DIVIDE-AND-CONQUER COT: RL FOR REDUCING LATENCY VIA PARALLEL REASONING

Arvind Mahankali, Kaiyue Wen & Tengyu Ma
Computer Science Department
Stanford University
Stanford, CA 94305, USA
{amahanka,kaiyuew,tengyuma}@stanford.edu

ABSTRACT

Long chain-of-thought reasoning (Long CoT) is now fundamental to state-of-the-art LLMs, especially in mathematical reasoning. However, LLM generation is highly sequential, and long CoTs lead to a high latency. We propose to train Divide-and-Conquer CoT (DC-CoT) to reduce the latency. With DC-CoT, the model can act as a director that identifies distinct subtasks that can be performed in parallel in its reasoning process, and then spawns workers to execute the subtasks. Our goal is to achieve high accuracy, with a low longest path length, which is a theoretical measure of the latency needed for the response. We start with a long CoT base model (DeepScaleR-1.5B-Preview), and first use SFT with a small curated demonstration set to initialize its ability to spawn workers in a certain format. Because SFT degrades the accuracy significantly, we design a multi-stage RL algorithm, with various data filtering strategies, to recover the accuracy while decreasing the longest path length. Across several benchmarks including AIME 2024 and HMMT 2025, DC-CoT achieves similar accuracy as DeepScaleR-1.5B-Preview while decreasing longest path length by 35-40%.

1 INTRODUCTION

Long chain-of-thoughts (CoTs) have led to a paradigm shift in language model (LM) reasoning (OpenAI et al., 2024; DeepSeek-AI et al., 2025). Long CoTs allow LMs to think for longer and try many useful problem solving strategies, such as checking their own work and enumerating cases. Thus, long CoT models trained with reinforcement learning (RL) excel in mathematical reasoning, drastically improving performance on benchmarks such as AIME (OpenAI, 2025).

However, long CoTs are quite slow, due to the sequential nature of LM generation (Leviathan et al., 2023) and the large number of tokens involved. Over the course of 2025, token budgets in long CoTs have increased from 32K for DeepSeek R1 (DeepSeek-AI et al., 2025) to 80K for Minimax M1 (MiniMax et al., 2025) and even 256K for Kimi K2 Thinking (Moonshot AI). Reducing the latency of generation using reasoning models is therefore a pressing question.

While long CoTs are all *sequential*, many tasks admit a *parallelizable* reasoning structure. For instance, many math problems naturally decompose into independent sub-cases that are solvable in parallel. Alternatively, there may be multiple unique plausible approaches that are worth trying in parallel as opposed to sequentially.

This paper aims to significantly reduce the latency of long CoTs by **training** models to perform parallel thinking. We hypothesize that prompting-based scaffolding methods **alone**—whether heuristic approaches (Hadfield et al., 2025; Yan, 2025; Lin, 2026) or principled prompt optimization (Agrawal et al., 2025), which do not change model weights—are fundamentally limited in achieving effective or optimal parallel thinking. This limitation arises from existing models' inability to decompose challenging goals into parallelizable sub-tasks, as they were never trained to do so. We posit that parallel thinking is an advanced capability that must be explicitly taught through targeted, large-scale training.

We introduce Divide-and-Conquer CoT (DC-CoT), a method to teach LMs to identify opportunities for parallelizable subtasks in their thinking processes, solve the subtasks in parallel threads, then aggregate the outputs of these threads. As shown in Figure 2, a single model will serve two roles: director and worker. The director performs sequential thinking until it identifies useful subtasks that can be performed in parallel. It will then spawn multiple workers, and specify a subtask for each worker. Each worker inherits its context from the director, and the workers complete their subtasks in parallel. Afterwards, the thinking processes of the workers will be appended to the context of the director. The director aggregates the workers' results, then thinks more before completing the response. This enables sequential reasoning for inherently sequential aspects, while allowing parallel reasoning when applicable to reduce latency.

We train our model to accurately identify parallelism and generate responses in this fashion. Starting from a sequential reasoning model (in our case DeepScaleR-1.5B-Preview (Luo et al., 2025)), we first perform SFT on synthetic demonstrations (Section 4.1 and Section 4.2) to teach the model to follow the parallel format of Figure 2. However, while SFT gives our model the ability to follow the format to a certain degree, it cannot fundamentally improve the parallel thinking capability of our model — in fact, it significantly degrades the accuracy compared to the base model.

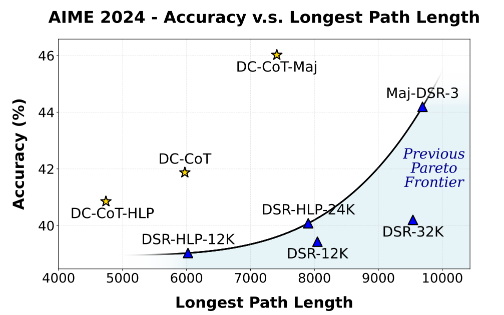


Figure 1: Accuracy (pass@1) and longest path length for DC-CoT and baselines on AIME 2024. **DSR-32K** and **DSR-12K** denote DeepScaleR-1.5B-Preview evaluated with response lengths $32,768$ and $12,000$ respectively. **DC-CoT-HLP** denotes DC-CoT trained with additional RL with a high length penalty, and **DSR-HLP-24K** and **DSR-HLP-12K** denote baselines trained similarly, starting from DeepScaleR-1.5B-Preview. **DC-CoT** achieves higher accuracy than **DSR-32K** while requiring less longest path length.

Therefore, we perform RL, which is critical to improving the model's parallel thinking capability and recovering the accuracy. We quantify the model's parallel thinking capability by its accuracy and the average **longest path length** of its responses, defined as the number of tokens on the longest path from the beginning to the end of the response (see Section 3 for details). This is a theoretical proxy for the *latency*. We design a multi-stage RL algorithm (Section 4.3), with the goal of maximizing accuracy and minimize longest path length.

We highlight that improving our model's parallel thinking capability through RL is nontrivial, as we face the following technical challenges. First, we initially use DAPO (Yu et al., 2025), but find that the accuracy plateaus at a level much lower than the accuracy of our sequential base model. We believe this is caused by DAPO's tendency to steadily increase the entropy of the policy for almost all tokens. We speculate that an excessively high entropy for all tokens is detrimental, since some tokens should be essentially deterministic under an ideal policy. We later find that continuing RL training with **CISPO** (MiniMax et al., 2025) improves the accuracy monotonically while keeping the entropy stable (unlike DAPO or GRPO (Shao et al., 2024)), though using CISPO by itself does not suffice to recover the accuracy to the level of the sequential base model.

We next find that the data filtering strategy affects whether the model prioritizes accuracy or longest path length. Initially, we filter out problems where all the responses are wrong, while still including problems where all the responses are correct, as such problems can still provide a signal for the model to reduce its longest path length, via the longest path length penalty term in the reward function (see Section 4.3). However, the model eventually plateaus in accuracy and starts to solely improve its longest path length. We speculate that there is a conflict between easier problems, for which the advantages in RL are dominated by the longest path length, and more difficult problems, where the advantage depends more on accuracy — since we include problems where all responses are correct, the former dominates. We address this issue by updating our data filtering strategy — partway through the training, we start filtering out those easy problems where all responses are correct.

Concurrent and independent works propose AsyncThink (Chi et al., 2025) and Native Parallel Reasoner (Wu et al., 2025b), or NPR, which both also train models end-to-end with RL to parallelize their reasoning. Unlike these works that use non-thinking variants of Qwen3-4B, we start from a long CoT reasoning model that is already trained with RL. We speculate that the long CoTs cause some of the technical challenges we describe above, due to the positive correlation between length and accuracy (and the resulting conflict between improving accuracy and reducing length). We show that it is possible to build on a long CoT model, recovering and continuously improving the accuracy while reducing the inference latency. Interestingly, the very recent Kimi K2.5 (Moonshot AI, 2026) also uses a multi-stage strategy to train an orchestrator to decompose and delegate subtasks, prioritizing parallelization initially and then accuracy later, reminiscent of the purpose of our multi-stage data filtering strategy. Unlike Kimi K2.5, we allow the workers/sub-agents to learn from RL as well.

Our results on AIME 2024 ((Mathematical Association of America, 2024)) are shown in Figure 1. Our method, DC-CoT, achieves a better accuracy and longest path length compared to DeepScaleR-1.5B-Preview evaluated at response lengths 32K and 12K (denoted as **DSR-32K** and **DSR-12K** respectively) — in particular, DC-CoT achieves a 37.4% reduction in longest path length compared to **DSR-32K**. Additionally, continuing RL from DC-CoT with a high length penalty (denoted as **DC-CoT-HLP**) gives a Pareto improvement compared to similarly-trained baselines trained from DeepScaleR-1.5B-Preview (**DSR-HLP-24K** and **DSR-HLP-12K**). Finally, DC-CoT is complementary to test-time scaling through independent sampling, achieving further gains with majority voting. DC-CoT with majority voting (denoted as **DC-CoT-Maj**) achieves better accuracy and longest path length compared to DeepScaleR-1.5B-Preview with response length 12K and majority voting (denoted as **Maj-DSR-3**). We observe similar benefits when using DC-CoT on other benchmarks (see Figure 11, Table 1, and Table 2 in Section J).

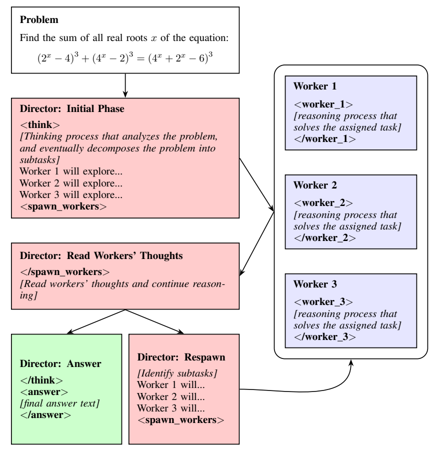


Figure 2: Our parallel inference procedure for **DC-CoT**. The model performs some reasoning before assigning tasks to workers. The workers perform the assigned tasks. The director then reads the workers' outputs. Then, the director may either finalize its answer or do another round of parallel generation.

2 RELATED WORK

We discuss works on reinforcement learning with verifiable rewards (RLVR) and long CoT for mathematical reasoning in Section A, as well as works on parallel sampling for inference-time scaling. These works are orthogonal and complementary to ours, since we aim to identify *distinct* and parallelizable subtasks, which cannot be achieved by independent sampling. Indeed, we show in Section 5.2 that parallel sampling can be combined with our approach for further gains.

Delegation in Multi-agent Systems. Several works study multi-agent systems (Hadfield et al., 2025; Kim et al., 2025; Zou et al., 2025; Gao et al., 2025), where different LMs collaborate to solve problems, dividing an over-arching task into subtasks. Hadfield et al. (2025) find that multi-agent systems can lead to large improvements in deep research compared to sequential single-agent systems — we emphasize that our contribution compared to Hadfield et al. (2025) is to train our model end-to-end with RL. Other works (Nguyen et al., 2025; Gao et al., 2025; Kim et al., 2025) study the circumstances under which multi-agent systems will outperform single-agent systems, and vice versa. Training-free methods using multi-agent systems have been proposed to improve efficiency (Narayan et al., 2025; Zou et al., 2025) and accuracy (Du et al., 2023) compared to single-agent systems. Other works train multi-agent systems with RL (Dang et al., 2025; Zhou et al., 2025; Sun et al., 2025). Dang et al. (2025) train an orchestrator with RL to call LLM sub-agents while minimizing total resource usage. Sun et al. (2025) train a coordinator and its sub-agents end-to-end to manage the context length of the coordinator, while Zhou et al. (2025) train a single agent with RL to compress its own context length. Hu et al. (2025d) train a multi-agent system with a planner and workers to accurately perform tasks such as coding and deep research. Our goal is to reduce the inference latency, which Dang et al. (2025), Sun et al. (2025), Zhou et al. (2025) and Hu et al. (2025d) do not study — additionally, we train the sub-agents through RL, unlike Dang et al. (2025).

Parallel Decoding with LLMs to Reduce Latency. Improving the latency of LM inference is an active area of research (Leviathan et al., 2023; Cai et al., 2024; Chen et al., 2023; Zhang et al., 2024b). Several works specifically study how an LM can identify opportunities for parallelization in its own generation, through prompting (Ning et al., 2024), training on demonstrations (Liu et al., 2024), and preference optimization using solution quality and latency to compute a reward (Jin et al., 2025).

These works study instruction following models. Parallelization of inference has also been studied for reasoning models that use long CoTs (Biju et al., 2025; Yang et al., 2025b; Zheng et al., 2025; Pan et al., 2025). Biju et al. (2025) and Yang et al. (2025b) curate demonstrations, used for SFT, which can teach reasoning models to parallelize their reasoning traces. Zheng et al. (2025) train a reasoning model with RL to perform parallel inference, as part of a curriculum used to improve sequential reasoning. Pan et al. (2025) show that in the Countdown task, LMs trained with RL to search in parallel achieve a better accuracy-latency tradeoff than sequential methods.

3 Parallel Inference Architecture

We now explain our architecture for parallelizable inference. In the following, we use `[EOS]` to denote the end-of-sequence token. We define a single-thread inference primitive $\textsc{Infer}(M, P, \texttt{finish}, L)$, which performs standard LM inference. Here M is a language model, P is a prompt, `finish` is a termination string, and L is an upper bound on the amount of generated tokens. The subroutine $\textsc{Infer}(M, P, \texttt{finish}, L)$ performs generation using the model M and prompt P until one of the following occurs: (1) L tokens are generated, (2) `finish` is output, or (3) the `[EOS]` token is output. The output is a list of tokens S, whose length is denoted by $|S|$.

Our architecture is implemented entirely using calls to Infer. Thus, we directly use vLLM (Kwon et al., 2023), without any under-the-hood modifications, unlike Multiverse (Yang et al., 2025b), which uses non-standard attention masks during generation, thus requiring a customized version of SGLang. We adjust our training-time algorithm to maintain consistency with inference (see Section 4.2).

An illustration of our architecture is shown in Figure 2. A more detailed version with examples of thoughts generated by the model is shown in Figure 7 in Section C. Formal pseudocode is in Algorithm 1 in Section B.

Initial Single-Thread Phase. We begin by using $\textsc{Infer}(M, P, \texttt{"<spawn\_workers>"}, L)$ to perform inference in single-thread mode. Here the model performs initial reasoning, through which it may identify subtasks that are useful towards solving the problem and can be done in parallel. This corresponds to the "Director: Initial Phase" box in Figure 2, and in the rest of the paper, we use S_1 to refer to this part. This call to Infer terminates when `"<spawn_workers>"` is output — in essentially all cases, our model will output `"<spawn_workers>"` in this stage rather than `[EOS]`.

Parallel Generation with Workers. After reaching `"<spawn_workers>"`, our algorithm spawns multiple workers, which will each continue generation *in parallel* based on the contents of S_1. In our experiments, we use 3 workers. For each worker $i \in [3]$, the prompt of the ith worker will be $P_i := P + S_1 + \texttt{"<worker\_i>"}$. In other words, each worker sees the original prompt and the result of the initial single-thread generation, as well as a special tag to distinguish this worker from other workers. The ith worker then performs standard inference by calling $\textsc{Infer}(M, P_i, \texttt{"</worker\_i>"}, L)$ — these 3 calls to Infer are executed in parallel. In the rest of the paper, we use W_1 to refer to `"<worker_1>"`, together with the portion generated by Infer — this is the text corresponding to "Worker 1" in Figure 2. We define W_2 and W_3 analogously.

Returning to Single-Thread. After all of the workers finish their generation, the model (in its role as director) sees the work done by the workers, and performs further reasoning (see the "Director: Read Workers' Thoughts" box in Figure 2). The model may finalize its answer and end generation (see "Director: Answer" box) or may identify new subtasks that can be done in parallel, output `"<spawn_workers>"`, and start a new round of multi-thread generation (see "Director: Respawn" box). While there can be arbitrarily many rounds of parallel generation, our model typically spawns workers for 1 or sometimes 2 rounds. In the case where only 1 round occurs, we use S_2 to refer to all the text after `"</spawn_workers>"` inclusive, corresponding to boxes "Director: Read Workers' Thoughts" and "Director: Final Answer" in Figure 2.

Longest Path Length. We use the longest path length of a response, defined below, as a proxy for the wall-clock time for generating it. Suppose for simplicity that workers are only spawned for a single round — the definition generalizes naturally to multiple rounds. Let S_1, W_1, W_2, W_3 and S_2 be as defined earlier in this section. Then, we define the longest path length of o as $\mathrm{lpl}(o) := |S_1| + \max_{i \in [3]} |W_i| + |S_2|$. We enforce a longest path length budget during inference by setting the allowed output length at each call to Infer.

4 Training Methods

This section describes our training methods, which are summarized in Figure 3.

4.1 SFT DATASET

Initial Generation of Sequential Long CoT Rollouts. We generate sequential CoTs using M_{base}, which can be any reasoning model. For an initial set of problems $\mathcal{P}_{\text{init}}$, we perform rejection sampling, generating up to 3 responses per problem, until we obtain a correct response. If a problem p has at least one correct response o_{seq}, we include (p, o_{seq}) in $\mathcal{D}_{\text{init}}$ — otherwise, we omit p.

Rewriting Sequential CoTs to Include Parallelization. Given the (p, o_{seq}) pairs in $\mathcal{D}_{\text{init}}$, we rewrite the responses o_{seq} to spawn workers where useful, obtaining a dataset $\mathcal{D}_{\text{rewrite}}$ of responses (p, o) where the response o performs parallel thinking.

Given a sequential CoT response o_{seq}, we tell our rewriting model M_{rewrite} to largely preserve the content and thinking process of o_{seq}, while producing o. The response o should satisfy the following criteria. (A) It should begin by reasoning in a sequential manner, in the same way as o_{seq} does — this is shown in the "Director: Initial Phase" box in Figure 2. (B) The reasoning processes of the workers in o should be excerpted almost verbatim from o_{seq}. Intuitively, criteria (A) and (B) mean that the solutions in $\mathcal{D}_{\text{rewrite}}$ have a similar structure to responses produced by M_{base}. We hypothesize that training M_{base} on rewritten responses satisfying (A) and (B) mitigates the degradation in the accuracy of M_{base} caused by SFT, though not eliminating it entirely.

Figure 3: Training Procedure

1. Prepare SFT Dataset (Section 4.1)
- Sample 1 correct solution with sequential CoT for each given problem using M_{base}
- Use a more powerful model M_{rewrite} to rewrite CoTs into the parallel format as in Figure 2, obtaining dataset $\mathcal{D}_{\text{rewrite}}$

2. SFT (Section 4.2)
- SFT starting from M_{base} on $\mathcal{D}_{\text{rewrite}}$ using attention masks which respect the parallelism shown in Figure 2

3. RL (Section 4.3)
- RL with 4 stages using correctness reward and format/longest path length penalties
- Each stage has varying hyperparameters and data filtering criteria

We also ask M_{rewrite} to include in o, prior to spawning workers, a justification for why the tasks it has assigned to the workers can be performed in parallel. One goal is to reduce cases where the tasks are not in fact parallelizable, e.g. where the task assigned to Worker 2 requires the answer to Worker 1's task. This also allows for the possibility that our model M_{SFT} trained on $\mathcal{D}_{\text{rewrite}}$ can self-reflect, and after proposing an initial assignment of tasks to workers, it can decide that the assignment is suboptimal, rather than committing to spawning workers.

To convey the above information to M_{rewrite}, we give M_{rewrite} a detailed prompt, shown in Box 4 in Section K. The prompt contains an example of a sequential CoT o_{seq}, together with an example of a rewritten response o that spawns workers.

We again use rejection sampling when generating the rewritten responses with M_{rewrite}. For each pair (p, o_{seq}) in $\mathcal{D}_{\text{init}}$, we attempt up to 3 times to generate a rewritten response o using M_{rewrite}. We verify that o has the correct answer. Additionally, we verify that o satisfies the format shown in Figure 2. We consider o to satisfy the format if it spawns workers, if it includes all relevant tags shown in Figure 2, and if these tags are nested and matched appropriately.

4.2 SFT TRAINING SEQUENCES, POSITION IDS, AND ATTENTION MASK

We now elaborate on step 2 of Figure 3. For simplicity, we consider a response where workers are only called for 1 round, and define S_1, W_1, W_2, W_3 and S_2 as in Section 3.

In our inference procedure, two distinct operations take place. (A) When W_1, W_2 and W_3 are initially generated, each of them has S_1 as the prefix. (B) W_1, W_2 and W_3 are later included in the prefix used to generate S_2.

For $i \in [3]$, let $W_i^{(\text{gen})}$ denote the KV cache vectors corresponding to W_i in operation (A), and let $W_i^{(\text{prefill})}$ denote the KV cache vectors corresponding to W_i in step (B). Then, $W_i^{(\text{gen})}$ and $W_i^{(\text{prefill})}$ are distinct vectors, for the following reason. When the W_i are initially generated during operation (A), they are generated in parallel — they share the original prompt and S_1 as their context, and are otherwise separate LM generation tasks. Thus, the W_i only attend to S_1 and not each other. Meanwhile, in operation (B), we perform standard LM generation to generate S_2, using S_1, W_1, W_2 and W_3 as the context. Thus, during the prefill stage, a standard causal mask will be applied to the context, meaning W_2 attends to W_1, and W_3 attends to W_1 and W_2.

Thus, for the sake of consistency with inference-time behavior, we include W_1, W_2 and W_3 twice in the training sequences, with the first occurrence corresponding to operation (A) (i.e. the $W_i^{(\text{gen})}$ for

$i \in [3]$) and the second corresponding to (B) (i.e. the $W_i^{\text{(prefill)}}$). To enforce the independence of the $W_i^{\text{(gen)}}$, as well as the fact that S_2 attends to the $W_i^{\text{(prefill)}}$ but not the $W_i^{\text{(gen)}}$, we use the attention mask in Figure 4. Similarly, at training time, we let the position IDs corresponding to the $W_i^{\text{(gen)}}$ all start from $|S_1|$, while those of $W_1^{\text{(prefill)}}$ start from $|S_1|$, those of $W_2^{\text{(prefill)}}$ start from $|S_1| + |W_1|$, and so on.

4.3 Reinforcement Learning (RL)

We next perform RL starting from M_{SFT}. We find that the model's accuracy degrades after performing SFT with $\mathcal{D}_{\text{rewrite}}$ — thus, we aim to recover the accuracy. Additionally, many of M_{SFT}'s responses do not spawn workers (for instance, on a batch of problems for our RL training dataset, M_{SFT} only spawns workers for roughly 65% of responses). Thus, we would like to teach M_{SFT} to spawn workers more often, and in a way which reduces its longest path length. Thus, the reward for a response o is determined by the correctness and longest path length of o, as well as whether o satisfies the format defined by Figure 2.

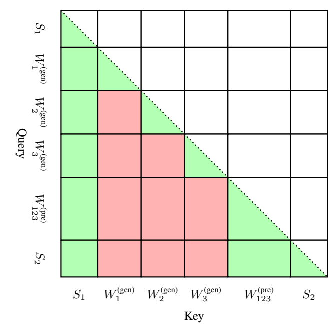


Figure 4: Attention mask at training time. Blocks which attend to each other are shown in green, while blocks which do not attend to each other are shown in red. The attention mask is also causal — entries above the diagonal are masked out. Here, $W_{123}^{\text{(pre)}}$ refers to the concatenation of $W_1^{\text{(prefill)}}$, $W_2^{\text{(prefill)}}$ and $W_3^{\text{(prefill)}}$.

Given a problem and response generated by π_θ, we construct a training sequence, as well as the accompanying attention mask and position IDs, in a manner similar to Section 4.2. We do not compute the loss on tokens which are not generated by the policy itself — for instance, using the terminology of Section 4.2, we compute the loss on the tokens in $W_i^{\text{(gen)}}$, but not on those in $W_i^{\text{(prefill)}}$, for $i \in [3]$.

In our RL training, we use DAPO (Yu et al., 2025) (a variant of GRPO (Shao et al., 2024)), and CISPO (MiniMax et al., 2025) — implementation details are deferred to Section F.

Penalty for overlong responses. We use an overlong length penalty similar to Yu et al. (2025). Here, we penalize responses with excessive longest path length. For a response o, we add to the reward a length penalty $\ell(o) := C_L \cdot \frac{\text{lpl}(o) - L_{\text{cutoff}}}{L_{\max} - L_{\text{cutoff}}}$ for some cutoff L_{cutoff}, if $\text{lpl}(o) > L_{\text{cutoff}}$. We let $C_L = 0.1$ for most of our experiments.

Reward function. Given a response o, its reward $g(o)$ is defined as follows. We first define a *correctness-format* reward as follows:

$$f(o) := \begin{cases} 1.0 & \text{if } o \text{ is correct and satisfies format} \\ 0.5 & \text{if } o \text{ is correct but does not satisfy format} \\ 0.0 & \text{otherwise} \end{cases}$$

Then, the overall reward is $g(o) := \max(0, f(o) - \ell(o))$.

Data filtering. Following DAPO (Yu et al., 2025), we filter problems p according to certain criteria in each iteration of RL. We use two different data filtering strategies: (1) ***Include-Easy***, where we filter out problems p where there are no responses o that are both correct *and* satisfy the format, i.e. where none of the responses o satisfy $f(o) = 1.0$, and (2) ***Remove-Easy***, where we filter out problems p where all of the responses are correct or all of the responses are wrong. One key difference is that *Include-Easy* includes problems where all of the responses are correct, unlike *Remove-Easy*. Filtering strategy *Remove-Easy* does not take format into account. We use *Include-Easy* in the initial stages of training. Many of the SFT model M_{SFT}'s responses do not spawn workers at all, or do not satisfy the format. Thus, even problems p for which all of the responses have the correct answer provide a useful signal for the model to learn to spawn workers, follow the format, and reduce its longest path length. However, *Include-Easy* eventually leads to a plateau in the accuracy, as the model prioritizes longest path length. Thus, we switch to *Remove-Easy* partway through RL training, as discussed below in more detail.

Multi-stage training to address accuracy plateaus. We make changes to our RL algorithm at multiple intermediate steps, each time with the goal of addressing accuracy plateaus. Due to compute constraints, each time we make a change to our training algorithm, we build on our latest checkpoint, rather than starting from scratch. Similar multi-stage training is done by other works, such as ProRL (Liu et al., 2025a). Each time we make a change to our algorithm and begin a new stage, we reset the reference model π_{ref}, as well as the optimizer state, similar to ProRL (Liu et al., 2025a). Our RL training run has 4 stages in total.

In Stage 1, we use DAPO, and filtering strategy *Include-Easy*. We set the upper clipping parameter $\varepsilon_{\text{high}}$ to 0.28, as done by Yu et al. (2025). The maximum allowed longest path length L_{max} for any response o is $7,500$ in this stage (and in stages 2 and 3, as well). The length penalty cutoff L_{cutoff} is $2,000$. For many responses, the SFT model does not spawn workers at all, or does not follow the format. Over the course of Stage 1, the accuracy improves, and the model learns to follow the format and spawn workers more often, while the longest path length decreases. However, we eventually observe a plateau in the accuracy after many steps (Figure 8 in Section G). We hypothesize that this accuracy plateau is caused by DAPO's tendency to keep increasing the entropy of our model. In Figure 10 in Section H, we show various statistics related to the entropy of π_θ over the course of Stage 1. We find that the mean entropy increases monotonically, and even the 60th percentile of entropy rises drastically. Intuitively, this is harmful — it suggests that DAPO increases the entropy for tokens for which the model should have a relatively high certainty. This analysis of entropy statistics is motivated by the work of Wang et al. (2025), who recommend only performing gradient updates for tokens in the higher percentiles of entropy, during RLVR.

Thus, in Stage 2, we continue from our Stage 1 checkpoint and use CISPO instead of DAPO. We find that CISPO improves the accuracy monotonically for several more steps, and CISPO does not exhibit the phenomenon of monotonically increasing entropy as DAPO does. Stage 2 eventually also encounters a plateau, and then a decline, in accuracy (Figure 9 in Section G). The longest path length continues to decrease sharply, at the expense of accuracy. We hypothesize that this is because of filtering strategy *Include-Easy*— for a problem p where all responses are correct, the advantage of a response is solely a function of its longest path length. Thus, in Stage 3, in order to resolve the plateau in accuracy, we switch to filtering strategy *Remove-Easy*.

Stage 3 improves the accuracy, but eventually reaches a plateau, and the amount of truncated responses increases. Thus, in Stage 4, we now increase the allowed longest path length L_{max} from $7,500$ to $12,000$, and increase the longest path length penalty cutoff L_{cutoff} from $2,000$ to $6,500$.

High length penalty (HLP). After finishing stage 4, we aim to decrease the longest path length significantly, without significantly reducing accuracy. Thus, we increase C_L from 0.1 to 0.9, and we let $L_{\text{cutoff}} = 2000$ — in other words, all responses with longest path length more than 2000 are penalized, and the length penalty can range up to 0.9. The formal definition of our HLP reward function, and additional details, are deferred to Section F.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

The baselines we evaluate are discussed below. We defer details on our SFT dataset $\mathcal{D}_{\text{rewrite}}$, SFT and RL hyperparameters, and additional experimental details to Section I. For all experiments, we use 8 H100 GPUs. During RL, we use the DeepScaleR training dataset (Luo et al., 2025), and AIME 2024 (Mathematical Association of America, 2024) as our validation dataset.

Baselines. We report the accuracy and longest path length of DeepScaleR-1.5B-Preview (Luo et al., 2025), using a response length of 32K tokens, denoted as **DSR-32K**, and 12K tokens, denoted as **DSR-12K**. We use the prompt shown in Box 2, following DeepScaleR (Luo et al., 2025). To further increase the length efficiency of the baseline, we apply an HLP stage to DeepScaleR-1.5B-Preview. We train and evaluate DeepScaleR-1.5B-Preview with a high length penalty allowing response length 24K (denoted as **DSR-HLP-24K**) and 12K (denoted as **DSR-HLP-12K**). Hyperparameters for training DSR-HLP-24K and DSR-HLP-12K are given in Section I. We also evaluate DSR-12K using maj@3, and refer to this method as **Maj-DSR-3** (additional details in Section I).

5.2 RESULTS

Main results — accuracy and longest path length. Figure 11 and Table 1, in Section J, compare our methods (**DC-CoT**, **DC-CoT-HLP**) with the aforementioned baselines. DC-CoT consistently improves the accuracy-longest path length tradeoff compared to baselines. DC-CoT-HLP further improves the longest path length significantly, at the cost of a slight decrease in accuracy compared to DC-CoT. Across our benchmarks, DC-CoT matches (and sometimes exceeds) the accuracy of the strongest non-HLP baseline DSR-32K, while reducing the longest path length by roughly 35-40% — see Table 2 in Section J for percentage decreases in longest path length. DC-CoT-HLP achieves comparable accuracy to the HLP baseline DSR-HLP-12K trained and evaluated with the same longest path length, while achieving a roughly 20% reduction in longest path length on all benchmarks. Additionally, we evaluate DC-CoT using maj@3, using an identical setup to Maj-DSR-3 — we refer to this method as **DC-CoT-Maj**. As shown in Table 2, DC-CoT-Maj achieves similar or higher accuracy than Maj-DSR-3, while achieving significant reductions in longest path length. Table 3 in Section J shows the total tokens used per response by our methods.

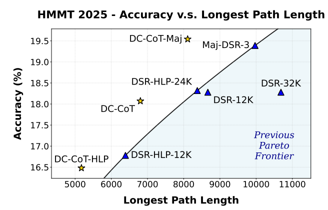
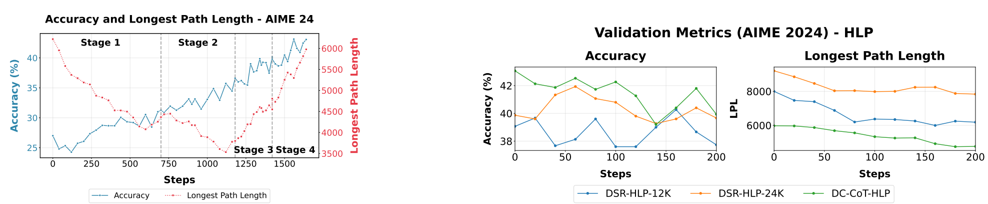


Figure 5: Accuracy and longest path length for all methods on HMMT 2025.

On AIME 2024 (Figure 1), the methods DC-CoT, DC-CoT-HLP and DC-CoT-Maj are beyond the Pareto frontier of accuracy and longest path length achieved by previous methods — in particular, both DC-CoT and DC-CoT-HLP are a Pareto improvement compared to DSR-32K, DSR-12K, and DSR-HLP-24K. On HMMT 2025 (Figure 5), DC-CoT achieves similar accuracy (<0.2% lower) to DSR-32K, DSR-12K and DSR-HLP-24K, while achieving a 36% reduction in longest path length compared to DSR-32K, a 21.5% reduction compared to DSR-12K, and an 18.8% reduction compared to DSR-HLP-24K. Similarly, DC-CoT-HLP achieves a similar accuracy as DSR-HLP-12K, with 19% lower longest path length. Finally, DC-CoT-Maj is a Pareto improvement over Maj-DSR-3, achieving slightly higher accuracy with 18.7% less longest path length. On Minerva Math (MM), DC-CoT and DC-CoT-HLP obtain worse accuracy than the baselines. We speculate that problems in MM involve applying several calculations in a purely sequential manner, making them less amenable to parallelization. We leave improving DC-CoT's performance on MM to future work.



(a) Accuracy and longest path length of **DC-CoT** during all stages of RL.

(b) Accuracy and longest path length for DSR-HLP-12K, DSR-HLP-24K, and DC-CoT-HLP.

Figure 6: RL validation metrics on AIME 2024 for (a) DC-CoT and (b) HLP baselines/DC-CoT-HLP

RL training dynamics. Figure 6a shows the accuracy and longest path length of DC-CoT over the course of RL, on AIME 2024. Figure 12 in Section J shows the degree of parallelism (DP), which for a response o is defined as the ratio between the total tokens in o and the longest path length. The validation accuracy of our model improves monotonically, due to the multi-stage training discussed in Section 4.3. The longest path length decreases, and the degree of parallelism increases, during Stages 1 and 2, when we use filtering strategy *Include-Easy*. However, when we switch to *Remove-Easy* during Stages 3 and 4, the model uses more sequential thinking to help improve the accuracy. As a result, the longest path length increases and the degree of parallelism drops in these two stages.

The accuracy and longest path length of the methods trained with HLP (DSR-24K-HLP, DSR-12K-HLP, and DC-CoT-HLP) are shown in Figure 6b, and the DP of DC-CoT-HLP is shown in Figure 13 in Section J. The accuracy of DSR-HLP-12K and DSR-HLP-24K stays roughly constant, while there is a slight decline in the accuracy of DC-CoT-HLP. There is a steady decrease in the longest path lengths for all three methods. Corresponding to this decrease in longest path length, the degree of parallelism for DC-CoT-HLP also increases during HLP, from slightly under 1.2, to around 1.37. We evaluate checkpoints from towards the end of HLP for all methods, based on our assessment of validation accuracy and longest path length. We use step 160 for DSR-12K-HLP, step 180 for DSR-24K-HLP, and step 180 for DC-CoT-HLP in all tables and figures.

6 ACKNOWLEDGMENTS

The authors would like to thank Luke Bailey, Neil Band, Caroline Choi and Thomas Chen for helpful discussions. The authors would also like to thank support from NSF CIF 2212263 and the National Defense Science and Engineering Graduate (NDSEG) Fellowship, and the Marlowe cluster at Stanford University (Kapfer et al., 2025) for providing the H100 GPUs required for the experiments.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025. URL https://arxiv.org/abs/2507.19457.

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2025. URL https://arxiv.org/abs/2505.23281.

Emil Biju, Shayan Talaei, Zhemin Huang, Mohammadreza Pourreza, Azalia Mirhoseini, and Amin Saberi. Sprint: Enabling interleaved planning and parallelized execution in reasoning models, 2025. URL https://arxiv.org/abs/2506.05745.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Zewen Chi, Li Dong, Qingxiu Dong, Yaru Hao, Xun Wu, Shaohan Huang, and Furu Wei. The era of agentic organization: Learning to organize with language models, 2025. URL https://arxiv.org/abs/2510.26658.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models, 2025. URL https://arxiv.org/abs/2505.22617.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration, 2025. URL https://arxiv.org/abs/2505.19591.

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, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL `https://arxiv.org/abs/2305.14325`.

Mingyan Gao, Yanzi Li, Banruo Liu, Yifan Yu, Phillip Wang, Ching-Yu Lin, and Fan Lai. Single-agent or multi-agent systems? why not both?, 2025. URL `https://arxiv.org/abs/2505.18286`.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL `https://arxiv.org/abs/2506.04178`.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system, June 2025. URL `https://www.anthropic.com/engineering/multi-agent-research-system`.

Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. Don't overthink it. preferring shorter thinking chains for improved llm reasoning, 2025. URL `https://arxiv.org/abs/2505.17813`.

Bingxiang He, Zekai Qu, Zeyuan Liu, Yinghao Chen, Yuxin Zuo, Cheng Qian, Kaiyan Zhang, Weize Chen, Chaojun Xiao, Ganqu Cui, Ning Ding, and Zhiyuan Liu. Justrl: Scaling a 1.5b llm with a simple rl recipe, 2025a. URL `https://arxiv.org/abs/2512.16649`.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi:

10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

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. Skywork open reasoner 1 technical report, 2025b. URL https://arxiv.org/abs/2505.22312.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Jian Hu, Mingjie Liu, Shizhe Diao, Ximing Lu, Xin Dong, Pavlo Molchanov, Yejin Choi, Jan Kautz, and Yi Dong. Prorl v2: Prolonged training validates rl scaling laws, August 2025a. URL https://hijkzzz.notion.site/prorl-v2. First published on Notion.

Jian Hu, Mingjie Liu, Ximing Lu, Fang Wu, Zaid Harchaoui, Shizhe Diao, Yejin Choi, Pavlo Molchanov, Jun Yang, Jan Kautz, and Yi Dong. Brorl: Scaling reinforcement learning via broadened exploration, 2025b. URL https://arxiv.org/abs/2510.01180.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, Jie Cheng, Fanqi Wan, Xiangwen Kong, Chengyuan Yao, Ailin Huang, Hongyu Zhou, Qi Han, Zheng Ge, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Pacore: Learning to scale test-time compute with parallel coordinated reasoning, 2025c. URL https://github.com/stepfun-ai/PaCoRe/blob/main/pacore_report.pdf.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025d. URL https://arxiv.org/abs/2505.23885.

Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding, 2025. URL https://arxiv.org/abs/2502.11517.

Craig Kapfer, Kurt Stine, Balasubramanian Narasimhan, Christopher Mentzel, and Emmanuel Candes. Marlowe: Stanford's gpu-based computational instrument, January 2025. URL https://doi.org/10.5281/zenodo.14751899.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms, 2025. URL https://arxiv.org/abs/2510.13786.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems, 2025. URL https://arxiv.org/abs/2512.08296.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Wilson Lin. Scaling long-running autonomous coding, 2026. URL https://cursor.com/blog/scaling-agents. Cursor Blog (research).

Mingdao Liu, Aohan Zeng, Bowen Wang, Peng Zhang, Jie Tang, and Yuxiao Dong. Apar: Llms can do auto-parallel auto-regressive decoding, 2024. URL https://arxiv.org/abs/2401.06761.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models, 2025a. URL https://arxiv.org/abs/2505.24864.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Dler: Doing length penalty right - incentivizing more intelligence per token via reinforcement learning, 2025b. URL https://arxiv.org/abs/2510.15110.

Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, Shengyi Huang, Johan Obando-Ceron, Siran Yang, Jiamang Wang, Wenbo Su, and Bo Zheng. Part i: Tricks or traps? a deep dive into rl for llm reasoning, 2025c. URL https://arxiv.org/abs/2508.08221.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2, 2025. Notion Blog.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: Llms as improvement operators, 2025. URL https://arxiv.org/abs/2510.01123.

Mathematical Association of America. Amc 12 problems and solutions, 2023. URL https://artofproblemsolving.com/wiki/index.php/AMC_12_Problems_and_Solutions. Accessed: 2026-01-16.

Mathematical Association of America. American invitational mathematics examination 2024, 2024. URL https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination.

Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems, 2024. URL https://arxiv.org/abs/2412.09413.

MiniMax, :, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du, Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui Du, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Enwei Jiao, Haigang Zhou, Haimo Zhang, Han Ding, Haohai Sun, Haoyu Feng, Huaiguang Cai, Haichao Zhu, Jian Sun, Jiaqi Zhuang, Jiaren Cai, Jiayuan Song, Jin Zhu, Jingyang Li, Jinhao Tian, Jinli Liu, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kaiyi Feng, Ke Yang, Kecheng Xiao, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Li, Lin Zheng, Linge Du, Lingyu Yang, Lunbin Zeng, Minghui Yu, Mingliang Tao, Mingyuan Chi, Mozhi Zhang, Mujie Lin, Nan Hu, Nongyu Di, Peng Gao, Pengfei Li, Pengyu Zhao, Qibing Ren, Qidi Xu, Qile Li, Qin Wang, Rong Tian, Ruitao Leng, Shaoxiang Chen, Shaoyu Chen, Shengmin Shi, Shitong Weng, Shuchang Guan, Shuqi Yu, Sichen Li, Songquan Zhu, Tengfei Li, Tianchi Cai, Tianrun Liang, Weiyu Cheng, Weize Kong, Wenkai Li, Xiancai Chen, Xiangjun Song, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xinzhu Hou, Xuan Lu, Xun Zou, Xuyang Shen, Yan Gong, Yan Ma, Yang Wang, Yiqi Shi, Yiran Zhong, Yonghong Duan, Yongxiang Fu, Yongyi Hu, Yu Gao, Yuanxiang Fan, Yufeng Yang, Yuhao Li, Yulin Hu, Yunan Huang, Yunji Li, Yunzhi Xu, Yuxin Mao, Yuxuan Shi, Yuze Wenren, Zehan Li, Zelin Li, Zhanxu Tian, Zhengmao Zhu, Zhenhua Fan, Zhenzhen Wu, Zhichao Xu, Zhihang Yu, Zhiheng Lyu, Zhuo Jiang, Zibo Gao, Zijia Wu, Zijian Song, and Zijun Sun. Minimax-m1: Scaling test-time compute efficiently with lightning attention, 2025. URL https://arxiv.org/abs/2506.13585.

Moonshot AI. Introducing kimi k2 thinking. https://moonshotai.github.io/Kimi-K2/thinking.html. Accessed: 2026-01-23.

Moonshot AI. Kimi k2.5: Visual agentic intelligence. https://www.kimi.com/blog/kimi-k2-5.html, 2026. Accessed: 2026-01-27.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Re. Minions: Cost-efficient collaboration between on-device and cloud language models, 2025. URL https://arxiv.org/abs/2502.15964.

Xuan-Phi Nguyen, Shrey Pandit, Revanth Gangi Reddy, Austin Xu, Silvio Savarese, Caiming Xiong, and Shafiq Joty. Sfr-deepresearch: Towards effective reinforcement learning for autonomously reasoning single agents, 2025. URL https://arxiv.org/abs/2509.06283.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation, 2024. URL https://arxiv.org/abs/2307.15337.

OpenAI. Openai o3 and o4-mini system card, 2025. URL https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf.

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, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie

Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models, 2025. URL https://arxiv.org/abs/2504.15466.

Jianing Qi, Xi Ye, Hao Tang, Zhigang Zhu, and Eunsol Choi. Learning to reason across parallel samples for llm reasoning, 2025. URL https://arxiv.org/abs/2506.09014.

Amrith Setlur, Matthew Y. R. Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms, 2025. URL https://arxiv.org/abs/2506.09026.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Mingyang Song, Mao Zheng, Zheng Li, Wenjie Yang, Xuan Luo, Yue Pan, and Feng Zhang. Fastcurl: Curriculum reinforcement learning with stage-wise context scaling for efficient training r1-like reasoning models, 2025. URL https://arxiv.org/abs/2503.17287.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding, 2025. URL https://arxiv.org/abs/2510.11967.

Howe Tissue. Blog of entropy scheduling, 2025.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning, 2025. URL https://arxiv.org/abs/2506.01939.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute, 2025. URL https://arxiv.org/abs/2509.04475.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Chen Henry Wu, Sachin Goyal, and Aditi Raghunathan. Mode-conditioning unlocks superior test-time scaling, 2025a. URL https://arxiv.org/abs/2512.01127.

Tong Wu, Yang Liu, Jun Bai, Zixia Jia, Shuyi Zhang, Ziyong Lin, Yanting Wang, Song-Chun Zhu, and Zilong Zheng. Native parallel reasoner: Reasoning in parallelism via self-distilled reinforcement learning, 2025b. URL https://arxiv.org/abs/2512.07461.

Walden Yan. Don't build multi-agents, June 2025. URL https://cognition.ai/blog/dont-build-multi-agents#applying-the-principles. Cognition Blog.

Chenxiao Yang, Nathan Srebro, David McAllester, and Zhiyuan Li. Pencil: Long thoughts with short memory, 2025a. URL https://arxiv.org/abs/2503.14337.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation, 2025b. URL https://arxiv.org/abs/2506.09991.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

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, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b, 2024a. URL https://arxiv.org/abs/2406.07394.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft&; verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.acl-long.607. URL http://dx.doi.org/10.18653/v1/2024.acl-long.607.

Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. The majority is not always right: Rl training for solution aggregation, 2025. URL https://arxiv.org/abs/2509.06870.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. Parallel-r1: Towards parallel thinking via reinforcement learning, 2025. URL https://arxiv.org/abs/2509.07980.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems, 2025. URL https://arxiv.org/abs/2511.20639.

A Additional Related Work

Parallel Sampling for Inference-Time Scaling. Several works study how to improve the accuracy of LMs, via *parallel sampling* (Wang et al., 2023; Yao et al., 2023; Brown et al., 2024; Snell et al., 2024; Zhang et al., 2024a; Wu et al., 2025a; Wen et al., 2025; Qi et al., 2025; Zhao et al., 2025; Hassid et al., 2025) or via parallel sampling followed by iterative summarization and refinement (Madaan et al., 2025; Hu et al., 2025c). Wang et al. (2023) show that majority voting can achieve good performance when combined with CoT. Yao et al. (2023) propose Tree-of-Thoughts (ToT), and show that allowing an LM to backtrack can lead to performance improvements compared to standard CoT (for instance, greatly improving the performance of GPT-4 on Game of 24). Brown et al. (2024) study how independent sampling aggregation methods, such as majority voting and best-of-N with a reward model, scale as the number of samples increases. Snell et al. (2024) study compute-optimal allocation of test-time compute, comparing sequential refinement with strategies such as beam search with a process reward model. Wu et al. (2025a) aim to improve parallel sampling by heuristically guiding different samples to be drawn from different modes of the distribution, with the goal of ensuring that different samples pursue different strategies. Wen et al. (2025) train a model with SFT to generate multiple independent samples in parallel, then generate a summary — by prepending the token <think_i> to the start of the ith sample, Wen et al. (2025) aim to have the samples pursue different strategies. Qi et al. (2025) and Zhao et al. (2025) instead use RL to teach their models to aggregate independent samples from another model to arrive at the correct answer. In our method, the orchestrator does reasoning after seeing the problem and prior to assigning tasks. Thus, the workers perform distinct tasks and can terminate once their assigned task is done, leading to an improvement in latency, unlike with parallel sampling. Hassid et al. (2025) propose *short*-$m@k$ — this method is an alternative to majority voting, where k independent samples are generated, and once the shortest m samples are finished, generation is terminated. The final answer is then obtained by majority voting among these m responses. Our work is orthogonal to Hassid et al. (2025), as it is possible for our method to further improve the latency required by the shortest sample.

RLVR and Long CoT for Mathematical Reasoning OpenAI et al. (2024) and DeepSeek-AI et al. (2025) introduce reasoning models, which are trained using RL to output long chains-of-thought (CoTs) which dramatically improve the models' capabilities, including in mathematical reasoning. DeepSeek-AI et al. (2025) train their model with RL using only verifiable rewards (RLVR), using GRPO (Shao et al., 2024). Min et al. (2024) and Guha et al. (2025) curate demonstrations to teach models to perform long CoTs. Many works aim to achieve open-source reproductions of the RL procedure of DeepSeek-AI et al. (2025) using RLVR, as well as understand the impact of various hyperparameters and design choices (Luo et al., 2025; Yu et al., 2025; Liu et al., 2025a; He et al., 2025b; Hu et al., 2025a;b; An et al., 2025; Liu et al., 2025c; Song et al., 2025; Liu et al., 2025b; Setlur et al., 2025; Cui et al., 2025; Tissue, 2025). He et al. (2025a) aim to simplify existing recipes and identify which of the design choices are necessary for training stability and monotonic improvement. MiniMax et al. (2025) propose the CISPO algorithm, which is different from GRPO, using an objective akin to REINFORCE (Williams, 1992) but with truncated importance sampling — Khatri et al. (2025), among other findings, show that CISPO works well. Yang et al. (2025a) show that on synthetic tasks, long CoTs can benefit from removing information that is no longer useful from the context.

B FORMAL PSEUDOCODE FOR INFERENCE ALGORITHM

Algorithm 1 Algorithm for inference with a language model M that spawns worker threads.

Require: Language model M, Prompt P, number of workers K, longest path length budget L

S $\leftarrow$ INFER($M, P,$ "<spawn_workers>", L)
$P_{\text{temp}} \leftarrow P + \text{S}$
$L \leftarrow L - |\text{S}|$
RESPONSE $\leftarrow S$

while True **do**
    if P_{temp} contains `[EOS]` token **then**
        return P_{temp}
    for $i \leftarrow [1 \ldots K]$ in parallel **do**
        # Create prompts for each worker thread.
        $P_i \leftarrow P_{\text{temp}} +$ "<worker_i>"
        # Do inference for each worker thread.
        $W_i \leftarrow$ INFER($M, P_i,$ "</worker_i>", L) — this is done asynchronously
    Wait for W_i for $i \in [K]$
    $L \leftarrow L - \max_{i \in [K]} |W_i|$
    for $i \leftarrow [1 \ldots K]$ **do**
        $P_{\text{temp}} \leftarrow P_{\text{temp}} +$ "<worker_i>" $+ W_i$
        RESPONSE $\leftarrow$ RESPONSE + "<worker_i>" $+ W_i$
    $P_{\text{temp}} \leftarrow P_{\text{temp}} +$ "</spawn_workers>"
    RESPONSE $\leftarrow$ RESPONSE + "</spawn_workers>"
    $S \leftarrow$ INFER($M, P_{\text{temp}},$ "<spawn_workers>", L)
    $P_{\text{temp}} \leftarrow P_{\text{temp}} + S$
    RESPONSE $\leftarrow$ RESPONSE $+ S$
    $L \leftarrow L - |S|$
Return RESPONSE

C DETAILED INFERENCE EXAMPLE

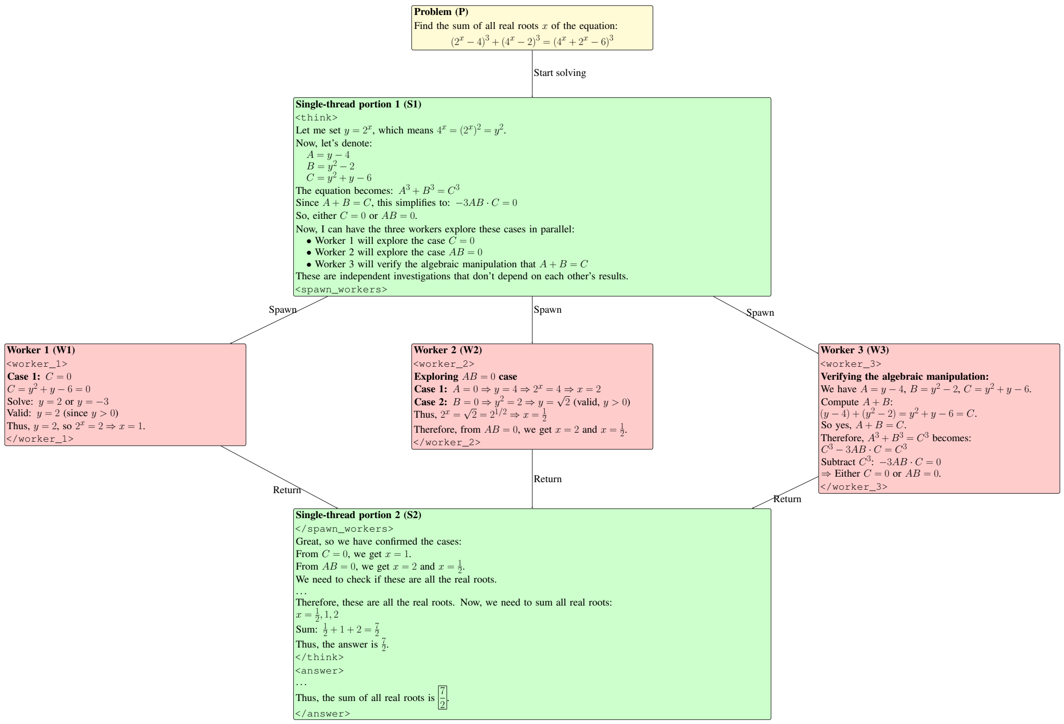


Figure 7: An instantiation of the inference architecture in Figure 2 with example thoughts. These are a condensed version of the thoughts that our model produced when given this problem during RL training.

D Prompts for SFT and RL

Box 1: Prompt Used During SFT and RL

```
prefix = "<|begin_of_sentence|><|User|>"
suffix = "<|Assistant|>"
instruction = "Let's think step by step and output the final
   answer within \\boxed{{}}."
parallel_instruction = "You can spawn multiple workers to
   solve this problem in parallel." \
                       " The workers' thoughts are enclosed
                          within <spawn_workers></
                          spawn_workers> tags, and each
                          worker's" \
                       " thought is enclosed within <worker_i
                          ></worker_i> tags, where i is the
                          worker number, i.e." \
                       " <spawn_workers><worker_1>worker 1's
                          thought</worker_1><worker_2>worker
                           2's thought</worker_2>..." \
                       "</spawn_workers>."
deepseek_r1_prompt_parallel = prefix + "{question} " +
   parallel_instruction + " " + instruction + suffix
```

E PROMPT FOR EVALUATING DEEPSCALER-1.5B-PREVIEW

Box 2: Prompt Used to Evaluate DeepScaleR-1.5B-Preview

```
prefix = "<|begin_of_sentence|><|User|>"
suffix = "<|Assistant|>"
instruction = "Let's think step by step and output the final
   answer within \\boxed{{}}."
think_token = "<think>\n"
deepseek_r1_prompt = prefix + "{question} " + instruction +
   suffix + think_token
```

F RL METHODS — ADDITIONAL DETAILS

GRPO/DAPO. Within an iteration, for each question, we generate G responses. For each response o_i for $i = 1, \ldots, G$, we calculate a reward R_i. Following Shao et al. (2024) and Yu et al. (2025), we calculate the normalized advantages $\hat{A}_i := \frac{R_i - \mu}{\sigma}$, where μ and σ are the mean and standard deviation, respectively, of R_i for $i = 1, \ldots, G$.

For a mini-batch with problems $p^{(j)}$ for $j = 1, \ldots, K$, with each question $p^{(j)}$ having responses $o_i^{(j)}$ for $i = 1, \ldots, G$, the objective is

$$\frac{1}{\sum_{j=1}^{K}\sum_{i=1}^{G}|o_i^{(j)}|}\sum_{j=1}^{K}\sum_{i=1}^{G}\sum_{t=1}^{|o_i^{(j)}|}\Big(\min(r_{j,i,t}(\theta)\hat{A}_i^{(j)},\ \text{clip}(\min(r_{j,i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}})\hat{A}_i^{(j)}) \\ - \beta \text{KL}(\pi_\theta \mid \pi_{\text{ref}})\Big)$$

where the ratios $r_{j,i,t}(\theta) := \frac{\pi_\theta(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})}{\pi_{\text{old}}(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})}$, π_θ denotes the current policy with up-to-date parameters, π_{old} denotes the policy used to generate the responses, $o_{i,t}^{(j)}$ denotes the tth token of the response, and $o_{i,<t}^{(j)}$ denotes the prefix up to (but not including) the tth token. We estimate $\text{KL}(\pi_\theta \mid \pi_{\text{ref}})$ in a manner similar to Shao et al. (2024), using the formula $\frac{\pi_{\text{ref}}(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})}{\pi_\theta(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})} - \log \frac{\pi_{\text{ref}}(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})}{\pi_\theta(o_{i,t}^{(j)}|p, o_{i,<t}^{(j)})} - 1$. The questions $p^{(j)}$ included in the batch are filtered according to certain criteria — details of the filtering are discussed below. Following Yu et al. (2025), we set $\varepsilon_{\text{high}} = 0.28$ and $\varepsilon_{\text{low}} = 0.2$.

CISPO. Partway through RL training, we change the objective — instead of the GRPO/DAPO objective, we use the objective of CISPO, introduced by MiniMax et al. (2025) and further found to work well by Khatri et al. (2025). For a mini-batch with questions $p^{(j)}$ for $j = 1, \ldots, K$, with each question $p^{(j)}$ having responses $o_i^{(j)}$ for $i = 1, \ldots, G$, the objective is now

$$\frac{1}{\sum_{j=1}^{K}\sum_{i=1}^{G}|o_i^{(j)}|}\sum_{j=1}^{K}\sum_{i=1}^{G}\sum_{t=1}^{|o_i^{(j)}|}\Big(\min(r_{j,i,t}(\theta), \varepsilon_{\text{high}})\hat{A}_i^{(j)} \log \pi_\theta(o_{i,t}^{(j)} \mid p, o_{i,<t}^{(j)}) - \beta \text{KL}(\pi_\theta \mid \pi_{\text{ref}})\Big)$$

Observe that if only a single optimization step is performed for each rollout batch, then this objective reduces to REINFORCE (Williams, 1992), due to being fully on-policy — we perform 2 optimization steps per rollout batch. We continue computing the normalized advantages $\hat{A}$ by taking the standard deviation of the rewards across all responses for a pproblem p. This is different from Khatri et al. (2025), who find that normalizing the advantages across the entire mini-batch performs slightly better.

HLP Reward Function. Our reward function in the HLP stage is

$$g(o) := \begin{cases} 1 - \ell(o) & \text{if } o \text{ is correct and satisfies format} \\ 0.01 & \text{if } o \text{ is correct but does not satisfy format} \\ 0 & \text{otherwise} \end{cases}$$

Here, we only apply the length penalty to responses that are both correct and satisfy the format — our reasoning is as follows. With our reward function $g(o)$, we ensure that responses that are correct, but do not satisfy the format, will not have higher reward than responses that are correct and satisfy the format. Since $1 - \ell(o)$ could be as low as 0.1 for a response o that is correct and satisfies the format, we must define g so that responses which are correct and do not satisfy the format always have reward less than 0.1. Additionally, we would like all correct responses (regardless of format) to have higher reward than incorrect responses. Thus, in summary, all responses o that are correct, but do not satisfy the format, should have reward strictly greater than 0 and less than 0.1. We believe any value between 0 and 0.1 would work well — we choose 0.01 arbitrarily. Finally, we do not apply the length penalty to responses that are correct, but do not satisfy the format, to ensure that their reward does not fall below 0. We continue to filter problems p for which all responses are correct or all responses are incorrect.

G Plateaus in Accuracy — Stage 1 and Stage 3

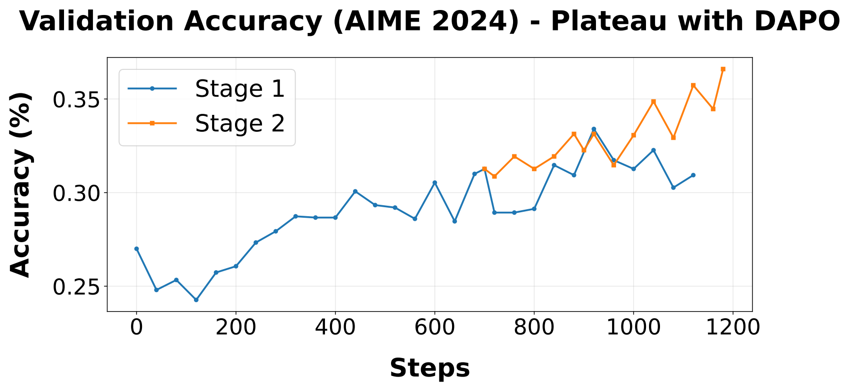


Figure 8: DAPO plateaus in accuracy during Stage 1, while accuracy continues improving monotonically once we switch to CISPO.

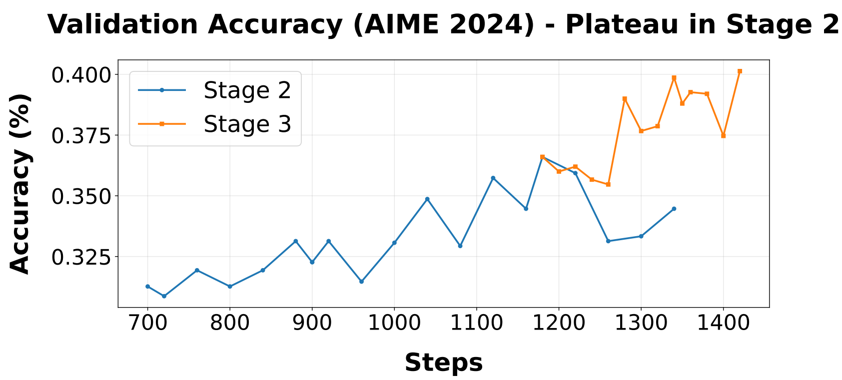


Figure 9: We encounter a plateau in accuracy in Stage 2, where we use filtering strategy *Include-Easy*. Accuracy continues improving monotonically in Stage 3 when we switch to filtering strategy *Remove-Easy*.

H ENTROPY STATISTICS DURING STAGE 1

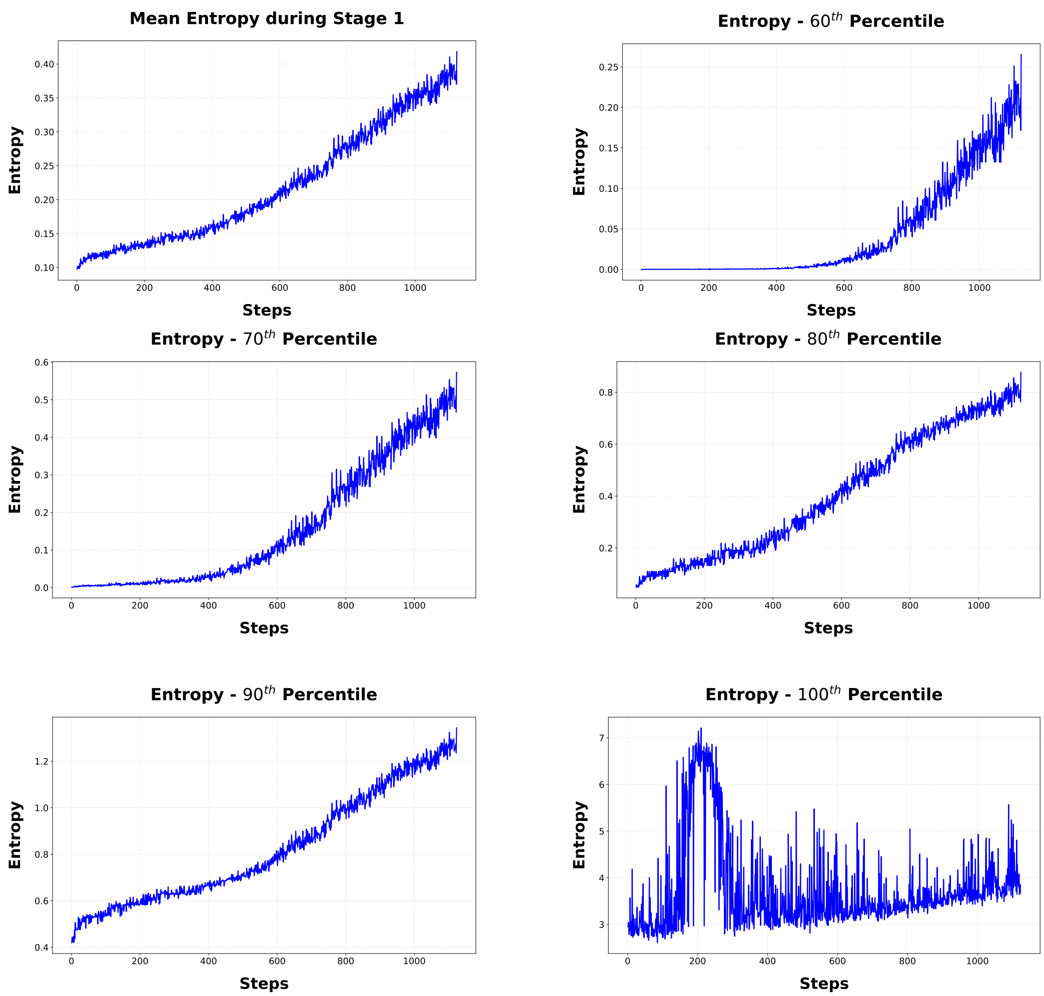


Figure 10: Entropy statistics during stage 1, when using DAPO with $\varepsilon_{\text{high}} = 0.28$.

I Additional Experimental Details

In this section, we specify hyperparameters and experimental details that were omitted from Section 5.1.

SFT Dataset — Creation of $\mathcal{D}_{\text{init}}$ and $\mathcal{D}_{\text{rewrite}}$. Our base model M_{base} is DeepScaleR-1.5B-Preview (Luo et al., 2025) and we obtain its responses on the first $15K$ problems in the DeepScaleR training dataset to obtain $\mathcal{D}_{\text{init}}$. We verify the accuracy of the responses using `math_verify`. [1] We use temperature 0.6 and set `top_p` $= 0.95$ when sampling with vLLM (Kwon et al., 2023). We allow response length up to $24K$ and use the prompt shown in Box 3. After rejection sampling, we end up with 9968 problem-answer pairs — we include the first $4K$ of these in $\mathcal{D}_{\text{init}}$. We use Claude Sonnet 4.5 as our rewriting model M_{rewrite}. [2] When generating with M_{rewrite}, we use temperature 0.7 and response length $12K$. Our final dataset $\mathcal{D}_{\text{rewrite}}$ contains 3411 examples.

SFT Hyperparameters. For SFT, we use the prompt in Box 1. We use batch size 8, and learning rate $5 \cdot 10^{-6}$, for 1800 steps. We use linear learning rate warmup for 105 steps. We construct the training sequences as described in Section 4.2, and remove any training sequences with length more than $10,000$ tokens.

RL Details and Hyperparameters. We use `verl` (Sheng et al., 2024), with modifications for our inference architecture (Section 3) and the customized training sequences, attention mask and position IDs described in Section 4.2. We use the DeepScaleR training dataset (Luo et al., 2025), and AIME 2024 (Mathematical Association of America, 2024) for our validation dataset, performing 50 rollouts per problem during each validation step. We use `math_verify` to verify accuracy. Our prompt is given in Box 1. Stages 1, 2, 3 and 4 last for 700, 480, 240 and 220 steps respectively. The HLP stage lasts for 200 steps.

The hyperparameters that we use in each stage of our training run are as follows. In Stage 1, we use learning rate $7.071 \cdot 10^{-7}$, rollout batch size 288, and training batch size 96. In Stage 2, for the first 200 steps, we use learning rate $1.414 \cdot 10^{-6}$, rollout batch size 288, and training batch size 96. Afterwards, we increase the training batch size to 192, and the learning rate to $2.828 \cdot 10^{-6}$. In Stage 3, the rollout batch size is 288, the training batch size is 96, and the learning rate is $2 \cdot 10^{-6}$. Finally, in Stage 4, the rollout batch size is 360, the training batch size is 96, and the learning rate is $2 \cdot 10^{-6}$. For each rollout batch, we perform 2 optimization steps, making our algorithm off-policy. In each stage that uses CISPO, we set $\varepsilon_{\text{high}} = 5.0$.

For our high length penalty (HLP) stage, we begin from the final step of Stage 4. The learning rate is $2 \cdot 10^{-6}$, the rollout batch size is 360, and the training batch size is 96.

Evaluation Benchmarks. For evaluation, we use AIME 2024 (Mathematical Association of America, 2024), AMC 23 (Mathematical Association of America, 2023), HMMT Feb 2025 (Balunović et al., 2025), MATH 500 (Hendrycks et al., 2021), Minerva Math (Lewkowycz et al., 2022), and Olympiad-Bench (He et al., 2024). To evaluate pass@1 and maj@3, for AIME 2024, AMC 23 and HMMT 2025, we use 200 rollouts per problem, and for MATH 500, Minerva Math, and Olympiad-Bench, we use 16 rollouts per problem. Results on all benchmarks are reported in Figure 11 and Table 1 in Section J.

HLP with DeepScaleR-1.5B-Preview. For **DSR-HLP-24K** and **DSR-HLP-12K**, we use learning rate $2 \cdot 10^{-6}$, training batch size 96 and rollout batch size 360. We filter problems p based solely on the correctness of the final answer of the responses to p, as in **DC-CoT-HLP**. We use a length penalty with $C_L = 0.9$ and $L_{\text{cutoff}} = 2000$. As in the original training of DeepScaleR-1.5B-Preview (Luo et al., 2025), we assign reward 1.0 to responses which are correct, which include a thinking process between <think> and </think> tags, and for which the answer comes after </think>, and 0 otherwise. We ensure that <think> is included at the end of the prompt. We run **DC-CoT-HLP**, **DSR-HLP-24K**, and **DSR-HLP-12K** for 200 steps each, and for each method, we select the best checkpoint by assessing the accuracy and longest path length on AIME 2024. We train and evaluate **DSR-HLP-24K** and **DSR-HLP-12K** using the prompt shown in Box 2.

Majority Voting Baseline — Implementation Details. As mentioned in Section 5.1, we generate N responses per problem, where N is 200 or 16 depending on the evaluation benchmark, using **DSR-12K**. To determine the pass rate and longest path length of **Maj-DSR-3** for a given problem,

[1] `https://github.com/huggingface/Math-Verify`

[2] The specific version we use is `claude-sonnet-4-5@20250929`.

we take all $\binom{N}{3}$ subsets $S = \{o_1, o_2, o_3\}$ of size 3 from the responses to this problem, and compute the accuracy and longest path length when using S for majority voting. For a particular subset S, we compute the accuracy and longest path length as follows. We first use `math_verify` to determine the number of equivalence classes of final answers. If there is only a single equivalence class, we compute the accuracy of each of the 3 responses and take the average (corresponding to sampling a representative uniformly at random). If there are two equivalence classes, one of which has 2 elements, we use the equivalence class with 2 elements, and take the average of the accuracies of those two elements. If there are 3 equivalence classes, then we again take the average accuracy of all 3 responses (corresponding to sampling a class uniformly at random). In all cases, the longest path length of majority voting is calculated as the maximum longest path length out of these 3 responses.

J Additional Tables and Plots

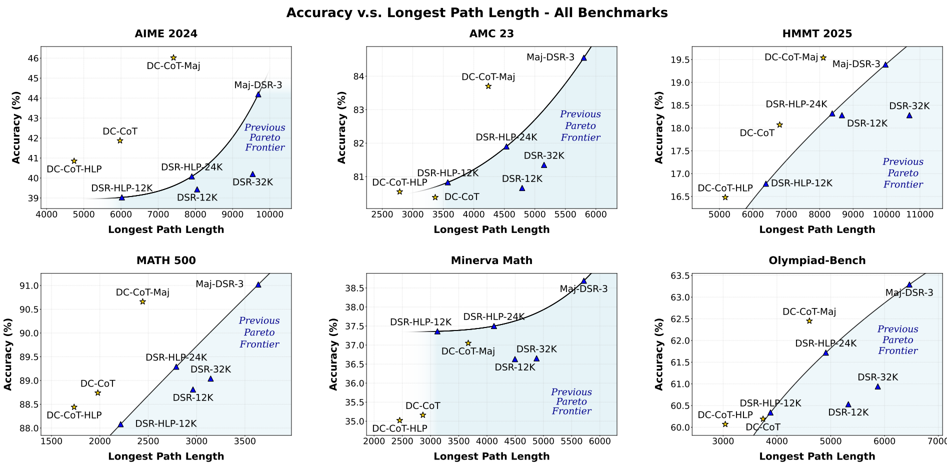


Figure 11: Accuracy (pass@1) and longest path length for all methods on all benchmarks.

Table 1: Accuracy and longest path length (LPL) for all methods and benchmarks discussed in Section 5.1. Error bars for all accuracies and longest path lengths shown in Section J, Table 4 and Table 5.

Method	AIME 24		AMC		HMMT		MATH		MM		OB	
	Acc (%)	LPL	Acc (%)	LPL	Acc (%)	LPL	Acc (%)	LPL	Acc (%)	LPL	Acc (%)	LPL
DSR-32K	40.20	9540	81.35	5151	18.28	10685	89.04	3146	36.65	4879	60.95	5875
DSR-12K	39.43	8047	80.66	4792	18.28	8666	88.81	2963	36.63	4496	60.53	5327
Maj-DSR-3	44.19	9690	84.55	5803	19.39	9970	91.02	3638	38.69	5716	63.29	6462
DSR-HLP-24K	40.08	7903	81.90	4535	18.32	8375	89.29	2790	37.50	4124	61.72	4909
DSR-HLP-12K	39.03	6023	80.83	3574	16.78	6391	88.08	2215	37.36	3122	60.34	3880
DC-CoT	41.87	5974	80.38	3368	18.07	6803	88.74	1979	35.16	2865	60.19	3742
DC-CoT-HLP	40.85	4739	80.55	2786	16.48	5175	88.44	1734	35.08	2454	60.07	3040
DC-CoT-Maj	46.02	7412	83.70	4239	19.54	8109	90.66	2443	37.04	3668	62.45	4603

Table 2: Change in accuracy and longest path length of our methods compared to appropriate baselines. For **DC-CoT** we use **DSR-32K** as the baseline, for **DC-CoT-HLP** we use **DSR-HLP-12K** as the baseline, and for **DC-CoT-Maj** we use **Maj-DSR-3** as the baseline.

Method	Baseline	AIME 24		AMC		HMMT		MATH		MM		OB	
		Acc	LPL	Acc	LPL	Acc	LPL	Acc	LPL	Acc	LPL	Acc	LPL
DC-CoT	**DSR-32K**	+1.67	-37.4%	-0.97	-34.6%	-0.21	-36.3%	-0.3	-37.1%	-1.49	-41.3%	-0.76	-36.3%
DC-CoT-HLP	**DSR-12K-HLP**	+1.82	-21.3%	-0.28	-22.0%	-0.3	-19.02%	+0.36	-21.7%	-2.28	-21.4%	-0.27	-21.6%
DC-CoT-Maj	**Maj-DSR-3**	+1.83	-23.5%	-0.85	-27.0%	+0.15	-18.67%	-0.36	-32.8%	-1.65	-35.8%	-0.84	-28.8%

Table 3: Total tokens used by each response, averaged across responses. For **DC-CoT-Maj**, we average across all subsets of 3 responses to the same problem. For each subset of 3 responses, we compute the sum of the tokens used by each of the 3 responses.

Method	AIME 24	AMC	HMMT	MATH	MM	OB
DC-CoT	6831	4185	7603	2554	3526	4489
DC-CoT-HLP	6185	4022	6411	2537	3401	4154
DC-CoT-Maj	20493	12554	22809	7662	10579	13468

Table 4: Results in Table 1 with sample mean and sample standard deviation (AIME 2024, AMC 23, HMMT 2025).

Method	AIME 24		AMC		HMMT	
	Acc (%)	PL	Acc (%)	PL	Acc (%)	PL
DSR-32K	$40.20_{\pm 0.63}$	$9540_{\pm 83}$	$81.35_{\pm 0.44}$	$5151_{\pm 51}$	$18.28_{\pm 0.50}$	$10685_{\pm 89}$
DSR-12K	$39.43_{\pm 0.63}$	$8047_{\pm 43}$	$80.66_{\pm 0.44}$	$4792_{\pm 38}$	$18.28_{\pm 0.50}$	$8666_{\pm 41}$
Maj-DSR	$44.19_{\pm 0.01}$	$9690_{\pm 0.0}$	$84.55_{\pm 0.0}$	$5803_{\pm 1}$	$19.39_{\pm 0.01}$	$9970_{\pm 0.0}$
DSR-HLP-24K	$40.08_{\pm 0.63}$	$7903_{\pm 54}$	$81.90_{\pm 0.43}$	$4535_{\pm 40}$	$18.32_{\pm 0.50}$	$8375_{\pm 55}$
DSR-HLP-12K	$39.03_{\pm 0.63}$	$6023_{\pm 38}$	$80.83_{\pm 0.44}$	$3574_{\pm 29}$	$16.78_{\pm 0.48}$	$6391_{\pm 38}$
DC-CoT	$41.87_{\pm 0.64}$	$5974_{\pm 38}$	$80.38_{\pm 0.44}$	$3368_{\pm 29}$	$18.07_{\pm 0.50}$	$6802_{\pm 38}$
DC-CoT-HLP	$40.85_{\pm 0.63}$	$4740_{\pm 27}$	$80.55_{\pm 0.44}$	$2786_{\pm 21}$	$16.48_{\pm 0.48}$	$5175_{\pm 28}$
DC-CoT-Maj	$46.02_{\pm 0.01}$	$7412_{\pm 0.0}$	$83.70_{\pm 0.00}$	$4239_{\pm 0.43}$	$19.54_{\pm 0.01}$	$8109_{\pm 0.0}$

Table 5: Results in Table 1 with sample mean and sample standard deviation (MATH 500, Minerva Math, Olympiad Bench).

Method	MATH		MM		OB	
	Acc (%)	PL	Acc (%)	PL	Acc (%)	PL
DSR-32K	$89.04_{\pm 0.35}$	$3146_{\pm 39}$	$36.65_{\pm 0.73}$	$4879_{\pm 71}$	$60.94_{\pm 0.47}$	$5875_{\pm 49}$
DSR-12K	$88.81_{\pm 0.35}$	$2963_{\pm 29}$	$36.62_{\pm 0.73}$	$4496_{\pm 47}$	$60.53_{\pm 0.47}$	$5327_{\pm 33}$
Maj-DSR	$91.02_{\pm 0.05}$	$3638_{\pm 6}$	$38.69_{\pm 0.12}$	$5716_{\pm 9}$	$63.29_{\pm 0.08}$	$6462_{\pm 6}$
DSR-HLP-24K	$89.29_{\pm 0.35}$	$2790_{\pm 30}$	$37.50_{\pm 0.73}$	$4124_{\pm 49}$	$61.72_{\pm 0.47}$	$4909_{\pm 34}$
DSR-HLP-12K	$88.08_{\pm 0.36}$	$2215_{\pm 22}$	$37.36_{\pm 0.73}$	$3122_{\pm 33}$	$60.34_{\pm 0.47}$	$3880_{\pm 25}$
DC-CoT	$88.74_{\pm 0.35}$	$1979_{\pm 22}$	$35.16_{\pm 0.72}$	$2865_{\pm 32}$	$60.19_{\pm 0.47}$	$3742_{\pm 26}$
DC-CoT-HLP	$88.44_{\pm 0.36}$	$1734_{\pm 16}$	$35.02_{\pm 0.72}$	$2454_{\pm 25}$	$60.07_{\pm 0.47}$	$3040_{\pm 19}$
DC-CoT-Maj	$90.66_{\pm 0.05}$	$2443_{\pm 5}$	$37.05_{\pm 0.12}$	$3668_{\pm 6}$	$62.45_{\pm 0.08}$	$4603_{\pm 5}$

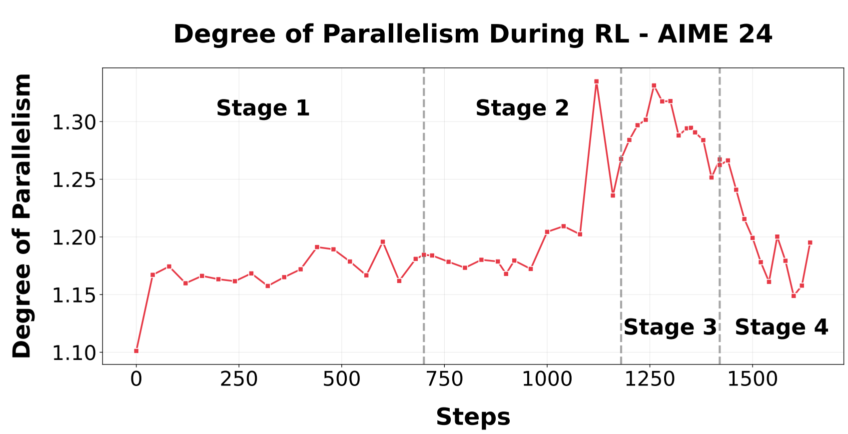


Figure 12: Degree of parallelism of **DC-CoT** on AIME 2024 during all stages of RL.

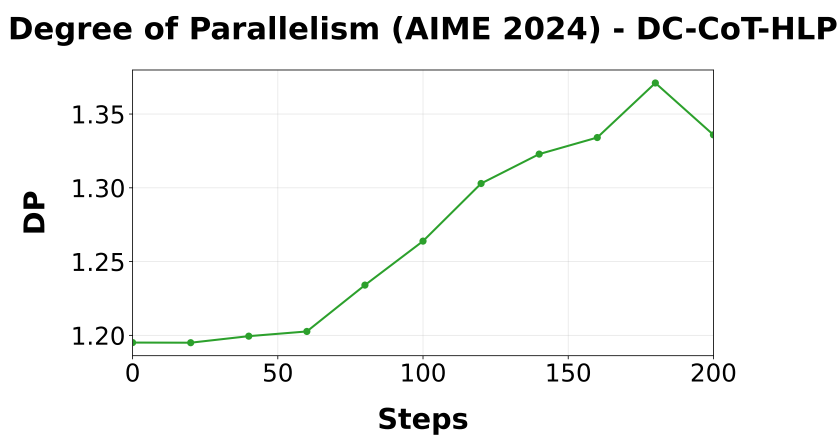


Figure 13: Degree of parallelism on AIME 2024 for DC-CoT-HLP

K PROMPTS FOR CREATING SFT DATASET

Box 3: Prompt for Generating Sequential CoTs

```
instruction = "Let's think step by step and output the final
 ↪  answer within \\boxed{{}}."
deepseek_r1_prompt = "{question} " + instruction +
 ↪  "<think>\n"
```

Box 4: Prompt for `claude-sonnet-4-5@20250929` to Rewrite Sequential CoTs with Parallelization

```
Given a problem, Alice provides a solution in the format
 ↪  <think>some thoughts here</think>final answer here.
Once the <think></think> tags are done, she includes a little
 ↪  bit of discussion, followed by the final answer inside
 ↪  \boxed{{}}.
Your job is to rewrite Alice's solution in a new format,
 ↪  where multiple workers can sometimes collaborate to solve
 ↪  parts of the problem in parallel.
NOTE: When I say "rewrite" Alice's solution, I mean that you
 ↪  should copy most parts verbatim, but insert some
 ↪  parallelism when it makes sense.
There are 3 workers. At any point in the solution where you
 ↪  feel it is appropriate, you come up with 3 tasks, each of
 ↪  which is a subtask of the problem. The 3 workers then
 ↪  solve this task in parallel. To invoke the workers, you
 ↪  use the token <spawn_workers>. When the workers begin
 ↪  thinking, they output <worker_1>, <worker_2>, and
 ↪  <worker_3>, respectively. When the workers are done
 ↪  thinking, they output </worker_1>, </worker_2>, and
 ↪  </worker_3>, respectively. Multiple rounds of parallel
 ↪  thinking and answering may take place. Your response
 ↪  should thus have the following format:
<think>
some thoughts (which could potentially be very long) and an
 ↪  outline of what each worker should do
The thoughts here should be copied directly from Alice's
 ↪  solution - but spawn workers when it makes sense
Each time before you spawn workers - give a justification of
 ↪  why the calculation can be parallelized. (This
 ↪  justification should preserve the writing style of
 ↪  Alice's original solution as much as possible, and spell
 ↪  out the thought process step by step.)
- Feel free to make this justification as detailed as you
 ↪  would like. This justification might include some kind of
 ↪  thinking/reflection (for example, in Alice's original
 ↪  solution, you might see some thinking/second-guessing,
 ↪  such as "Wait, what if..." or other phrases along those
 ↪  lines). However if it is truly trivial then it doesn't
 ↪  have to be too long.
```

```
Additionally, when you propose subtasks for the workers, give
 ↪  a justification of why they are independent. Worker 1,
 ↪  Worker 2, and Worker 3's tasks should be precise, and
 ↪  should not rely on each other's results. E.g. worker 2
 ↪  should be able to do its task without the results of
 ↪  Worker 1 or Worker 3's tasks. (This justification should
 ↪  preserve the writing style of Alice's original solution
 ↪  as much as possible, and spell out the thought process
 ↪  step by step.)
- Feel free to make this justification as detailed as you
 ↪  would like. This justification might include some kind of
 ↪  thinking/reflection (for example, in Alice's original
 ↪  solution, you might see some thinking/second-guessing,
 ↪  such as "Wait, what if..." or other phrases along those
 ↪  lines). However if it is truly trivial then it doesn't
 ↪  have to be too long.
<spawn_workers>
<worker_1> worker 1 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers </worker_1>
<worker_2> worker 2 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers</worker_2>
<worker_3> worker 3 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers </worker_3>
</spawn_workers>
Do some further reflection based on the workers' thoughts. If
 ↪  needed, use the parallel workers again and again - but if
 ↪  you decide to call the workers, then you must always
 ↪  explicitly state what worker 1, worker 2 and worker 3
 ↪  should do in the upcoming block
Additionally, this portion should be almost directly copied
 ↪  from Alice's solution, aside from the part where you
 ↪  assign tasks to workers.
For assigning tasks to workers, again you should give
 ↪  justifications of why the calculations can be
 ↪  parallelized, why the tasks are independent, etc.
<spawn_workers>
<worker_1> worker 1 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers </worker_1>
<worker_2> worker 2 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers </worker_2>
<worker_3> worker 3 thinking - this should be copied directly
 ↪  from Alice's solution almost, but you can edit it very
 ↪  slightly ONLY to remove all references to tasks that are
 ↪  supposed to be done by other workers</worker_3>
</spawn_workers>
Further reflection (which can potentially be long), and
 ↪  optionally more rounds of parallel workers.
```

```
<spawn_workers>
<worker_1> worker 1 thinking </worker_1>
<worker_2> worker 2 thinking </worker_2>
<worker_3> worker 3 thinking </worker_3>
</spawn_workers>
....
....
Finally, wrap up the thought process after the last round of
 ↪  parallel workers is done
</think>
<answer>
answer here - this part should not contain any other tags
 ↪  such as the thinking tags/worker tags.
Within this block, you may include some discussion prior to
 ↪  the final answer.
The final answer should always be included in \boxed{{}}.
This part should be essentially copied from the portion of
 ↪  Alice's solution that comes after </think>.
</answer>

From a worker's perspective, it will not see the other
 ↪  workers' thoughts. Thus, worker 2 does not see
 ↪  <worker_1>, </worker_1>, or what goes between these two
 ↪  tags, for example.

====================

VERY IMPORTANT:
- As far as possible, copy text only from Alice's solution,
 ↪  instead of adding your own language.
- Only add your own language (that is not present in Alice's
 ↪  solution) when it is absolutely necessary. E.g. when you
 ↪  give justification for using parallel workers, since this
 ↪  will not be present in Alice's solution. Or for example,
 ↪  if a worker is performing a partial calculation, but does
 ↪  not have enough information to say exactly what Alice
 ↪  would have said.
- Basically, when writing the tasks that each worker is
 ↪  supposed to do, always keep thinking about what
 ↪  information this worker is supposed to have access to.
 ↪  For example, in some algebra problem, if Worker 1 finds
 ↪  that x=5, then within that same round of workers, Workers
 ↪  2 and 3 should NOT have access to the knowledge that x=5.
- Within workers, make sure to use the same style as Alice.
 ↪  The portion WITHIN the workers should also be COPIED FROM
 ↪  ALICE AS MUCH AS POSSIBLE (with some caveats, as I
 ↪  discussed previously).

VERY IMPORTANT:
- Pretend you are Alice, but with access to multiple workers.
 ↪  Do NOT use phrases such as "Alice said," etc. You are
 ↪  rewriting the solution so that someone who has not read
 ↪  Alice's solution will be able to completely follow it.
```

```
- Include all the trial and error that was done by Alice. In
 ↪  particular, Alice uses a lot of phrases such as "Wait,
 ↪  let me double-check" and "wait, no. Let's think again."
 ↪  Please include all of these phrases in your re-written
 ↪  version
- However, the solution should be completely self-contained
 ↪  with no references to Alice's original work or to an
 ↪  'original solution.'
- Write as if you are solving the problem for the first time
 ↪  using the parallel workers approach, not rewriting
 ↪  someone else's solution.
- Avoid any meta-commentary about the solution being a
 ↪  rewrite or referring to other solutions.
- Within a <spawn_workers></spawn_workers> block, each worker
 ↪  functions independently and cannot see the work of other
 ↪  workers until after the </spawn_workers> tag. Workers
 ↪  should never reference each others' work or conclusions
 ↪  withi
n a <spawn_workers> block.
- In other words, when writing worker sections, imagine each
 ↪  worker is in a separate room with no communication until
 ↪  their work is collected at the end of a spawn cycle.

==================== EXAMPLE OF REWRITING TASK

The following is an example of the task that you must
 ↪  accomplish.

The math problem is as follows.

The problem was The matrices\n\\[\\begin{{pmatrix}} a & 1 & b
 ↪  \\\\ 2 & 2 & 3 \\\\ c & 5 & d \\end{{pmatrix}} \\quad
 ↪  \\text{{and}} \\quad \\begin{{pmatrix}} -5 & e & -11 \\\\
 ↪  f & -13 & g \\\\ 2 & h & 4 \\end{{pmatrix}}\\]are
 ↪  inverses.
 Find $a + b + c + d + e + f + g + h.$

Alice's solution is as follows:

<think>\nOkay, so I have this problem where two matrices are
 ↪  given, and they're supposed to be inverses of each other.
 ↪  I need to find the sum of all these variables: a, b, c,
 ↪  d, e, f, g, and h. Hmm, let me think about how to
 ↪  approach
this.\n\nFirst, I remember that if two matrices are inverses,
 ↪  their product should be the identity matrix. So, if
 ↪  matrix A multiplied by matrix B gives the identity
 ↪  matrix, then B is the inverse of A, and vice versa. So,
 ↪  my plan is to
 multiply these two matrices together and set each element of
  ↪  the resulting matrix equal to the corresponding element
  ↪  of the identity matrix. That should give me a system of
  ↪  equations to solve for all the variables.\n\nLet me
  ↪  write do
```

```
wn the matrices again to make sure I have them
 ↪  correctly:\n\nMatrix A:\n\\[\n\\begin{{pmatrix}} \na & 1
 ↪  & b \\\\ \n2 & 2 & 3 \\\\ \nc & 5 & d
 ↪  \n\\end{{pmatrix}}\n\\]\n\nMatrix
 ↪  B:\n\\[\n\\begin{{pmatrix}} \n-5 & e & -11 \\\\ \nf & -13
& g \\\\ \n2 & h & 4 \n\\end{{pmatrix}}\n\\]\n\nAnd the
 ↪  identity matrix, which is:\n\\[\n\\begin{{pmatrix}} \n1 &
 ↪  0 & 0 \\\\ \n0 & 1 & 0 \\\\ \n0 & 0 & 1
 ↪  \n\\end{{pmatrix}}\n\\]\n\nSo, when I multiply A and B, I
 ↪  should get the identit
y matrix. Let's denote the product of A and B as matrix C,
 ↪  where C = A * B. Then, C should equal the identity
 ↪  matrix. So, I can set up equations for each element of C
 ↪  by equating them to the corresponding elements of the
 ↪  identity matr
ix.\n\nLet me compute each element one by one.\n\nFirst,
 ↪  let's compute the element in the first row and first
 ↪  column of matrix C.\n\nElement C11 = (a)(-5) + (1)(f) +
 ↪  (b)(2) = -5a + f + 2b\n\nThis should equal 1, because the
 ↪  identity m
atrix has 1 in the (1,1) position.\n\nSo, equation 1: -5a + f
 ↪  + 2b = 1\n\nNext, element C12 = (a)(e) + (1)(-13) +
 ↪  (b)(h) = ae - 13 + bh\n\nThis should equal 0, since the
 ↪  identity matrix has 0 in the (1,2) position.\n\nEquation
 ↪  2: ae -
 13 + bh = 0\n\nThen, element C13 = (a)(-11) + (1)(g) +
  ↪  (b)(4) = -11a + g + 4b\n\nThis should equal 0 as
  ↪  well.\n\nEquation 3: -11a + g + 4b = 0\n\nMoving on to
  ↪  the second row.\n\nElement C21 = (2)(-5) + (2)(f) +
  ↪  (3)(2) = -10 + 2f + 6
= 2f - 4\n\nThis should equal 0.\n\nEquation 4: 2f - 4 =
 ↪  0\n\nElement C22 = (2)(e) + (2)(-13) + (3)(h) = 2e - 26 +
 ↪  3h\n\nThis should equal 1.\n\nEquation 5: 2e - 26 + 3h =
 ↪  1\n\nElement C23 = (2)(-11) + (2)(g) + (3)(4) = -22 + 2g
 ↪  + 12
= 2g - 10\n\nThis should equal 0.\n\nEquation 6: 2g - 10 =
 ↪  0\n\nProceeding to the third row.\n\nElement C31 =
 ↪  (c)(-5) + (5)(f) + (d)(2) = -5c + 5f + 2d\n\nThis should
 ↪  equal 0.\n\nEquation 7: -5c + 5f + 2d = 0\n\nElement C32
 ↪  = (c)(e) +
 (5)(-13) + (d)(h) = ce - 65 + dh\n\nThis should equal
  ↪  0.\n\nEquation 8: ce - 65 + dh = 0\n\nFinally, element
  ↪  C33 = (c)(-11) + (5)(g) + (d)(4) = -11c + 5g +
  ↪  4d\n\nThis should equal 1.\n\nEquation 9: -11c + 5g + 4d
  ↪  = 1\n\nOkay, so now
I have a system of 9 equations with 8 variables: a, b, c, d,
 ↪  e, f, g, h. Wait, 9 equations and 8 variables? That seems
 ↪  overdetermined, but maybe the equations are
 ↪  consistent.\n\nLet me write down all the equations
 ↪  again:\n\n1. -5a + f
 + 2b = 1\n2. ae - 13 + bh = 0\n3. -11a + g + 4b = 0\n4. 2f -
  ↪  4 = 0\n5. 2e - 26 + 3h = 1\n6. 2g - 10 = 0\n7. -5c + 5f
  ↪  + 2d = 0\n8. ce - 65 + dh = 0\n9. -11c + 5g + 4d =
  ↪  1\n\nHmm, okay, let's try to solve these step by
  ↪  step.\n\nStartin
```

```
g with equation 4: 2f - 4 = 0. That should be
 ↪  straightforward.\n\nEquation 4: 2f - 4 = 0 => 2f = 4 => f
 ↪  = 2.\n\nOkay, so f = 2.\n\nNow, equation 2: ae - 13 + bh
 ↪  = 0. Since we know f = 2, we can plug that in
 ↪  later.\n\nEquation 6: 2g -
10 = 0 => 2g = 10 => g = 5.\n\nSo, g = 5.\n\nEquation 3: -11a
 ↪  + g + 4b = 0. We know g = 5, so:\n\n-11a + 5 + 4b = 0 =>
 ↪  -11a + 4b = -5. Let's call this equation 3a.\n\nEquation
 ↪  7: -5c + 5f + 2d = 0. f = 2, so:\n\n-5c + 10 + 2d = 0 =>
 ↪  -
5c + 2d = -10. Let's call this equation 7a.\n\nEquation 5: 2e
 ↪  - 26 + 3h = 1 => 2e + 3h = 27. Let's call this equation
 ↪  5a.\n\nEquation 1: -5a + f + 2b = 1. f = 2, so:\n\n-5a +
 ↪  2 + 2b = 1 => -5a + 2b = -1. Let's call this equation
 ↪  1a.\n
\nEquation 8: ce - 65 + dh = 0. Let's keep this as equation
 ↪  8.\n\nEquation 9: -11c + 5g + 4d = 1. g = 5, so:\n\n-11c
 ↪  + 25 + 4d = 1 => -11c + 4d = -24. Let's call this
 ↪  equation 9a.\n\nSo now, let's recap the equations:\n\n1a:
 ↪  -5a + 2b
= -1\n2: ae - 13 + bh = 0\n3a: -11a + 4b = -5\n4: f = 2\n5a:
 ↪  2e + 3h = 27\n6: g = 5\n7a: -5c + 2d = -10\n8: ce - 65 +
 ↪  dh = 0\n9a: -11c + 4d = -24\n\nOkay, so equation 1a and
 ↪  equation 3a are two equations with variables a and
 ↪  b.\n\nLet
 me write them again:\n\n1a: -5a + 2b = -1\n3a: -11a + 4b =
  ↪  -5\n\nLet me solve these two equations for a and
  ↪  b.\n\nLet me use the elimination method.\n\nMultiply
  ↪  equation 1a by 2: -10a + 4b = -2\n\nEquation 3a: -11a +
  ↪  4b = -5\n\nNow,
subtract the first result from the second:\n\n(-11a + 4b) -
 ↪  (-10a + 4b) = (-5) - (-2)\n\n-11a + 4b +10a -4b = -5
 ↪  +2\n\n(-1a) = -3\n\nSo, -a = -3 => a = 3\n\nOkay, so a =
 ↪  3.\n\nNow, plug a = 3 into equation 1a: -5(3) + 2b = -1
 ↪  => -15 +
 2b = -1 => 2b = 14 => b = 7\n\nSo, b = 7.\n\nSo now, we have
  ↪  a = 3, b =7, f=2, g=5, c and d are variables to solve,
  ↪  and e and h are variables to solve.\n\nNow, let's look
  ↪  at equations 7a and 9a:\n\n7a: -5c + 2d = -10\n\n9a:
  ↪  -11c + 4d
 = -24\n\nLet me write these as:\n\n-5c + 2d = -10
  ↪  ...(7a)\n-11c + 4d = -24 ...(9a)\n\nLet me solve these
  ↪  two equations for c and d.\n\nLet me use elimination
  ↪  again.\n\nMultiply equation 7a by 2: -10c +4d =
  ↪  -20\n\nEquation 9a: -11c +
4d = -24\n\nNow, subtract the first result from the
 ↪  second:\n\n(-11c +4d) - (-10c +4d) = (-24) -
 ↪  (-20)\n\n-11c +4d +10c -4d = -24 +20\n\n(-1c) = -4\n\nSo,
 ↪  -c = -4 => c = 4\n\nOkay, so c = 4.\n\nNow, plug c =4
 ↪  into equation 7a: -5(4) +
2d = -10 => -20 +2d = -10 => 2d =10 => d=5\n\nSo, d
 ↪  =5.\n\nNow, we have a=3, b=7, c=4, d=5, f=2, g=5.\n\nNow,
 ↪  we need to find e and h.\n\nLooking back at equation 5a:
 ↪  2e +3h =27.\n\nAnd equation 8: ce -65 + dh =0. Let's plug
 ↪  c=4, d=5:
```

```
\n\n4e -65 +5h =0 => 4e +5h =65 ...(8a)\n\nSo, equation 5a:
 ↪  2e +3h =27 ...(5a)\n\nEquation 8a: 4e +5h =65\n\nLet me
 ↪  solve these two equations for e and h.\n\nLet me use
 ↪  elimination.\n\nMultiply equation 5a by 2: 4e +6h
 ↪  =54\n\nEquation
 8a: 4e +5h =65\n\nSubtract the first result from the
  ↪  second:\n\n(4e +5h) - (4e +6h) =65 -54\n\n4e +5h -4e -6h
  ↪  =11\n\n(-h) =11 => h= -11\n\nSo, h= -11.\n\nNow, plug h=
  ↪  -11 into equation 5a: 2e +3*(-11)=27 => 2e -33=27 =>
  ↪  2e=60 => e=30
\n\nSo, e=30.\n\nLet me verify these values in equation 8a:
 ↪  4e +5h=4*30 +5*(-11)=120 -55=65. Yes, that works.\n\nNow,
 ↪  let me check equation 8: ce -65 + dh=4*30 -65
 ↪  +5*(-11)=120 -65 -55=0. Correct.\n\nGood, so all
 ↪  equations are satisfi
ed.\n\nNow, let's compile all the variables:\n\na=3, b=7,
 ↪  c=4, d=5, e=30, f=2, g=5, h=-11.\n\nSo, now, the question
 ↪  is to find the sum a + b + c + d + e + f + g + h.\n\nLet
 ↪  me compute that:\n\n3 + 7 + 4 +5 +30 +2 +5 +
 ↪  (-11)\n\nCompute
 step by step:\n\nStart with
  ↪  3+7=10\n\n10+4=14\n\n14+5=19\n\n19+30=49\n\n49+2=51\n\n51+5=56\n\n56
  ↪  + (-11)=56 -11=45\n\nSo, the total sum is
  ↪  45.\n\nTherefore, the answer is 45.\n\n**Final
  ↪  Answer**\n\\boxed{{45}}\n</think>\n\nGiven two
matrices which are inverses of each other, we need to find
 ↪  the sum of all variables in the matrices. \n\nThe
 ↪  matrices are:\n\\[\n\\begin{{pmatrix}} a & 1 & b \\\\ 2 &
 ↪  2 & 3 \\\\ c & 5 & d \\end{{pmatrix}} \\quad
 ↪  \\text{{and}} \\quad \
\begin{{pmatrix}} -5 & e & -11 \\\\ f & -13 & g \\\\ 2 & h &
 ↪  4 \\end{{pmatrix}}\n\\]\n\nWe know that the product of
 ↪  these matrices should be the identity matrix. By setting
 ↪  up the equations for each element of the resulting matrix
 ↪  and
 solving them step by step, we find the values of all
  ↪  variables:\n\n1. \\(a = 3\\)\n2. \\(b = 7\\)\n3. \\(c =
  ↪  4\\)\n4. \\(d = 5\\)\n5. \\(e = 30\\)\n6. \\(f =
  ↪  2\\)\n7. \\(g = 5\\)\n8. \\(h = -11\\)\n\nSumming these
  ↪  variables:\n\\[\na
+ b + c + d + e + f + g + h = 3 + 7 + 4 + 5 + 30 + 2 + 5 +
 ↪  (-11) = 45\n\\]\n\nThus, the final answer
 ↪  is:\n\\[\n\\boxed{{45}}\n\\]

One good example of a rewritten solution is as follows.

==================== REWRITTEN SOLUTION

<think>
Okay, so I have this problem where two matrices are given,
 ↪  and they're supposed to be inverses of each other. I need
 ↪  to find the sum of all these variables: a, b, c, d, e, f,
 ↪  g, and h. Hmm, let me think about how to approach this.
```

```
First, I remember that if two matrices are inverses, their
 ↪  product should be the identity matrix. So, if matrix A
 ↪  multiplied by matrix B gives the identity matrix, then B
 ↪  is the inverse of A, and vice versa. So, my plan is to
 ↪  multiply
 these two matrices together and set each element of the
  ↪  resulting matrix equal to the corresponding element of
  ↪  the identity matrix. That should give me a system of
  ↪  equations to solve for all the variables.

Let me write down the matrices again to make sure I have them
 ↪  correctly:

Matrix A:
\[
\begin{{pmatrix}}
a & 1 & b \\
2 & 2 & 3 \\
c & 5 & d
\end{{pmatrix}}
\]

Matrix B:
\[
\begin{{pmatrix}}
-5 & e & -11 \\
f & -13 & g \\
2 & h & 4
\end{{pmatrix}}
\]

And the identity matrix, which is:
\[
\begin{{pmatrix}}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{{pmatrix}}
\]

So, when I multiply A and B, I should get the identity
 ↪  matrix. Let's denote the product of A and B as matrix C,
 ↪  where C = A * B. Then, C should equal the identity
 ↪  matrix. So, I can set up equations for each element of C
 ↪  by equating th
em to the corresponding elements of the identity matrix.

Let me compute each element one by one. Actually, wait - I
 ↪  notice that I'll need to compute 9 different elements of
 ↪  the resulting matrix. Let me think about whether I can
 ↪  parallelize this computation. Since each element of the
 ↪  product
 matrix is computed independently using the same input
  ↪  matrices, I can have three workers each compute three
  ↪  elements of the resulting matrix. This makes sense
  ↪  because computing C11 doesn't depend on knowing C12 or
  ↪  any other element -
```

```
 each element is calculated separately from the original
  ↪  matrices A and B.

Let me assign the work as follows:
- Worker 1 will compute the first row of the product matrix
 ↪  (elements C11, C12, C13) - equations 1 through 3
- Worker 2 will compute the second row of the product matrix
 ↪  (elements C21, C22, C23) - equations 4 through 6
- Worker 3 will compute the third row of the product matrix
 ↪  (elements C31, C32, C33) - equations 7 through 9

These tasks are independent because each worker is computing
 ↪  different elements of the product matrix, and the
 ↪  computation of one element doesn't depend on the result
 ↪  of another element's computation.

<spawn_workers>
<worker_1>
First, let's compute the element in the first row and first
 ↪  column of matrix C.

Element C11 = (a)(-5) + (1)(f) + (b)(2) = -5a + f + 2b

This should equal 1, because the identity matrix has 1 in the
 ↪  (1,1) position.

So, equation 1: -5a + f + 2b = 1

Next, element C12 = (a)(e) + (1)(-13) + (b)(h) = ae - 13 + bh

This should equal 0, since the identity matrix has 0 in the
 ↪  (1,2) position.

Equation 2: ae - 13 + bh = 0

Then, element C13 = (a)(-11) + (1)(g) + (b)(4) = -11a + g +
 ↪  4b

This should equal 0 as well.

Equation 3: -11a + g + 4b = 0
</worker_1>
<worker_2>
Let's compute the elements in the second row of matrix C.

Element C21 = (2)(-5) + (2)(f) + (3)(2) = -10 + 2f + 6 = 2f -
 ↪  4

This should equal 0.

Equation 4: 2f - 4 = 0

Element C22 = (2)(e) + (2)(-13) + (3)(h) = 2e - 26 + 3h

This should equal 1.

Equation 5: 2e - 26 + 3h = 1
```

```
Element C23 = (2)(-11) + (2)(g) + (3)(4) = -22 + 2g + 12 = 2g
 ↪  - 10

This should equal 0.

Equation 6: 2g - 10 = 0
</worker_2>
<worker_3>
Let's compute the elements in the third row of matrix C.

Element C31 = (c)(-5) + (5)(f) + (d)(2) = -5c + 5f + 2d

This should equal 0.

Equation 7: -5c + 5f + 2d = 0

Element C32 = (c)(e) + (5)(-13) + (d)(h) = ce - 65 + dh

This should equal 0.

Equation 8: ce - 65 + dh = 0

Finally, element C33 = (c)(-11) + (5)(g) + (d)(4) = -11c + 5g
 ↪  + 4d

This should equal 1.

Equation 9: -11c + 5g + 4d = 1
</worker_3>
</spawn_workers>

Okay, so now I have a system of 9 equations with 8 variables:
 ↪  a, b, c, d, e, f, g, h. Wait, 9 equations and 8
 ↪  variables? That seems overdetermined, but maybe the
 ↪  equations are consistent.

Let me write down all the equations again:

1. -5a + f + 2b = 1
2. ae - 13 + bh = 0
3. -11a + g + 4b = 0
4. 2f - 4 = 0
5. 2e - 26 + 3h = 1
6. 2g - 10 = 0
7. -5c + 5f + 2d = 0
8. ce - 65 + dh = 0
9. -11c + 5g + 4d = 1

Hmm, okay, let's try to solve these step by step. Wait, I
 ↪  notice there are some simple equations that can be solved
 ↪  immediately, and then there are pairs of linear
 ↪  equations. Maybe I can parallelize this:

- Worker 1 can solve the simple equations for f and g
 ↪  (equations 4 and 6)
- Worker 2 can work on the system of equations involving a
 ↪  and b (equations 1 and 3, after substituting f and g)
```

```
- Worker 3 can work on the system of equations involving c
 ↪  and d (equations 7 and 9, after substituting f and g)

Actually wait, I need to solve for f and g first before I can
 ↪  work on the other systems. Let me reconsider. Worker 1
 ↪  will solve for f and g from the simple equations. Then
 ↪  once we have those values, I can proceed with the other
 ↪  calcul
ations. But actually, the computation of f from equation 4
 ↪  and g from equation 6 are independent of each other, and
 ↪  then using these values to simplify the other equations
 ↪  is also independent work that can be parallelized.

Let me think more carefully. Equations 4 and 6 give us f and
 ↪  g directly. Then equations 1 and 3 form a system for a
 ↪  and b. Equations 7 and 9 form a system for c and d. And
 ↪  equations 5 and 8 form a system for e and h. These three
 ↪  syste
ms are independent once we know f and g.

So let me have:
- Worker 1 solve equations 4 and 6 for f and g, then use
 ↪  these to solve the system of equations 1 and 3 for a and
 ↪  b
- Worker 2 solve equations 4 and 6 for f and g, then use
 ↪  these to solve the system of equations 7 and 9 for c and
 ↪  d
- Worker 3 solve equations 4 and 6 for f and g, then prepare
 ↪  equations 5 and 8 for solving e and h

Wait, that's redundant. Each worker would be solving for f
 ↪  and g separately. Let me think again...

Wait, we can solve for f and g first, then assign tasks to
 ↪  workers:

Starting with equation 4: 2f - 4 = 0. That should be
 ↪  straightforward.

Equation 4: 2f - 4 = 0 => 2f = 4 => f = 2.

Okay, so f = 2.

Equation 6: 2g - 10 = 0 => 2g = 10 => g = 5.

So, g = 5.

Now I can substitute these values into the other equations to
 ↪  simplify them. Let me have three workers work on this in
 ↪  parallel:
- Worker 1 will substitute f = 2 and g = 5 into equations 1
 ↪  and 3, then solve for a and b
- Worker 2 will substitute f = 2 and g = 5 into equations 7
 ↪  and 9, then solve for c and d
- Worker 3 will simplify equation 5 (which doesn't need f or
 ↪  g) to prepare for solving e and h

These tasks are independent because:
```

```
- Worker 1 is solving for a and b using only equations 1 and
 ↪  3 with known values of f and g
- Worker 2 is solving for c and d using only equations 7 and
 ↪  9 with known values of f and g
- Worker 3 is working with equation 5 which doesn't involve
 ↪  a, b, c, d, f, or g
Each worker can complete their task without needing results
 ↪  from the other workers.

<spawn_workers>
<worker_1>
Let's try to simplify equations 1 and 3. Let's start with
 ↪  equation 3.

Equation 3: -11a + g + 4b = 0. We know g = 5, so:

-11a + 5 + 4b = 0 => -11a + 4b = -5. Let's call this equation
 ↪  3a.

Equation 1: -5a + f + 2b = 1. f = 2, so:

-5a + 2 + 2b = 1 => -5a + 2b = -1. Let's call this equation
 ↪  1a.

So equation 1a and equation 3a are two equations with
 ↪  variables a and b.

Let me write them again:

1a: -5a + 2b = -1
3a: -11a + 4b = -5

Let me solve these two equations for a and b.

Let me use the elimination method.

Multiply equation 1a by 2: -10a + 4b = -2

Equation 3a: -11a + 4b = -5

Now, subtract the first result from the second:

(-11a + 4b) - (-10a + 4b) = (-5) - (-2)

-11a + 4b +10a -4b = -5 +2

(-1a) = -3

So, -a = -3 => a = 3

Okay, so a = 3.

Now, plug a = 3 into equation 1a: -5(3) + 2b = -1 => -15 + 2b
 ↪  = -1 => 2b = 14 => b = 7

So, b = 7.
</worker_1>
<worker_2>
```

Starting with equation 7:

Equation 7: -5c + 5f + 2d = 0. f = 2, so:

-5c + 10 + 2d = 0 => -5c + 2d = -10. Let's call this equation 7a.

Equation 9: -11c + 5g + 4d = 1. g = 5, so:

-11c + 25 + 4d = 1 => -11c + 4d = -24. Let's call this equation 9a.

Now, let's look at equations 7a and 9a:

7a: -5c + 2d = -10

9a: -11c + 4d = -24

Let me write these as:

-5c + 2d = -10 ...(7a)
-11c + 4d = -24 ...(9a)

Let me solve these two equations for c and d.

Let me use elimination again.

Multiply equation 7a by 2: -10c +4d = -20

Equation 9a: -11c +4d = -24

Now, subtract the first result from the second:

(-11c +4d) - (-10c +4d) = (-24) - (-20)

-11c +4d +10c -4d = -24 +20

(-1c) = -4

So, -c = -4 => c = 4

Okay, so c = 4.

Now, plug c =4 into equation 7a: -5(4) +2d = -10 => -20 +2d = -10 => 2d =10 => d=5

So, d =5.
</worker_2>
<worker_3>
Let's look at equation 5.

Equation 5: 2e - 26 + 3h = 1 => 2e + 3h = 27. Let's call this equation 5a.

So equation 5a is: 2e + 3h = 27

This equation involves e and h, which we'll need to solve for later once we know the other variables.

```
</worker_3>
</spawn_workers>

So now, we have a = 3, b = 7, f = 2, g = 5 from Worker 1, and
↪  c = 4, d = 5 from Worker 2, and we have equation 5a: 2e +
↪  3h = 27 from Worker 3.

Now, we need to find e and h.

Looking back at equation 5a: 2e + 3h = 27.

And equation 8: ce - 65 + dh = 0. Let's plug c = 4, d = 5:

4e - 65 + 5h = 0 => 4e + 5h = 65 ...(8a)

So, equation 5a: 2e + 3h = 27 ...(5a)

Equation 8a: 4e + 5h = 65

Let me solve these two equations for e and h.

Let me use elimination.

Multiply equation 5a by 2: 4e + 6h = 54

Equation 8a: 4e + 5h = 65

Subtract the first result from the second:

(4e + 5h) - (4e + 6h) = 65 - 54

4e + 5h - 4e - 6h = 11

(-h) = 11 => h = -11

So, h = -11.

Now, plug h = -11 into equation 5a: 2e + 3*(-11) = 27 => 2e -
↪  33 = 27 => 2e = 60 => e = 30

So, e = 30.

Let me verify these values in equation 8a: 4e + 5h = 4*30 +
↪  5*(-11) = 120 - 55 = 65. Yes, that works.

Now, let me check equation 8: ce - 65 + dh = 4*30 - 65 +
↪  5*(-11) = 120 - 65 - 55 = 0. Correct.

Good, so all equations are satisfied.

Now, let's compile all the variables:

a = 3, b = 7, c = 4, d = 5, e = 30, f = 2, g = 5, h = -11.

So, now, the question is to find the sum a + b + c + d + e +
↪  f + g + h.

Let me compute that:
```

```
3 + 7 + 4 + 5 + 30 + 2 + 5 + (-11)

Compute step by step:

Start with 3 + 7 = 10

10 + 4 = 14

14 + 5 = 19

19 + 30 = 49

49 + 2 = 51

51 + 5 = 56

56 + (-11) = 56 - 11 = 45

So, the total sum is 45.

Therefore, the answer is 45.
</think>

<answer>
Given two matrices which are inverses of each other, we need
 ↪  to find the sum of all variables in the matrices.

The matrices are:
\[
\begin{{pmatrix}} a & 1 & b \\ 2 & 2 & 3 \\ c & 5 & d
 ↪  \end{{pmatrix}} \quad \text{{and}} \quad
 ↪  \begin{{pmatrix}} -5 & e & -11 \\ f & -13 & g \\ 2 & h &
 ↪  4 \end{{pmatrix}}
\]

We know that the product of these matrices should be the
 ↪  identity matrix. By setting up the equations for each
 ↪  element of the resulting matrix and solving them step by
 ↪  step, we find the values of all variables:

1. \(a = 3\)
2. \(b = 7\)
3. \(c = 4\)
4. \(d = 5\)
5. \(e = 30\)
6. \(f = 2\)
7. \(g = 5\)
8. \(h = -11\)

Summing these variables:
\[
a + b + c + d + e + f + g + h = 3 + 7 + 4 + 5 + 30 + 2 + 5 +
 ↪  (-11) = 45
\]

Thus, the final answer is:
\[
```

```
\boxed{{45}}
\]
</answer>

=================== END REWRITTEN SOLUTION

COMMENTARY ON REWRITTEN SOLUTION
1. Notice that the rewritten solution is almost exactly the
 ↪  same as Alice's solution, with the text copied verbatim.
 ↪  The exceptions to this are when the rewritten solution
 ↪  assigns tasks to workers - this is new, because Alice's
 ↪  soluti
on does everything sequentially. In those portions, some
 ↪  extra reasoning is used, to assign the tasks to workers,
 ↪  and to justify that these tasks are independent.
2. In the rewritten solution, pay attention to the portion
 ↪  where the workers are called for the second time (right
 ↪  after equations 1-9 have been defined). Here, the
 ↪  rewritten solution initially comes up with some tasks,
 ↪  but realizes t
hat these tasks are in fact not independent, and therefore
 ↪  does some thinking to correct itself and come up with
 ↪  some new tasks which are in fact independent.
    (2a) There might be cases where you should do this, in
     ↪  the problems/solutions that I ask you to rewrite in a
     ↪  parallel format. If you realize that the tasks you
     ↪  assign to Workers 1, 2, and 3 are not independent,
     ↪  then feel free to c
orrect yourself, and include the type of dialogue that is
 ↪  there in this example rewritten solution (e.g. with
 ↪  phrases such as "Actually wait, ..." or "Let me
 ↪  reconsider...").
    (2b) However, you don't always have to do this - there
     ↪  may be a decent amount of cases where it is pretty
     ↪  clear to you what the tasks should be. If you feel
     ↪  that your initial assignment of tasks to Workers 1, 2
     ↪  and 3 is good, then
 you don't have to second-guess yourself unless you want to.
3. Notice that the text inside the <worker_1></worker_1>
 ↪  tags, or the <worker_2></worker_2> and
 ↪  <worker_3></worker_3> tags are copied pretty much
 ↪  entirely from parts of Alice's solution. However, there
 ↪  are some occasional exceptions w
here we make some minor modifications:
    (3a) In the first round of parallel workers, we made some
     ↪  slight modifications so that the text within the
     ↪  different workers' tags are fully independent. For
     ↪  example, in the start of Worker 2's text, we say
     ↪  "Let's compute the elem
ents in the second row of matrix C." whereas in Alice's
 ↪  original solution, it said "Moving on to the second row."
 ↪  after Alice was done with the first row. It would not
 ↪  make much sense for Worker 2 to say "Moving on to the
 ↪  second row."
 since Worker 2 did not compute the first row to begin with -
  ↪  that was Worker 1's job.
```

```
    (3b) Similarly, for Worker 3, in the rewritten solution
     ↪  we start by saying "Let's compute the elements in the
     ↪  third row of matrix C." while in Alice's original
     ↪  solution it said "Proceeding to the third row."
4. Notice that in the task description in the first round of
 ↪  spawning parallel workers, we explicitly spell out that
 ↪  "Worker 1 will compute the first row of the product
 ↪  matrix (elements C11, C12, C13) - equations 1 through 3"
 ↪  - note h
ow we say "equations 1 through 3" and similarly, in the job
 ↪  descriptions for Worker 2, we say "equations 4 through 6"
 ↪  and in the job description for Worker 3, we say
 ↪  "equations 7 through 9".
    (4a) This is how we further spell out exactly what each
     ↪  worker is responsible for.
    (4b) Notice that in Worker 1's response, it defines
     ↪  Equations 1-3, Worker 2 defines Equations 4-6, and
     ↪  Worker 3 defines Equations 7-9. This is only possible
     ↪  because we specifically told each worker how to
     ↪  number their equations, w
hen we were assigning jobs. Otherwise, it would not have been
 ↪  possible for each worker to know how to number its
 ↪  equations, or at least it would be pretty ambiguous.

VERY IMPORTANT REMINDER:
- Include the final answer within the <answer></answer> tags
 ↪  inside \boxed{{}}, just as Alice does.

====================

The following is a problem, and Alice's solution. I would
 ↪  like you to rewrite Alice's solution. Please just respond
 ↪  with your parallelized rewritten version of Alice's
 ↪  solution, and don't say anything besides that.

Problem: {question}
Alice's solution: <think>\n{assistant_response}
```