

Extended Abstract

Motivation Large language models can often produce plausible mathematical reasoning traces while making small arithmetic mistakes or committing to an incorrect intermediate plan. These failures are especially problematic in verifiable tasks such as Countdown, where a final answer must use a fixed set of numbers exactly once and evaluate to a target value. Our project asks whether reinforcement learning can improve this kind of reasoning and whether explicit calculator tool use provides an additional benefit beyond reinforcement learning alone.

Method We study no-tool and tool-integrated post-training pipelines for Countdown reasoning. Starting from Qwen2.5-0.5B-style supervised fine-tuning baselines, we construct a tool-format dataset that allows the model to issue calculator calls with `<tool>...</tool>` tags and receive executable results in `<obs>...</obs>` tags. We then train reinforcement-learning variants using automatically verifiable rewards from the Countdown evaluator. Our main RL method is RLOO, a REINFORCE-style leave-one-out estimator over groups of sampled responses. We also evaluate Tool-RLOO++, which normalizes advantages across the update batch, Tool-GRPO, and Tool-DAPO, which combines group-normalized advantages, asymmetric policy-ratio clipping, and dynamic sampling that removes all-correct or all-wrong prompt groups before policy updates.

Implementation All RL experiments use vLLM to sample multiple rollouts per prompt and a HuggingFace policy model for gradient updates. Tool-augmented rollouts execute calculator calls during generation, exclude tool observations from policy-loss tokens, and score the final `<answer>` expression with the same Countdown verifier used for no-tool evaluation. We evaluate both the original 50-problem public test split and a new 1,024-problem fresh held-out benchmark with no exact overlap with the training data under the `(target, sorted(numbers))` key.

Results On the fresh 1,024-problem benchmark, no-tool RLOO improves pass@1 from 26.4% for the SFT baseline to 50.6%. Tool integration improves both SFT and RL: Tool-SFT reaches 35.8% pass@1, Tool-RLOO reaches 56.6% pass@1 with a matched 100-step training budget, and extended Tool-RLOO reaches 60.7% pass@1 after 200 steps. Tool-DAPO further improves over Tool-RLOO, reaching stronger pass@1 and pass@16 performance in fewer training hours. Error analysis shows that calculation mistakes are common in the SFT model, while correctness-distribution analysis shows that RL mainly shifts probability mass toward already-discoverable correct trajectories.

Discussion The results support two complementary conclusions. First, reinforcement learning with verifiable rewards substantially improves low- k performance by increasing the probability of correct solution traces. Second, calculator tool use directly targets arithmetic and verification errors, giving additional gains when combined with RL. The remaining hard cases are examples whose sampled groups contain no correct trajectories, since these groups provide little positive signal for policy improvement.

Conclusion Tool-integrated RL improves Countdown mathematical reasoning more than either tool-format SFT or no-tool RL alone. It enables LLM to produce correct math reasoning via finding possible combinations, validating answers and correcting errors by using calculator. Future work should focus on combining tool execution with stronger planning support, especially for prompts where the model cannot initially sample any correct trajectory.

Learning to Use Tools: Reinforcement Learning for Tool-Integrated Mathematical Reasoning

Minghui Xu, Zi Wang

Department of Energy Science and Engineering
Stanford University
minghuix@stanford.edu; ziwang3@stanford.edu

Abstract

Current large language models (LLMs) increasingly benefit from external tool integration, especially for tasks requiring reliable computation and verification. Motivated by this, we study calculator tool calling for improving mathematical reasoning on the Countdown task. We first analyze reasoning failures and find that calculation errors account for a substantial portion of incorrect responses. We then construct supervised fine-tuning datasets to teach the model useful tool-use patterns and how to interpret returned outputs. Building on this tool-formatted policy, we apply several on-policy reinforcement learning methods, including RLOO, RLOO++, GRPO, and DAPO, using automatically verifiable final-answer rewards. To enable a more reliable evaluation, we construct a fresh 1,024-problem held-out Countdown benchmark with no exact overlap with the training data. Our results show that calculator tool integration consistently improves both SFT and RL baselines, yielding roughly 10 percentage-point gains across pass@k. Among the RL methods, Tool-DAPO achieves the strongest performance, improving pass@1 from 35.8% for Tool-SFT to 66.0%. Further analysis shows that RL encourages more effective tool use even when only final-answer rewards are provided. These findings suggest that tool integration reduces arithmetic and verification errors, while RL increases the probability of correct reasoning traces.

1 Introduction

Large language models (LLMs) have become increasingly capable in coding and mathematical reasoning, especially when prompted or trained to generate intermediate reasoning steps. However, even reasoning-oriented LLMs can still make arithmetic and logical mistakes, particularly in multi-step problems that require exact numerical computation. This limitation arises partly from tokenization and autoregressive next-token prediction, which do not provide an explicit mechanism for reliable calculation or verification. As a result, integrating external tools has become an important direction for improving LLM reliability on tasks that require precise computation (Schick et al., 2023; Yao et al., 2023).

In this work, we study tool-integrated mathematical reasoning on the Countdown task, where the model must combine a set of given numbers using arithmetic operations to reach a target value, as shown in Fig. 1. Countdown is a useful testbed because success requires both combinatorial reasoning and exact arithmetic, and the final answer can be automatically verified. During the LLM math reasoning process, if it gives the correct answer, then it will be firstly validated by calculator and then output the final answer. If the math reasoning reaches an arithmetic error, it will use the calculator to correct it and continue the math reasoning based on the correct answer, preventing model deviating from correct solutions. We investigate this problem using a sequence of no-tool and tool-integrated training pipelines. We first construct supervised fine-tuning (SFT) baselines and then apply several on-policy reinforcement learning (RL) methods, including RLOO (Ahmadian

et al., 2024a), RLOO++ (Hu, 2025), GRPO (Shao et al., 2024), and DAPO (Yu et al., 2025), using automatically computed rewards from the Countdown verifier (Ahmadian et al., 2024a; Shao et al., 2024; Yu et al., 2025). For tool-augmented models, rollouts may include calculator calls enclosed by `<tool>...</tool>` tags; the environment executes the expression and inserts the result as an `<obs>` observation before generation continues. We evaluate all methods on both the original 50-problem public test split and a larger fresh 1,024-problem held-out benchmark designed to remove exact training overlap.

Our main contributions are:

- We build a tool-integrated Countdown training and evaluation pipeline with live calculator execution during model rollouts.
- We show that tool integration consistently improves both SFT and RL models, and that Tool-DAPO achieves the strongest performance among the tested methods.
- We analyze error categories, tool-call dynamics, and correctness distributions to explain when RL and tool use improve mathematical reasoning and where they still fail.

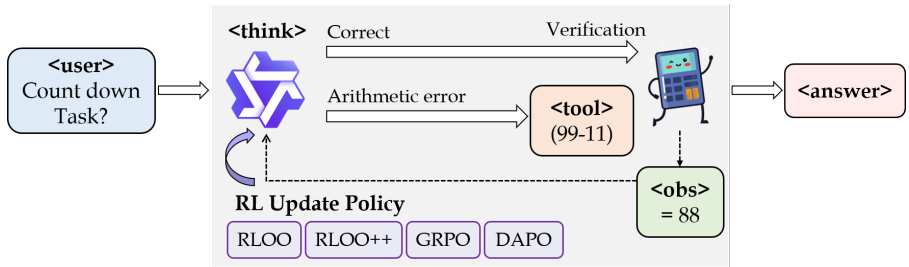


Figure 1: Schematic of RL for tool-integrated LLM on math reasoning.

2 Related Work

Beyond supervised fine-tuning, preference optimization and online RL provide sequence-level training signals. Recent work has shown that reinforcement learning can substantially improve the reasoning capabilities of large language models. Models such as OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025) demonstrate that large-scale RL can encourage long-chain reasoning.

Identity Preference Optimization (IPO) (Azar et al., 2023) offers an alternative to DPO-style preference learning, while recent REINFORCE-style methods such as RLOO (Ahmadian et al., 2024b) simplify online policy optimization by avoiding a learned critic. These methods are particularly relevant for reasoning tasks where final answers can be automatically verified.

Tool-augmented reasoning has also become an important direction for improving LLM reliability. Early approaches such as ReAct (Yao et al., 2023) interleave natural-language reasoning with external actions. More recent work, such as ReTool (Feng et al., 2025) and In-Context Reinforcement Learning for Tool Use (Ye et al., 2026), begins to train tool invocation through reinforcement learning.

Recent studies suggest that the gains from reinforcement learning with verifiable rewards (RLVR) may arise more from probability reallocation than from a genuine expansion of reasoning coverage. Yue et al. (2025) show that RLVR-trained models often improve pass@1 but fail to improve, or even underperform, base models at large pass@k, suggesting that RL mainly increases the sampling probability of correct trajectories already present in the base model distribution. Similarly, Liu et al. (2025) argue that R1-Zero-like training strongly depends on the base model’s existing exploration ability and pretraining-induced reasoning behavior. Together, these findings indicate that RLVR can sharpen the model’s existing solution distribution, but may not necessarily create substantially new reasoning capabilities.

3 Method

3.1 Construction of Tool-Integrated SFT Dataset

The base Countdown task asks the model to produce an expression inside `<answer>...</answer>` tags. A response receives full reward only if the expression is syntactically valid, uses exactly the provided numbers, and evaluates to the target; malformed or incorrect expressions receive lower reward. This verifier makes the task well suited to reinforcement learning because final-answer rewards can be computed without human labels.

For the tool setting, we convert the prompt and completion format so the model can call a calculator before producing the final answer. A tool call is written as `<tool>expression</tool>`, and the environment returns an observation such as `<obs>= 44</obs>`. During SFT, this format teaches the model where to place executable arithmetic checks and how to condition on the returned observation. Based on the original SFT data, the calculation errors are firstly detected. Then the correct answer together with prior reasoning contexts are given to SFT reference model and the afterwards reasoning process is generated by SFT reference model. Then the prior reasoning and afterwards reasoning contexts are concatenated together. In between these two reasoning context, the tool-integration is injected with the above format, formulating tool-integrated data for Tool-SFT model. The whole process is given in Algorithm 1. During RL, tool observations are treated as environment outputs rather than model-generated tokens, so the policy loss is applied only to the model-controlled parts of the trajectory.

Algorithm 1 Tool-Integrated Countdown SFT Data Construction

Require: Original SFT data \mathcal{D} ; trained SFT model π ; samples per row K

Ensure: Tool-augmented dataset \mathcal{D}'

```
Phase A: build correction prefixes
1: for each row  $(q, c)$  in  $\mathcal{D}$  do
2:   if the reasoning contains a wrong arithmetic step then
3:     prefix  $\leftarrow$  reasoning up to that step, with the value corrected
4:   else if the final answer is wrong then
5:     prefix  $\leftarrow$  full reasoning + a calculator check that flags it
6:   end if
7: end for
Phase B: model continuations (GPU)
8: for each prefix do
9:   sample  $K$  continuations from  $\pi$  that finish the reasoning and answer
10: end for
Phase C: assemble completions
11: for each row  $(q, c)$  in  $\mathcal{D}$  do
12:   add the tool instruction to the query  $q$ 
13:   if  $q$  is an error row then
14:     pick the shortest continuation whose answer is correct
15:     if none is correct then use the exact solver's answer
16:     new completion  $\leftarrow$  prefix + calculator call + chosen continuation
17:   else if the original answer is already correct then
18:     append a confirming calculator call to the reasoning
19:   else
20:     keep the original completion unchanged
21:   end if
22: end for
Phase D: split
23: train  $\leftarrow$  all rows; test  $\leftarrow$  small held-out monitoring set
24: return  $\mathcal{D}'$ 
```

This setup lets us separate two questions. Tool-SFT tests whether demonstrations of calculator use alone improve the baseline. Tool-RL methods test whether online RL can further increase the frequency and usefulness of tool calls when the only optimization signal is final-answer correctness.

3.2 RLOO

RLOO (Ahmadian et al., 2024b) is a REINFORCE-based policy-gradient estimator that reduces variance using a leave-one-out baseline. For each prompt x , we sample k responses and compute the advantage for response y_i as its reward relative to the average reward of the other $k - 1$ responses. In our implementation, trajectories are sampled using vLLM for efficiency, while policy-gradient updates are computed with a trainable HuggingFace policy initialized from the SFT checkpoint.

The overall training loss is

$$\mathcal{L} = -\mathbb{E} [w(y_i, x) A_i \log \pi_\theta(y_i | x)] - \beta_{\text{ent}} \mathcal{H}(\pi_\theta) + \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}),$$

where A_i is the leave-one-out RLOO advantage, $w(y_i, x)$ is the clipped sequence-level importance weight, π_θ is the trainable policy, and π_{ref} is the frozen SFT reference policy. We report the RLOO loss as the policy-gradient component and separately track the entropy and KL terms during training. We use the default values $\beta_{\text{ent}} = 10^{-3}$ and $\beta_{\text{KL}} = 10^{-3}$.

3.3 RLOO++

RLOO++ (Hu, 2025) keeps the same leave-one-out reward baseline but changes how advantages are scaled before the policy update. Instead of using raw leave-one-out advantages directly, we apply global advantage normalization, subtracting the batch mean and dividing by the batch standard deviation,

$$A_{q,o_t}^{\text{norm}} = \frac{A_{q,o_t} - \text{mean}(A | A \in \mathcal{D}_{\text{batch}})}{\text{std}(A | A \in \mathcal{D}_{\text{batch}}) + \epsilon},$$

and rescale them to a target standard deviation of 0.3. This modification is intended to stabilize updates when rewards are sparse and many sampled groups contain either no correct responses or mostly correct responses, since normalizing across the entire batch $\mathcal{D}_{\text{batch}}$ rather than the small leave-one-out group keeps the denominator stable even when within-group rewards are nearly uniform. We evaluate RLOO++ only in the tool-integrated setting.

3.4 GRPO

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is a policy-gradient method that forgoes the value model used in PPO, instead estimating the baseline directly from group scores to reduce memory and computational cost. For each prompt x , we sample a group of G responses and compute the advantage for response y_i by normalizing its reward relative to the group: subtracting the group mean and dividing by the group standard deviation. Since the sampled trajectories are associated with behavior-policy log probabilities, we apply token-level importance weighting, where the ratio between the current policy and behavior policy is computed in log space and clipped for stability. The overall training loss is

$$\mathcal{L} = -\mathbb{E} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min(\rho_{i,t} \hat{A}_{i,t}, \text{clip}(\rho_{i,t}, 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t}) \right] + \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}),$$

where

$$\rho_{i,t} = \frac{\pi_\theta(y_{i,t} | x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} | x, y_{i,<t})}$$

is the clipped token-level importance weight, $\hat{A}_{i,t}$ is the group-normalized advantage assigned to all tokens of response y_i , π_θ is the trainable policy, and π_{ref} is the frozen SFT reference policy. Unlike PPO, GRPO adds the KL penalty directly to the loss rather than to the token-level reward. We report the clipped surrogate as the policy-gradient component and separately track the KL term during training. We use the values $\epsilon = 10^{-3}$ and $\beta_{\text{KL}} = 0.01$ for our tool-integration training.

3.5 DAPO

Tool-DAPO (Yu et al., 2026) modifies the policy update in two main ways. First, it applies group-normalized advantages, so rewards are normalized within each sampled prompt group before updating

the model, i.e., for the i -th response the advantage is computed as

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)}.$$

Second, it uses dynamic sampling: prompt groups whose sampled responses are all correct or all wrong are filtered out before the update because they provide little within-group preference signal, which corresponds to enforcing the constraint $0 < |\{o_i \mid \text{is_equivalent}(a, o_i)\}| < G$. In our run, 60.7% of generated prompt groups were filtered, increasing the density of informative groups in the training batch. Tool-DAPO also uses asymmetric policy-ratio clipping, with lower and upper bounds of $1 - 0.2$ and $1 + 0.28$, respectively. Concretely, the policy is optimized via the token-level objective

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}_{(q,a) \sim \mathcal{D}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \varepsilon_{\text{low}}, 1 + \varepsilon_{\text{high}}) \hat{A}_{i,t} \right) \right],$$

where $r_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}$ and $(\varepsilon_{\text{low}}, \varepsilon_{\text{high}}) = (0.2, 0.28)$. Compared with Tool-RLOO, this gives a more constrained update rule while still allowing the policy to increase the probability of high-reward tool-using trajectories.

4 Experimental Setup

We evaluate on the Countdown arithmetic reasoning task. Each example contains a target and either three or four numbers. A model response is correct only if the final expression inside `<answer>...</answer>` uses each provided number exactly once and evaluates to the target under the allowed arithmetic operations. We report $\text{pass}@k$, computed from k sampled responses per problem.

Our main comparisons are no-tool SFT, Tool-SFT, no-tool RLOO, Tool-RLOO, Tool-RLOO++, Tool-GRPO, and Tool-DAPO. No-tool models generate a reasoning trace and final answer directly. Tool models are allowed to issue calculator calls during generation and receive observations before producing the final answer. Unless otherwise stated, RLOO and Tool-RLOO use the same RL hyperparameters, batch size, group size, sampling temperature, entropy coefficient, and KL coefficient; the main difference is whether live tool execution is enabled during rollouts.

The original public test split contains only 50 examples, which makes $\text{pass}@k$ estimates noisy. To quantify the effect of tool integration on mathematical reasoning more reliably, we construct an additional 1,024-problem fresh held-out Countdown test set. The held-out problems have no exact overlap with the training data under the (target, sorted(numbers)) key, allowing us to evaluate performance on new, non-overlapping problems from the same task distribution with substantially lower statistical uncertainty than the original 50-problem public test split. All reported confidence intervals are 95% bootstrap intervals over problem instances.

5 Results

5.1 Evaluation of the SFT Model without Tools

To diagnose the failure modes of the SFT model, we decompose incorrect responses into calculation errors, target/planning errors, and other final-answer validity errors. Calculation errors are evaluated over the full reasoning trace, while target/planning, number-use, and format errors are identified from the final answer. This distinction allows us to separate local arithmetic mistakes from broader failures in reasoning and solution construction.

As shown in Figure 2, the results reveal two complementary failure modes. Calculation errors occur frequently across both difficulty levels, motivating the use of an external arithmetic tool or verifier. In contrast, target/planning errors become substantially more common in 4-number problems, suggesting that harder instances require not only more reliable calculation but also stronger target-aware planning.

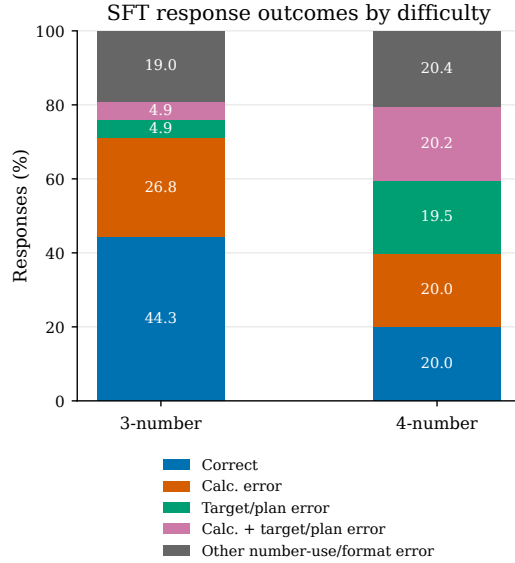


Figure 2: SFT response outcomes by difficulty. Bars show the percentage of responses for 3-number and 4-number Countdown problems, grouped by correct answers, calculation errors, target/planning errors, their overlap, and other number-use/format errors.

5.2 Tool Integration

As shown in Figure 3, we compare no-tool SFT, tool-augmented SFT, no-tool RLOO, and tool-augmented RLOO variants. Error bars denote 95% bootstrap confidence intervals computed over problem instances.

The small 50-problem test set in the right panel exhibits wide and largely overlapping confidence intervals, especially at larger k , making fine-grained comparisons difficult. In contrast, the larger held-out set in the left panel provides a clearer comparison. RLOO substantially improves over the SFT baseline at low k , increasing pass@1 from 26.4% to 50.6% without tool use and from 35.8% to 56.6% with tool use.

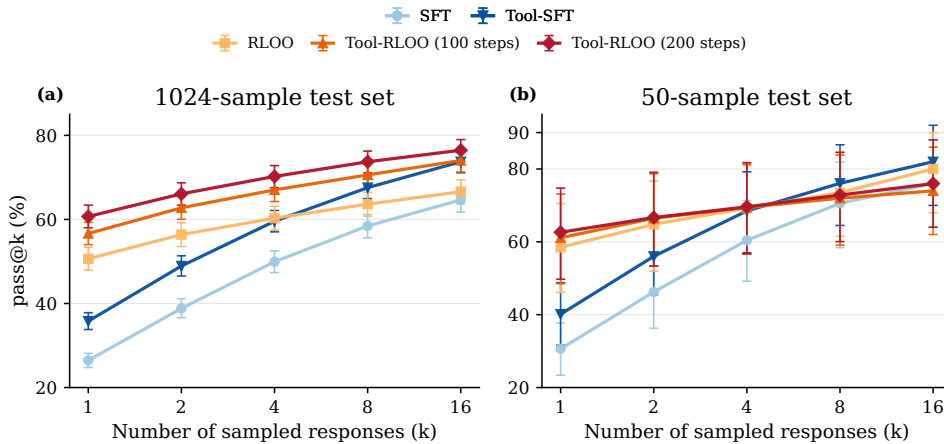


Figure 3: Pass@ k comparison on Countdown reasoning. The left panel reports results on our 1,024-problem fresh held-out benchmark, while the right panel reports results on the original 50-problem public test split.

The gains from RLOO become smaller at high k , likely because pass@ k approaches saturation once the model can sample at least one correct trajectory from multiple attempts. Thus, improvements in

trajectory quality or sampling probability may lead to only modest changes in high- k performance. We revisit this behavior in Section 6.1 through a correctness-distribution analysis, which suggests that reinforcement learning benefits most from prompt groups that already contain some correct sampled responses, since these groups provide a useful policy-update signal. In contrast, prompt groups with zero correct samples remain challenging, because they provide little direct signal for reinforcing a correct reasoning trace.

Tool integration improves both the SFT and RLOO settings. With a matched training budget, Tool-RLOO increases pass@1 from 50.6% to 56.6% and pass@16 from 66.6% to 74.0%. Extending Tool-RLOO training to 200 steps further improves performance to 60.7% pass@1 and 76.5% pass@16, corresponding to a roughly 10 percentage-point pass@1 gain over no-tool RLOO. Together, these results provide strong evidence that RLOO improves verifiable mathematical reasoning over SFT, and that tool-augmented RLOO adds a further benefit by reducing computation and verification errors. Representative RLOO and Tool-RLOO responses are documented in Section A for qualitative analysis.

5.3 Comparison of Different Reinforcement Learning Methods

We also compare different reinforcement learning methods under tool integration, as shown in Figure 4 and Table. 1. Tool-RLOO and Tool-RLOO++ both substantially improve pass@1 over the Tool-SFT baseline. However, Tool-RLOO++ does not consistently improve high- k performance and even reduces pass@16 relative to Tool-RLOO, which is consistent with the observation in Yue et al. (2025) that reinforcement learning may improve low- k performance without necessarily expanding the overall pass@ k coverage. For Tool-GRPO, on the 50-sample test set, it exhibits better performance than Tool-SFT model with only one shot as shown by higher pass@1 accuracy in Figure 4, but it performs weaker with multiple shots, which is likely caused by the variance of small test set. Further testing on 1024-sample test set, Tool-GRPO model consistently exceeds the baseline Tool-SFT model, as shown in Figure 4. Although the Tool-GRPO shows relatively weaker performance than other RL algorithm, it reaches the first highest pass@16 accuracy. This behavior should be interpreted with caution, since Tool-GRPO was trained for only 30 steps, followed by deviating from Tool-SFT reference model. Although all four methods sample a group of responses per prompt and therefore produce a vanishing advantage when the rewards within a group are uniform, the instability is confined to Tool-GRPO because of how it scales and filters these groups. Tool-GRPO alone both retains these zero-gradient groups and divides by an unstable per-group standard deviation, so the resulting noisy updates push the policy off the SFT initialization until reward and tool usage collapse—here after roughly 30 steps, versus 100 for the others.

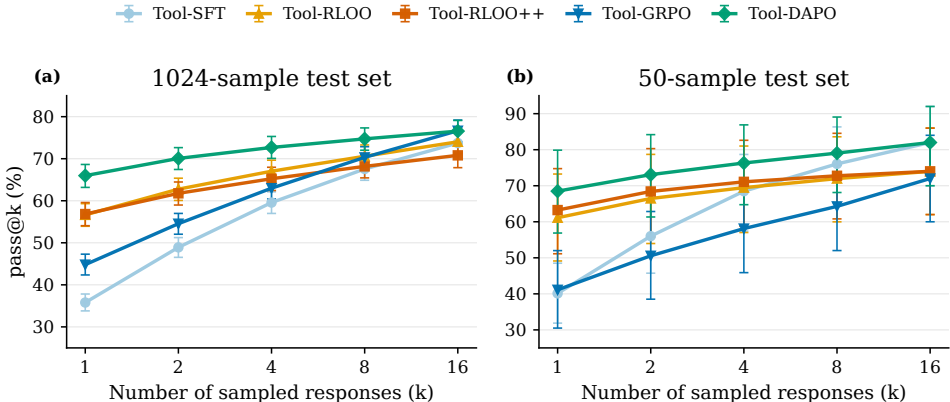


Figure 4: Comparison of reinforcement learning methods under tool integration. The curves show pass@ k for Tool-SFT, Tool-RLOO, Tool-RLOO++, and Tool-DAPO on the 1,024-problem fresh test set and the 50-problem public test set.

In contrast, Tool-DAPO further improves RL performance, increasing pass@1 by 9.3 percentage points over Tool-RLOO. Pass@16 also improves, suggesting that DAPO not only increases the probability of sampling a correct response early but also modestly improves the broader sampling distribution. This improvement is likely related to DAPO’s dynamic sampling strategy, which

filters out zero-variance prompt groups before the policy update. In our run, 60.7% of generated prompt groups were filtered, substantially increasing the density of informative training batches by removing all-correct or all-wrong groups. Although this filtering introduces additional resampling overhead, Tool-DAPO reaches higher pass@1 in 100 steps and 6.29 hours than the 200-step extended Tool-RLOO baseline, which requires 16.15 hours, as documented in Section B.

Table 1: Pass@ k (%) on the 1024-sample and 50-sample Countdown test sets. Best in **bold**.

Method	1024-sample test set					50-sample test set				
	$k=1$	$k=2$	$k=4$	$k=8$	$k=16$	$k=1$	$k=2$	$k=4$	$k=8$	$k=16$
Tool-SFT	35.8	48.9	59.6	67.6	73.7	40.1	56.0	68.5	76.1	82.0
Tool-RLOO	56.6	62.8	67.0	70.6	74.0	61.1	66.5	69.5	72.0	74.0
Tool-RLOO++	56.8	61.8	65.2	68.2	70.8	63.2	68.4	71.1	72.8	74.0
Tool-GRPO	44.8	54.5	63.0	70.3	76.7	41.1	50.6	58.1	64.3	72.0
Tool-DAPO	66.0	70.1	72.7	74.7	76.6	68.5	73.1	76.3	79.1	82.0

6 Discussion

6.1 Where RL Improves

To better understand where reinforcement learning improves model behavior, we examine the distribution of correct responses among 16 sampled responses per prompt across training checkpoints.

As shown in Figures 5 and 6, RL primarily increases the mass near fully correct response sets, suggesting that training reinforces trajectories that already contain correct reasoning traces. In contrast, examples with zero correct responses remain challenging: when the model fails to produce any correct trace among sampled responses, RL has limited signal for improving that example.

The Tool-DAPO distribution in Figure 7 increases the fraction of fully correct examples with fewer optimization steps. It also reduces the fraction of examples with zero correct samples, which is encouraging because these examples represent the most difficult cases for policy improvement.

6.2 Reinforcement Learning Encourages Tool Use

As shown in Figure 8, starting from the Tool-SFT model, Tool-RLOO increases the average number of executable tool calls per sampled rollout from below one call to approximately one call per rollout, while also improving Countdown accuracy. This suggests that reinforcement learning does not merely preserve the tool-use behavior acquired during SFT; instead, it further encourages tool invocation when tool use is beneficial for solving the task.

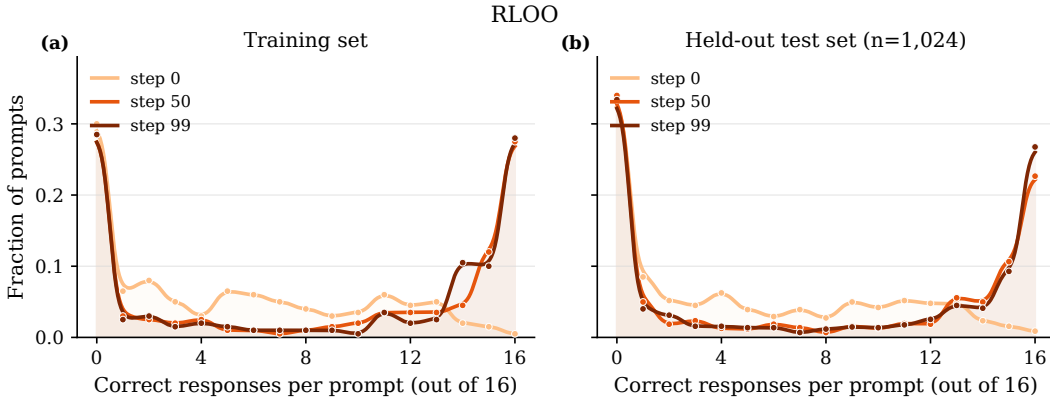


Figure 5: Correctness distribution across RLOO training. Each curve shows the fraction of examples with a given number of correct responses among 16 sampled responses.

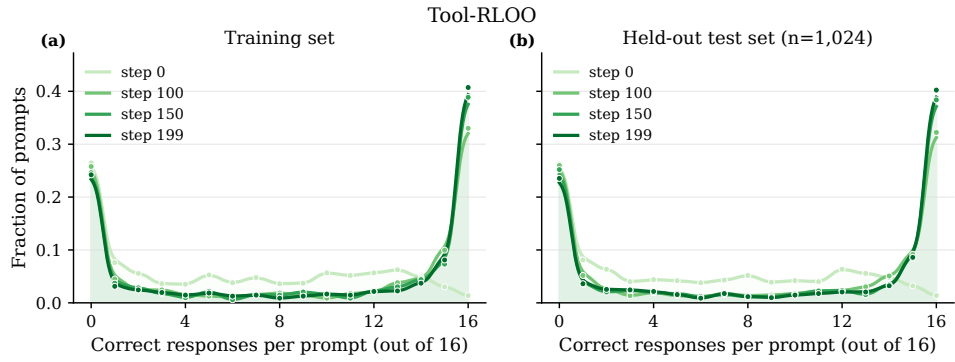


Figure 6: Correctness distribution across Tool-RLOO training. The tool-augmented model shows a similar shift toward high-correctness examples, with extended training further increasing the fraction of examples for which most sampled responses are correct.

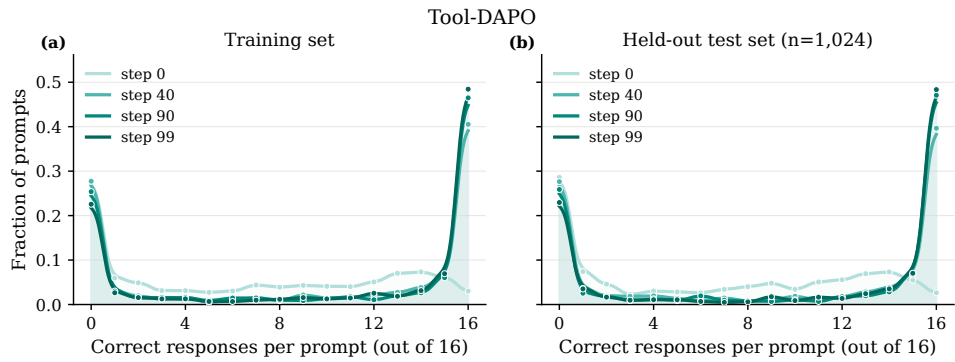


Figure 7: Correctness distribution across Tool-DAPO training. Compared with the RLOO-based baselines, Tool-DAPO substantially increases the fraction of examples with 16 out of 16 correct sampled responses, while more effectively reducing the fraction of examples with 0 out of 16 correct responses.

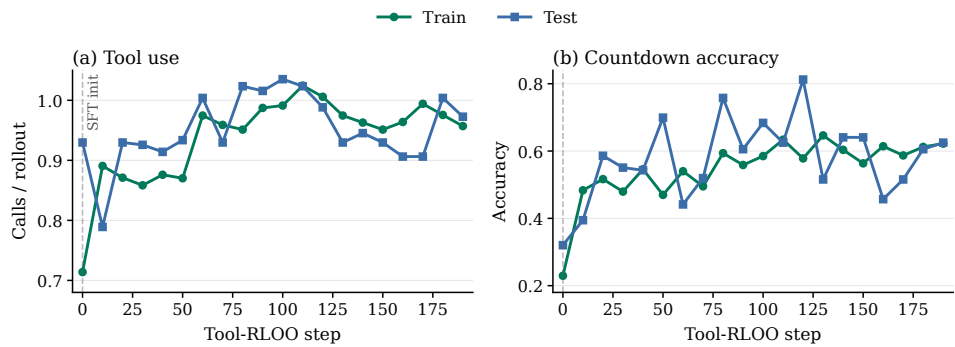


Figure 8: Tool-use and accuracy dynamics during Tool-RLOO training.

7 Conclusion

This project shows that reinforcement learning and calculator tool use are complementary for Count-down mathematical reasoning. RLOO substantially improves over SFT by increasing the probability of correct reasoning traces, while tool integration further improves both SFT and RL by reducing arithmetic and verification failures. Tool-DAPO provides the strongest result among the tested RL variants, suggesting that dynamic sampling and clipped updates can make training more efficient when many prompt groups provide little useful policy-gradient signal.

The analysis also identifies an important limitation. RL helps most when the model already samples at least one correct trajectory within a group; prompts with zero correct samples remain difficult because they provide little positive signal. Future work should therefore combine tool execution with stronger planning or search mechanisms, so the model can discover correct trajectories for hard prompts rather than only reallocating probability mass among trajectories it can already sample.

8 Team Contributions

- **Minghui Xu:** Tool-format data construction, RL implementation (RLOO++, DAPO), and report writing.
- **Zi Wang:** Tool-format data construction, SFT implementation, RL implementation (GRPO), and report writing.

References

- Arash Ahmadian, Chris Cremer, Matthias Gall'e, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet "Ust"un, and Sara Hooker. 2024a. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 12248–12267. doi:10.18653/v1/2024.acl-long.662
- Arash Ahmadian, Chris Cremer, Matthias Gall'e, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024b. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.LG] <https://arxiv.org/abs/2402.14740>
- Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2023. A General Theoretical Paradigm to Understand Learning from Human Preferences. arXiv:2310.12036 [cs.AI] <https://arxiv.org/abs/2310.12036>
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. arXiv:2504.11536 [cs.CL] <https://arxiv.org/abs/2504.11536>
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, 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, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, 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, Mingxu Zhou, 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, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, 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, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, 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. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638. doi:10.1038/s41586-025-09422-z
- Jian Hu. 2025. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv e-prints* (2025), arXiv-2501.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv preprint arXiv:2503.20783* (2025).
- OpenAI. 2024. OpenAI o1 System Card. arXiv:2412.16720 [cs.AI] <https://arxiv.org/abs/2412.16720>
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint arXiv:2302.04761* (2023).
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] <https://arxiv.org/abs/2210.03629>
- Yaoqi Ye, Yiran Zhao, Keyu Duan, Zeyu Zheng, Kenji Kawaguchi, Cihang Xie, and Michael Qizhe Shieh. 2026. In-Context Reinforcement Learning for Tool Use in Large Language Models. arXiv:2603.08068 [cs.AI] <https://arxiv.org/abs/2603.08068>
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. 2026. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems* 38 (2026), 113222–113244.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiase Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, and Yonghui Wu. 2025. DAP0: An Open-Source LLM Reinforcement Learning System at Scale. *arXiv preprint arXiv:2503.14476* (2025).
- Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? *arXiv preprint arXiv:2504.13837* (2025).

A Additional Analysis

Plain RLOO failure trace.

<think>

Let me analyze this step by step:

1. First, let's look at factors of 44:

Table 2: Example where plain RLOO fails but Tool-RLOO succeeds.

Numbers	[99, 11, 46, 48]
Target	44
Reference solution	$(99 - 11)/(48 - 46)$
RLOO pass@16	0/16
Tool-RLOO pass@16	4/16

$$44 = 2 * 22$$

$$44 = 4 * 11$$

2. Looking at the given numbers: 99, 11, 46, 48
46 is close to 44, but we need to add 2 somehow.
3. Let's try to work with 99 first:
99 is much larger than 44, so we need to reduce it.
 $99 - 48 = 51$ (too large)
 $99 - 46 = 53$ (too large)
4. Let's try a different approach:
 $48 - 46 = 2$
 $99 + 11 = 110$
 $110 - 2 = 108$ (too far from 44)
5. Let's try another combination:
 $46 + 48 = 94$
 $94 - 11 = 83$ (too far)
6. What if we start with 99 and subtract something?
 $99 - 46 = 53$
 $53 - 48 = 5$ (too small)
7. Let's try:
 $99 - 48 = 51$
 $51 - 11 = 40$
 $40 + 46 = 86$ (too far)
8. What about:
 $46 + 11 = 57$
 $57 - 48 = 9$
 $99 - 9 = 90$ (too far)
9. Let's try:
 $99 - 48 = 51$
 $51 - 46 = 5$
 $5 * 11 = 55$ (too far)
10. What if we divide a larger number?
 $99 / 11 = 9$
 $9 * 46 = 414$ (too large)
11. Let's try:
 $48 + 46 = 94$
 $94 - 99 = -5$
 $-5 + 11 = -4$ (not helpful)
12. New approach:
 $99 - 46 = 53$

$53 - 48 = 5$
 $5 * 11 = 55$ (too far)

13. Let's try:
 $99 - 48 = 51$
 $51 - 46 = 5$
 $5 * 11 = 55$ (too far)

14. One more attempt:
 $46 + 48 = 94$
 $94 - 99 = -5$
 $-5 + 11 = 6$ (not helpful)

15. Final try:
 $99 - 48 = 51$
 $51 - 46 = 5$
 $5 * 11 = 55$ (too far)

16. Let's try:
 $48 + 46 = 94$
 $94 - 99 = -5$
 $-5 + 11 = 6$ (not helpful)

17. One last attempt:
 $99 - 48 = 51$
 $51 - 11 = 40$
 $40 + 46 = 86$ (too far)

</think>

<answer> $(99 - 48) - 11 + 46$ </answer>

<think>

Let's verify:
 $99 - 48 = 51$
 $51 - 11 = 40$
 $40 + 46 = 86$ (too far)

Let me try a different sequence:

$99 - 46 = 53$
 $53 - 48 = 5$
 $5 * 11 = 55$ (too far)

Let's try:
 $46 + 48 = 94$
 $94 - 99 = -5$
 $-5 + 11 = 6$ (not helpful)

One last attempt:
 $99 - 48 = 51$
 $51 - 46 = 5$
 $5 * 11 = 55$ (too far)

Final try:
 $48 + 46 = 94$
 $94 - 99 = -5$
 $-5 + 11 = 6$ (not helpful)

Let's try:
99

Tool-RLOO success trace.

Let me try to reach 44 using these numbers.

First attempt:

$$99 - 48 = 51$$

$$51 - 46 = 5$$

$$5 * 11 = 55 \text{ (too high)}$$

Second attempt:

$$48 - 46 = 2$$

$$99 - 11 = 88$$

$$88 / 2 = 44 \text{ (This works!)}$$

Let me verify:

1. $48 - 46 = 2$

2. $99 - 11 = 88$

3. $88 / 2 = 44$

This works! Let me write it formally.

Let me double-check the final answer with the calculator.

```
<tool>(99 - 11) / (48 - 46)</tool>
```

```
<obs>= 44</obs>
```

That equals 44, so the answer is correct.

```
</think>
```

```
<answer>(99 - 11) / (48 - 46)</answer>
```

Analysis. Plain RLOO repeatedly explores nearby subtractive patterns such as $99 - 48$ and $99 - 46$, but it does not identify the key decomposition $48 - 46 = 2$ and $99 - 11 = 88$. Tool-RLOO finds this decomposition and then uses the external tool at the verification point:

```
<tool>(99 - 11) / (48 - 46)</tool>
```

```
<obs>= 44</obs>
```

The tool is useful here because it converts a proposed plan into a checked arithmetic result before the final answer is emitted. This helps distinguish a valid expression from the many near-miss expressions generated in the plain RLOO trace.

B Implementation Details

All RL experiments were run on a single H100 GPU. RLOO without tool use took 5.73 hours for 100 training steps, while Tool-RLOO with the matched 100-step training budget took 9.21 hours. The extended Tool-RLOO run continued from step 100 to step 200, requiring an additional 6.97 hours; therefore, the full 200-step Tool-RLOO training took 16.15 hours in total. Tool-RLOO++ took 5.29 hours for 100 steps, Tool-GRPO took about 5 hours, and Tool-DAPO took 6.29 hours for 100 steps. It should be noted that Tool-GRPO significantly deviates from SFT model after about 35 training steps, thus the Tool-GRPO is taken at the checkpoint with training step as 30.

Table 3: Hyperparameters for RLOO and Tool-RLOO. The two settings share the same RL hyperparameters; Tool-RLOO additionally enables tool execution during rollout.

Hyperparameter	Value
Initial policy	SFT checkpoint
Dataset	Countdown / Tool-Countdown
Tool execution	No / Yes
Max turns	- / 10
Training steps	100/200
Batch size	128
Group size	8
Gradient accumulation steps	128
Learning rate	1×10^{-5}
LR schedule	constant
Warmup ratio	0.0
Weight decay	1×10^{-4}
Gradient clipping	0.0
Entropy coefficient	1×10^{-3}
KL coefficient	1×10^{-3}
Temperature	1.0
Top- p	1.0
Top- k	-1
Min- p	0.0
Max generation tokens	1024
Max model length	2048
Checkpoint interval	10 steps
Policy-ratio clipping	No PPO-style clipping

Table 4: Hyperparameters for Tool-RLOO++. Tool-RLOO++ follows Tool-RLOO and modifies the advantage normalization.

Hyperparameter	Value
Initial policy	Tool-SFT checkpoint
Dataset	Tool-Countdown
Tool execution	Yes
Max turns	10
Training steps	100
Batch size	128
Group size	8
Gradient accumulation steps	128
Learning rate	1×10^{-5}
LR schedule	constant
Warmup ratio	0.0
Weight decay	1×10^{-4}
Gradient clipping	0.0
Entropy coefficient	1×10^{-3}
KL coefficient	1×10^{-3}
Temperature	1.0
Top- p	1.0
Top- k	-1
Min- p	0.0
Max generation tokens	1024
Max model length	2048
Advantage normalization	Batch normalization
Advantage target std.	0.3
Policy-ratio clipping	No PPO-style clipping
Checkpoint interval	10 steps

Table 5: Hyperparameters for Tool-GRPO. The trainer is a faithful single-update GRPO (DeepSeek-Math, $\mu = 1$): the importance ratio is identically 1 and PPO-style clipping is inert.

Hyperparameter	Value
Initial policy	SFT checkpoint
KL reference	Frozen SFT checkpoint
Dataset	Tool-Countdown
Tool execution	Yes
Max tool calls	5
Training steps	250
Batch size	128
Group size	8
Gradient accumulation steps	128
Inner epochs (μ)	1
Learning rate	1×10^{-5}
LR schedule	constant
Warmup ratio	0.0
Weight decay	1×10^{-4}
Gradient clipping	1.0
Entropy coefficient	1×10^{-3}
KL coefficient (β)	0.01
Advantage	group-relative $(r - \mu)/(\sigma + 10^{-4})$
Temperature	0.6
Top- p	0.95
Top- k	20
Min- p	0.0
Max generation tokens	1024
Max prompt length	512
Max model length	4096
Checkpoint interval	10 steps
Policy-ratio clipping	$\epsilon = 0.2$ (inert at $\mu = 1$)

Table 6: Hyperparameters for Tool-DAPO.

Hyperparameter	Value
Initial policy	Tool-SFT checkpoint
Dataset	Tool-Countdown
Tool execution	Yes
Max turns	10
Training steps	100
Batch size	128
Group size	8
Gradient accumulation steps	128
Learning rate	2×10^{-5}
LR schedule	constant
Warmup ratio	0.0
Weight decay	1×10^{-4}
Gradient clipping	1.0
Entropy coefficient	1×10^{-3}
KL coefficient	1×10^{-3}
Temperature	1.0
Top- p	1.0
Top- k	-1
Min- p	0.0
Max generation tokens	1024
Max model length	3072
Advantage normalization	Group normalization
Policy-ratio clipping	$[1 - 0.2, 1 + 0.28]$
Dynamic sampling	Yes
Max resampling iterations	8
Overlong penalty	Disabled
Checkpoint interval	10 steps