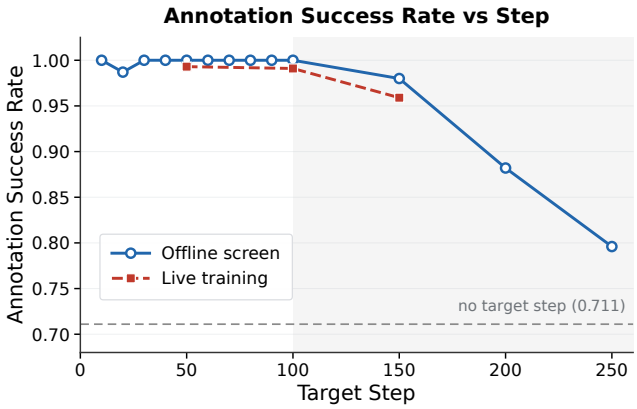


Figure 5: **Annotation reliability bounds the step target from above.** whose dependency annotation is successful (all steps are annotated) versus the per-rollout step target. *Offline screen*: 128 frozen R1-Distill-7B rollouts re-annotated by Haiku 4.5, 3 passes per setting. *Live training*: the same metric during the three runs of the training (`MAX_STEPS_PER_ROLLOUT`$= 50, 100, 150, 512$ rollouts $\times$ 4 training steps). Dashed line: the raw split (0.711). Reliability saturates at 100 and is still 0.980 at 150.

Table 13: **Training runs at the three step targets** (DeepSeek-R1-Distill-Qwen-7B, four steps, validation every two steps).

	target 50	**target** 100	**target** 150
Offline annotation success	1.000	1.000	0.980
Offline tokens / step	155	86	73
Task accuracy	**0.614**	0.596	0.590
Live annotation success	**0.992–0.994**	0.986–0.996	0.953–0.963
Live mean tokens / step	152	87	73
Training Time (s/step)	**461–644**	491–677	495–836

A.4.5 Atomic Reasoning Operation Annotation

To distinguish a reduction in reasoning content from a change in how reasoning is segmented or expressed, we manually annotate atomic reasoning operations on a subset of training rollouts. This analysis is independent of the semantic-step boundaries used by RECAP: annotators inspect the reasoning trace itself and count the underlying operations, regardless of whether several operations appear in the same semantic step or one operation spans multiple textual fragments.

We define an *atomic reasoning operation* as a minimal action that changes the current solution state or introduces new information that is subsequently used in the derivation. The details are shown in Table 16.

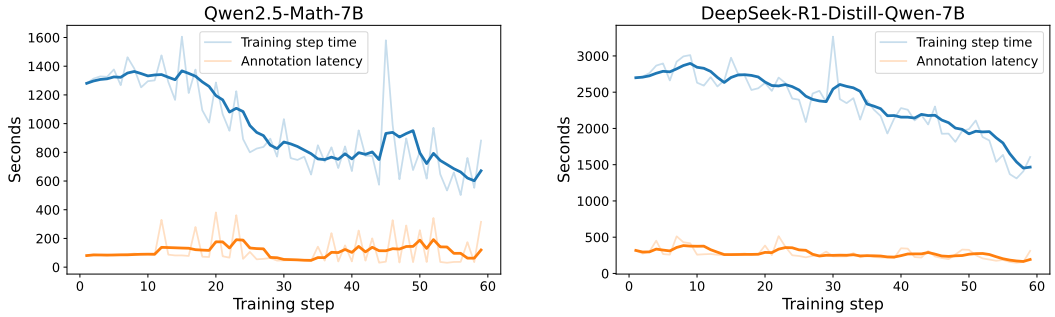
A.4.6 Prompts

Annotator Prompts Figures 9 and 10 show the two prompts for the annotator models.

Policy Model Prompt Figure 11 shows the prompt the *policy* model itself is trained and evaluated on (`use_model_chat_template=True`). To enable R1-Distill-Qwen-7B's thinking mode, we add `<think>` to its chat template only. Qwen2.5-Math-7B does not have the explicit thinking mode.

Table 14: **Approximate wall-clock training time** on 8× H100 or 8× A100 GPUs. "–" denotes configurations that were not measured.

Method	8× **H100**	8× **A100**
Qwen2.5-Math-7B		
GRPO	5h	–
Simple Length	4h 40m	–
DDCA	5h 20m	–
LEASH	3h 35m	–
BINGO	–	6h
ThoughtFold	10h	–
RECAP (ours)	8h 30m	16h 23m
DeepSeek-R1-Distill-Qwen-7B		
GRPO	15h 12m	–
Simple Length	–	19h 55m
DDCA	–	32h 20m
LEASH	–	26h
BINGO	–	11h 40m
ThoughtFold	–	32h 10m
RECAP (ours)	22h	38h 41m



(a) Qwen2.5-Math-7B. (b) DeepSeek-R1-Distill-Qwen-7B.

Figure 6: **Per-step training time and annotation latency of RECAP on** 8× **A100.** Light curves show raw measurements and dark curves show 5-step rolling averages. Annotation latency measures the wall-clock time spent waiting for dependency annotations.

Table 15: **Dependency-DAG Analysis of Dead-End Reasoning.** Structural properties of the dependency DAGs for Qwen2.5-Math-7B rollouts over the full test sets of all four benchmarks, measured on the *same* $n = 8$ samples that produced the accuracy tables. *ML* = share of rollouts with more than one sink in the step sub-graph, i.e. containing work whose result no later step uses and that never reaches the answer. *ISO* = share with at least one step of total degree zero, connected to nothing at all. **Bold** marks the best column value.

	mean	Haiku 4.5†		Opus 4.8		GPT-5.5	
Model	steps	ML↓	ISO↓	ML↓	ISO↓	ML↓	ISO↓
Base	11.7	43.2	26.0	43.3	38.0	42.7	34.3
Prompt-guided	10.0	37.2	22.3	37.7	31.5	37.9	30.6
GRPO	8.2	71.9	60.0	76.6	69.9	78.2	74.5
Simple Length	6.9	37.5	20.0	40.3	30.0	38.8	31.2
DDCA	9.3	46.8	13.3	52.4	34.1	50.2	35.7
LEASH	8.9	44.7	11.8	48.5	29.6	47.0	33.0
Bingo	9.3	44.9	20.5	47.9	33.1	45.5	34.3
ThoughtFold	8.4	43.7	11.7	49.7	32.8	49.2	36.0
RECAP (ours)	**4.0**	**11.0**	**5.9**	**10.8**	**7.0**	**15.6**	**12.8**

Table 16: **Annotation rubric for atomic reasoning operations.** An atomic reasoning operation is a minimal action that changes the current solution state or introduces new information used in the derivation. Pure planning statements, restatements, exact repetitions, and off-task text are excluded. Incorrect operations are still counted.

Operation	Definition	Example
Setup / notation	Introduces a variable, object, or representation that is subsequently used.	$BD = x,\ DC = 6 - x$
Fact recall	Invokes a mathematical fact or named theorem needed by the derivation.	Vieta's formulas; angle-bisector theorem
Equation derivation	Derives a new mathematical relation from existing information.	$x^2 - 14x + 48 \leq 0 \Rightarrow (x-6)(x-8) \leq 0$
Substitution	Substitutes an established value or relation into another expression.	Substitute $b = 6c - 3$ into the third equation.
Numerical computation	Performs an arithmetic or numerical calculation.	$1503 \times 4 = 6012$
Case analysis	Evaluates one distinct branch of an enumeration or case split.	Analyze the case of four-digit even numbers.
Verification	Checks a previously derived result, analytically or computationally.	Verify $\binom{23}{4} + \binom{23}{5} = \binom{24}{5}$.
Conclusion	Converts the established derivation into the requested result.	Conclude that the answer is $x = 6$.

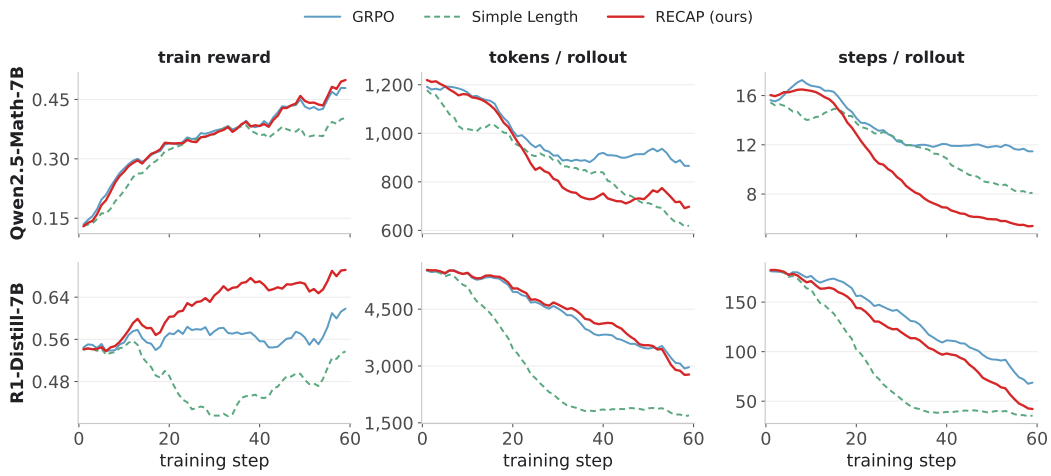


Figure 7: Training dynamics.

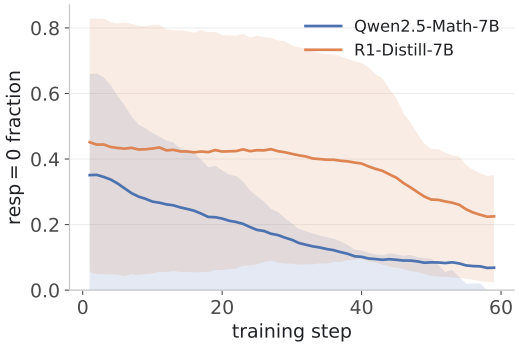


Figure 8: Fraction of reasoning steps that cannot reach the final answer ($resp = 0$) during RECAP training: mean over each step's ∼2000 rollouts, band = inter-quartile range across rollouts, both smoothed over 5 steps.

System Prompt — Dependency Annotator `ANNOTATE_SYSTEM_TEMPLATE`

TASK

You are an annotator of typed dependency structure in step-by-step math solutions.
You will be given:

1. A math problem.
2. A numbered step-by-step solution.
3. A final boxed answer.

Your job is to annotate each step as written. Do not solve, correct, rewrite, explain, add, or remove any step.
Output only the required annotation lines.

OUTPUT FORMAT

For each numbered step, output exactly one line:

```
STEP N    support_depends_on: [...] context_depends_on: [...] restate_of: [...]
```

After all step lines, output exactly one final line:

```
Final answer depends on: [...]
```

Formatting rules:

- Use exact field names.
- Use square brackets for all lists.
- Use [] for empty dependency lists.
- Dependency lists contain step ids only.
- Step ids must be integers.
- Single dependency format: [1]
- Multiple dependency format: [1, 3, 5]
- Put one comma and one space between ids.
- Use only earlier step ids in support_depends_on, context_depends_on, and restate_of.
- Sort ids increasingly.
- Do not use JSON.
- Do not use markdown.
- Do not output explanations.

Example output:

```
Step 1 support_depends_on: [] context_depends_on: [] restate_of: []
Step 2 support_depends_on: [] context_depends_on: [] restate_of: []
Step 3 support_depends_on: [1, 2] context_depends_on: [] restate_of: []
Final answer depends on: [3]
```

EDGE DEFINITIONS

support_depends_on — hard support

Include Step i in Step N's support_depends_on only if Step N directly needs Step i for mathematical truth, validity, applicability, parseability, candidate selection, constraint filtering, verification, or answer extraction.

Test: If Step i were removed, would Step N become invalid, unjustified, underdetermined, inapplicable, impossible to parse, or unsupported as written? If yes, include Step i.

Examples of support:

- using a previous result
- transforming a previous equation
- using a variable or definition introduced earlier
- using a case condition or assumption
- using a domain restriction
- using a candidate set
- filtering candidates
- extracting the requested final answer
- verifying a previous candidate

Important: case conditions, assumptions, definitions, equations, constraints, and candidate sets are support dependencies whenever they are used.

Example — Step 1: Consider $x > 0$. Step 2: Then $|x| = x$. → Step 2 support_depends_on: [1]

context_depends_on — local framing

May overlap support_depends_on: if Step i both mathematically supports Step N and provides local context or scope for it, include Step i in both fields.

Use for:

- case scope
- subproblem scope
- goal transition
- answer target
- organizational framing
- concrete discourse framing

Not used for *mere adjacency, topical similarity, vague narrative flow, or low-information transitions ("Now we continue," "Next, we solve," "This gives the result") unless the step establishes a concrete goal, scope, case, subproblem, or answer target.*

restate_of — pure restatement

Include Step i if Step N merely repeats, summarizes, or lightly rephrases it without adding a new function.

Do not mark as restatement if *the step performs derivation, transformation, selection, filtering, verification, answer extraction, required formatting, or combination.*

FINAL ANSWER

Final answer depends on lists the step(s) that directly produce the boxed answer in the required form — never automatically the last step.
Include a step if it performs answer extraction, selection, filtering, final combination, or required formatting, and the boxed answer follows from it.
Direct dependencies only, not transitive ones:
If Step 4 directly uses Step 3, and Step 3 directly uses Step 2, Step 4's dependency is [3], not [2, 3].
Include Step 2 only if Step 4 directly uses its content again.

PRIORITY RULE

Always choose the smallest sufficient set of earlier steps for each field.
Facts stated directly in the original problem are implicit roots. If a step uses only the problem statement and no earlier step, use [].

For each Step N, inspect each earlier Step i independently:

- `support_depends_on` if Step N needs Step i for truth, validity, applicability, parseability, selection, filtering, verification, or answer extraction.
- `context_depends_on` if Step i establishes local context, scope, goal, case frame, subproblem frame, answer target, or concrete discourse frame for Step N.
- `restate_of` if Step N merely repeats or lightly rephrases Step i without adding a new function.

These fields may overlap when appropriate.

When uncertain:

- *prefer no context edge over a weak artificial context edge;*
- *include support if the step would otherwise be invalid, unjustified, or impossible to parse.*

Figure 9: System prompt given to the annotator.

User Prompt — Per-Rollout Message `ANNOTATE_USER_TEMPLATE`

Sent once per rollout as the chat user turn, filling the two template variables from that rollout's parsed stage-1 output.

```
Problem:
{problem}
Steps:
{numbered_steps}
```

INSTRUCTION

The solution above contains numbered steps and a final boxed answer. For each numbered step, output exactly one line:

```
STEP <ID>     support_depends_on: [<earlier step ids>] \
              context_depends_on: [<earlier step ids>] restate_of: [<earlier step ids>]
```

Then output exactly one final line:

```
FINAL         support_depends_on: [<step ids>] context_depends_on: [<earlier step ids>] \
              restate_of: [<earlier step ids>]
```

Output nothing else.

Figure 10: User prompt given to the annotator

Policy Model Prompt — Chat Template `use_model_chat_template=True`

system

```
Please reason step by step with steps separated by "\n\n", and put your final answer within \boxed{}.
```

+ Prompt-guided baseline addition only (not in the main RECAP prompt)

```
Be concise and avoid unnecessary steps.
```

user

```
{question}
```

assistant

```
<think>  # added for R1-Distill-Qwen-7B only
```

Figure 11: **Policy model prompt (chat-template mode).** System (blue) is the unified instruction that replaces the default template. The gray band directly below is the Prompt-guided-baseline-only addition. User (teal) is the unmodified question. Assistant (violet) is where generation begins. `<think>` is used for R1-Distill-Qwen-7B only, to enable its thinking capability. Qwen2.5-Math-7B does not have explicit thinking mode.