Graph-Based Chain-of-Thought Pruning for Reducing Redundant Reflections in Reasoning LLMs

Anonymous ACL submission

Abstract

Extending CoT through RL has been widely used to enhance the reasoning capabilities of LLMs. However, due to the sparsity of reward signals, it can also induce undesirable thinking patterns such as overthinking, i.e., generating redundant intermediate reasoning content. In this work, we argue that a major source of such redundancy is inefficient reflection, which often manifests in two problematic patterns: Indiscriminate Reflection, where the model performs broad, low-impact checks throughout reasoning, and Repetitive Reflection, where it repeatedly re-verifies an already established conclusion. To address this, we introduce a graph-based CoT optimization framework. Specifically, we convert each linear CoT into a directed acyclic graph (DAG) with explicit dependency edges, and design a dual pruning strategy: branch-level pruning removes weakly contributing reflection branches, while depth-level pruning eliminates late-stage reverification. We distill this behavior via a three-stage pipeline: (1) SFT to initialize the policy on pruned concise traces, (2) DPO to prefer correct but less redundant trajectories, and (3) GRPO with length penalty to jointly optimize answer correctness and efficiency. Experiments show that our approach reduces the average reasoning tokens by 42% while maintaining or improving accuracy.

1 Introduction

Recently, test-time scaling has emerged as an important pathway for improving the reasoning capabilities of LLMs. Representative works such as o1(OpenAI et al., 2024) and R1(DeepSeek-AI et al., 2025) employ reinforcement learning with verifiable rewards to encourage the generation of longer CoT, leading to substantial gains on challenging tasks such as mathematics, code, and logical reasoning. However, reward signals in RL training are often sparse and delayed, making credit assignment over long trajectories more difficult(Tran et al., 2025); as a result, models may develop inefficient thinking patterns such as overthinking, namely producing a large amount of redundant intermediate reasoning that contributes little to the final answer and increases inference cost(Chen et al., 2025a; Sui et al., 2025).

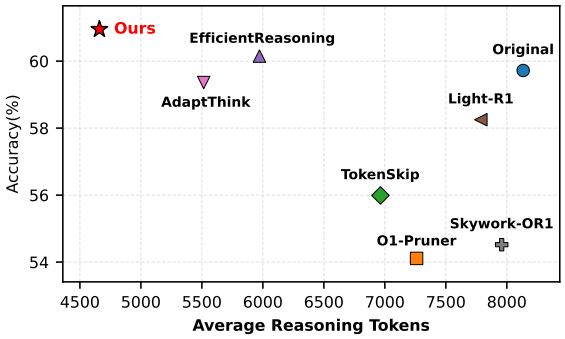


Figure 1: Relationship between average reasoning tokens and accuracy averaged across all benchmarks for different methods.

To mitigate this issue, prior work has explored different ways to constrain and suppress redundancy in CoT. One line of methods focuses on length and reasoning budget control, with typical examples like TALE(Han et al., 2025) and CoT-Valve(Ma et al., 2025b). Another stream focuses on detecting and removing redundant content within the generated trajectory, for example, TokenSkip(Xia et al., 2025) performs controllable compression based on token-level semantic importance, while Think Clearly(Choi et al., 2025) and TRAAC(Singh et al., 2025) leverages attention-based criteria to detect redundant tokens.

In this work, we argue that redundancy largely stems from inefficient reflection behavior. We identify two major forms: Indiscriminate Reflection, where the model checks every intermediate step even when the step is trivial; and Repetitive Reflection, where the model repeatedly re-checks in-

termediate conclusions that have already been verified. The common issue is that such reflection does not contribute new useful information to the main reasoning chain. To precisely identify redundant reflection, we need to recover the dependency relations among reasoning steps. Since practical reasoning often involves non-linear behaviors such as branching exploration, backtracking, and cross-step repetition, we propose to explicitly model CoT with a graph structure, representing reasoning units and their dependencies as nodes and edges, and identifying redundant reflection based on graph properties.

Specifically, we first split the raw CoT into a sequence of steps using keyword triggers, and then perform iterative graph construction with a LLM. With a carefully designed prompt, the model processes each step in sequence, and conditioned on the partially constructed graph and the current step, it creates semantically complete and functionally atomic nodes, determines their semantic roles (Progress nodes that advance reasoning or Review nodes that verify intermediate states) and their logical predecessors, and finally reconstructs the linear text into a DAG with explicit dependencies.

On top of this topology, we design a dual pruning strategy to precisely remove two types of redundancy. For Indiscriminate Reflection, we apply branch-level pruning: when a Review node produces only a small number of child nodes, it forms a narrow side branch that is unlikely to develop into the main reasoning chain and contributes little to the overall solution, and thus is removed. For Repetitive Reflection, we apply depth-level pruning: when a Review node appears in the later part of the CoT, it often corresponds to repeatedly re-verifying conclusions that have already been validated, which is unlikely to introduce new useful information; such nodes are treated as redundant and are removed.

During training, we first perform cold-start supervised fine-tuning on the pruned high-quality CoTs, so that the model initially acquires an efficient reasoning paradigm. To further internalize this strategy, we adopt a two-stage reinforcement learning procedure. In the first stage, we use DPO for preference alignment by constructing preference pairs (concise trajectories versus redundant trajectories), encouraging the model to reduce the probability of generating inefficient reasoning steps. In the second stage, we introduce GRPO with length penalty, using final-answer correctness as the primary signal while jointly optimizing reasoning efficiency, thereby further compressing redundant steps at inference time and improving robustness and accuracy on complex reasoning tasks. Experiments show that compared with strong baselines, our method can substantially reduce ineffective reasoning overhead while maintaining or improving reasoning accuracy.

2 Related Work

2.1 Large Reasoning Models

To elicit human-like step-by-step reasoning, researchers proposed Chain-of-Thought (CoT) prompting, which substantially improves LLM performance on complex reasoning tasks (Wei et al., 2023; Kojima et al., 2023). Later work attempted to internalize this capability via SFT on CoT-annotated data, enabling models to naturally generate intermediate reasoning steps at inference time (Shridhar et al., 2023; Liao et al., 2025). However, further gains are often limited by the scale and quality of available CoT datasets.

Recently, models such as (OpenAI et al., 2024; DeepSeek-AI et al., 2025) have shifted toward Reinforcement Learning, which uses outcome-based supervision in domains with verifiable answers (e.g., mathematics and code) and has proven effective for long-chain reasoning (Wen et al., 2025; Su et al., 2025). Along this direction, post-training with reinforcement learning has become increasingly central, with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) emerging as a widely adopted recipe for improving reasoning performance.

2.2 Efficient Reasoning

Although longer CoT can improve LLM reasoning, it may also induce overthinking, i.e., generating redundant reasoning content (Sui et al., 2025). To mitigate this issue, existing methods can be broadly categorized. The first stream directly regulate the overall reasoning budget by predicting/assigning a token budget(Han et al., 2025; Ma et al., 2025b) or introducing length-aware training objectives in reinforcement learning (Luo et al., 2025b; Arora and Zanette, 2025). Another stream involves identifying redundancy within a trajectory, for instance, Think Clearly (Choi et al., 2025) and TRAAC(Singh et al., 2025) leverages attention-based criteria to detect redundant tokens, TokenSkip (Xia et al., 2025) relies on ex-

plicit importance scoring, while SPIRIT and Step-Entropy (Cui et al., 2025; Li et al., 2025a) use perplexity-/entropy-related cues to identify and remove low-information steps/tokens. In contrast to the above methods, we instead focus on the reflective behaviors of the model, and directly reduce redundant self-reflection steps in the reasoning trace.

3 Methodology

3.1 Problem Definition

Formally, let M_θ be a large reasoning model. For a given problem q, the model generates its reasoning r and produces a final answer a: $r, a \sim M_\theta(q)$, where the reasoning part can be split into many chunks $r = [c_1, c_2, ...c_n]$ at special control tokens such as "wait", "alternative", etc.(complete list is shown in Appendix B). Each chunk represents an individual step generated by the model and n denotes the total number of reasoning chunks.

Our objective is to propose a method to locate and remove redundant reflective segments in r:

$$\tilde{r} \;=\; \mathrm{Prune}(r) \subseteq r, \quad \tilde{r} = [\tilde{c}_1, \ldots, \tilde{c}_{\tilde{n}}],$$

and finetune the model M_θ using the dataset $D = \{(q, \tilde{r}, a)\}$ so that the finetuned model learns to generate more efficient and concise reasoning paths at inference time, while maintaining or even enhancing its overall task performance and accuracy.

3.2 Graph Construction and Pruning

3.2.1 Formatting Chain-of-Thought into a Graph

Given a problem q and its corresponding CoT $r = [c_1, \ldots, c_n]$, we progressively convert the linear reasoning sequence into a DAG

$$G_t = (V_t, E_t, \ell_t), \quad t = 0, 1, \ldots, n,$$

where V_t is the node set, $E_t \subseteq V_t \times V_t$ is the edge set encoding logical dependencies between nodes, and $\ell_t : V_t \to \{progress, review\}$ is the node-type labeling:

- **progress**: advances the active reasoning frontier, producing information that later steps rely on.
- **review**: reads, checks, restates, deletes, or rewinds existing material without advancing the frontier.

We initialize the graph as $G_0 = (\varnothing, \varnothing, \varnothing)$ and update it sequentially with each chunk c_i where c_i is a step-level chunk obtained by splitting the CoT. At step i, conditioned on the current graph G_{i-1} and the chunk c_i, we prompt an external LLM (e.g., qwen-turbo) to predict an operation:

$$o_i \sim f_L(G_{i-1}, c_i) \tag{1}$$

where f_L is the prompting process and $o_i \in \{insert, merge\}$ indicate where to insert a new node or merge the current chunk into an existing node. Detailed prompts are provided in Appendix A

When $o_i = insert$, the model will create a new node $v_i = (s_i, l_i)$, where s_i is the summarization of the node, and l_i is the type of the node, and then connect v_i to existing nodes based on their dependencies. When $o_i = merge$, the model will select a target node $v^\star \in S_i$ and update its content based on the content of the current chunk.

To validate the constructed graphs, we further conduct a human evaluation on randomly sampled nodes, assessing both node-type labeling accuracy and node-level semantic atomicity. Detailed evaluation protocols and quantitative results are reported in Appendix C.

3.2.2 Graph-Based Pruning Criteria

After formatting the linear CoT into a DAG $G = (V, E, \ell)$, we aim to identify and prune low-utility `review` nodes, yielding a more compact and compute-efficient reasoning trajectory.

We define the (directed) descendant set of a node v as

$$\mathrm{Desc}(v) \;=\; \{\, u \in V \mid v \rightsquigarrow u,\; u \neq v \,\},$$

i.e., all nodes reachable from v (excluding v itself). The descendant count is

$$B(v) \;=\; |\mathrm{Desc}(v)|.$$

We also define the depth of v, denoted $d(v)$, is the length of a shortest path from the source node (in-degree 0) to v, and we define the maximum depth $d_{\max} = d(t)$ where t is the terminal node.

We define two types of redundancy in review nodes and prune nodes that fall into either category:

Branch-Level Redundancy. A review node is considered redundant when it has fewer than k descendants, i.e., $B(v) < k$. In this case, the node initiates only a narrowly expanding side branch that rarely develops into the main reasoning trajectory.

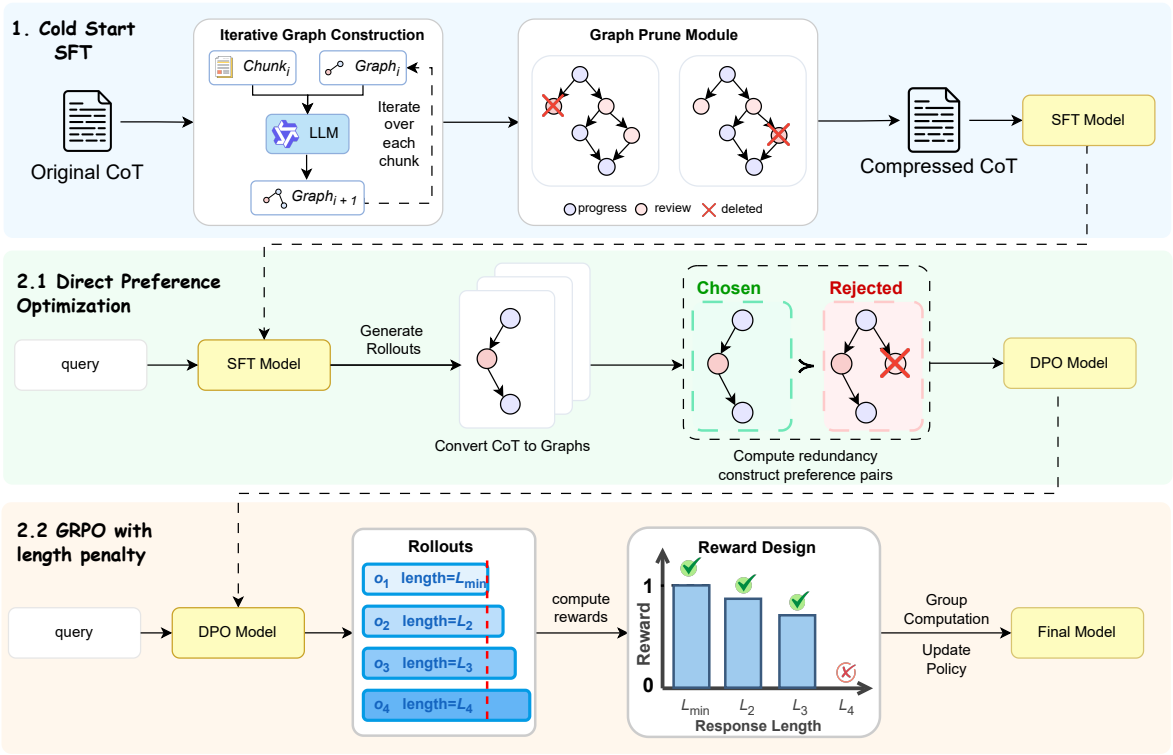


Figure 2: Training pipeline of our graph-based redundancy-aware post-training framework. We first perform cold-start SFT using compressed chain-of-thoughts obtained via iterative graph construction and redundancy pruning, which instills an inductive bias toward concise reasoning. On top of the SFT model, we conduct DPO by ranking correct trajectories according to redundancy, preferring low-redundancy outputs over highly redundant ones. Finally, we apply GRPO with length penalty, jointly optimizing answer correctness and reasoning efficiency to obtain the final model.

Such nodes contribute little to the global problem-solving process and are therefore pruned.

Depth-Level Redundancy. LLMs often reach the correct answer at an intermediate depth but continue producing additional rounds of reflection or self-checking afterward. We treat a review node as redundant if it appears in the late part of the trajectory, i.e., its relative depth exceeds a threshold m, $\frac{d(v)}{d_{\max}} > m$. Such nodes typically correspond to post-answer backtracking and add no new information to the reasoning chain, and are thus removed.

Finally, we locate and prune redundant review nodes, then relinearize the remaining DAG into a CoT sequence for training; we set $m = 0.9$ and $k = 2$ throughout all experiments.

3.3 Training Pipeline

To effectively suppress redundant reflection while preserving correct reasoning, we adopt a three-stage training pipeline for the policy model M_θ: (1) cold-start supervised fine-tuning, which provides a cold-start initialization by training the model to follow concise reasoning traces; (2) preference-based optimization, which aligns the policy toward correct reasoning trajectories with lower reflective redundancy; and (3) GRPO with length penalty, which further refines the policy by jointly optimizing answer correctness and reasoning efficiency. The overall workflow is summarized in Algorithm 1.

3.3.1 Cold-Start Supervised Fine-Tuning

In this stage, we instill an explicit inductive bias toward concise, non-redundant reasoning, which is essential for effective preference optimization and reinforcement learning in later stages. Starting from raw CoT annotations, we construct reasoning graphs and prune reflective redundancies to obtain concise reasoning traces $\tilde{r}$. For each problem x, we pair the pruned trace $\tilde{r}$ with its corresponding final answer a, and train the policy to generate the combined output sequence $y = (\tilde{r}, a)$. The supervised fine-tuning objective is the standard next-token negative log-likelihood:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,\tilde{r},a)} \sum_t \log \pi_\theta(y_t \mid x, y_{<t}),$$

where π_θ is the policy model parameterized by θ, y_t denotes the t-th token in y, and $y_{<t}$ denotes the preceding token sequence.

3.3.2 Two-stage Reinforcement Learning

Stage I: Preference Optimization over Reasoning Trajectories. To further align the policy with concise reasoning behavior, we perform DPO on trajectory pairs ranked by redundancy. For each question x, we sample multiple trajectories and compute a redundancy score:

$$R(y) = \frac{|\{v \in V : \ell(v) = \texttt{review}\}|}{|V|} + \frac{|y|}{\overline{|y|}_x},$$

where $|y|$ is the number of generated tokens and $\overline{|y|}_x$ is the mean length of sampled trajectories for the same question x; Among trajectories that yield correct final answers, we treat those with lower redundancy as preferred y^+ and those with heavier redundancy as dispreferred y^-. We then train the policy model with DPO to increase the relative likelihood of concise, low-redundancy trajectories. The DPO loss is

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)} \log \sigma\Big(\beta\big[\log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}\big]\Big).$$

Here π_θ is the current policy, π_{ref} is a fixed reference policy (e.g., the SFT model), and $\beta > 0$ controls the sharpness of the preference.

Stage II: GRPO with Length Penalty. Finally, we apply GRPO-based reinforcement learning, the reward is computed in two steps. We first assign a correctness reward $V(x, y) \in \{0, 1\}$ based on whether the final answer is correct. We then apply length regularization only to correct trajectories:

$$R(x, y) = V(x, y) - \lambda \, \mathbf{1}_{\{V(x,y)=1\}} \, R_{\text{length}}(x, y),$$

To compute the length penalty, we define $L(y)$ as the number of reasoning tokens and the shortest correct trajectory length among the sampled candidates is

$$L^\star(x) = \min_{y:\, V(x,y)=1} L(y).$$

We measure the normalized excess length as

$$\delta(x, y) = \frac{[L(y) - L^\star(x) - \Delta]_+}{L^\star(x) + \Delta},$$

and define the length regularizer as

$$R_{\text{length}}(x, y) = \delta(x, y)^\gamma,$$

where $[z]_+ = \max(z, 0)$, Δ is a small tolerance margin, and $\gamma \geq 1$ controls the sharpness of the penalty. In this formulation, trajectories whose length is close to the shortest correct one incur almost no penalty, while substantially longer correct trajectories receive increasingly larger penalties, encouraging concise yet accurate reasoning.

4 Experiments

4.1 Experiment Setup

Benchmarks and Metrics: We evaluate our method on five widely used mathematical reasoning benchmarks covering different difficulty levels. AIME24 and AIME25 are derived from the American Invitational Mathematics Examinations, representing competition-level reasoning problems (AIME, 2024, 2025). AMC23 consists of questions from American Mathematics Competitions, reflecting moderately challenging contest problems (AMC, 2023). MATH500 is a curated subset of 500 problems from the MATH benchmark, spanning algebra, number theory, geometry, and probability (Hendrycks et al., 2021). OlympiadBench further tests advanced mathematical and scientific reasoning with bilingual Olympiad-level problems (He et al., 2024). For all the datasets, we sample 10 solutions for each problem, we then calculate the average accuracy of the 10 solutions using math-verify and report the average number of generated tokens across all samples.

Baselines: We compare our method with several efficient reasoning methods. (1) **O1-Pruner**(Luo et al., 2025b): a length-harmonizing fine-tuning approach that reduces long CoT traces while preserving performance. (2) **TokenSkip**(Xia et al., 2025): a controllable CoT compression method that estimates token-level importance and removes low-utility tokens to shorten reasoning traces. (3) **EfficientReasoning**(Arora and Zanette, 2025): an RL-based objective that favors correct yet concise reasoning, improving efficiency while maintaining accuracy. (4) **AdaptThink**(Zhang et al., 2025): an RL method that trains the model to adaptively decide whether to generate an explicit reasoning trace or respond directly based on input difficulty. Additionally, we also compare against several open-source R1-like reasoning models, includ-

Method	AIME24		AIME25		AMC23		OlympiadBench		MATH500		Average	
	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Len↓
Open-Source R1-Style Models												
Skywork-OR1-7B(He et al., 2025)	37.08	11890	24.67	12831	73.50	5241	51.41	6075	85.96	3753	54.52	7958
OREAL-7B(Lyu et al., 2025)	38.75	10907	28.75	11309	81.25	4510	56.44	5164	88.83	3023	58.80	6983
AReaL-boba-RL-7B(Mei et al., 2025)	36.67	11407	25.42	12365	71.56	5337	52.91	6037	87.12	3904	54.74	7810
Light-R1-DS-7B(Liang Wen, 2025)	39.67	12036	31.00	12498	78.25	6073	54.74	5306	87.58	3020	58.25	7787
DeepSeek-R1-Distill-Qwen-1.5B												
Base(DeepSeek-AI et al., 2025)	**30.83**	11755	22.92	11797	63.12	5205	43.89	5632	72.65	2822	46.68	7442
O1-Pruner*(Luo et al., 2025b)	20.00	12680	18.75	12953	59.38	5798	38.56	6645	66.81	3291	40.70	8273
TokenSkip*(Xia et al., 2025)	18.33	12635	17.92	12630	60.62	5452	38.63	6692	72.31	3139	41.56	8110
EfficientReasoning(Arora and Zanette, 2025)	24.17	9380	20.00	9383	68.75	3475	49.48	4032	**81.50**	1600	48.78	5574
AdaptThink(Zhang et al., 2025)	22.08	**5129**	17.50	**3854**	65.31	**2039**	46.67	**2019**	78.90	**987**	46.09	**2806**
Ours	26.67	8278	**23.33**	7739	**69.38**	2774	**49.78**	3221	80.40	1798	**49.91**	4762
DeepSeek-R1-Distill-Qwen-7B												
Base(DeepSeek-AI et al., 2025)	42.67	12723	29.00	12779	81.00	6849	56.77	5252	**89.16**	3065	59.72	8134
O1-Pruner*(Luo et al., 2025b)	36.67	11191	27.50	11475	75.62	5128	54.15	5695	76.62	2812	54.11	7260
TokenSkip*(Xia et al., 2025)	39.58	10841	30.83	11259	79.06	4811	52.94	5795	77.56	2115	55.99	6964
EfficientReasoning(Arora and Zanette, 2025)	43.75	9568	30.83	9293	**83.44**	3401	56.52	3920	82.25	**1392**	59.36	5515
AdaptThink(Zhang et al., 2025)	**44.00**	9424	30.00	10290	80.75	3531	58.76	5011	87.23	1603	60.15	5972
Ours	42.08	**7244**	**31.67**	**6977**	82.34	**3211**	**59.85**	**3786**	88.80	2080	**60.95**	**4660**

Table 1: Performance comparison across multiple math reasoning benchmarks (Accuracy ↑ and Average Length ↓). * We adapt the original implementation of O1-Pruner and TokenSkip to the DeepSeek-R1-Distill-Qwen models.

ing **Skywork-OR1-7B**(He et al., 2025), **OREAL-7B**(Lyu et al., 2025), **AReaL-base-R1-7B**(Mei et al., 2025), and **Light-R1-DS-7B**(Liang Wen, 2025).

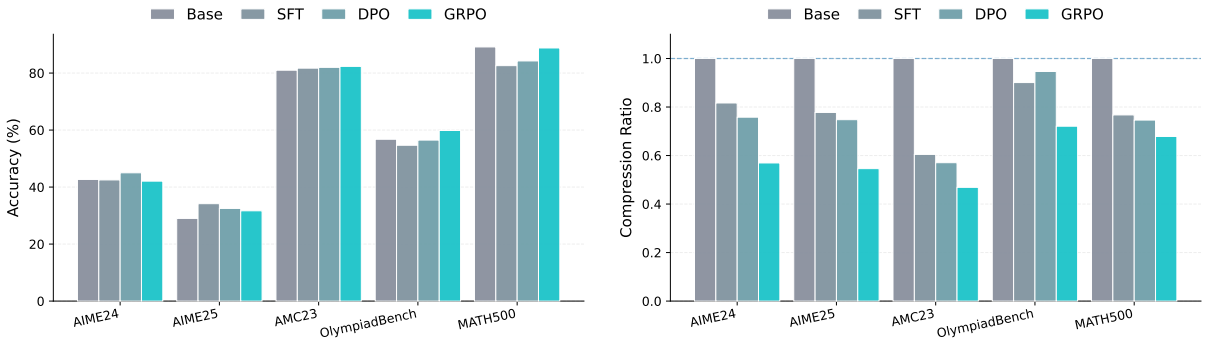
Training Details: We conduct all training on DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B(DeepSeek-AI et al., 2025). We start with supervised SFT on the Light-R1 (Liang Wen, 2025) dataset. We prune all CoT traces with our graph-based method to remove redundant reflection nodes while preserving the core reasoning structure, and then fine-tune the model to imitate these concise reasoning traces. Next, we sample problems from AIME (pre-2024) and AMC (pre-2023)(LI et al., 2024) and use the SFT model to generate responses. For each problem, we construct preference pairs by ranking sampled correct trajectories by our redundancy score, treating low-redundancy trajectories as preferred and high-redundancy ones as dispreferred, and train a DPO model that favors shorter yet effective reasoning. Finally, we perform GRPO with length penalty on the dapo-17k (Yu et al., 2025) dataset with verifiable rewards and length penalty to further improve correctness and robustness. All experiments are conducted on a single compute node with $4\times$ NVIDIA A800 GPUs.

4.2 Main Results

We compare our method against four widely used efficiency-oriented baselines at two model scales, with results reported in Table 1. Overall, our approach achieves the best accuracy–efficiency trade-off: it attains the highest average accuracy while producing substantially shorter reasoning traces. On DeepSeek-R1-Distill-Qwen-7B, our method improves the average accuracy from 59.72 to 60.95 while reducing the average reasoning length from 8134 to 4660 tokens (42.7% reduction). The gains are most evident on difficult benchmarks: for AIME25, accuracy increases from 29.00% to 31.67% while the reasoning length drops from 12779 to 6977 tokens; for OlympiadBench, accuracy increases from 56.77% to 59.85% with shorter traces (5252 → 3786). On DeepSeek-R1-Distill-Qwen-1.5B, our method also improves the average accuracy from 46.68 to 49.91 and reduces the average length from 7442 to 4762 tokens (a 36% reduction), with notable gains on AMC23 (63.12% → 69.38%) and MATH500 (72.65% → 80.40%). These results hold across datasets and model sizes, indicating that our method scales well and consistently reduces redundant computation without sacrificing overall performance.

4.3 Ablation Study

We conduct a stage-wise ablation to disentangle the contributions of each component in our training recipe to both accuracy and efficiency. Starting from the base policy (DS-7B), we progressively add SFT, DPO, and GRPO. Figure 3 summarizes



(a) (b)

Figure 3: **Stage-wise ablation across five benchmarks.** (a) Accuracy (%). (b) Relative reasoning length measured by the average number of generated tokens, normalized to the BASE setting (token ratio; BASE=1.0, lower is better). The compared configurations follow a cumulative training recipe: starting from BASE, we sequentially add SFT, then DPO, and finally GRPO.

the results across five benchmarks. We report (a) accuracy and (b) the average number of generated tokens normalized to the Base setting (token ratio; Base$=1.0$, lower is better). This protocol separates performance improvements from changes in reasoning length, allowing us to assess whether generation cost can be reduced without sacrificing accuracy.

5 Analysis of Graph-based CoT Pruning

In this section, we provide a detailed analysis of our graph-based CoT pruning framework from four perspectives: (i) dataset- and graph-level statistics, (ii) whether pruning preserves essential reasoning, and (iii) changes in model behavior before and after training.

5.1 Dataset and Graph Statistics

We first examine how graph-based pruning reshapes the structure of CoT supervision data. We report the average number of nodes per reasoning graph, the number of reflection nodes, the number of redundant reflection nodes identified by our pruning algorithm, the average CoT length in tokens, the proportion of main-path nodes, as well as the total number of training examples and the data synthesis cost. Table 2 shows the statistics that graph-based pruning removes a large fraction of redundant reflection nodes while keeping the main reasoning path relatively intact, substantially reducing supervision length without discarding most of the essential steps and incurring only a low data synthesis cost.

Statistic	Full CoT	Pruned CoT
Total Samples	3335	3335
Avg. Nodes	27.8	15.6
Avg. Review Nodes	16.8	4.5
Avg. Tokens	6468	4439
Total Cost	–	$20

Table 2: Dataset and graph statistics before and after pruning.

5.2 Does Pruning Preserve Essential Reasoning?

To investigate whether pruning inadvertently removes crucial reasoning steps, we established three distinct experimental settings: (1) **Full-CoT**: the original, complete reasoning traces, (2) **Graph-Pruned**: traces processed via our graph-based pruning, and (3) **Len-Trunc**: traces truncated from the beginning to match the token length of the Graph-Pruned variants. We randomly sampled 1,000 examples from the training set. For each example, we employed `DeepSeek-R1-Distill-Qwen-7B` to generate answers conditioned on these different CoT types, performing $N = 8$ generation passes per question to assess robustness.

We evaluated the model using two metrics: **Accuracy**, representing the percentage of questions answered correctly, and **Consistency**, which measures the degree of consensus among the 8 generated answers. Consistency is defined as $C = \sum(n_y/N)^2$, where n_y denotes the frequency of a specific answer y across the N generations; a higher score indicates the model reliably converges

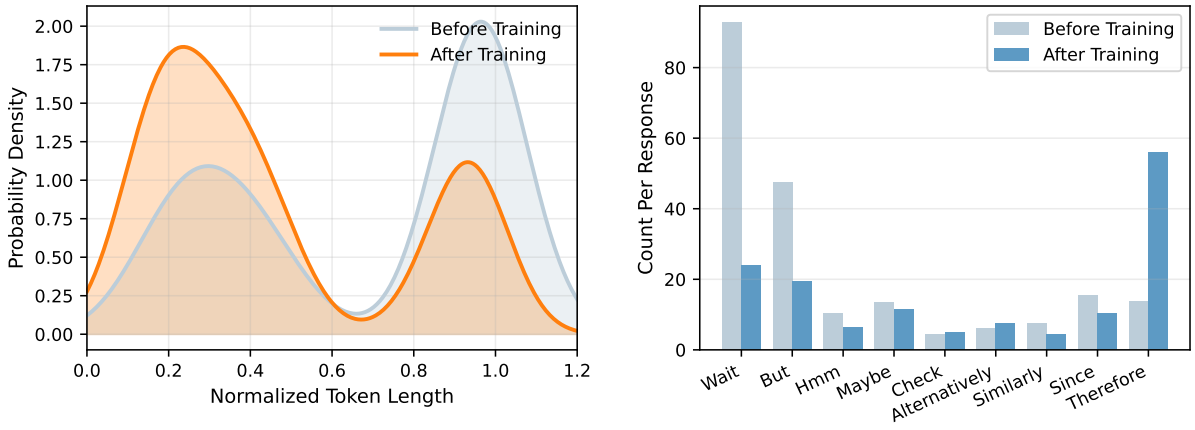


Figure 4: Changes in reasoning length and reasoning token usage before and after training on AIME24.

CoT Variant	Accuracy (↑)	Consistency (↑)
Full-CoT	98.95	99.60
Graph-Pruned	93.70	90.69
Len-Trunc	73.60	69.10

Table 3: Comparison of accuracy and consistency across different Chain-of-Thought (CoT) pruning methods.

on the same output. As shown in Table 3, **Full-CoT** sets a high baseline with 98.95% accuracy and 99.60% consistency. **Graph-Pruned** maintains high reliability, achieving 93.70% accuracy and 90.69% consistency. In contrast, **Len-Trunc** suffers a significant drop to 73.60% accuracy and 69.10% consistency. This stark difference indicates that while naive length truncation disrupts the logical flow, leading to divergent and incorrect answers, our graph-based pruning successfully preserves the core reasoning structure necessary for stable and accurate generation.

5.3 Impact on Training and Model Behavior

Reasoning length and reflection tokens. We analyze how the model's test-time reasoning behavior changes before and after training. We first examine the distribution of reasoning length, measured by the number of generated tokens and normalized across samples. As shown in Figure 4, the density curves indicate that, after training, the model produces noticeably shorter reasoning trajectories, with a clear suppression of the long-tail region corresponding to excessively long responses.

We then analyze the frequency of representative reasoning tokens (e.g., "wait", "but", "hmm", "maybe", and "check") on AIME24 by counting their occurrences per response. As shown in the bar plot, these reflection-oriented tokens consistently decrease after training, indicating reduced reflective behaviors. In contrast, progress-oriented connectives such as "therefore" become substantially more frequent, suggesting a shift from reflective verbosity toward more direct, decision-driven reasoning.

Qualitative case studies. Finally, we present qualitative case studies to illustrate how graph-based pruning reshapes the reasoning process. For selected problems, we visualize the original CoT graph and the pruned graph, highlighting nodes identified as redundant reflections. We also show the corresponding natural-language CoT segments with removed pieces struck out or shaded. In most cases, pruning eliminates repeated self-checks and digressions while keeping the core derivation intact, leading to cleaner and more stable reasoning trajectories. An example is shown in Figure 7 for reference.

6 Conclusion

We presented a graph-based approach for improving CoT efficiency by identifying and pruning redundant reflection steps that do not contribute to the main reasoning path. By converting linear CoTs into structured graphs, our method localizes and removes low-importance review nodes while preserving essential logic, yielding more concise CoTs without sacrificing accuracy. Our findings show that structural modeling of reasoning offers a promising direction for systematically reducing overthinking and enabling more efficient LLM reasoning.

Limitations

Our approach requires constructing reasoning graphs with a strong teacher model, which introduces preprocessing cost and may limit scalability. The progress–review labeling is coarse and may overlook fine-grained reasoning nuances. Additionally, while effective for mathematical reasoning, it remains unclear how well the method generalizes to more open-ended domains. Future work may explore lighter-weight graph construction, richer semantic labels, and broader domain evaluations.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Second Conference on Language Modeling*.

AIME. 2024. American invitational mathematics examination. https://huggingface.co/datasets/HuggingFaceH4/aime_2024.

AIME. 2025. American invitational mathematics examination. https://huggingface.co/datasets/opencompass/AIME2025.

AMC. 2023. American mathematics competitions. https://huggingface.co/datasets/math-ai/amc23.

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

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. 2025a. Do NOT think that much for 2+3=? On the overthinking of long reasoning models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 9487–9499. PMLR.

Zigeng Chen, Xinyin Ma, Gongfan Fang, Ruonan Yu, and Xinchao Wang. 2025b. Verithinker: Learning to verify makes reasoning model efficient. *Preprint*, arXiv:2505.17941.

Daewon Choi, Jimin Lee, Jihoon Tack, Woomin Song, Saket Dingliwal, Sai Muralidhar Jayanthi, Bhavana Ganesh, Jinwoo Shin, Aram Galstyan, and Sravan Babu Bodapati. 2025. Think clearly: Improving reasoning via redundant token pruning. *Preprint*, arXiv:2507.08806.

Karl Cobbe and 1 others. 2021. Gsm8k: A reasoning benchmark for grade-school math word problems. *arXiv preprint arXiv:2110.14168*.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, Suhang Wang, Yue Xing, Jiliang Tang, and Qi He. 2025. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18581–18597, Vienna, Austria. Association for Computational Linguistics.

Muzhi Dai, Chenxu Yang, and Qingyi Si. 2025. S-grpo: Early exit via reinforcement learning in reasoning models. *Preprint*, arXiv:2505.07686.

Renfei Dang, Zhening Li, Shujian Huang, and Jiajun Chen. 2025. The first impression problem: Internal bias triggers overthinking in reasoning models. *Preprint*, arXiv:2505.16448.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yunzhen Feng, Julia Kempe, Cheng Zhang, Parag Jain, and Anthony Hartshorn. 2025. What characterizes effective reasoning? revisiting length, review, and structure of cot. *Preprint*, arXiv:2509.19284.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. *Preprint*, arXiv:2412.18547.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. 2025. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*.

Dan Hendrycks and 1 others. 2021. Math-500: A curated subset of the math benchmark for mathematical reasoning. *arXiv preprint arXiv:2103.02789*.

Joonwon Jang, Jaehee Kim, Wonbin Kweon, Seonghyeon Lee, and Hwanjo Yu. 2025. Verbosity-aware rationale reduction: Sentence-level rationale reduction for efficient and effective reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20769–20784, Vienna, Austria. Association for Computational Linguistics.

Yuxuan Jiang, Dawei Li, and Frank Ferraro. 2025. Drp: Distilled reasoning pruning with skill-aware step

decomposition for efficient large reasoning models. *Preprint*, arXiv:2505.13975.

Liwei Kang, Yue Deng, Yao Xiao, Zhanfeng Mo, Wee Sun Lee, and Lidong Bing. 2025a. First try matters: Revisiting the role of reflection in reasoning models. *Preprint*, arXiv:2510.08308.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2025b. C3ot: generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. `https://huggingface.co/AI-MO/NuminaMath-CoT`.

Zeju Li, Jianyuan Zhong, Ziyang Zheng, Xiangyu Wen, Zhijian Xu, Yingying Cheng, Fan Zhang, and Qiang Xu. 2025a. Compressing chain-of-thought in llms via step entropy. *Preprint*, arXiv:2508.03346.

Zhong-Zhi Li, Xiao Liang, Zihao Tang, Lei Ji, Peijie Wang, Haotian Xu, Xing W, Haizhen Huang, Weiwei Deng, Yeyun Gong, Zhijiang Guo, Xiao Liu, Fei Yin, and Cheng-Lin Liu. 2025b. Tl;dr: Too long, do reweighting for efficient llm reasoning compression. *Preprint*, arXiv:2506.02678.

Fenrui Xiao Xin He Qi An Zhenyu Duan Yimin Du Junchen Liu Lifu Tang Xiaowei Lv Haosheng Zou Yongchao Deng Shousheng Jia Xiangzheng Zhang Liang Wen, Yunke Cai. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond.

Huanxuan Liao, Shizhu He, Yupu Hao, Xiang Li, Yuanzhe Zhang, Jun Zhao, and Kang Liu. 2025. SKIntern: Internalizing symbolic knowledge for distilling better CoT capabilities into small language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3203–3221, Abu Dhabi, UAE. Association for Computational Linguistics.

Haotian Luo, Haiying He, Yibo Wang, Jinluan Yang, Rui Liu, Naiqiang Tan, Xiaochun Cao, Dacheng Tao, and Li Shen. 2025a. Ada-r1: Hybrid-cot via bi-level adaptive reasoning optimization. *Preprint*, arXiv:2504.21659.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025b. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *Preprint*, arXiv:2501.12570.

Chengqi Lyu, Songyang Gao, Yuzhe Gu, Wenwei Zhang, Jianfei Gao, Kuikun Liu, Ziyi Wang, Shuaibin Li, Qian Zhao, Haian Huang, and 1 others. 2025. Exploring the limit of outcome reward for learning mathematical reasoning. *arXiv preprint arXiv:2502.06781*.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025a. Reasoning models can be effective without thinking. *Preprint*, arXiv:2504.09858.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025b. CoT-valve: Length-compressible chain-of-thought tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6025–6035, Vienna, Austria. Association for Computational Linguistics.

Zhiyu Mei, Wei Fu, Kaiwei Li, Guangju Wang, Huanchen Zhang, and Yi Wu. 2025. Real: Efficient rlhf training of large language models with parameter reallocation. *Preprint*, arXiv:2406.14088.

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

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25127–25152, Vienna, Austria. Association for Computational Linguistics.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Ziqing Qiao, Yongheng Deng, Jiali Zeng, Dong Wang, Lai Wei, Guanbo Wang, Fandong Meng, Jie Zhou, Ju Ren, and Yaoxue Zhang. 2025. ConCISE: Confidence-guided compression in step-by-step efficient reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8021–8040, Suzhou, China. Association for Computational Linguistics.

Swarnadeep Saha, Archiki Prasad, Justin Chen, Peter Hase, Elias Stengel-Eskin, and Mohit Bansal. 2025. System 1.x: Learning to balance fast and slow planning with language models. In *International Conference on Representation Learning*, volume 2025, pages 38652–38679.

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

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. DAST: Difficulty-adaptive slow-thinking for large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2322–2331, Suzhou (China). Association for Computational Linguistics.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. 2023. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7059–7073, Toronto, Canada. Association for Computational Linguistics.

Joykirat Singh, Justin Chih-Yao Chen, Archiki Prasad, Elias Stengel-Eskin, Akshay Nambi, and Mohit Bansal. 2025. Think right: Learning to mitigate under-over thinking via adaptive, attentive compression. *Preprint*, arXiv:2510.01581.

Feifan Song, Shaohang Wei, Bofei Gao, Yejie Wang, Wen Luo, Wei Li, Linli Yao, Weimin Xiong, Liang Chen, Tianyu Liu, and Houfeng Wang. 2025. Mitigating overthinking through reasoning shaping. *Preprint*, arXiv:2510.09535.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. 2025. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains. *Preprint*, arXiv:2503.23829.

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

Hieu Tran, Zonghai Yao, and Hong Yu. 2025. Exploiting tree structure for credit assignment in rl training of llms. *Preprint*, arXiv:2509.18314.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. 2025. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. *Preprint*, arXiv:2506.08343.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *Preprint*, arXiv:2506.14245.

Taiqiang Wu, Runming Yang, Tao Liu, Jiahao Wang, and Ngai Wong. 2025. Revisiting model interpolation for efficient reasoning. *Preprint*, arXiv:2510.10977.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. TokenSkip: Controllable chain-of-thought compression in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363, Suzhou, China. Association for Computational Linguistics.

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

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025. AdaptThink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730, Suzhou, China. Association for Computational Linguistics.

Zheng Zhao, Yeskendir Koishekenov, Xianjun Yang, Naila Murray, and Nicola Cancedda. 2025. Verifying chain-of-thought reasoning via its computational graph. *Preprint*, arXiv:2510.09312.

A Prompts

This section describes the instruction prompt used to construct a graph based on the model's original CoT. Given a reasoning step and a partial graph, the model updates the graph by either inserting a new node or merging the step into an existing one.

Each node represents an abstract reasoning step and is labeled as either progress, which advances the reasoning process, or review, which captures reflective behavior. Reflective steps are prohibited from being merged into progress nodes.

The prompt enforces fixed node identifiers, directed dependency edges, and a dedicated *final answer* node, with all updates returned in a structured JSON format for deterministic parsing. The full prompt is shown in Figure 6.

B Special Tokens to Split CoT to Chunks

We use a set of special tokens to split a long reasoning trajectory into multiple step-level chunks.

```
split_tokens = [ "Wait",
    "Alternatively", "Another angle",
    "Another approach", "But wait",
    "Hold on", "Hmm", "Maybe", "Looking
    back", "Okay", "Let me", "First",
    "Then", "Alright", "Compute",
    "Correct", "Good", "Got it", "I
    don't see any errors", "I think",
    "Let me double-check", "Let's see",
    "Now", "Remember", "Seems solid",
    "Similarly", "So", "Starting",
    "That's correct", "That seems
    right", "Therefore", "Thus" ]
```

C Node-level Evaluation of Graph Construction

To evaluate the accuracy of converting linear CoT into graph-structured representations, we conduct a human evaluation on a randomly sampled set of 100 graph nodes. Each node is assessed along two dimensions.

First, we evaluate node type correctness. Nodes are classified as either *progress* or *review*, and annotators judge whether the predicted type matches the node's functional role in the original reasoning process. We report per-class precision, recall, and F1 scores for both node types.

Second, we evaluate step atomicity. A node is considered atomic if it corresponds to a single, semantically independent reasoning step, without mixing multiple operations. We report the atomicity valid rate as a measure of structural quality.

We further define a node as valid only if it satisfies both criteria. Table 4 summarizes the results. The model achieves high accuracy in node type classification and a strong atomicity valid rate, indicating that the constructed graphs are both semantically faithful and structurally well-formed.

Node Type	Precision	Recall	F1
Review	0.9048	0.9661	0.9344
Progress	0.8947	0.8500	0.8718
Atomicity Valid (%)		**85.29**	

Table 4: Type classification performance and step atomicity validity of constructed graph nodes.

D Implementation Details

D.1 SFT Training Settings

We perform SFT using the LLaMA-Factory framework with LoRA-based parameter-efficient tuning. LoRA adapters are applied to all attention and linear layers. The hyper-parameters are summarized in Table 5.

Name	Value
Epochs	8
Global batch size	64
Max sequence length	8192
Optimizer	AdamW
Learning rate	1×10^{-5}
LoRA rank	32

Table 5: Hyper-parameters for SFT training.

D.2 DPO Training Settings

We perform DPO using the same training framework as SFT. The hyper-parameters are summarized in Table 6.

Name	Value
Epochs	5
Global batch size	64
Max sequence length	8192
Optimizer	AdamW
Learning rate	1×10^{-7}

Table 6: Hyper-parameters for DPO training.

D.3 GRPO Training Settings

We conduct GRPO with length penalty using the verl framework. The hyper-parameters are summa-

rized in Table 7.

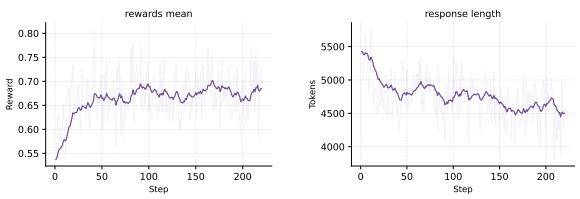
Name	Value
KL coefficient β	1×10^{-3}
Rollouts per input	8
Sampling temperature	1.0
Max response length	12000
Global batch size	64
Optimizer	AdamW
Learning rate	1×10^{-6}
Training steps	220

Table 7: Hyper-parameters for GRPO training.

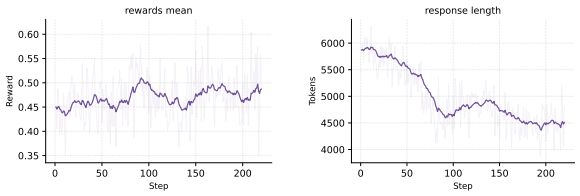
E RL Training Dynamics

Figure 5 presents the training dynamics of the model during reinforcement learning. We visualize the evolution of the reward signal and the average response length across training steps. To reduce high-frequency noise inherent to reinforcement learning, we apply exponential moving average (EMA) smoothing to the raw curves.

As shown in the figure, the reward exhibits an overall increasing trend despite noticeable fluctuations, which is typical for policy optimization with sparse or delayed rewards. At the same time, the response length does not grow monotonically with reward improvement, indicating that higher rewards are not solely achieved by generating longer responses. Instead, the model gradually learns more effective reasoning strategies under the given reward signal.



(a)



(b)

Figure 5: RL training dynamics for different model scales. (a) 7B; (b) 1.5B. Left: mean reward. Right: response length (tokens).

F Overall Training Algorithm

Algorithm 1 presents the pseudocode of our training algorithm, detailing how the policy is optimized across SFT, DPO, and GRPO-based RL.

Algorithm 1: Three-Stage Training with Graph Construction and Pruning

Input: Problems $\{q\}$, raw CoT $r = [c_1, \ldots, c_n]$ and answer a for each q, base policy M_θ
Output: Final policy M_θ

(A) Construct pruned traces for SFT.;
foreach *problem q with CoT $r = [c_1, \ldots, c_n]$* **do**
  $G_0 \leftarrow (\varnothing, \varnothing, \varnothing)$;
  for $i = 1$ **to** n **do**
    $o_i \sim f_L(G_{i-1}, c_i)$, $o_i \in \{\texttt{insert}, \texttt{merge}\}$;
    Update G_i by applying o_i (create/merge node $v_i = (s_i, l_i)$ and add dependency edges);
  $\tilde{G} \leftarrow \textsc{Prune}(G_n; m, k)$;
  $\tilde{r} \leftarrow \textsc{Relinearize}(\tilde{G})$;
  Add $(q, \tilde{r}, a)$ to $\mathcal{D}_{\text{SFT}}$;

(B) Cold-start SFT.;
Train M_θ on $\mathcal{D}_{\text{SFT}}$ by minimizing $\mathcal{L}_{\text{SFT}}(\theta)$ to obtain M_{SFT};

(C) DPO via rollout and preference pairing.;
Initialize $M_\theta \leftarrow M_{\text{SFT}}$;
foreach *problem x* **do**
  Sample rollouts $\mathcal{Y} = \{y^{(k)}\}$ from $\pi_\theta(\cdot \mid x)$;
  Keep correct set $\mathcal{Y}_{\text{ok}} = \{y \in \mathcal{Y} \mid V(x, y) = 1\}$;
  Score redundancy for each $y \in \mathcal{Y}_{\text{ok}}$;
  Form preference pairs (y^+, y^-) with lower-vs-higher redundancy;
  Update M_θ by minimizing $\mathcal{L}_{\text{DPO}}(\theta)$ on collected pairs to obtain M_{DPO};

(D) GRPO with length penalty.;
Initialize $M_\theta \leftarrow M_{\text{DPO}}$;
foreach *problem x* **do**
  Sample trajectories $\mathcal{Y} \sim \pi_\theta(\cdot \mid x)$;
  Compute shortest correct length $L^\star(x)$ in $\mathcal{Y}$;
  foreach $y \in \mathcal{Y}$ **do**
    Compute accuracy reward and length reward;
  Update M_θ with GRPO using reward R;

Prompt for Constructing Graph

You are a **chain-of-thought graph structure analysis and update module**.
Given an existing `graph` and the current text segment `current_step`, you must incorporate `current_step` into the graph by choosing **exactly one** operation: **Insert** or **Merge**, and output **strict JSON**.

1. Inputs

- `graph`: an existing partial reasoning graph (Mermaid code).
- `current_step`: the current reasoning text segment (continuous content from the CoT).

2. Node Definition (Reasoning Unit)

- A node represents an **abstract reasoning unit** with:
  - **Semantic completeness**: a clear intent and its reasoning product (e.g., introducing a constraint, deriving a conclusion, setting a sub-goal, establishing a framework).
  - **Abstraction**: do *not* record low-level arithmetic/symbolic manipulation.
  - **Dependability**: its product can be referenced by later reasoning and creates dependencies.
- **Forbidden**: purely operational steps (substitution, expansion, simplification, step-by-step calculations) cannot be standalone nodes.

3. Independence Criteria Treat `current_step` as a new reasoning unit (prefer **Insert**) if it satisfies any of:

- **Goal introduction**: introduces a new intermediate goal/subproblem.
- **Product generation**: yields a key conclusion/property/constraint/equivalence used later.
- **Method switch**: changes the reasoning strategy or framework.
- **Structural advancement**: adds structure (case split, construction, invariant/lemma framework).
- **Branch initiation**: starts a new attempt path (even if it fails); branch from still-valid ancestors.

If it only continues the same goal with minor details/restatement and produces no new product/structure, prefer **Merge**.

5. Update Operations

- **Merge**: Allowed only if `current_step` can be integrated into exactly one existing node while keeping it **abstract** and **single-purpose**. **Hard constraints**: Review content **cannot** be merged into a **progress** node; Do not merge if it causes one node to mix "advance" and "reflect" as major functions. Must specify `target_node` and `updated_node_description`.
- **Insert**: Required if `current_step` meets independence criteria, introduces a new branch/framework, or Merge would harm abstraction/readability. Must create a new node and add necessary dependency edges.

6. Edge Construction Rules

- **Dependency principle**: if node B uses products from node A, add edge $A \rightarrow B$ with a clear dependency label.
- **Branch origin principle**: new attempts must branch from **still-valid ancestor products**, not from negated/dead-end nodes.
- **Ordering constraints**:
  - Node IDs increase lexicographically: A–Z, AA–AZ, BA–BZ, ...
  - Edges must go from lexicographically smaller IDs to larger IDs.
  - The final node must be named *final answer*.

7. Output Format (Strict JSON)

```
{
  "decision": "Insert or Merge",
  "target_node": "(If Merge, the node ID to merge into; if Insert, leave empty)",
  "new_node": {
    "id": "(If Insert, a unique ID; if Merge, leave empty)",
    "description": "(If Insert, a concise description of the new node; if Merge, leave empty)",
    "type": "progress or review"
  },
  "edges": [
    {
      "from": "source node ID",
      "to": "target node ID",
      "label": "meaning of the edge"
    }
  ],
  "updated_node_description": "(If Merge, the new description; if Insert, leave empty)"
}
```

Your Turn:
Existing Graph: `{{ graph }}`
Current Step: `{{ current_step }}`
Your Response:

Figure 6: Instruction for updating a graph-structured chain-of-thought

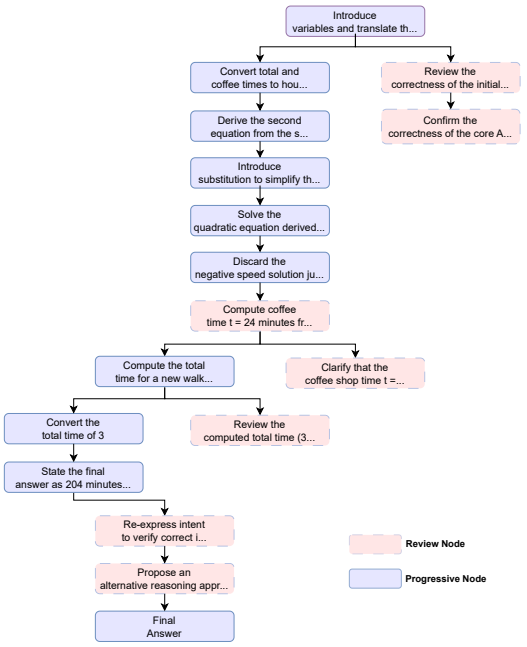
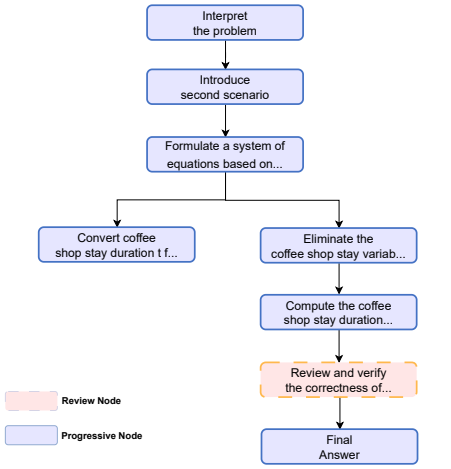
Question	Every morning Aya goes for a 9-kilometer-long walk and stops at a coffee shop afterwards. When she walks at a constant speed of s kilometers per hour, the walk takes her 4 hours, including t minutes spent in the coffee shop. When she walks $s+2$ kilometers per hour, the walk takes her 2 hours and 24 minutes, including t minutes spent in the coffee shop. Suppose Aya walks at $s+\\frac{1}{2}$ kilometers per hour. Find the number of minutes the walk takes her, including the t minutes spent in the coffee shop.	
Base Model	Okay, let me try to figure out this problem step by step... First, let me parse the information given... When she walks at a constant speed of (s) km/h, the total time is 4 hours including (t) minutes in the coffee shop; when she walks at (s+2) km/h, the total time is 2 hours 24 minutes including the same (t) minutes... So I set up the walking-time equations: (\frac{9}{s}=4-\frac{t}{60}) and (\frac{9}{s+2}=2.4-\frac{t}{60})... Let (x=\frac{t}{60}), then subtract the second from the first to eliminate (x): (1.6=\frac{9}{s}-\frac{9}{s+2}=\frac{18}{s(s+2)}\right)), so (\frac{18}{s(s+2)}=1.6), hence (s(s+2)=11.25) and (s^2+2s-11.25=0)... Solve the quadratic: discriminant (=49), so (s=\frac{-2\pm7}{2}), discard the negative solution and get (s=2.5)... Plug back into (,4-x=\frac{9}{s},) to get (4-x=3.6\Rightarrow x=0.4\Rightarrow t=24) minutes... Let me verify with equation 2 to ensure consistency... We have (s+2=4.5), (9/4.5=2), and (2.4-0.4=2), which matches, so that's correct. Now the required speed is (s+\frac{1}{2}=3) km/h, so walking time is (9/3=3) hours; add coffee time 24 minutes to get total time (3) hours (24) minutes (=204) minutes... But let me check if the problem is expecting the answer in hours or minutes... Wait... 3 hours is 180 minutes, plus 24 minutes is 204 minutes... Wait... let me make sure that the coffee shop time remains 24 minutes in all cases... the coffee shop time is fixed at t = 24 minutes regardless of her walking speed... So the total time for 3 km/h is confirmed as 3 hours 24 minutes, i.e., 204 minutes... Wait... in the first scenario (4-0.4=3.6) hours and (3.6\times2.5=9) km... in the second scenario (2.4-0.4=2) hours and (9/4.5=2) hours... Alternatively, maybe I can think of it another way... (9/s-9/(s+2)=1.6) which matches the equation I set up earlier... Therefore, the total time is (\boxed{204}) minutes.	
Ours	Alright, let's tackle this problem step by step. So, Aya has a routine where every morning she walks 9 kilometers to a coffee shop and then spends some time there. The problem gives us two different walking speeds and the total time taken each time, including the coffee shop time. ..... When she walks at speed s km/h, the total time is 4 hours, which includes t minutes in the coffee shop. Similarly, when she walks at s + 2 km/h, the total time is 2 hours and 24 minutes, still including t minutes in the coffee shop. ..... So, the walking time when she walks at speed s is 4 − t/60 hours, and when she walks at s + 2 km/h, the walking time is 2.4 − t/60 hours. ..... For the first case, 9/s = 4 − t/60, and for the second case, 9/(s + 2) = 2.4 − t/60. ..... Let's subtract equation (2) from equation (1): 9/s − 9/(s + 2) = 1.6, which gives s(s + 2) = 11.25. ..... Solve the quadratic equation and discard the negative solution to get s = 2.5 km/h. Plug s = 2.5 back into equation (1) to find t = 24 minutes. Let me check this with equation (2) to ensure consistency: 9 / 4.5 = 2 hours, and 2.4 − 0.4 = 2 hours, so this checks out. ..... Now, the problem asks for the total time when she walks at s + 1/2 km/h. The walking time at 3 km/h is 9 / 3 = 3 hours, and adding the coffee shop time gives a total of 204 minutes. Therefore, the final answer is (\boxed{204}).	

Figure 7: Qualitative comparison of reflection behavior between the base model and our trained model. The left column shows the original CoT, and the right column shows its graph-structured representation. Red-highlighted text indicate reflection-related content. Compared to the base model, our trained model exhibits reduced and more focused reflection.