

# Training LLMs for Divide-and-Conquer Reasoning Elevates Test-Time Scalability

Xiao Liang<sup>1</sup>, Zhong-Zhi Li<sup>2</sup>, Zhenghao Lin<sup>2</sup>, Eric Hancheng Jiang<sup>1</sup>, Hengyuan Zhang,

Yelong Shen<sup>2</sup>, Kai-Wei Chang<sup>1</sup>, Ying Nian Wu<sup>1</sup>, Yeyun Gong<sup>2†</sup>, Weizhu Chen<sup>2†</sup>

<sup>1</sup>University of California, Los Angeles    <sup>2</sup>Microsoft    <sup>†</sup> Corresponding Authors

Large language models (LLMs) have demonstrated strong reasoning capabilities through step-by-step chain-of-thought (CoT) reasoning. Nevertheless, at the limits of model capability, CoT often proves insufficient, and its strictly sequential nature constrains test-time scalability. A potential alternative is divide-and-conquer (DAC) reasoning, which decomposes a complex problem into subproblems to facilitate more effective exploration of the solution. Although promising, our analysis reveals a fundamental misalignment between general-purpose post-training and DAC-style inference, which limits the model's capacity to fully leverage this potential. To bridge this gap and fully unlock LLMs' reasoning capabilities on the most challenging tasks, we propose an end-to-end reinforcement learning (RL) framework to enhance their DAC-style reasoning capacity. At each step, the policy decomposes a problem into a group of subproblems, solves them sequentially, and addresses the original one conditioned on the subproblem solutions, with both decomposition and solution integrated into RL training. Under comparable training, our DAC-style framework endows the model with a higher performance ceiling and stronger test-time scalability, surpassing CoT by 8.6% in *Pass@1* and 6.3% in *Pass@32* on competition-level benchmarks.

 Date: February 3, 2026

 Code: <https://github.com/MasterVito/DAC-RL>

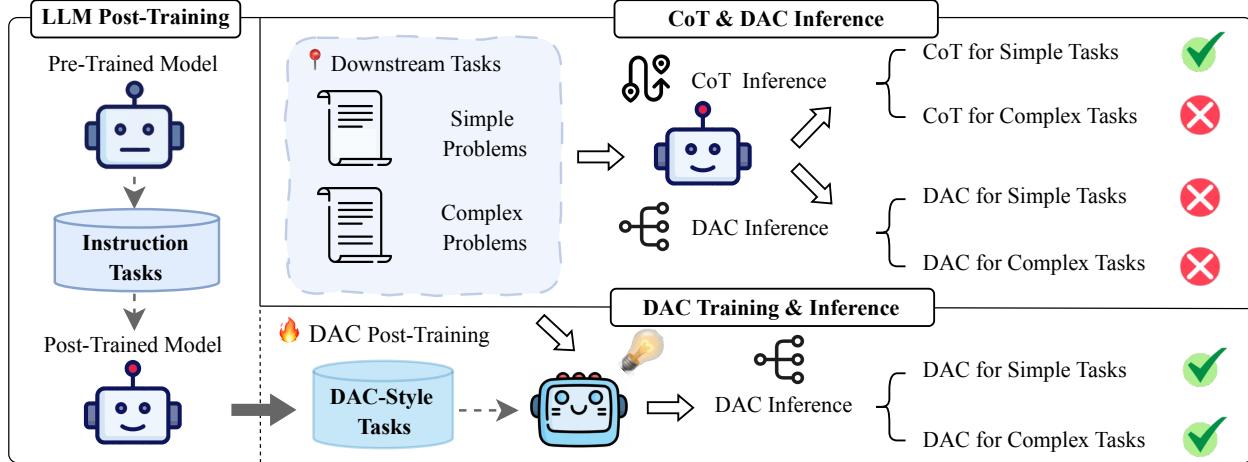
 Correspondence: Yeyun Gong ([yegong@microsoft.com](mailto:yegong@microsoft.com)); Weizhu Chen ([wzchen@microsoft.com](mailto:wzchen@microsoft.com))

## 1. Introduction

Large language models (LLMs) have shown remarkable capabilities in complex reasoning tasks, with chain-of-thought (CoT) (Wei et al., 2022) enabled by large-scale reinforcement learning (RL) during post-training. Models such as OpenAI O1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) can effectively solve competition-level mathematical problems like AIME (MAA) through long CoT reasoning with extensive self-reflection. However, on more challenging reasoning tasks, including International Mathematical Olympiad and advanced theorem proving, standard CoT prompting often exhibits limited effectiveness, motivating the development of more advanced reasoning paradigms for language models.

An intuitive and classical approach to solving complex problems is the **divide-and-conquer** (DAC) strategy, originally developed in computer programming (Cormen et al., 2022). For LLMs, early approaches such as Tree-of-Thought (Yao et al., 2023) and DeAR (Xue et al., 2024) adopt similar strategies by problem decomposition and employing structured reasoning to solve subproblems, thereby facilitating the final solution. Recent works such as Seed-Prover (Chen et al., 2025) and DeepSeek-Prover-V2 (Ren et al., 2025) also incorporate DAC-style problem-solving approaches to tackle frontier mathematical tasks, underscoring the promise of DAC-style test-time scalability in fully eliciting the reasoning capacity of LLMs.

Despite establishing a DAC reasoning framework, these approaches function only during inference and rely on complex prompt engineering, leaving the model misaligned between its general post-training and this specific reasoning style, as complex reasoning problems are typically trained using step-by-step CoT. This misalignment can constrain the model's DAC-style reasoning even on simpler problems that could be solved with CoT, as



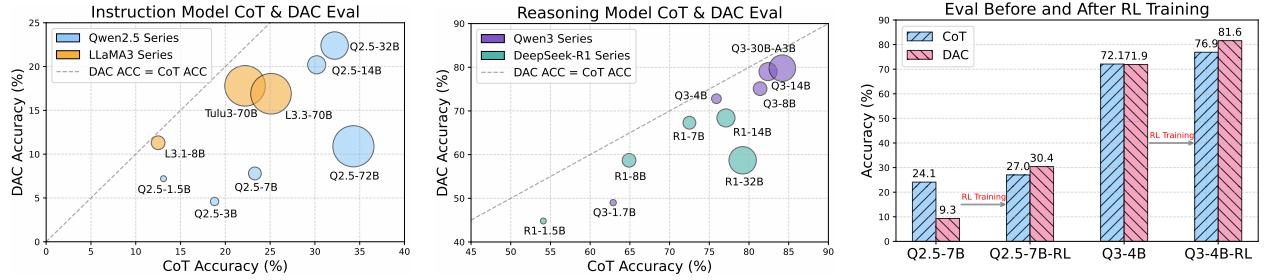
**Figure 1:** Overview of the LLM post-training pipeline, comparison of the DAC and CoT-style inference, and our proposed DAC post-training. Without dedicated training, DAC inference remains ineffective, whereas DAC-specific post-training makes this advanced reasoning paradigm robust across downstream tasks.

illustrated in Figure 1. Accordingly, we evaluate a wide range of instruction-following and reasoning models on mathematical benchmarks using DAC and CoT-style inference, and observe that most models exhibit inferior performance under direct DAC reasoning compared to CoT, as shown in Figure 2. This gap indicates that fully unlocking DAC-style problem-solving capabilities in general LLMs necessitates dedicated training.

To address this issue, we propose an end-to-end RL framework for equipping LLMs with DAC-style reasoning capabilities for tackling complex problems. We adopt RL over supervised fine-tuning because, when the training framework is applied to frontier models, stronger expert annotations are often unavailable or prohibitively expensive, whereas RL enables effective training through self-exploration. Specifically, at each RL iteration, the policy model decomposes every input problem into a set of subproblems. These are then aggregated with the original problem into a conquering prompt, which directs the model to solve the subproblems sequentially and then formulate the final answer. Both problem division and conquering are incorporated into unified policy training, with the corresponding optimization objectives defined in Sections 2.2 and 2.3, respectively. Furthermore, the evolution of distinct and improved subproblems over training epochs facilitates broader, non-repetitive exploration during conquering, which is crucial for keeping generation diversity and preventing entropy collapse (Liang et al., 2025b).

To assess the effectiveness of our DAC-RL framework, we experiment with two models and evaluate the *Pass@1* (averaged 32 times) and *Pass@32* performance on challenging, widely used competition-level reasoning benchmarks, including AIME 24 & 25 (MAA), Beyond-AIME ByteDance-Seed (2025) and HMMT-25 (Balunović et al., 2025). Models trained under our framework consistently outperform standard CoT on the adopted benchmarks, achieving absolute margins of 8.6% and 6.3% on the *Pass@1* and the *Pass@32* metrics when using *Qwen3-4B-Instruct-2507* as the initial policy, whereas CoT-style RL fails to improve performance over this post-trained model. Our analysis indicates that DAC-style training is even beneficial for the policy’s CoT reasoning ability. Moreover, DAC-trained models exhibit more flexible and greater test-time scalability ceilings, along with more compact reasoning traces. We also experiment with cold-start initialization and evaluate its effectiveness in training models for DAC reasoning. Our **contributions** are summarized as follows:

- (1) By comparing DAC and CoT at inference and integrating DAC into training, we reveal a critical misalignment between general-purpose post-training and DAC inference that constrains performance.
- (2) We introduce DAC-RL, a unified training framework that optimizes DAC reasoning via RL, thereby raising the performance ceiling of LLMs on complex reasoning problems and elevating their test-time scalability.
- (3) We perform extensive experiments on post-trained LLMs and evaluate the trained models on challenging benchmarks with detailed analysis, demonstrating the superiority of the proposed DAC-RL framework.



**Figure 2:** We evaluate the CoT and DAC Pass@32 performance on four competition-level benchmarks (Table 1) for both general instruction and reasoning post-trained models. The right panel presents Pass@32 performance for the [Qwen2.5-7B-Instruct](#) and [Qwen3-4B-Instruct-2507](#) models before and after task-specific RL training.

## 2. Method

### 2.1. Preliminaries

**Task Formalization** In this section, we formalize our DAC-style reasoning and CoT reasoning (Wei et al., 2022). In standard CoT reasoning, given an input problem  $x$ , the policy model  $\pi_\theta$  directly generates a step-by-step reasoning trajectory  $y$ , from which the final answer can be extracted as  $a = \text{Extract}(y)$ . For DAC-style reasoning, given an input  $x$ , the model first performs a *division* step, generating a response  $y_d$  consisting of a set of  $n_g$  subproblems derived from  $x$ , denoted as  $\mathcal{P} = \{p_i\}_{i=1}^{n_g} \sim \pi_\theta(\mathcal{P} | x)$ . Next, in the *conquering* step, the generated subproblems  $\mathcal{P}$  together with  $x$  are concatenated to construct a *conquering* prompt, which guides the model to sequentially generate solutions to the subproblems as  $\mathcal{S} = \{s_i\}_{i=1}^n \sim \pi_\theta(\mathcal{S} | x, \mathcal{P})$ , and subsequently solve the original problem conditioned on  $\mathcal{S}$ . The complete *conquering* response is denoted as  $y_c$ .

The *overview* of our DAC training and inference pipeline is illustrated in Figure 3. At each RL training step, the policy model processes every problem in the batch by performing two tasks: (1) *Division*: Dividing the problem into a set of subproblems (green), and (2) *Conquering*: Sequentially solving the subproblems and then the original problem (blue). The learning objective is to maximize the expected rewards of both the *division* and *conquering* responses, defined as:

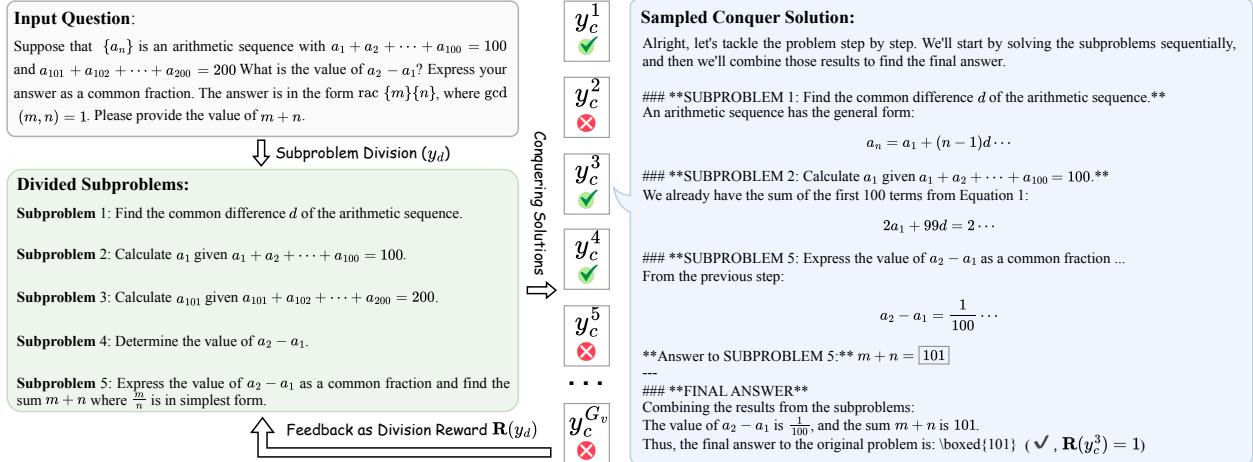
$$\mathcal{J}(\theta) = \mathbb{E}_{y_d, y_c \sim \pi_\theta} [\mathbf{R}(y_d) + \mathbf{R}(y_c)], \quad (1)$$

where  $\mathbf{R}(y_d)$  and  $\mathbf{R}(y_c)$  are the *division* and *conquering* rewards detailed in Section 2.2 and 2.3.

### 2.2. Subproblem Division

During each iteration, the policy is first prompted to decompose each problem  $x$  in the training set into  $G_d$  groups of subproblems  $\{\mathcal{P}_g\}_{g=1}^{G_d}$ , using the prompt in Figure 12. Notably, the policy is required to generate more than  $N_s$  subproblems for each input; without this constraint, it collapses to producing no useful subproblems in *division* response  $y_d$  as the training goes, and the *conquering* stage degrades to directly solving the original problem, as in standard CoT reasoning.

For training subproblem division, we adopt a combined reward scheme that integrates *format validity*, *quantity validity*, and its *helpfulness* in facilitating the solution of the original problem. Specifically, for *format validity*, we require that the subproblems generated by the policy in response to the *division* prompt must be parsable using regular expressions. For *quantity validity*, since the policy is instructed to generate at least  $N_s$  subproblems, a negative reward is assigned when the number of extractable subproblems is fewer than  $N_s$ . For *helpfulness*, we evaluate each generated subproblem group  $\mathcal{P}_g$  using the accuracy of the solutions in the conquering stage  $\text{CA}(\mathcal{P}_g)$  as an indicator of their helpfulness. Based on empirical analysis, we optimize a lower bound on subproblem helpfulness by requiring that each subproblem group  $\mathcal{P}_g$  yield at least one correct solution to the original problem, provided that the policy can generate a correct solution from any subproblem group in  $\{P_i\}_{i=1}^{G_d}$ . This encourages each subproblem group, when incorporated into the *conquering* prompt, to help



**Figure 3:** An overview of the DAC-style inference and reward assignments in training, illustrated with a case study. The policy decomposes the original problem into a group of subproblems, samples candidate *conquering* solutions in parallel, and leverages their correctness to compute *division* rewards for optimizing problem decomposition.

guide the policy toward producing at least one correct solution for simple problems, while rewarding groups that enable correct solutions for challenging problems where others fail. In summary, the reward scheme for division generation  $y_d$  is derived as:

$$\mathbf{R}(y_d) = \begin{cases} 0, & \underbrace{|\mathcal{P}_g| < N_s}_{\text{quantity}} \vee \underbrace{\neg \text{Format}(y_d)}_{\text{format}} \\ 0, & \underbrace{\text{CA}(\mathcal{P}_g) = 0 \wedge \text{CA}(\{\mathcal{P}_i\}_{i=1}^{G_d}) > 0}_{\text{helpfulness}} \\ 1, & \text{otherwise}, \end{cases} \quad (2)$$

where  $|\mathcal{P}_g|$  denotes the number of subproblems, CA represents the *conquering* accuracy, and  $\neg \text{Format}(y_d)$  indicates that  $y_d$  violates the required format, making  $\mathcal{P}_g$  non-extractable.

### 2.3. Subproblem and Original Conquering

Once the subproblem group  $\mathcal{P}_g$  is generated, it is combined with the original problem to form the *conquering* prompt (Figure 13), which guides the model to solve the subproblems sequentially and then tackle the original one. Since ground-truth answers for the subproblems are unavailable and therefore cannot serve as training signals, we instead rely solely on the correctness of the final answer extracted from  $y_c$  to the original problem as a surrogate reward for both subproblem solving and original problem solving. The underlying idea is that if the policy solves the subproblems correctly, their solutions can support answering the original problem, enabling the policy to solve it accurately. Conversely, incorrect solutions to the subproblems may lead to failure in solving the original problem. The theoretical validity of this reward scheme is established in Lemma 2.1. The final *conquering* reward is defined as follows:

$$\mathbf{R}(y_c) = \mathbf{1}\{\text{Extract}(y_c) = a\} \quad (3)$$

where  $a$  denotes the ground-truth answer to the original problem.

**Lemma 2.1 (Final-answer reward positively associates with subproblem correctness).** Let  $s_i \in \{0, 1\}$  indicate whether subproblem  $i$  is solved correctly and  $\mathbf{s} = (s_1, \dots, s_m)$ . Let  $C \in \{0, 1\}$  indicate whether the original problem is solved correctly. If the division-conquer pipeline induces the causal direction  $\mathbf{s} \rightarrow C$ , we have:

$$\text{Cov}_\theta(\mathbf{1}\{s_i = 1\}, \mathbf{1}\{C = 1\}) \geq 0, \quad (4)$$

---

**Algorithm 1** Divide-and-Conquer Reinforcement Learning for LLMs

---

**Require:** Training set  $\mathcal{D}$ ; initial policy  $\pi_\theta$ ; division group size  $G_d$ ; conquering group size  $G_c$ ; training steps  $T$   
**Ensure:** Trained policy  $\pi_{\theta'}$

- 1: Initialize experience buffer  $\mathcal{B} \leftarrow \emptyset$
- 2: **for**  $t = 1$  to  $T$  **do**
- 3:   Sample a mini-batch  $d \sim \mathcal{D}$
- 4:   **for** each problem–answer pair  $(x, a)$  in  $d$  **do**
- 5:     [Divide] Generate  $G_d$  subproblem groups  $\{\mathcal{P}_g\}_{g=1}^{G_d} \sim \pi_\theta(x)$ ,  $\mathcal{P}_g = \{p_{g,i}\}_{i=1}^{n_g}$ ,  $n_g$  varies per group.
- 6:     **for** each subproblem group  $\mathcal{P}_g$  **do**
- 7:       [Conquer] Generate  $G_c$  solution candidates  $\{y_{g,v}\}_{v=1}^{G_c} \sim \pi_\theta(\mathcal{P}_g; x)$
- 8:       Compute correctness rewards  $\{\mathbf{R}(y_{g,v})\}_{v=1}^{G_c}$  for conquering w.r.t. reference  $a$
- 9:       Store conquer tuples in buffer:  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(x; \mathcal{P}_g), y_{g,v}, \mathbf{R}(y_{g,v})\}_{v=1}^{G_c}$
- 10:      **end for**
- 11:     [Division Reward] Evaluate Format and Quantity validity  $\{f_g\}_{g=1}^{G_d}$ ,  $\{q_g\}_{g=1}^{G_d}$  for  $\{\mathcal{P}_g\}_{g=1}^{G_d}$
- 12:     Compute division rewards  $\{\mathbf{R}(\mathcal{P}_g)\}_{g=1}^{G_d}$  from  $\{\mathbf{R}(y_{g,v})\}_{v=1}^{G_c}$ ,  $\{q_g\}$ , and  $\{f_g\}$  via Eq. (2)
- 13:     Store division tuples in buffer:  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(x, \mathcal{P}_g, \mathbf{R}(\mathcal{P}_g))\}_{g=1}^{G_d}$
- 14:    **end for**
- 15:    [Policy Update] Update policy  $\pi_\theta$  using  $\mathcal{B}$  according to Eq. (6)
- 16:    Clear buffer:  $\mathcal{B} \leftarrow \emptyset$
- 17: **end for**
- 18: **return** trained policy  $\pi_{\theta'}$

---

where  $\theta$  denotes the parameters of the policy  $\pi_\theta$ . The detailed proof for Lemma 2.1 is depicted in Appendix B. Consequently, the scalar reward in Equation 3, which satisfies  $\mathbb{E}_{y_c \sim \pi_\theta}[\mathbf{R}(y_c)] = P_\theta(C = 1)$ , is a consistent surrogate signal for subproblem correctness: rewarding  $C = 1$  preferentially upweights trajectories with more correct subproblems, and vice versa.

### 3. Experiments

#### 3.1. Settings

**Models and Datasets.** We conduct experiments using two models, [Qwen2.5-7B-Instruct](#) and [Qwen3-4B-Instruct-2507](#), to evaluate the proposed DAC-RL framework. We use [DAPO-Math-17k](#) (Yu et al., 2025a) as the training dataset. Following rStar2 (Shang et al., 2025), which suggests that rule-based verifiers often struggle with certain open-ended mathematical formats, we evaluate model performance only on benchmarks with integer answers to ensure precise assessment. Specifically, we evaluate models on four widely used competition-level mathematical reasoning benchmarks: AIME 2024, AIME 2025 ([MAA](#)), Beyond-AIME ([ByteDance-Seed](#), 2025), and HMMT-25 ([Balunović et al., 2025](#)).

**Implementation Details.** For RL training, all experiments employ the GRPO (Shao et al., 2024) optimization strategy, extended with the Clip-Higher and token-level loss techniques (Yu et al., 2025a). In each iteration, every input problem is divided into  $G_d = 4$  subproblem groups containing varying numbers of subproblems, and each group, together with the original problem, is used to generate  $G_c = 8$  conquering solutions. The minimum requirement of subproblems is set to  $N_s = 3$ . The training batch size is set to 256, the maximum rollout length to 8,192, and the sampling temperature to 1.0. For policy optimization, the Clip-Higher upper bound  $\epsilon_h$  is fixed at 0.28, and the mini-batch size is 64. We conduct 400 training steps (nearly 6 epochs) for both models on the [DAPO-Math-17k](#) dataset, with an additional about 200 steps for the [Qwen3-4B-Instruct-2507](#) policy on a difficulty-filtered subset containing 3.7k problems that the initial policy solves with less than 50% accuracy across 16 responses, to better elicit its advanced reasoning capability.

Model	AIME 2024		AIME 2025		Beyond-AIME		HMMT 2025		Average	
	Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32
Qwen2.5-7B-Instruct										
Init-CoT	9.8	26.7	6.8	<b>36.7</b>	3.8	23.0	2.0	10.0	5.6	24.1
Init-DAC	0.5	13.3	0.2	6.7	0.7	10.0	0.2	6.7	0.4	9.2
RL-CoT	13.5	34.5	11.4	30.8	5.1	25.5	2.7	13.1	8.2	27.0
RL-DAC	<b>15.5</b>	<b>39.1</b>	<b>15.5</b>	34.2	<b>7.0</b>	<b>27.4</b>	<b>4.8</b>	<b>20.8</b>	<b>10.4</b>	<b>30.4</b>
$\Delta$ (RL)	<b>+2.0</b>	<b>+4.6</b>	<b>+4.1</b>	<b>+3.4</b>	<b>+1.9</b>	<b>+1.9</b>	<b>+2.1</b>	<b>+7.7</b>	<b>+2.2</b>	<b>+3.4</b>
Qwen3-4B-Instruct-2507										
Init-CoT	62.6	<b>90.0</b>	45.7	76.7	32.1	65.0	30.3	56.7	42.7	72.1
Init-DAC	59.6	<b>90.0</b>	43.2	73.3	29.6	61.0	28.2	63.3	40.2	71.9
RL-CoT	45.9	85.8	52.1	77.4	30.4	58.1	21.8	54.4	37.5	69.0
RL-DAC	<b>63.9</b>	87.7	<b>54.2</b>	<b>78.8</b>	<b>34.6</b>	<b>67.9</b>	<b>31.9</b>	<b>66.6</b>	<b>46.1</b>	<b>75.3</b>
$\Delta$ (RL)	<b>+18.0</b>	<b>+1.9</b>	<b>+2.1</b>	<b>+1.4</b>	<b>+4.2</b>	<b>+9.8</b>	<b>+10.1</b>	<b>+12.2</b>	<b>+8.6</b>	<b>+6.3</b>
Qwen3-4B-Instruct-2507 (Deep)										
RL-D-CoT	64.4	84.8	58.8	87.9	38.9	69.5	37.6	65.5	49.9	76.9
RL-D-DAC	66.3	91.6	61.5	87.6	38.8	70.7	38.7	76.4	51.3	81.6
$\Delta$ (RL)	<b>+1.9</b>	<b>+6.8</b>	<b>+2.7</b>	<b>-0.3</b>	<b>-0.1</b>	<b>+1.2</b>	<b>+1.1</b>	<b>+10.9</b>	<b>+1.4</b>	<b>+4.7</b>

Table 1: The results of the baselines and our DAC strategies are reported across six benchmarks using different models. The *Pass@1* metric is averaged over 32 runs to ensure a more stable and precise evaluation. RL-D refers to the *Deep DAC* setting with the baseline described in Section 3.2. Best results are highlighted in **bold**.

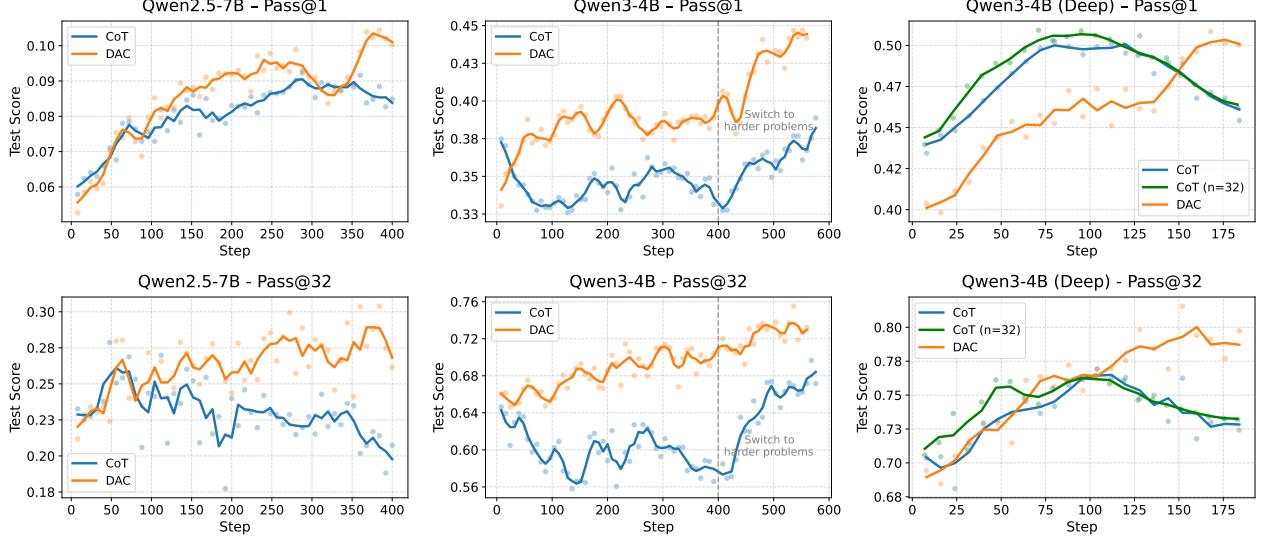
For inference, we set the maximum token length to 16,384 unless otherwise specified, with the temperature kept the same as training (1.0) and the top- $p$  value set to 0.7 following (Yu et al., 2025a). We report the *Pass@1* metric as the average over 32 runs. For *Pass@32* evaluation, we employ an unbiased estimation method (Chen et al., 2021) to reduce the high variance from single evaluations.

### 3.2. Main Results

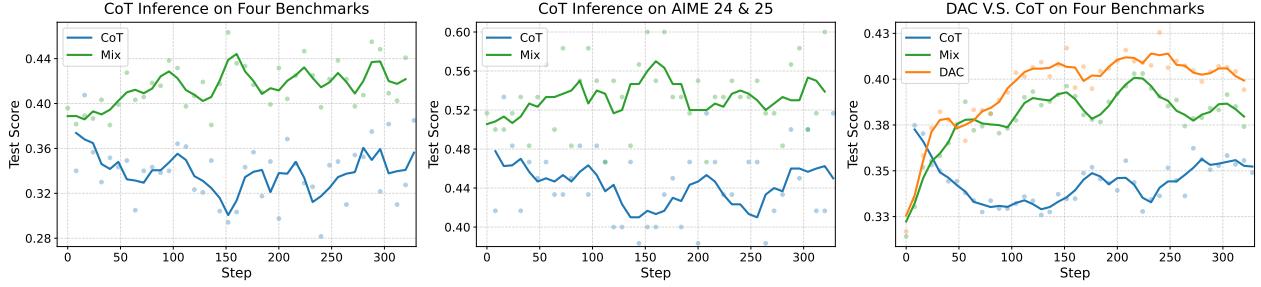
**DAC-style Reasoning Shows a Higher Ceiling.** We present the main results of our DAC-RL training and inference alongside the CoT baselines in Table 1. Compared with the standard CoT RL training, DAC-style training demonstrates substantially greater gains in *Pass@32* performance on competition-level benchmarks, achieving overall improvements of **3.4%** and **6.3%** for the *Qwen2.5-7B-Instruct* and *Qwen3-4B-Instruct-2507* models, respectively. Notably, these training improvements are achieved even when the initial policy’s DAC reasoning lags behind the CoT-style reasoning obtained through intensive post-training, particularly for the *Qwen2.5-7B-Instruct* model, which attains only 0.4% average DAC accuracy on competition-level problems.

To characterize training effects across the two reasoning styles, we evaluate intermediate checkpoints in Figure 4. Notably, both models begin with lower DAC accuracy than CoT, particularly on the *Pass@1* metric. However, as training progresses, DAC performance steadily grows, eventually outperforming CoT and increasing at a faster pace. The steady improvements, together with the observation that CoT-style RL fails to elicit further gains for *Qwen3-4B-Instruct-2507*, suggest that CoT reasoning is largely saturated in post-training for general instruction-tuned models. In contrast, DAC surpasses the CoT performance ceiling on the evaluated benchmarks, indicating a higher upper bound on reasoning and the necessity of dedicated DAC post-training to fully unlock it.

**Deep DAC Training Further Enhances Reasoning.** The motivation for DAC-style reasoning is to fully unleash the model’s potential to solve the most challenging problems at the edge of its capability. To this end, we conduct experiments of the *Deep DAC* training setting, training the *Qwen3-4B-Instruct-2507* model exclusively on the 3.7k most difficult subset described in Section 3.1 for ten epochs, with extended training and inference token budgets of 16,384 and 24,576, respectively. This setup equips the model with deeper mastery of DAC-style reasoning while enhancing test-time scalability, particularly for complex tasks. We further compare this



**Figure 4:** Intermediate evaluations across all four benchmarks during DAC-RL and CoT-RL training. **Middle:** For experiments using [Qwen3-4B-Instruct-2507](#), the training set is updated to the difficult subset after the 400th iteration, as detailed in Section 3.1. **Right:** Results of our Deep DAC training experiments, additionally including a comparison with the CoT-RL baseline using 32 rollouts in training.



**Figure 5:** An overall comparison among standard CoT-RL, Mix-RL, and DAC-RL trained on all problems. **Left:** Pass@1 accuracy on all benchmarks using CoT inference for both CoT-RL and Mix-RL. **Middle:** AIME score under the same settings as in Left. **Right:** Average performance across four competition-level benchmarks, where the CoT-trained policy uses CoT-style inference, while the other two employ DAC-style inference.

configuration with CoT-style RL training under an identical budget of 32 rollouts per training problem, with results shown in the right panel of Figure 4 and the bottom section of Table 1. It is worth noting that simply increasing the rollout budget for CoT reasoning during RL training does not yield performance improvements, while DAC-style training delivers a 4.7% improvement in Pass@32 over the CoT baseline. This underscores the superior scalability of DAC-style reasoning during both training and inference.

## 4. Analysis

### 4.1. Integrating CoT- and DAC-RL Training

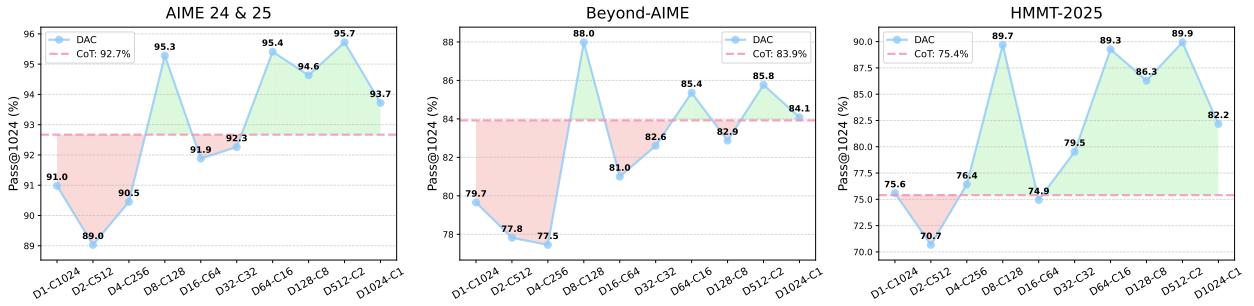
In this section, we experiment with integrating CoT- and DAC-RL training (**Mix-RL**), applying DAC-RL only to challenging problems while retaining CoT-style training for simpler ones, with experiments performed on [Qwen3-4B-Instruct-2507](#). Specifically, all batch problems are first answered in CoT-style rollouts, and those with accuracy below  $t_{acc} = 25\%$  are replaced by DAC-style solutions, which are mixed into the experiences with the reward assignment in Section 2. We compare this Mix-RL training with both the full CoT-RL and

DAC-RL strategies in Figure 5, from which we draw the following observations:

**DAC Training Enhances CoT Reasoning.** We find that substituting CoT training with DAC training on challenging problems can paradoxically enhance the policy’s CoT reasoning, even with reduced CoT training. As shown in Figure 5, the *Left* and *Middle* plots illustrate the intermediate performance under CoT-style inference for the CoT-RL and Mix-RL training across all benchmarks and on the AIME benchmarks, respectively. We find that further CoT-style RL training yields no additional gains, as the policy has already been extensively post-trained with CoT reasoning. However, under the Mix-RL setup, incorporating DAC-style training on challenging problems significantly enhances the model’s CoT performance—by over 10% across all benchmarks. These improvements highlight the effectiveness and robustness of DAC-style training in model’s reasoning capability.

**Mix-RL can Activate DAC Reasoning.** The DAC-style reasoning performance of Mix-RL is shown in Figure 5 (*Right*). Notably, applying DAC-style training only to complex problems can also equip the model with this reasoning capability, consistently surpassing its CoT counterpart on the benchmarks, which reinforces the motivation for adopting a DAC strategy for complex tasks. The inferior performance of Mix-RL relative to full DAC suggests that incorporating a broader range of problems into DAC training is generally beneficial for this advanced reasoning paradigm.

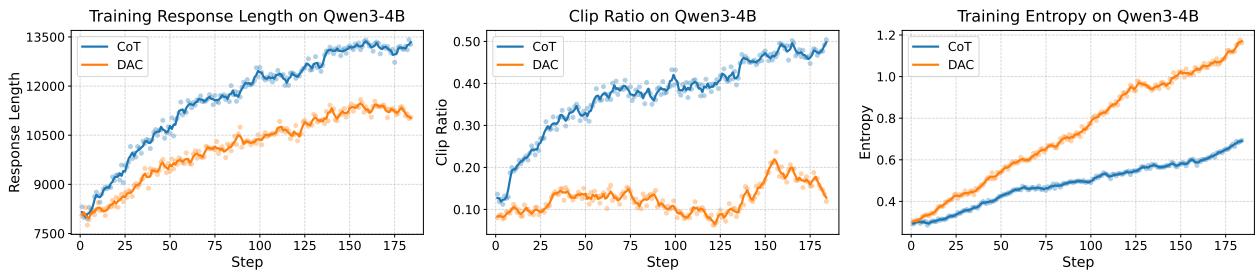
## 4.2. Test-time Scalability and Configurations



**Figure 6:** The *Pass@k* performance under different allocations between *division* and *conquering* with a fixed budget total of  $k = 1024$ . The CoT baseline corresponds to 1024 independent generations.

We evaluate test-time scalability and investigate the optimal configuration of DAC-style reasoning by measuring *Pass@k* under a fixed rollout budget of  $k = 1024$ , while varying the number of subproblem groups  $n$  and the number of conquering solutions per group  $m$  such that  $n \times m = k$ . As shown in Figure 6, allocating more groups (i.e., larger  $n$  and smaller  $m$ ) consistently improves performance on competition-level benchmarks compared to the CoT baseline. This indicates that increased subproblem diversity at test time expands the model’s exploration space, improving its chances to discover correct trajectories.

## 4.3. Concise and Diverse Reasoning with DAC



**Figure 7:** The batch-averaged response length, clip ratio of intermediate rollouts, and policy entropy in training.

Intuitively, guiding the model to generate responses to subproblems may introduce additional reasoning steps and increase output length. However, in our *Deep DAC* experiments, we observe that DAC training produces more concise reasoning than CoT-style RL. This results in fewer rollouts being clipped by the maximum length constraint, which may reduce false negatives arising from truncated positive rollouts, while lowers the per-step training time of DAC-RL. An example with detailed analysis is present in Appendix D.

Despite producing more compact training rollouts, DAC exhibits broader exploration, as reflected by its higher policy entropy compared with CoT in the right panel of Figure 7. This indicates that DAC-style reasoning compaction does not compromise exploration; instead, it removes redundant steps while preserving—and even enhancing—diversity in solution space exploration, thereby enabling more efficient training without premature convergence (Cui et al., 2025).

#### 4.4. Cold-Start Distillation for DAC-RL

Model	AIME 2024		AIME 2025		Beyond-AIME		HMMT 2025		Average	
	Pass@1	Pass@32								
Init-CoT	62.6	90.0	45.7	76.7	32.1	65.0	30.3	56.7	42.7	72.1
Init-DAC	59.6	90.0	43.2	73.3	29.6	61.0	28.2	63.3	40.2	71.9
CD-CoT	58.0	90.0	49.0	80.0	34.9	66.0	31.9	63.3	43.4	74.8
CD-DAC	60.2	86.7	53.5	83.3	35.9	68.0	35.9	66.7	46.4	76.2
CD-RL-CoT	66.5	<b>92.5</b>	60.5	88.2	41.2	66.1	<b>40.5</b>	71.8	52.2	79.7
CD-RL-DAC	<b>71.3</b>	91.7	<b>61.9</b>	<b>91.7</b>	<b>41.5</b>	<b>69.0</b>	39.3	<b>75.9</b>	<b>53.5</b>	<b>82.1</b>
$\Delta$ (RL)	+4.8	-0.8	+1.4	+3.5	+0.3	+2.9	-1.2	+4.1	+1.3	+2.4

Table 2: Experiments with cold-start (CD) initialization. For rows labeled “CD,” both CoT and DAC models are initialized via supervised distillation using the same amount of training samples, and models marked with “R” further undergo reinforcement learning as described in Section 2.

One concern when applying smaller models to DAC-RL training, as in our main experiments in Section 3.2, is their limited instruction-following capacity, which may hinder exploration during solution generation and policy optimization. Therefore, in this section, we investigate whether cold-starting DAC-style reasoning via distillation from a stronger instruction model can benefit subsequent RL training, and compare its performance with a CoT baseline under comparable settings. For distillation, we randomly select 3K problems from the *DeepMath-103K* dataset with difficulty labels greater than six and use a stronger instruction-following model, *Qwen3-235B-A22B-2507-Instruct*, to generate both CoT and DAC-style solutions, yielding a total of 6K problem-response pairs. We directly mix the CoT and DAC distillation data to form the training set, use *Qwen3-4B-Instruct-2507* for the experiment, set the maximum token length to 32K, and fine-tune the model for five epochs.

The experimental results are shown in Table 2, where we report model performance both after cold-starting and after subsequent RL fine-tuning. Overall, cold-start improves the performance of both DAC and CoT. Under an equal budget of 3k samples, the distilled model exhibits stronger DAC-style reasoning performance than CoT, even though the initial model performs better with CoT. After RL fine-tuning, the DAC model achieves a larger performance gain than the CoT baseline, with improvements increasing from 1.4% to 2.4%. The consistent improvements across both training paradigms indicate that DAC constitutes a more effective advanced reasoning paradigm. Moreover, the amplified gains observed after RL suggest that DAC enables richer and more diverse exploration during policy optimization, leading to more efficient performance improvement than standard CoT reasoning.

#### 4.5. The Effects of a Subproblem-Solving Format Constraint

In this section, we explore the impact of imposing a constraint that enforces the *conquering* solutions to explicitly solve subproblems one by one, assigning positive rewards only to solutions that adhere to this rule.

Model	Format Following	AIME 2024		AIME 2025		Beyond-AIME		HMMT 2025		Average	
		Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32	Pass@1	Pass@32
Init	72.1%	60.9	86.7	44.6	76.7	30.4	62.0	27.0	63.3	40.7	72.2
↳ RL w.o. FC	42.6%	66.3	91.6	61.5	87.6	38.8	70.7	38.7	76.4	51.3	81.6
↳ RL w.i. FC	92.1%	56.2	86.3	55.5	87.1	34.3	66.6	34.6	69.2	45.2	77.3

Table 3: Evaluating the alignment tax from enforcing strict subproblem answer formats in conquering solutions on the *format following rate* and *model performance*. *w.i. FC* refers to imposing the format constraint in RL.

Specifically, we require the responses to include all “subproblem  $i$ ” for  $i = 1, \dots, n_g - 1$ , where  $n_g$  denotes the number of subproblems generated during the divide stage. A response receives a positive reward only if it adheres to the constrained format with a correct final answer; otherwise, it receives a negative reward.

The results are reported in Table 2. Although the policy learns to explicitly answer all subproblems in a strict format—unlike models without this constraint—its evaluation performance is worse than models without this constraint. This finding aligns with the conclusions in (Lin et al., 2023; Wang et al., 2025) that training LLMs for alignment, especially under strict formatting constraints, can introduce an *alignment tax*, which enhances instruction-following behavior while degrading downstream performance.

## 5. Related Work

### 5.1. Divide-and-Conquer in LLM Reasoning

Divide-and-conquer is a fundamental algorithm design paradigm (Cormen et al., 2022). The core idea is to break a complex problem into smaller, manageable subproblems, solve them independently, and subsequently combine their solutions to derive the final answer. In LLM reasoning, several studies (Zhou et al., 2022; Ling et al., 2023; Khot et al., 2022; Huang et al., 2022; Dua et al., 2022; Chen et al., 2022; Zelikman et al., 2023; Ye et al., 2023; Xue et al., 2024; Meng et al., 2024) focus on decomposing complex problems into subproblems using prompting-based strategies on different levels of reasoning tasks. For example, Least-to-Most Prompting (Zhou et al., 2022) incorporates a few in-context examples to guide LLMs in generating a sequence of simpler subproblems from the original one. Another line of work (Yao et al., 2023; Besta et al., 2024; Chen et al., 2022; Yang et al., 2024b) intrinsically guides LLMs to decompose reasoning beyond the standard CoT (Wei et al., 2022) through mechanisms such as expansion, search, or reflection, while Yu et al. (2025b) proposes decomposing LLM reasoning into multiple stages.

Although effective, existing studies apply the DAC strategy only during inference, suffering from a misalignment between the DAC inference and general post-training, which primarily emphasizes direct problem answering (Yang et al., 2024a). Ladder (Simonds and Yoshiyama, 2025) proposes decomposing integral problems into simpler sub-tasks and incorporating the solutions into training. In this work, we train LLMs’ DAC reasoning with a unified RL framework and show that this paradigm offers a higher reasoning ceiling.

### 5.2. Reinforcement Learning for LLMs

Large-scale reinforcement learning (RL) has significantly improved LLMs in complex reasoning tasks (Luong et al., 2024; Guo et al., 2025). Algorithms such as PPO (Schulman et al., 2017), GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025a) have shown strong generalization and effectiveness in LLM post-training, while Wen et al. (2025) demonstrates that RLVR encourages correct reasoning. Existing efforts in scaling up RL optimization have focused on enhancing exploration (Yu et al., 2025a; Yuan et al., 2025; Liu et al., 2025; Yeo et al., 2025; Liang et al., 2025a; Huang et al., 2025) and adapting RL to long CoT (System 2 reasoning (Li et al., 2025)) conditions (Jaech et al., 2024; Guo et al., 2025). However, these methods overlook the limitation that models cannot improve on problems they consistently answer incorrectly under CoT reasoning, while we address this by integrating the DAC strategy into RL training, enabling the model to learn and solve them via subproblem decomposition.

---

## 6. Conclusion

In this paper, we conduct a comprehensive study of the divide-and-conquer (DAC) reasoning paradigm beyond standard CoT for LLMs in addressing complex reasoning problems. We identify a gap between models trained under general or CoT-centric post-training and their DAC reasoning capabilities at inference time, and propose enhancing DAC reasoning through RL-based training. The experiments shows that DAC-style reasoning demonstrates a higher performance upper bound and stronger test-time scalability than traditional step-by-step CoT, leading to improved performance on competition-level benchmarks.

## References

- Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawska, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczek, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690, 2024.
- ByteDance-Seed. BeyondAime: Advancing math reasoning evaluation beyond high school olympiads. <https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME>, 2025.
- Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, et al. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harry Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.
- Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
- Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.
- Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. Successive prompting for decomposing complex questions. *arXiv preprint arXiv:2212.04092*, 2022.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Fanding Huang, Guanbo Huang, Xiao Fan, Yi He, Xiao Liang, Xiao Chen, Qinting Jiang, Faisal Nadeem Khan, Jingyan Jiang, and Zhi Wang. Beyond the exploration-exploitation trade-off: A hidden state approach for llm reasoning in rlvr. *arXiv preprint arXiv:2509.23808*, 2025.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.
- 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.

- 
- Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.
- Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, et al. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*, 2025.
- Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Yelong Shen, Ying Nian Wu, and Weizhu Chen. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.08989*, 2025a.
- Xiao Liang, Zhongzhi Li, Yeyun Gong, Yelong Shen, Ying Nian Wu, Zhijiang Guo, and Weizhu Chen. Beyond pass@ 1: Self-play with variational problem synthesis sustains rlvr. *arXiv preprint arXiv:2508.14029*, 2025b.
- Yong Lin, Hangyu Lin, Wei Xiong, Shizhe Diao, Jianmeng Liu, Jipeng Zhang, Rui Pan, Haoxiang Wang, Wenbin Hu, Hanning Zhang, et al. Mitigating the alignment tax of rlhf. *arXiv preprint arXiv:2309.06256*, 2023.
- Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 36:36407–36433, 2023.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.
- Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning with reinforced fine-tuning. *arXiv preprint arXiv:2401.08967*, 3, 2024.
- MAA. American invitational mathematics examination (AIME). <https://maa.org/math-competitions/aime>, 2024.
- Zijie Meng, Yan Zhang, Zhaopeng Feng, Yang Feng, Gaoang Wang, Joey Tianyi Zhou, Jian Wu, and Zuozhu Liu. Divide and conquer for large language models reasoning. *CoRR*, 2024.
- ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanja Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Ning Shang, Yifei Liu, Yi Zhu, Li Lyra Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, et al. rstar2-agent: Agentic reasoning technical report. *arXiv preprint arXiv:2508.20722*, 2025.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Toby Simonds and Akira Yoshiyama. Ladder: Self-improving llms through recursive problem decomposition. *arXiv preprint arXiv:2503.00735*, 2025.
- Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

- 
- Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*, 2025.
- Shangzi Xue, Zhenya Huang, Jiayu Liu, Xin Lin, Yuting Ning, Binbin Jin, Xin Li, and Qi Liu. Decompose, analyze and rethink: Solving intricate problems with human-like reasoning cycle. *Advances in Neural Information Processing Systems*, 37:357–385, 2024.
- An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024a.
- Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024b.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
- Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 174–184, 2023.
- Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*, 2025.
- Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025a.
- Yiying Yu, Yuxiang Zhang, Dongdong Zhang, Xiao Liang, Hengyuan Zhang, Xingxing Zhang, Ziyi Yang, Mahmoud Khademi, Hany Awadalla, Junjie Wang, et al. Chain-of-reasoning: Towards unified mathematical reasoning in large language models via a multi-paradigm perspective. *arXiv preprint arXiv:2501.11110*, 2025b.
- Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.
- Eric Zelikman, Qian Huang, Gabriel Poesia, Noah Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions. *Advances in Neural Information Processing Systems*, 36: 31466–31523, 2023.
- Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

## A. Preliminaries on GRPO for LLMs

**Group Relative Policy Optimization (GRPO)** (Shao et al., 2024) is an efficient algorithm for reinforcement learning in LLMs, where the advantages for each token in a rollout are computed in a group-relative manner without requiring an additional critic model to estimate token values. Specifically, given an input prompt  $x$ , the policy model  $\pi_{\theta_{\text{old}}}$  generates a group of  $G$  responses  $\mathbf{Y} = \{y_i\}_{i=1}^G$ , with acquired rewards  $\mathbf{R} = \{r_i\}_{i=1}^G$ . The advantage  $A_{i,t}$  for each token in response  $y_i$  is computed as the group-normalized rewards:

$$A_{i,t} = \frac{r_i - \text{mean}(\{r_i\}_{i=1}^G)}{\text{std}(\{r_i\}_{i=1}^G)}. \quad (5)$$

To improve the stability of policy optimization, GRPO clips the probability ratio  $k_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})}$  within a trust region (Schulman et al., 2017), and constrains the policy distribution from deviating too much from the reference model using a KL term. The final optimization objective is defined as follows:

$$\mathcal{J}(\theta) = \mathbb{E}_{x \sim \mathcal{D}, \mathbf{Y} \sim \pi_{\theta_{\text{old}}}(\cdot|x)} \left[ \frac{1}{G} \sum_{i=1}^G \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \left( \min(k_{i,t}(\theta) A_{i,t}, \text{clip}(k_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon) A_{i,t}) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right) \right] \quad (6)$$

In this work, we incorporate several techniques from (Yu et al., 2025a) into training, including Clip-Higher and Token-Level Loss, all of which are widely adopted to enhance training efficiency.

## B. Proof for Lemma 2.1

Let  $s_i \in \{0, 1\}$  indicate whether the solution to subproblem  $i$  is correct, and let  $\mathbf{s} = (s_1, \dots, s_m)$  denote a realization of the random vector  $S = (S_1, \dots, S_m)$ . Let  $C \in \{0, 1\}$  indicate whether the final answer to the original problem is correct. The *conquering* inference first sequentially solves the  $m$  subproblems and then addresses the original problem, thereby inducing the causal structure  $\mathbf{s} \rightarrow C$ , where subproblem correctness determines the correctness of the final solution.

**Assumption 1: Monotonicity.**  $P(C = 1 | S = \mathbf{s})$  is nondecreasing in each coordinate of  $\mathbf{s}$  and strictly increasing in at least one coordinate. Intuitively, holding all other subproblem outcomes fixed, solving one additional subproblem correctly cannot decrease the probability of producing a correct final answer.

**Assumption 2: Policy control.** The policy  $\pi_\theta$  determines the distribution of subproblem correctness, i.e.,  $P_\theta(S = \mathbf{s})$  depends on  $\theta$ .

**Proof.** To analyze how optimizing for  $C = 1$  affects subproblem correctness, we expand  $P_\theta(C = 1)$  using the law of total probability:

$$P_\theta(C = 1) = \sum_{\mathbf{s} \in \{0,1\}^m} P(C = 1 | S = \mathbf{s}) P_\theta(S = \mathbf{s}) = \mathbb{E}_{S \sim P_\theta}[g(S)], \quad g(\mathbf{s}) := P(C = 1 | S = \mathbf{s}).$$

Under **Assumption 1**,  $g$  is an increasing function. Therefore, any policy update that increases  $P_\theta(C = 1)$  must, on average, allocate more probability mass to configurations with higher numbers of correct subproblems. For any  $i$ , Bayes' rule gives:

$$P_\theta(S_i = 1 | C = 1) = \frac{\sum_{\mathbf{s}: s_i=1} P(C = 1 | S = \mathbf{s}) P_\theta(S = \mathbf{s})}{P_\theta(C = 1)}.$$

Because  $P(C = 1 | S = \mathbf{s})$  is larger whenever  $s_i = 1$  (by **Assumption 1**), it follows that

$$P_\theta(S_i = 1 | C = 1) \geq P_\theta(S_i = 1),$$

with strict inequality whenever  $S_i$  influences  $C$ . Equivalently,

$$\text{Cov}_\theta(\mathbf{1}\{S_i = 1\}, \mathbf{1}\{C = 1\}) \geq 0.$$

---

Hence, conditioning on (and rewarding)  $C = 1$  preferentially upweights trajectories with more correct subproblems. Under **Assumption 1-2**, the scalar reward

$$\mathbf{R}(y_c) = \mathbf{1}\{\text{Extract}(y_c) = a\}, \quad \mathbb{E}_{y_c \sim \pi_\theta}[\mathbf{R}(y_c)] = P_\theta(C = 1),$$

serves as a consistent surrogate for subproblem correctness. The converse also holds for  $C = 0$ , where incorrect final answers correspond to lower subproblem correctness.

### Generate Solution Steps Instead of Subproblems

> > > Problem:

Add 53.463 to 12.9873 and round to the nearest thousandth.

> > > Divide Response:

<SUBPROBLEM 1>

This subproblem is about adding the two numbers: 53.463 and 12.9873.

53.463 + 12.9873 = 66.4503

</SUBPROBLEM 1>

<SUBPROBLEM 2>

Now, we need to round 66.4503 to the nearest thousandth. The thousandth place is the third digit after the decimal point. Looking at the fourth digit (which is 3 in this case), we determine if we need to round up or stay the same.

66.4503 rounded to the nearest thousandth is 66.450 since the fourth digit (3) is less than 5.

</SUBPROBLEM 2>

<SUBPROBLEM 3>

So, rounding 66.4503 to the nearest thousandth gives us 66.450.

</SUBPROBLEM 3>

<SUBPROBLEM 4>

We have confirmed that 66.4503 rounded to the nearest thousandth is 66.450.

The final answer is 66.450.

</SUBPROBLEM 4>

<SUBPROBLEM 5>

Thus, the final rounded number is 66.450.

</SUBPROBLEM 5>

**Figure 8:** A case that the policy generates intermediate solution steps instead of dividing subproblems.

## C. Discussion on Conquering Accuracy as Feedback for Division Reward

Intuitively, rewarding the divided subproblems based on how effectively they help the policy solve the original problem encourages the model to generate higher-quality subproblems. Motivated by this intuition, our initial implementation adopts a straightforward strategy: using the average accuracy of the  $G_c$  conquer solutions given the subproblem group  $\mathcal{P}_g$  as the reward for divide responses instead of Equation 2, denoted as  $\frac{1}{|\mathcal{G}_c|} \sum_{\{y_{g,v}\}_{v=1}^{G_c}} \{\mathbf{R}(y_{g,v}) \mid \mathcal{P}_g\}_{v=1}^{G_c}$ . However, the experiments show that when conquer-accuracy-based rewards are used to reward subproblems division, the model tends to solve the problem prematurely during the *division* stage (as illustrated in Case 8), rather than decomposing it into subproblems. This indicates that when the model discovers outputs that better support subsequent conquer-stage solution generation—i.e., yield higher rewards—it may ignore the instructions specified in the prompt. *More generally, these results suggest that when prompt instructions substantially conflict with the reward assignment scheme in RL training, the policy tends to prioritize reward maximization over strict adherence to the instructions, highlighting the importance of aligning prompt design with reward specification in RL settings.* This initial failure motivates us to adopt a relaxed division reward that guarantees only a lower bound on subproblem helpfulness, rather than strictly optimizing for exact accuracy, as formalized in Equation 2. This relaxation reduces greedy behavior in early training and prevents the policy from prematurely optimizing for the original problem, thereby preventing this failure mode.

## D. A Case Study on How DAC Reduces Reasoning Redundancy

In this section, we present a case study based on a problem from the DAPO-Math-17k dataset. The original problem and its decomposed subproblems under DAC reasoning are shown in Figure 9. As the full responses of both reasoning styles are lengthy, we present partial solutions targeting the same linear system to clearly distinguish the two reasoning styles and highlight the compression effect of DAC. Partial reasoning traces produced by DAC and CoT are shown in Figures 10 and 11, respectively, with the corresponding full solutions in Listings 1 (3,328 tokens) and 2 (5,072 tokens).

Both DAC and CoT address the same algebraic structure by introducing auxiliary variables to linearize and solve the original system. However, the DAC sub-solution adheres to a predefined subproblem decomposition, mapping each subproblem directly to a necessary transformation or computation, thereby avoiding redundant restatements and self-corrections. In contrast, CoT reasoning involves *repeated verification and re-derivation* of intermediate results, which leads to longer and less structured outputs. Consequently, DAC compresses reasoning primarily by reducing narrative redundancy rather than simplifying the underlying mathematics, resulting in shorter, more stable, and more token-efficient reasoning traces.

### The Original Problem and Divided Subproblems

#### > > > Original Problem:

Let  $a, b, c$  be positive real numbers for which:

$$\frac{5}{a} = b + c, \quad \frac{10}{b} = c + a, \quad \text{and} \quad \frac{13}{c} = a + b.$$

If  $a + b + c = \frac{m}{n}$  for relatively prime positive integers  $m$  and  $n$ , compute  $m + n$ .

#### > > > Divided Subproblems:

<SUBPROBLEM 1>

Rewrite the given equations into an equivalent algebraic system involving  $a, b$ , and  $c$ .

</SUBPROBLEM 1>

<SUBPROBLEM 2>

Transform the system into a simpler form by introducing appropriate substitutions or auxiliary variables.

</SUBPROBLEM 2>

<SUBPROBLEM 3>

Solve the resulting system to determine the values of the substituted variables.

</SUBPROBLEM 3>

<SUBPROBLEM 4>

Recover the values of  $a, b$ , and  $c$  from the solved variables.

</SUBPROBLEM 4>

<SUBPROBLEM 5>

Compute  $a + b + c$  from the obtained solution and express it in lowest terms.

</SUBPROBLEM 5>

**Figure 9:** The original problem from DAPO-Math-17k and the decomposed subproblems in DAC reasoning.

### Partial Response from DAC Reasoning

Now denote the variables: let:

- $x = ab$
- $y = bc$
- $z = ac$

But notice that now:

- (1')  $\rightarrow ab + ac = x + z = 5$
- (2')  $\rightarrow ab + bc = x + y = 10$
- (3')  $\rightarrow ac + bc = z + y = 13$

Now we have a simpler linear system in terms of  $x, y, z$ :

$$\begin{cases} x + z = 5 \\ x + y = 10 \\ y + z = 13 \end{cases}$$

Add all three equations:

Left-hand side:  $x + z + x + y + y + z = 2x + 2y + 2z = 2(x + y + z)$

Sum of RHS =  $5 + 10 + 13 = 28$

Thus:

$$2(x + y + z) = 28 \Rightarrow x + y + z = 14$$

Now use individual equations:

- $x + z = 5 \rightarrow x = 5 - z$
- From this, plug into  $x + y = 10$ :  $5 - z + y = 10 \rightarrow y = 5 + z$
- Now use  $y + z = 13$ : left side becomes  $5 + z + z = 5 + 2z = 13 \rightarrow 2z = 8 \rightarrow z = 4$

Then  $x = 5 - z = 1$ ,  $y = 5 + 4 = 9$

So:

- $ab = x = 1$
- $bc = y = 9$
- $ac = z = 4$

**Figure 10:** A partial solution produced during DAC conquering for the problem illustrated in Figure 9.

#### Partial Response from CoT Reasoning

So now, we have:

From first:  $ab + ac = 5$

From second:  $ab + bc = 10$

From third:  $bc + ca = 14$ ? Wait no.

Wait solve carefully:

Wait first:  $ab + ac = 5$

Second:  $ab + bc = 10$

Third: from  $c(a + b) = 13$ :  $c(a + b) = ca + cb = 13$

So indeed:

- $ab + ac = 5 \rightarrow$  (i)
- $ab + bc = 10 \rightarrow$  (ii)
- $bc + ca = 13 \rightarrow$  (iii)

But wait — we previously found  $ab + bc + ca = 14$

Wait now we have:

Add all three:

(i)+(ii)+(iii):

Left:  $ab + ac + ab + bc + bc + ca$

$$= 2ab + 2bc + 2ca$$

So:

$$2(ab + bc + ca) = 5 + 10 + 13 = 28$$

So  $2(ab + bc + ca) = 28 \rightarrow ab + bc + ca = 14 \rightarrow$  matches what we got earlier.

Good — consistent.

Now, let's label:

Let:

- $x = ab$
- $y = bc$
- $z = ca$

Then from above:

- $x + z = 5 \rightarrow$  (from  $ab+ac=ab+ac=x+z=5$ )
- $x + y = 10$
- $y + z = 13$

And  $x + y + z = 14$

Now solve this system.

From:

- (1)  $x + z = 5$
- (2)  $x + y = 10$
- (3)  $y + z = 13$

Subtract (1) from (2): (2)-(1):  $(x + y) - (x + z) = 10 - 5 \Rightarrow y - z = 5$

From (3):  $y + z = 13$

Now solve:

From (3) and  $(y-z=5)$ :

Add them:  $(y + z) + (y - z) = 13 + 5 = 18 \Rightarrow 2y = 18 \Rightarrow y = 9$

Then from  $y + z = 13$ ,  $z = 13 - 9 = 4$

From (2):  $x=10 - y = 10 - 9 = 1$

From (1):  $x + z = 1+4=5$ : yes, checks.

So:

- $ab = 1$
- $bc = 9$
- $ca = 4$

Nice. So now we know:

- $ab = 1$
- $bc = 9$
- $ca = 4$

**Figure 11:** A partial solution generated by CoT-style reasoning for the problem illustrated in Figure 9.

## E. Prompts

We present the prompts used in our framework, including the *division* prompt and the *conquering* prompt, in Figures 12 and 13. During both RL training and inference, the subproblem placeholders in the *conquering* prompt are replaced with the actual decomposed subproblems, following the format “<SUBPROBLEM i> ··· </SUBPROBLEM i>” as illustrated in Figure 13.

**The Division Prompt**

You are given the following mathematical problem:

{REPLACE}

Your task is NOT to solve the problem. Instead, break it down into **more than 3 subproblems** that, if solved, would naturally lead to the solution of the original problem.

Please follow this output format strictly (include more than 3 subproblems as appropriate):

```

<SUBPROBLEM 1>
<first subproblem>
</SUBPROBLEM 1>

<SUBPROBLEM 2>
<second subproblem>
</SUBPROBLEM 2>

<SUBPROBLEM 3>
<third subproblem>
</SUBPROBLEM 3>

<SUBPROBLEM 4>
<fourth subproblem, if needed>
</SUBPROBLEM 4>

<SUBPROBLEM 5>
<fifth subproblem, if needed>
</SUBPROBLEM 5>
```

Do not provide any explanations or answers. Only list the subproblems using the tags <SUBPROBLEM x> and </SUBPROBLEM x> as shown.

**Figure 12:** The *division* prompt for dividing the original problem into subproblems.

**The Conquering Prompt**

You are given a challenging math problem and its decomposition into subproblems.

## Your task is to:

1. Solve the \*\*subproblems sequentially\*\* (in the given order).
2. Show detailed reasoning for each subproblem.
3. \*\*Do not use ‘\boxed’ when solving subproblems.\*\*
4. After solving all subproblems, combine their results to solve the \*\*original problem\*\*.
5. Present the final answer to the original problem \*\*only once\*\*, enclosed in `\\boxed{}`.

## Input

\*\*Original Problem:\*\*  
{REPLACE}

\*\*Subproblems:\*\*

```

<SUBPROBLEM 1>
<first subproblem>
</SUBPROBLEM 1>

<SUBPROBLEM 2>
<second subproblem>
</SUBPROBLEM 2>

<SUBPROBLEM 3>
<third subproblem>
</SUBPROBLEM 3>

<SUBPROBLEM 4>
<fourth subproblem, if needed>
</SUBPROBLEM 4>

<SUBPROBLEM 5>
<fifth subproblem, if needed>
</SUBPROBLEM 5>
```

—

## Final Instruction

- \* First, go through the subproblems one by one with detailed reasoning.
- \* Finally, derive the answer to the \*\*original problem\*\* and present it in the format:

```

$$
\boxed{\text{final answer here}}
$$
```

**Figure 13:** The *conquering* prompt for solving the subproblems and the original problem.

Listing 1: The full solution generated during DAC conquering for the problem illustrated in Figure 9.

We are given that  $a, b, c > 0$  are real numbers satisfying:

```
$$
\frac{5}{a} = b+c \tag{1}
$$
$$
\frac{10}{b} = c+a \tag{2}
$$
$$
\frac{13}{c} = a+b \tag{3}
$$
```

Our goal is to determine  $a + b + c$  in the form  $\frac{m}{n}$  with  $\gcd(m, n) = 1$ , then compute  $m+n$ .

---

## <SUBPROBLEM 1> Rewrite the given equations into an equivalent algebraic system involving  $a$ ,  $b$ , and  $c$ .

Start by treating this system algebraically as a consistent system with three equations and three variables.

We are to solve for three positive real numbers  $a, b, c$  satisfying:

```
$$
(b+c)a = 5 \Rightarrow ab + ac = 5 \tag{1'}
$$
$$
(c+a)b = 10 \Rightarrow bc + ab = 10 \tag{2'}
$$
$$
(a+b)c = 13 \Rightarrow ac + bc = 13 \tag{3'}
$$
```

Now we have the symmetric-looking expressions.

We have:

```
(1') $ ab + ac = 5 $
(2') $ ab + bc = 10 $
(3') $ ac + bc = 13 $
```

Now denote the variables: let:

```
- $ x = ab $
- $ y = bc $
- $ z = ac $
```

But notice that now:

```
- (1') -> $ ab + ac = x + z = 5 $
- (2') -> $ ab + bc = x + y = 10 $
- (3') -> $ ac + bc = z + y = 13 $
```

Now we have a simpler linear system in terms of  $x, y, z$ :

```
$$
\begin{cases}
x + z = 5 \\
x + y = 10 \\
y + z = 13
\end{cases}
$$
```

Add all three equations:

Left-hand side:  $x + z + x + y + y + z = 2x + 2y + 2z = 2(x+y+z)$

Sum of RHS =  $5 + 10 + 13 = 28$

Thus:

```
$$
2(x+y+z)=28 \Rightarrow x+y+z=14
$$
```

Now use individual equations:

```
- $ x + z = 5 $ -> $ x = 5 - z $
- From this, plug into $ x + y = 10 $: $ 5 - z + y = 10 $ -> $ y = 5 + z $
- Now use $ y + z = 13 $: left side becomes $ 5 + z + z = 5 + 2z = 13 $
-> $ 2z = 8 $ -> $ z = 4 $
```

Then  $x = 5 - z = 1$ ,  $y = 5 + 4 = 9$

So:

```
- $ ab = x = 1 $
```

---

```

- $ bc = y = 9 $
- $ ac = z = 4 $

Now use these derived values.

From:
- $ ab = 1 $ -> $ b = \frac{1}{a} $
- $ ac = 4 $ -> $ c = \frac{4}{a} $
- $ bc = 9 $

Let's now compute $ bc $ using those:
$ b = 1/a $, $ c = 4/a \Rightarrow bc = (1/a)(4/a) = 4 / a^2 $

But from equation, $ bc = 9 \Rightarrow \frac{4}{a^2} = 9 \Rightarrow a^2 = \frac{4}{9} \Rightarrow a = \frac{2}{3} $, since $ a>0 $

Now we can get:
- $ b = \frac{1}{a} = \frac{1}{\frac{2}{3}} = \frac{3}{2} $
- $ c = \frac{4}{a} = \frac{4}{\frac{2}{3}} = 6 \cdot \frac{3}{2} = 6 $

### Check if all equations hold

Now compute:
1. $ \frac{5}{a} = 5 / (2/3) = 5 \cdot 3/2 = 15/2 = 7.5 $

Left: $ b + c = 3/2 + 6 = 3/2 + 12/2 = 15/2 \rightarrow OK

2. $ \frac{10}{b} = 10 \div (3/2) = 10 \cdot 2 / 3 = 20/3 \approx 6.666 $

Left: $ c + a = 6 + 2/3 = (18+2)/3 = 20/3 \rightarrow OK

3. $ \frac{13}{c} = 13 / 6 \approx 2.1666 $

Left: $ a + b = 2/3 + 3/2 = $ find common denominator:
LCM of 3 and 2 is 6 -> $ \frac{4}{6} + \frac{9}{6} = 13/6 \rightarrow Yes

So left = 13/6, right = 13 / 6 -> OK

Perfect.

### Final values:
$$
a = \frac{2}{3}, b = \frac{3}{2}, c = 6
$$

Now compute $ a + b + c $:

Compute sum:
First write everything over 6:
- $ a = 2/3 = 4/6 $
- $ b = 3/2 = 9/6 $
- $ c = 6 = 36/6 $

Sum: $ \frac{4+9+36}{6} = \frac{49}{6} $

So $ a + b + c = \frac{49}{6} $

Now since $ \gcd(49, 6) = 1 $, this fraction is already reduced.

Then $ m = 49, n = 6 \rightarrow $ so answer is $ m+n = 49 + 6 = 55 $

Wait - this seems to directly solve the problem.

But to ensure completeness, let's go through the other subproblems to confirm structure and uniqueness.

---

## <SUBPROBLEM 2> Transform the system into a simpler form by introducing appropriate substitutions or auxiliary variables.

This question is really looking to see whether an interpretation in geometric terms, such as triangle-like inequalities, symmetry structures, monotonicity, etc., exists.

But observe:
The equations involve symmetry broken by coefficients: 5,10,13 -> not symmetric.
Equations like:
- $ 5/a = b + c \Rightarrow b + c > 0 $ (always true), and $(b + c)$ is inversely proportional to $a \rightarrow$ so if $a$ is small, $b + c$ is large -> but constrained.

Interestingly, we have:
```

Each variable appears in an inversion (via reciprocal), such as  $1/a$  is proportional to  $(b + c)$ . So all variables appear in such reciprocal-sum form.

We might interpret this as a symmetric-looking "harmonic-type" system.

Think about coefficients:

If  $a, b, c$  were symmetrically related, total expressions might balance.

But here the constants: 5,10,13  $\rightarrow$  increasing  $\rightarrow$  related to  $\frac{5}{a}, \frac{10}{b}, \frac{13}{c}$

So the reciprocals have degrees increasing with index.

Alternatively, for such problems involving harmonic variables, it's common to transform into symmetric forms.

In fact, our solution found in subproblem 1 completely worked out algebraically.

So geometric or structural insight might not be essential; instead this problem is algebraic.

But note the \*\*structure\*\*:

Define  $ab + ac = 5$ , etc. - each pair of products involving product between two variables equals something.

Indeed, such expressions (sums of pairwise products) with three variables suggest symmetric systems.

That's why we used substitutions.

In such systems, under positivity, solutions may be unique.

Moreover, as the functions involved are rational, and the equations are algebraic, and only one solution exists (we found one via direct substitution), we may expect uniqueness.

But we'll confirm.

This soul-bond between pairs may suggest a symmetric substitution.

Note that when we treated in terms of  $ab, bc, ca$ , we got linear equations - very structured.

So the structure is algebraic and symmetric in form with triple product structure. In particular, the system is symmetric under reindexing where the input constraints (5,10,13) are associated with  $a,b,c$  respectively -not symmetric.

So not symmetric.

But the symmetric-like pairing ( $ab + ac, ab + bc, ac + bc$ ) does suggest a linear algebraic structure.

We derived the three values uniquely under  $a > b > c$  (verify:  $a = 2/3 = 0.67, b = 1.5, c = 6 \rightarrow c > b > a \rightarrow$  actually monotonically decreasing in value), and all values positive, so uniquely determined via constraints.

So, structural constraints include:

- All variables  $> 0 \rightarrow$  positivity applies strictly because all terms positive.
- Substitution leads to a system solvable via linear system in products.
- Each of the pairwise products has unique definition under positivity  $\rightarrow$  forcing unique solution.

So no ambiguity.

Note also that the total sum  $S = a + b + c$  is uniquely determined.

So there can be only one such positive solution.

So geometric picture - maybe not meaningful?

Alternatively, perhaps there's a physical interpretation: like resistors or harmonic mean -but seems forced.

Rather than deep geometry, we see the equations impose independent constraints.

So key structural insight:

- The problem is constrained to three variables with three equations involving symmetric pairwise products via linear conversion.
- The constraints mix variables in such ways that the variables can be uniquely solved under positivity.

Also, each expression is proportional to sum of the \*other two variables\*:

- E.g.,  $5/a = b + c \rightarrow$  so  $b$  and  $c$  are parts forming  $5/a \rightarrow$  similar to "sum of two variables is reciprocal"

This suggests that each sum is related to the reciprocal of variable.

Think of it in terms of harmonic means:

We know  $\frac{1}{x} = \frac{1}{a}$ , and  $b + c \propto \frac{1}{a} \Rightarrow \frac{1}{a} = \frac{b+c}{5} = \frac{1}{a} \Rightarrow \frac{1}{a} \propto \text{sum of } b+c$

This suggests some harmonic-like character (a variable's reciprocal is linear in sum of others), hence it's similar to dual-type identities.

---

Nothing strong geometric, so likely algebraic in nature and solved directly as done.

---

## <SUBPROBLEM 3> Solve the resulting system to determine the values of the substituted variables.

Note that we are only asked for such an \$ a,b,c \$: and it is uniquely determined under positivity, since:

Since all variables are positive and the constraint equations are algebraic and fully determined, \*\*the solution must be unique\*\*.

Moreover, the function \$ f(a,b,c) = a + b + c \$ is strictly minimized or maximized not really, because we are asked to find one such triple, and it's uniquely determined.

Moreover, if we tried to perturb any variable slightly, the constraint will be broken - because the functions are strictly decreasing or increasing nonlinearly.

For example: The function \$ \frac{5}{a} \$ is strictly decreasing in \$ a \$, while sum \$ b + c \$ is increasing when \$ a \$ falls.

But in practice, we have determined a stable point.

So this triple is structural - the only satisfying one for positive reals.

Therefore, minimal or maximal configuration - such as minimal sum? But already the sum is uniquely fixed.

So parts:

- There is \*\*only one feasible solution for positive reals\*\*, hence no minimal or maximal; only one.

So such configurations are unique.

Hence, under constraints, only one such triple.

So answer so far is well-defined and unique.

---

## <SUBPROBLEM 4> Recover the values of \$a\$, \$b\$, and \$c\$ from the solved variables.

Done completely above, via algebra.

We found:

- \$ a = \frac{2}{3}, b = \frac{3}{2}, c = 6 \$

We already verified the constraints and the sum is:

\$\$ a + b + c = \frac{2}{3} + \frac{3}{2} + 6 \$\$

We can compute step-by-step:

Convert all to sixths:

- \$ \frac{2}{3} = \frac{4}{6} \$
- \$ \frac{3}{2} = \frac{9}{6} \$
- \$ 6 = \frac{36}{6} \$

Add  $\rightarrow \frac{4+9+36}{6} = \frac{49}{6}$

Thus:

\$ a + b + c = \frac{49}{6}\$

So the sum is \$ \frac{49}{6} \$, a fraction in lowest terms.

So this is the requested value for \$ m, n \$.

---

## <SUBPROBLEM 5> Compute \$a+b+c\$ from the obtained solution and express it in lowest terms.

We address uniqueness.

We have 3 equations in 3 variables.

The equations were:

- (1): \$ ab + ac = 5 \$
- (2): \$ ab + bc = 10 \$
- (3): \$ ac + bc = 13 \$

In effect: three equations involving the symmetric products \$ ab, bc, ac \$.

We solved the linear system for \$ ab, bc, ac \rightarrow \$ that system was triangular, with unique solution.

---

Then from  $ab=1$ ,  $bc=9$ ,  $ac=4$ , and from symmetry of variables, we derived only one positive solution (all variables positive), due to unilateral constraints (only positive reals allowed).

Suppose we try a negative variable  $\rightarrow$  but variables are defined as positive  $\rightarrow$  all constraints violated.

For instance, if one of the variables were negative,  $b+c$  could be negative or positive - but  $\frac{a}{b+c} > 0$ , so RHS must be positive  $\rightarrow$  so each side is positive  $\rightarrow$  so  $b+c > 0$  etc.

So all values are forced to be positive.

And each equation contains two variables in products (whose sum is linear), so movement in one variable entails constraints on others.

Now, we derived all entries from linear system uniquely.

So the entire system, given positivity and real numbers, has only one such triple.

Therefore:

- \*\*Uniqueness\*\*: yes - due to linearly independent equations in the pairwise products; and the positivity constraints.

- \*\*Existence\*\*: we have one such triple.

- Conditions: positivity, and algebraic equation solvability as above.

Hence, the solution is uniquely determined.

---

## Final Computation

We have found that, under the constraints, the only triple of positive reals is:

$a = \frac{2}{3}$ ,  $b = \frac{3}{2}$ ,  $c = 6$

Sum:  $\frac{49}{6}$

Express as reduced fraction: already reduced since  $\gcd(49, 6) = 1$

So  $m = 49$ ,  $n = 6$ ,  $m+n = 55$

---

\$\$\boxed{55}\$\$

---

Listing 2: The full solution generated by CoT-style reasoning for the problem illustrated in Figure 9.

We are given the following system of equations with positive real numbers  $a, b, c$ :

\$\$\begin{aligned} \frac{5}{a} &= b + c \quad \text{(1)} \\ \frac{10}{b} &= c + a \quad \text{(2)} \\ \frac{13}{c} &= a + b \quad \text{(3)} \end{aligned}\$\$

Our goal is to find  $a + b + c = \frac{m}{n}$ , with  $m, n$  relatively prime positive integers, and then compute  $m+n$ .

---

### Step 1: Set  $S = a + b + c$

From (1):  
 $\frac{5}{a} = b + c$

But  $b + c = S - a$ , so:

\$\$\frac{5}{a} = S - a \quad \text{(1')}

Multiply both sides by  $a$ :

\$\$\begin{aligned} 5 &= a(S - a) = aS - a^2 \\ \Rightarrow aS - a^2 &= 5 \quad \text{(A)} \end{aligned}\$\$

Similarly from (2):

$\frac{10}{b} = c + a = S - b$

---

So:

```
$$
\frac{10}{b} = S - b \Rightarrow 10 = b(S - b) = bS - b^2 \quad \text{(2')}
$$
```

\$\$
bS - b^2 = 10 \quad \text{(B)}
\$\$

And from (3):

$$\frac{13}{c} = S - c \Rightarrow 13 = c(S - c) = cS - c^2 \quad \text{(3')}$$

\$\$
cS - c^2 = 13 \quad \text{(C)}
\$\$

So we now have:

- (A):  $aS - a^2 = 5$
- (B):  $bS - b^2 = 10$
- (C):  $cS - c^2 = 13$

We can solve each for  $a^2, b^2, c^2$  or directly solve for  $a, b, c$  in terms of  $S$ , then sum them.

---

### Step 2: Express  $a, b, c$  from each equation

From (A):

```
$$
aS - a^2 = 5 \Rightarrow a^2 - aS + 5 = 0 \quad \text{wait: sign}
$$
```

Wait:  $aS - a^2 = 5 \Rightarrow -a^2 + aS - 5 = 0 \Rightarrow a^2 - aS + 5 = 0$ ? No: signs reversed.

Wait:

Equation:  
 $aS - a^2 = 5$

Bring all to left:

$$aS - a^2 - 5 = 0 \Rightarrow -a^2 + aS - 5 = 0$$

Multiply both sides by  $-1$ :

$$a^2 - aS + 5 = 0$$
? Wait:

No:

Negative times:  $a^2 - aS + 5 = 0$ ? Let's plug in.

Actually:

$$-a^2 + Sa - 5 = 0$$

Multiply both sides by  $-1$ :

$$a^2 - Sa + 5 = 0$$

Wait, that would be  $a^2 - Sa + 5 = 0$ , but original is  $Sa - a^2 = 5$

Check:  
Left side:  $Sa - a^2$ , then  $Sa - a^2 - 5 = 0 \Rightarrow -a^2 + Sa - 5 = 0 \Rightarrow$  multiply by  $-1$ :  $a^2 - Sa + 5 = 0$

But that means  $a^2 - Sa + 5 = 0$

Wait, but if real roots: discriminant  $S^2 - 4*1*5 = S^2 - 20$

Can't be negative. Since  $a > 0$ , so that's okay.

But go back - is this equation right?

Wait:

Equation (A):  $Sa - a^2 = 5$

Then  $-a^2 + Sa - 5 = 0$

---

Yes  $\rightarrow$  multiply both sides by -1:  $a^2 - Sa + 5 = 0$   
 Wait - that's inconsistent: from  $Sa - a^2 = 5$ , then  $Sa - a^2 = 5$ .  
 Then  $-a^2 + Sa = 5$ .  
 Then rearranged:  $a^2 - Sa = -5$ ? Yes.  
 That's better.  
 Wait, I think I made a sign error in the switch.  
 So instead:  
 From (A):  
 $Sa - a^2 = 5$   
 $\rightarrow -a^2 + Sa - 5 = 0$   
 Multiply through by -1:  
 $a^2 - Sa + 5 = 0$ ? Wait  $\rightarrow$  no:  
 $-(a^2 - Sa - 5) = a^2 - Sa + 5$ ? Wait: only if we negate the entire equation.  
 Yes.  
 If  $-a^2 + Sa - 5 = 0$ , then multiplying by -1:  
 $a^2 - Sa + 5 = 0$   
 Wait but original is positive. So if a solution exists, check.  
 But for small  $S$ , maybe  $a$  is small.  
 Wait, better not to multiply - keep as:  
 From  $Sa - a^2 = 5$   
 We can solve quadratic:  
 $a^2 - Sa + 5 = 0$ ? Wait, no.  
 Wait:  
 $-a^2 + Sa - 5 = 0$   
 Multiply both sides by -1:  
 $a^2 - Sa + 5 = 0$   
 Yes.  
 So quadratic:  

$$a^2 - Sa + 5 = 0 \quad \text{(A')}$$
  
 Wait - if this is correct, then roots of  $a^2 - Sa + 5 = 0$   
 But this means  $a = \frac{\pm \sqrt{S^2 - 20}}{2}$   
 Similarly, for (B):  $bS - b^2 = 10$   
 So  $-b^2 + bS - 10 = 0$   
 $\rightarrow$  Multiply by -1:  $b^2 - Sb + 10 = 0$   
 $\rightarrow b = \frac{\pm \sqrt{S^2 - 40}}{2}$   
 Similarly (3):  $cS - c^2 = 13$   
 $\rightarrow c^2 - Sc + 13 = 0$   
 Wait, let's check:  
 From  $cS - c^2 = 13$   
 $\rightarrow -c^2 + cS = 13$   
 $\rightarrow c^2 - cS + 13 = 0$ ? No:  
 Again: move things:  
 $-c^2 + Sc - 13 = 0$   
 $\rightarrow$  multiply by -1:  $c^2 - Sc + 13 = 0$   
 Wait same process.  
 Wait  $\rightarrow$  from  $Sc - c^2 = 13$

---

```

-> $ -c^2 + Sc -13 =0 $
Multiply by -1: $ c^2 - Sc +13 =0 $
Wait -> so yes:
We get quadratics:
- $ a^2 - S a + 5 = 0 $ -> (A')
- $ b^2 - S b + 10 = 0 $ -> (B')
- $ c^2 - S c + 13 = 0 $ -> (C')

Wait - but that suggests a, b, c satisfy such quadratics, but we also know $ a + b + c = S $

So from each:

We solve for $ a, b, c $:
- $ a = \frac{S \pm \sqrt{S^2 - 20}}{2} $
- $ b = \frac{S \pm \sqrt{S^2 - 40}}{2} $
- $ c = \frac{S \pm \sqrt{S^2 - 52}}{2} $

Wait, because quadratic:
$ x^2 - Sx + k = 0 $ -> solutions $ x = \frac{S \pm \sqrt{S^2 - 4k}}{2} $

Wait, in (A): equation: $ a^2 - Sa + 5 = 0 $ -> so discriminant $ S^2 - 4*1*5 = S^2 - 20 $, so roots:
$ a = \frac{S \pm \sqrt{S^2 - 20}}{2} $

Similarly, for (B): $ S^2 - 40 $

c: $ S^2 - 52 $

But since $ a,b,c > 0 $, we must pick the correct sign.

Let us see: Assume that values are less than S (since they are positive). Since $ a + b + c = S $, and all positive, each must be less than S.

Now, the roots of $ x^2 - Sx + k = 0 $:
The sum is $ S $, so average is $ S/2 $. For each pair.

So the roots are symmetric about $ S/2 $.

But we have $ a = \frac{S \pm \sqrt{S^2 - 20}}{2} $, with $ d_a = \sqrt{S^2 - 20} $

Let's calculate both roots:
$ a_+ = \frac{S + \sqrt{S^2 - 20}}{2} $
$ a_- = \frac{S - \sqrt{S^2 - 20}}{2} $

Now, since $ a < S $, and both roots are less than S? First, $ a_+ < S $? Yes.

But is $ a_+ > S/2 $? Yes.

We need $ a + b + c = S $. So now suppose we pick all roots equal to the minus version? Unlikely.

But since $ a, b, c > 0 $, and sum to $ S $, we need $ a, b, c < S $.

Now, the two roots: one is $ S/2 + d/2 $, one $ S/2 - d/2 $

Since $ d = \sqrt{S^2 - 4k} $, and $ k > 0 $, small.

But if we take the larger root: for a, $ a_+ > S/2 $

Likewise, $ a_- < S/2 $

Similarly for others.

But if we take the larger root for all three, their total sum can be?

Wait: suppose we take:
a: $ \frac{S + \sqrt{S^2 - 20}}{2} $
b: $ \frac{S + \sqrt{S^2 - 40}}{2} $
c: $ \frac{S + \sqrt{S^2 - 52}}{2} $

Then sum:
$ S/2 * 3 + \frac{1}{2}(\sqrt{S^2 - 20} + \sqrt{S^2 - 40} + \sqrt{S^2 - 52}) = $
= $ \frac{3S}{2} + \frac{1}{2}(\dots) $

But this would be $ \frac{3S}{2} > S $ unless $ S < 0 $ - not possible.

```

So impossible. So cannot pick all as larger root.

Similarly, if pick all as smaller roots:

```
$ a_- = \frac{S - \sqrt{S^2 - 20}}{2}
```

Then sum:

```
$ \frac{3S}{2} - \frac{1}{2}(\sqrt{S^2 - 20} + \sqrt{S^2 - 40} + \sqrt{S^2 - 52})
```

This sum is likely less than  $S$ , since the subtracted term is positive.

So we need to pick two big and one small, or other combinations.

But maybe none of them work. Perhaps only one of them is chosen.

Since  $a, b, c$  are symmetric in sum, maybe the values are such that the terms are picked consistently.

Alternative idea: add the three equations:

We have:

From (1'), (2'), (3'):

We can write equations:

```
$ aS - a^2 = 5
```

```
$ bS - b^2 = 10
```

```
$ cS - c^2 = 13
```

Add them together:

```
$ S(a + b + c) - (a^2 + b^2 + c^2) = 5 + 10 + 13 = 28
```

But  $a + b + c = S$ , so first term:  $S \cdot S = S^2$

So:

```
$$ S^2 - (a^2 + b^2 + c^2) = 28 \quad \text{(D)}
```

Also, we can find  $a^2 + b^2 + c^2$  in terms of  $(a+b+c)^2$

```
$ (a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ca)
```

So:

```
$ S^2 = (a^2 + b^2 + c^2) + 2(ab + bc + ca)
```

From (D):

```
$ a^2 + b^2 + c^2 = S^2 - 28
```

Plug in:

```
$ S^2 = (S^2 - 28) + 2(ab + bc + ca)
```

Then:

```
$ S^2 = S^2 - 28 + 2(ab + bc + ca)
```

Then:

```
$ -28 = -2(ab + bc + ca)
```

So  $ab + bc + ca = 14$

Nice.

So now we know:

- $S = a+b+c$
- $ab + bc + ca = 14$
- $a^2 + b^2 + c^2 = S^2 - 28$

Now go back to the equations.

From earlier, we have expressions for  $b + c = \frac{5}{a}$ , etc.

From (1):  $b + c = \frac{5}{a}$

But  $b + c = S - a$

So again  $S - a = \frac{5}{a} \Rightarrow a(S - a) = 5$

Same as before:  $aS - a^2 = 5$

Since  $(b + c) = S - a = 5/a$

Similarly,  $S - b = 10/b$ , etc.

Now, we already know that  $ab + bc + ca = 14$

Now, we can use:

We know  $a(S - a) = 5$

Similarly  $b(S - b) = 10$ ,  $c(S - c) = 13$

Wait that's more useful.

Let's write:

$a(S - a) = 5$

$\rightarrow aS - a^2 = 5$

But also we know  $ab + bc + ca = 14$

Now, perhaps we can express  $ab + bc + ca$  using these expressions.

We know from above:

We also know from equations that:

Since  $b + c = S - a$ , we can write  $ab + ac = a(b+c) = a(S - a) = 5$

Wait! Spot:

From  $b + c = S - a$ , multiply by  $a$ :

$a(b + c) = a(S - a)$

But left side is  $ab + ac$ , so:

$\begin{aligned} ab + ac &= a(S - a) = 5 \\ \end{aligned}$

Similarly, by symmetry:

From equation (2),  $c + a = 10 / b$

Then, multiply both sides by  $b$ :

$b(c + a) = 10$ , so  $bc + ab = 10$

Wait - so:

$ab + bc = 10$

Wait no:

Wait:  $c+a = S - b$ , and from (2),  $S - b = 10 / b$   $\rightarrow$  so:

$b(c + a) = 10$

But  $b(c + a) = bc + ab$

Thus:

$ab + bc = 10$

Similarly, equation (3):  $a + b = 13 / c$ , so  $c(a+b) = 13$

So  $ca + cb = 13$

That is:

$ca + cb = 13$

So now, we have:

From first:  $ab + ac = 5$

From second:  $ab + bc = 10$

From third:  $bc + ca = 14$ ? Wait no.

Wait solve carefully:

Wait first:  $ab + ac = 5$

Second: \$ ab + bc =10 \$

Third: from \$ c(a+b)=13 \$: \$ c(a+b)=ca+cb=13 \$

So indeed:

- \$ ab + ac = 5 \$ -> (i)
- \$ ab + bc =10 \$ -> (ii)
- \$ bc + ca =13 \$ -> (iii)

But wait - we previously found \$ ab+bc+ca =14 \$

Wait now we have:

Add all three:

(i)+(ii)+(iii):

Left: \$ ab + ac + ab + bc + bc + ca \$

= \$ 2ab + 2bc + 2ca \$

So:

\$ 2(ab + bc + ca) = 5+10+13=28 \$

So \$ 2(ab+bc+ca)=28 \$ -> \$ ab+bc+ca =14 \$ -> matches what we got earlier.

Good - consistent.

Now, let's label:

Let:

- \$ x = ab \$
- \$ y = bc \$
- \$ z = ca \$

Then from above:

- \$ x + z =5 \$ -> (from ab+ac=ab+ac=x+z=5)
- \$ x + y =10 \$
- \$ y + z =13 \$

And \$ x+y+z =14 \$

Now solve this system.

From:

- (1) \$ x + z =5 \$
- (2) \$ x + y =10 \$
- (3) \$ y + z =13 \$

Subtract (1) from (2): (2)-(1):  
\$ (x + y) - (x+z) =10 -5 \Rightarrow y - z = 5 \$

From (3): \$ y + z =13 \$

Now solve:

From (3) and (y-z=5):

Add them:  
\$ (y+z)+(y-z)=13+5=18 \Rightarrow 2y=18 \Rightarrow y=9 \$

Then from \$ y + z=13 \$, \$ z = 13 -9 =4 \$

From (2): \$ x=10 - y =10 -9=1 \$

From (1): \$ x + z =1+4=5\$: yes, checks.

So:

- \$ ab = 1 \$
- \$ bc =9 \$
- \$ ca =4 \$

Nice.

So now we know:

- \$ ab = 1 \$
- \$ bc = 9 \$
- \$ ca = 4 \$

Now we can find \$ a, b, c \$ from these?

We know:

```
From $ ab=1 $ -> $ b = \frac{1}{a} $
From $ ca=4 $ -> $ c = \frac{4}{a} $
Now plug into another: $ bc = 9 $
$ b = \frac{1}{a} $, $ c = \frac{4}{a} $
So $ b \cdot c = \frac{1}{a} \cdot \frac{4}{a} = \frac{4}{a^2} $
We know this should be 9.
```

So:

```
$$
\frac{4}{a^2} = 9
\Rightarrow a^2 = \frac{4}{9}
\Rightarrow a = \frac{2}{3} \quad (\text{positive since } a > 0)
$$
```

Now compute \$ b \$ and \$ c \$:

```
From $ ab = 1 $ -> $ b = \frac{1}{a} = \frac{3}{2} $
From $ ca = 4 $ -> $ c = \frac{4}{a} = 4 \times \frac{3}{2} = 6 $
Wait: $ a = 2/3 $, so $ c = 4 / (2/3) = 4 * 3 / 2 = 6 $
Yes.
```

Check \$ bc \$: \$ b = 3/2 \$, \$ c = 6 \$, so \$ bc = 3/2 \* 6 = 9 \$: yes.

Check \$ ab = (2/3)(3/2)=1 \$: yes.

Now we have \$ a = \frac{2}{3} \$, \$ b = \frac{3}{2} \$, \$ c = 6 \$

Now compute \$ S = a + b + c \$

```
$$
S = \frac{2}{3} + \frac{3}{2} + 6
$$
```

We compute this:

Common denominator: 6

```
$ \frac{2}{3} = \frac{4}{6} $
$ \frac{3}{2} = \frac{9}{6} $
$ 6 = \frac{36}{6} $
```

Sum: \$ \frac{4+9+36}{6} = \frac{49}{6} \$

So \$ S = \frac{49}{6} \$

Now - check if this satisfies original equations.

```
### Check equation (1): $ \frac{5}{a} = b + c $?
$ a = 2/3 $ -> $ 5 / (2/3) = 5 * 3 / 2 = 15/2 = 7.5 $
```

Now \$ b + c = 3/2 + 6 = 3/2 + 12/2 = 15/2 \$: yes -> 7.5 -> correct

```
### Equation (2): $ \frac{10}{b} = c + a $?
$ b = 3/2 $: $ 10 / (3/2) = 10 \cdot 2/3 = 20/3 \approx 6.666 $
```

\$ c + a = 6 + 2/3 = 18/3 + 2/3 = 20/3 \$ -> yes

```
### Equation (3): $ \frac{13}{c} = a + b $?
$ c = 6 $: $ 13 / 6 \approx 2.166 $
```

\$ a + b = 2/3 + 3/2 = 4/6 + 9/6 = 13/6 \$ -> equals \$ 13/6 \$: yes

Perfect -> all match.

Thus, \$ a + b + c = \frac{49}{6} \$

Now \$ m = 49 \$, \$ n = 6 \$: relatively prime?  
Yes, gcd(49, 6)=1 since 49=7^2, 6=2 \times 3 -> no common factors.

Thus \$ m+n=49+6 = \boxed{55} \$

### Final Answer:

\$ \boxed{55} \$