

Extended Abstract

Motivation Reinforcement learning with verifiable rewards (RLVR) has become the dominant recipe for eliciting multi-step reasoning from large language models. In this paradigm, a model samples candidate solutions, a programmatic verifier scores them, and a policy-gradient algorithm reinforces high-reward samples. However, verifier rewards are often sparse: the verifier returns 1 for a correct final answer and 0 otherwise. In Countdown, where the model must generate an arithmetic expression that evaluates to a target using given numbers, a completion that is a single operator away from a valid solution receives the same reward as one that is completely wrong. This gives no information about where an incorrect rollout went wrong or how close the sampled answer was to an ideal one.

Method We propose an AST-shaped dense reward for RLVR on Countdown. We treat the model’s final `<answer>` expression as a symbolic arithmetic program, parse it into an abstract syntax tree (AST), and compute a commutativity-aware tree-edit distance from the generated tree to the nearest valid solution tree. The set of valid solutions is enumerated exactly by a symbolic solver, so the reward remains verifiable and does not require a learned process reward model. Near misses receive partial credit through an exponentially decaying term $\lambda_a e^{-d_i/\tau}$, while correct answers still receive exactly λ_c and malformed answers receive zero. The reward is therefore dense, model-free, and a strict generalization of the sparse baseline, when the structural reward weight is set to zero, it collapses back to the standard Countdown reward.

Implementation We build on REINFORCE Leave-One-Out (RLOO), warm-started from a supervised fine-tuned Qwen2.5-0.5B checkpoint. For each prompt, we sample a group of $K = 8$ rollouts and compute the leave-one-out advantage using the mean reward of the other rollouts as the baseline. Our extension changes only the reward computation. We first perform SFT on `Asap7772/cog_behav_all_strategies`, then run RLOO on prompts from `asingh15/countdown_tasks_3to4`. We evaluate on the held-out 50-prompt test split, verify the main comparison on a larger 400-prompt held-out set, and construct a harder 5-number Countdown dataset to test whether the benefit persists as the task becomes harder. We sweep $\tau \in \{1, 2, 3, 4\}$ while keeping $\lambda_c = 1.0$, $\lambda_p = 0.1$, and $\lambda_a = 0.5$ fixed.

Results Dense AST shaping trades single-shot accuracy for stronger multi-sample coverage. On the 3-4 number task, sparse RLOO achieves the best Pass@1, reaching 57.81%, compared with 54.00% for AST $\tau=2$. However, AST $\tau=2$ starts to outperform sparse RLOO as the sampling budget increases where it overtakes sparse RLOO at $K = 4$ and widens the gap to +4.77% at $K = 16$. AST $\tau=2$ also achieves the best mean Pass@ K over both $k \leq 16$ and $k \leq 128$. On the independent 400-prompt held-out set, the same ranking holds, with the Pass@1 gap shrinking and AST $\tau=2$ leading from $K = 2$ onward. The temperature sweep shows that $\tau = 2$ performs best among the AST variants.

Discussion The Pass@1 versus Pass@ K trade-off appears to come from an exploration-exploitation difference. Sparse RLOO behaves more conservatively, its reasoning is cleaner and its intermediate arithmetic is more precise, which improves single-shot accuracy but narrows the search over possible solutions. AST $\tau=2$ produces more candidate equations per response and more distinct equations. This broader search leads to more distinct correct solutions per prompt, which is exactly what Pass@ K rewards. On the harder 5-number task, the benefit of AST shaping becomes more pronounced: AST $\tau=2$ is the only RL method that improves over the SFT initialization, while sparse RLOO mode collapses near the format-only reward.

Conclusion We explored an AST-shaped dense reward for RLVR on Countdown that grades a model’s final answer by its commutativity-aware tree-edit distance to the nearest exact solution. Our central finding is that dense AST shaping does not dominate sparse RLOO but trades off against it, sparse RLOO is stronger for single-shot accuracy, while AST $\tau=2$ is stronger for multi-sample inference. For any inference framework that draws several samples, such as best-of- N or verifier-guided reranking, dense AST shaping is preferable.

Dense Rewards for RLVR on Countdown via ASTs

Ahmed Elbakry

Department of Computer Science
Stanford University
bakry@stanford.edu

Abstract

Reinforcement learning with verifiable rewards (RLVR) has become a dominant recipe for eliciting multi-step reasoning from language models, but its rewards are often sparse. Typically, a verifier returns 1 for a correct final answer and 0 otherwise. In Countdown, this means a completion that is a single operator away from a valid solution receives the same reward as one that is completely wrong, giving little learning signal to RLOO when all samples fail. We propose an AST-shaped dense reward for RLVR on Countdown. We treat the model’s final `<answer>` expression as a symbolic arithmetic program, parse it into an abstract syntax tree, and measure a commutativity-aware tree-edit distance from the generated tree to the nearest valid solution tree, where the set of valid solutions is enumerated exactly by a symbolic solver. Near misses receive partial credit that decays smoothly with distance, while correctness and formatting rewards are preserved. Our results show that dense AST shaping does not dominate sparse RLOO, but introduces a trade-off, sparse RLOO achieves higher Pass@1, while AST shaping with $\tau = 2$ gives better multi-sample coverage for Pass@ K with $K \geq 4$. On the 3-4 number task, AST $\tau=2$ overtakes sparse RLOO at $K = 4$ and leads by +4.77% at $K = 16$. On the harder 5-number task, AST $\tau=2$ is the only RL method that improves over the SFT initialization, while sparse RLOO mode collapses to the format reward. Our analysis suggests that AST shaping encourages broader exploration, producing more candidate equations and more distinct correct solutions per prompt, at the cost of slightly lower single-shot precision.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has become the dominant recipe for eliciting multi-step reasoning from large language models. In this paradigm, a model samples several candidate solutions that are scored using a programmatic verifier, and a policy gradient algorithm is then used to reinforce generating samples with high rewards. Recent techniques such as DeepSeek-R1 Guo et al. (2025) demonstrate that this closed verifiable loop can induce sophisticated search and self-correction behaviors. Because the reward is verified programmatically rather than through a learned reward model, RLVR massively reduces the training and annotation cost compared to preference based techniques such as RLHF Ouyang et al. (2022), this makes it especially attractive for tasks that have a clear notion of correctness such as maths, code and tool calling.

Despite the benefits of RLVR, it also has its weaknesses. The sample rewards are often *sparse*, the verifier typically returns 1 for a correct final answer and 0 otherwise, this gives no information about where an incorrect rollout went wrong, and more importantly how close was the sampled answer to an ideal one.

In the multi-step arithmetic task Countdown, the model is given a few numbers, arithmetic symbols and a target, the model is expected to generate a mathematical expression that evaluates to the target. Early in training the overwhelming majority of rollouts fail, and a completion that is a single operator

away from a valid solution receives exactly the same reward as one that is completely wrong or even uses the wrong numbers. Even though a warm start using a supervised finetuned checkpoint can reduce this, it does not eliminate the problem, especially the more difficult that task gets. Even adding a small formatting reward of 0.1 for producing a parseable answer does not fix this, all formatted answers receive the same 0.1 regardless of how close they are to a solution, and a steep jump remains between a correctly formatted answer and a fully correct one.

In REINFORCE Leave-One-Out (RLOO) Ahmadian et al. (2024), which uses a leave-one-out variance reducing baseline to compute the advantages, batches in which every sample fails contribute zero advantage and therefore provide little to no meaningful learning signal. The direct result of this is slower learning, and if the task is difficult enough, even complete training failure. This can be partially addressed by using a larger batch size so that occasional successes appear more often, or by using a running mean baseline over the whole dataset instead of the per-batch one, but each of these has its own issues, larger batches increase the sampling cost and a global baseline gives up some of the variance reduction that the leave-one-out baseline provides.

A natural remedy is to make the reward *denser*. One way is process supervision, which uses a Process Reward Model (PRM) to score intermediate reasoning steps rather than relying solely on the final answer, and has been shown to outperform outcome supervision on mathematical reasoning Lightman et al. (2023); Wang et al. (2024). That PRM is itself a learned model, trained on a large set of human step-level annotations. This learned reward model reintroduces noise and is prone to reward hacking that RLVR aims to avoid. We ask whether the structure of the Countdown task itself can supply a dense, *exact* reward without any learned component. Unlike process supervision, we do not score the reasoning steps we keep an outcome-level reward but make it dense.

In this project, we propose an AST-shaped reward for RLVR on the Countdown task. We treat the model’s final `<answer>` expression as a symbolic arithmetic program, parse it into an abstract syntax tree (AST), and measure a commutativity-aware tree-edit distance from the generated tree to the nearest valid solution tree, where the set of valid solutions is enumerated exactly by a symbolic solver. Near misses receive partial credit that decays smoothly with distance, while the original correctness and formatting rewards are preserved. We aim to study and answer the following research questions:

- **RQ1.** Does AST-based dense reward shaping improve accuracy over the sparse RLOO on the Countdown task?
- **RQ2.** How does dense reward shaping affect the exploration-exploitation trade-off?
- **RQ3.** Does the benefit of dense AST shaping persist as the Countdown task becomes harder?

2 Related Work

RL fine-tuning of language models. RL fine-tuning is commonly used as a post-training step to align a language model with a human preference. The language model is the policy that is conditioned on a prompt and the tokens generated so far, it samples the next token, and model is trained to make better/ more preferred responses more likely under that policy. The main distinction across methods is where this reward comes from. *Preference-based* methods are used when correctness is hard to specify but responses can still be ranked relative to each other, deriving supervision from preferred and dispreferred completions. RLHF Ouyang et al. (2022) fits an explicit reward model and optimizes it with PPO Schulman et al. (2017), whereas DPO Rafailov et al. (2024) and IPO Azar et al. (2023) drop the explicit reward model and the online sampling loop, optimizing a classification objective directly against a frozen reference policy. Despite their effectiveness, these methods require high quality human labeled datasets and in cases like PPO where a reward model is trained, it may be complex to tune and may be more prone to reward hacking.

Verifier-based methods instead target tasks with programmatically verifiable answers, such as mathematics, code. The model samples rollouts on-policy and a rule-based verifier scores each one. In this setting, PPO relies on a learned value network as its baseline, while RLOO Ahmadian et al. (2024) and GRPO Shao et al. (2024) completely get rid of the critic and instead form the baseline from a group of rollouts for the same prompt, the leave-one-out mean for RLOO, and the group mean and standard deviation for GRPO, in order to reduce the variance of the gradient estimate. In this project we build on RLOO but our contribution is orthogonal to this choice: we change the reward

the estimator consumes, not the estimator itself, so the same AST signal could be used with PPO or GRPO unchanged.

Reward design for reasoning. A core limitation of verifier based RL is that the verifier usually provides only an outcome reward, a binary reward for whether the completion is correct or incorrect, with no indication of which intermediate choices helped or hurt. One way to increase the density of the learning signal is process supervision, where a Process Reward Model (PRM) scores individual steps in a reasoning trace rather than only the final answer Lightman et al. (2023); Wang et al. (2024). This can provide useful intermediate feedback, but it requires step-level supervision and introduces a learned reward model whose judgments may be noisy or exploitable via reward hacking.

In this project we explore another approach to increase the density of the reward. Instead of scoring the reasoning trace, we exploit the fact that for maths tasks the answer is usually structured with well defined symbols. For Countdown, the generated expression can be parsed into an abstract syntax tree (AST), a tree representation of the expression in which internal nodes are operators and leaves are numbers. Because the task is small enough for all valid solution expressions to be enumerated exactly, we can compare the model generated AST against the nearest valid solution AST and assign partial credit according to their structural distance. The resulting reward is dense, but still verifiable.

Our method is connected to classical reward shaping, which augments a sparse task reward with an auxiliary signal to guide exploration. Our shaping term gives graded credit to near miss expressions while preserving the original correctness reward. The distance computation is a simple, position-aligned tree distance between abstract syntax trees, with an additional commutativity-aware matching step so that equivalent arithmetic expressions are not penalized for superficial operand order differences.

The Countdown task. In the Countdown task, a mathematical solution is trivially checkable, yet the space of expressions is combinatorially large, so coming up with a correct expression requires multi-step search. The Countdown task was also studied in Stream of Search Gandhi et al. (2024), which trains models to emit explicit search traces, and in the TinyZero Pan et al. (2025) reproduction of R1-style RL on Countdown, where the rewards are 1.0 if correct, 0.1 if it has only correct formatting and 0 otherwise. We use this formulation as our sparse baseline. Both reward only the final outcome, we explore whether grading that outcome by its structural proximity to the nearest valid solution yields a more useful learning signal.

3 Method

3.1 Preliminaries: RLOO with a verifiable reward

In our RLOO setup, we fine-tune a policy π_θ , warm started from a supervised fine-tuned (SFT) checkpoint, with REINFORCE Leave-One-Out (RLOO). For each prompt x we sample a group of K rollouts $y_1, \dots, y_K \sim \pi_\theta(\cdot | x)$ and score each with a scalar reward $R(x, y_i)$. In RLOO the advantage of a rollout is formed by using the mean reward of the other $K - 1$ rollouts in the group as a variance reducing baseline:

$$A_i = R(x, y_i) - \frac{1}{K-1} \sum_{j \neq i} R(x, y_j). \tag{1}$$

The policy is updated with the policy-gradient loss

$$\mathcal{L}_{\text{PG}} = -\frac{1}{K} \sum_{i=1}^K \rho_i A_i \sum_{t=1}^{|y_i|} \log \pi_\theta(y_{i,t} | x, y_{i,<t}), \tag{2}$$

where ρ_i is a per-sample importance weight which is added to account for: rollouts that are generated by vLLM (a behavior policy μ) but the gradient is taken under the PyTorch copy of π_θ , and the two can assign slightly different token log-probabilities. We therefore reweight by

$$\rho_i = \min\left(\exp\left(\sum_t \log \pi_\theta(y_{i,t} | x, y_{i,<t}) - \log \mu(y_i | x)\right), c\right), \tag{3}$$

computed from the sequence log-probability, detached from the gradient graph, and clipped at c for stability. Our extension changes only the reward R and the estimator above is left intact.

3.2 Sparse outcome reward (baseline)

The default Countdown reward is sparse and given by the following piecewise function:

$$R_{\text{sparse}}(x, y) = \begin{cases} 1.0 & y \text{ is correct,} \\ 0.1 & y \text{ is parseable but is not correct,} \\ 0 & y \text{ is not parseable,} \end{cases} \quad (4)$$

where a rollout is correct when the extracted expression uses each provided number exactly once and evaluates to the given target. This is the sparsity issue which we aim to address, every incorrect but well formatted rollout collapses to 0.1, regardless of how close it is to a solution, so a group in which all K rollouts fail yields near-zero advantages in Eq. (1).

3.3 AST-shaped dense reward

Our extension replaces R_{sparse} with a reward that scores an incorrect answer by how structurally close it is to a valid solution (Figure 1).

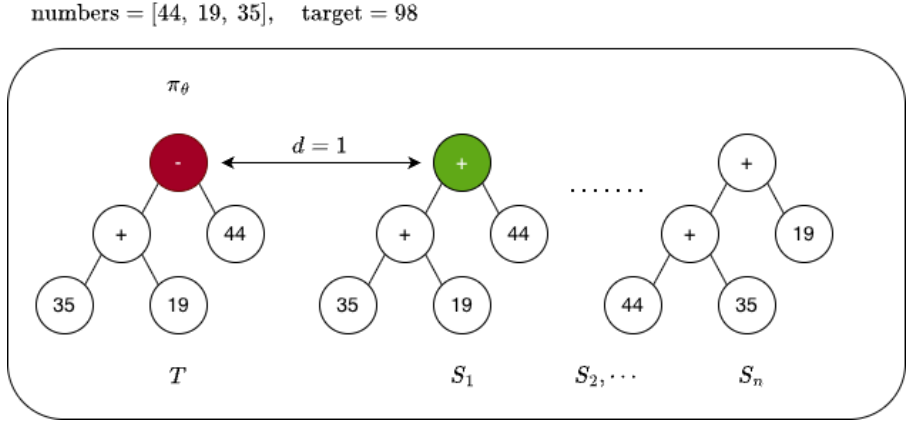


Figure 1: Method overview. The model’s final `<answer>` expression is parsed into an AST T (here $T = (35 + 19) - 44$ for numbers [44, 19, 35] and target 98), while a symbolic solver enumerates the set of all valid solution trees $\mathcal{S}(x) = \{S_1, \dots, S_n\}$ (e.g. $S_1 = (35 + 19) + 44 = 98$). We score T by its minimum commutativity-aware tree edit distance to this set. Here T differs from the nearest solution S_1 by a single operator ($-$ vs. $+$), so $d = 1$ and the rollout earns partial credit $\lambda_p + \lambda_a e^{-1/\tau} \approx 0.4$ rather than the sparse format-only $\lambda_p = 0.1$.

The reward is computed in 4 steps:

(1) Extract and parse. From rollout y_i we take the last `<answer> . . . </answer>` span and parse its contents as an arithmetic expression over $\{+, -, \times, \div\}$ and integer literals, producing a binary abstract syntax tree T_i whose leaves are numbers and whose internal nodes are operators. Contents that are not valid arithmetic give $T_i = \emptyset$.

(2) Enumerate valid solutions. A symbolic solver builds the set $\mathcal{S}(x)$ of all solution trees for a given problem. We generate a binary tree for every permutation, we keep valid trees only that use each number once and evaluate to the target. $\mathcal{S}(x)$ is computed once per prompt and cached.

(3) Structural distance. We score T_i by its minimum distance to the nearest tree in solution set $\mathcal{S}(x)$,

$$d_i = \min_{S \in \mathcal{S}(x)} d_{\text{tree}}(T_i, S). \quad (5)$$

Rather than a general tree edit distance, d_{tree} is a position-aligned distance that compares two trees root-to-root and recurses on matched children, incurring a cost of 1 for every node whose label differs

and when the full size of any subtree has no counterpart. Denoting a node’s label (operator or value) by $\ell(\cdot)$, T_L, T_R for its children, $|\cdot|$ for node count (a leaf’s children are \emptyset), and $d \equiv d_{\text{tree}}$,

$$d_{\text{tree}}(T, S) = \begin{cases} 0 & T = \emptyset, S = \emptyset, \\ |S| & T = \emptyset, S \neq \emptyset, \\ |T| & T \neq \emptyset, S = \emptyset, \\ \mathbf{1}[\ell(T) \neq \ell(S)] + a(T, S) & \text{otherwise.} \end{cases} \quad (6)$$

The child alignment cost is $a(T, S) = d(T_L, S_L) + d(T_R, S_R)$ in general, but at nodes that share a commutative operator $\text{op}(T) = \text{op}(S) \in \{+, \times\}$ we also try the swapped pairing and keep the cheaper of the two, $a(T, S) = \min(d(T_L, S_L) + d(T_R, S_R), d(T_L, S_R) + d(T_R, S_L))$. This commutativity awareness lets algebraically equivalent expressions (e.g. $a + b$ and $b + a$) reach distance 0. Associativity, by contrast, is not included in d_{tree} , for example the trees for $(a + b) + c$ and $a + (b + c)$ differ, but it need not be, because $S(x)$ already enumerates every tree shape (step 2), a validly grouped near-miss still matches some valid solution exactly, so the policy is never penalized for its choice of parenthesis. Thus d_i captures genuine structural error rather than surface syntax: an expression that differs from a solution by a single operator has $d_i = 1$, while a malformed or unrelated expression is many edits away.

(4) Shape the reward. To convert the distance into a reward, we pass it through $e^{-d_i/\tau}$ which gives us a desirable exponentially decaying property, where τ is a hyperparameter controlling the smoothness of the function. Concretely, we use the following modified piecewise function for computing the reward:

$$R_{\text{AST}}(x, y_i) = \begin{cases} \lambda_c & y \text{ is correct,} \\ \lambda_p + \lambda_a e^{-d_i/\tau} & y \text{ is parseable but is not correct,} \\ 0 & y \text{ is not parseable,} \end{cases} \quad (7)$$

Equivalently, with indicators for a correct answer, a present <answer> span, and a valid expression (parseable and using the given numbers),

$$R_{\text{AST}}(x, y_i) = \lambda_c \mathbf{1}[\text{correct}] + \lambda_p \mathbf{1}[\text{ans}] (1 - \mathbf{1}[\text{correct}]) + \lambda_a \mathbf{1}[\text{valid}] (1 - \mathbf{1}[\text{correct}]) e^{-d_i/\tau}. \quad (8)$$

Four main properties make this a desirable reward shaping term:

- **Suppressed at the goal.** The shaping term is switched off on correct rollouts, so a correct answer always receives exactly λ_c and the verifier’s decision is never distorted.
- **Order-preserving.** We require $\lambda_a < \lambda_c - \lambda_p$, so the best possible near-miss ($\lambda_p + \lambda_a$) still scores strictly below a correct answer (λ_c). Specifically, a near miss will earn more than a far miss, but never as much as a true solution.
- **Gated.** Structural credit requires using the prompt’s exact numbers, as we observed in initial unsuccessful experiments, without this gate the policy can learn to hack the reward with trivial parseable expressions such as <answer>1+1</answer>.
- **Tunable decay.** The temperature τ controls how far credit extends into the space of near misses, small τ rewards essentially only the $d_i = 1$ neighbors, as τ increases the ladder is flattened towards a constant (Figure 4, Appendix A). We treat τ as a hyperparameter and sweep it.

Unless stated otherwise we use $\lambda_c = 1.0$, $\lambda_p = 0.1$, $\lambda_a = 0.5$, and $\tau = 2$. With these same values but $\lambda_a = 0$, the structural term vanishes and the ladder collapses back onto the sparse baseline,

$$R_{\text{AST}}|_{\lambda_a=0, \lambda_c=1.0, \lambda_p=0.1} = R_{\text{sparse}}, \quad (9)$$

so our reward is a strict generalization of Eq. (4): it keeps the same ladder and differs only by the $\lambda_a e^{-d_i/\tau}$ credit added to incorrect but parseable near misses.

4 Experimental Setup

Task and data. The main task explored is Countdown with 3-4 input numbers. We initialize from the Qwen2.5-0.5B Base model Yang et al. (2024); Team (2024). We first perform supervised fine-tuning (SFT) on the `Asap7772/cog_behav_all_strategies`¹ demonstrations, then run RLOO on prompts from `asingh15/countdown_tasks_3to4`², this SFT checkpoint also serves as the frozen reference π_{ref} while computing the KL penalty to ensure that the trainable policy π_{θ} does not diverge. We use the test split of 50 prompts (24 three-number, 26 four-number) that is never seen in training for evaluation. Because 50 prompts is a relatively small sample, we additionally construct a larger held-out set of 400 3-4 number prompts that we use to verify the main comparison (Appendix C.2), we verify there are no sample overlaps between this and the original train/test sets. To test how the shaped reward behaves on harder problems (RQ3), we additionally construct `aelbakry/countdown_5num_tasks`³, a 5-number version of Countdown that we created for this work, and train and evaluate on it under the same protocol. Sizes for all splits are summarized in Table 5 (Appendix B).

Metrics and evaluation. Our primary metrics are **Pass@1** and **Pass@K** for $k = 1, \dots, 16$, and the mean of **Pass@K** over this range. We report both because they capture two different aspects. **Pass@1** measures single-shot accuracy, whereas **Pass@K** measures the coverage a model has to offer when using multiple samples for selection as in best-of- N , or verifier-guided reranking, where sample diversity, not single-shot accuracy, is what matters. In the Countdown task it is trivial to generate multiple samples and programmatically verify their correctness, without the need of additional trained verifiers.

For evaluation, we sample 128 responses for each of the 50 test prompts in order to obtain a more reliable estimate of **Pass@K**, which we report up to $K = 16$, a reasonable budget for multi-sample inference. Evaluation uses temperature 0.6, top- k 20, top- p 0.95. We cap generation at 1024 tokens in distribution and up to 4096 for the longer 5- and 6-number problems.

Training details. All RL runs use RLOO with a group size of $K = 8$ rollouts per prompt, an overall effective batch size of 128 through 128 gradient accumulation steps, and 100 update steps (due to compute constraints), optimized with AdamW at a constant learning rate of 10^{-5} and weight decay of 10^{-4} . We include an entropy bonus of 0.001, a KL coefficient of 0.001 to the SFT reference, mainly for tracking these metrics. Rollouts are generated with vLLM, so we include importance weight ρ_i and clip at 100. We trained on H100 GPUs and final inference was on A100. Using this setup a single RLOO training run took ≈ 350 minutes.

5 Results

We organize results around the three research questions from the introduction: how dense AST shaping reshapes the single-shot versus multi-sample trade-off (RQ1), the exploration mechanism behind it (RQ2), and behavior on harder problems (RQ3).

5.1 Quantitative Evaluation

Dense shaping trades single-shot accuracy for multi-sample coverage (RQ1). Table 1 and Figure 2 give the main comparison. Sparse RLOO achieves the best performance for **Pass@1**, reaching 57.81%, +3.81% above AST $\tau=2$ which achieves 54.00%. But AST starts to outperform sparse RLOO as k increases. AST $\tau=2$ overtakes sparse RLOO at $K = 4$ and the gap widens to +4.77% at $K = 16$. AST also achieves the best mean **Pass@K**, both restricted to the practical inference range (72.29% vs 69.85% over $k \leq 16$) and over the full sample budget (81.22% vs 75.24% over $k \leq 128$). A second pattern reinforces this, at $K = 16$ the SFT initialization checkpoint already performs better than sparse RLOO, it is likely that sparse RL improves single-shot accuracy by collapsing its sample diversity below its own starting point, whereas AST $\tau=2$ is the only method whose $K = 16$ coverage improves on SFT. For any inference protocol that draws $K \geq 4$ samples,

¹https://huggingface.co/datasets/Asap7772/cog_behav_all_strategies

²https://huggingface.co/datasets/asingh15/countdown_tasks_3to4

³https://huggingface.co/datasets/aelbakry/countdown_5num_tasks

Table 1: Evaluation results on the 3-4 number task. The last column reports the mean Pass@ K over two averaging ranges, $k \leq 16 / k \leq 128$.

Method	Pass@1	Pass@8	Pass@16	mean Pass@ K $k \leq 16 / k \leq 128$
SFT	32.78	70.30	75.63	66.37 / 79.29
RLOO (sparse)	57.81	71.01	72.65	69.85 / 75.24
AST $\tau=2$ (ours)	54.00	74.16	77.42	72.29 / 81.22

AST $\tau=2$ is preferable. To gain more confidence in these results, we re-ran the evaluation on the 400 held-out set (Appendix C.2). The ranking is the same, with the Pass@1 gap shrinking to within noise (28.6% vs 28.1%) and AST $\tau=2$ leading from $K = 2$ to +3.75% at $K = 16$.

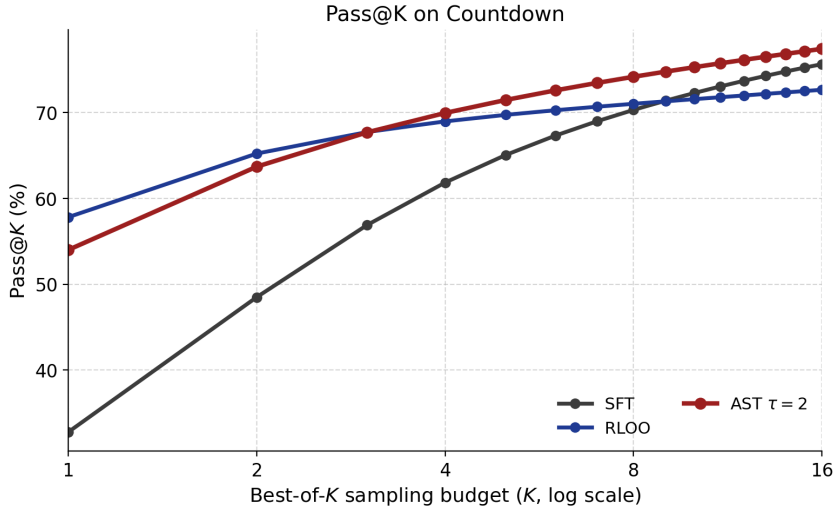


Figure 2: Pass@ K on the 3-4-number test set ($k \leq 16$, K on a log scale) for SFT, sparse RLOO, and AST $\tau=2$. Sparse RLOO leads at small K ; AST $\tau=2$ overtakes it at $K = 4$ and the gap widens through $K = 16$.

$\tau = 2$ produces the best results. In Table 2 we show the results for the sweep $\tau \in \{1, 2, 3, 4\}$. $\tau = 2$ simultaneously achieves the highest Pass@1 and the highest mean Pass@ K of any AST variant and the earliest crossover over sparse RLOO ($K^* = 4$), while its neighbors are strictly worse. Too sharp a ladder ($\tau = 1$) over-rewards the single nearest miss and collapses into repetition, while too flat a ladder ($\tau = 3, 4$) dilutes the correctness signal and barely crosses RLOO within $k \leq 16$.

Table 2: Sweep over τ . K^* is the smallest budget at which AST overtakes sparse RLOO.

Method	Pass@1	mean Pass@ K	K^*
RLOO (sparse)	57.81	69.85	—
AST $\tau=1$	50.12	67.76	16
AST $\tau=2$	54.00	72.29	4
AST $\tau=3$	52.64	67.68	> 16
AST $\tau=4$	51.47	68.74	11

Dense shaping is most important when the task is harder (RQ3). Within the 3-4-number distribution the AST advantage is stable across difficulty (Appendix C.1). A more challenging test is to train and evaluate on the harder 5-number set (Table 3). Benefits of AST reward shaping are more pronounced, $\tau=2$ is the only method that improves over the SFT initialization, roughly doubling its Pass@1 (0.98% vs 0.41%) and leading on Pass@ K , whereas 5-number trained sparse RLOO

mode-collapses and drops below the zero-shot SFT model. This failure is clear from the training metrics (Figure 6), sparse RLOO’s mean eval reward stays almost constant at the format-only reward of 0.1 for the entire run, the policy learns only to emit a well-formed but incorrect answer and never reaches a correct one, while AST’s reward climbs above the floor as it discovers structurally closer expressions. Overall, accuracy stays low, even after increasing the maximum token length to 4096 to avoid running out of tokens, the results do not improve much. We think this can be attributed to the limited number of training steps as well as the base model size.

Pushing further to out-of-distribution (OOD) evaluation to 6 operands drives every method to 0% (Appendix C.4), where the base model simply cannot execute the arithmetic.

Table 3: Training and evaluating on the harder 5-number Countdown task. AST $\tau=2$ is the only RL method that improves over the zero-shot SFT baseline.

Method	Pass@1	Pass@16
SFT (zero-shot)	0.41	4.28
RLOO (5-num)	0.16	1.81
AST $\tau=2$ (5-num)	0.98	5.02

5.2 Qualitative Analysis

Why the trade-off exists: exploration vs. exploitation (RQ2). To understand the reason behind the Pass@1 / Pass@K trade-off, we parse the intermediate arithmetic inside each <think> block. The main difference is not simply that AST-shaped models write more, it is that they search more broadly. Compared with sparse RLOO, AST $\tau=2$ produces more candidate equations per response and a larger set of distinct equations, while repeating a smaller fraction of them. This suggests that the dense reward is not merely encouraging verbosity. Instead, it keeps partially promising steps to explore new solutions. Sparse RLOO behaves more conservatively as its traces are cleaner and its intermediate arithmetic is more precise, but this precision comes with a narrower search over possible solutions.

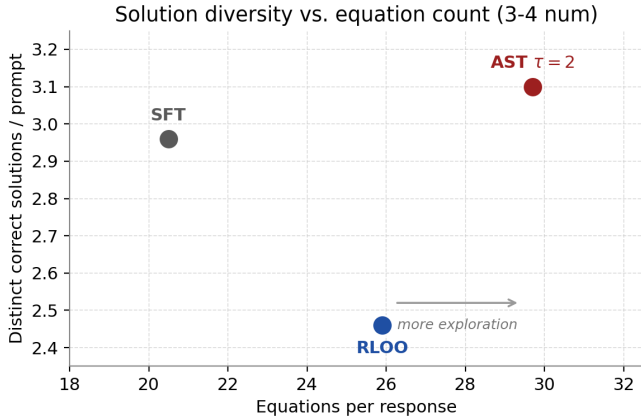


Figure 3: Equations per response versus distinct correct solutions per prompt.

This difference explains why sparse RLOO is stronger in the single-sample setting, while AST $\tau=2$ becomes stronger once multiple samples are allowed. Sparse RLOO tends to exploit a smaller set of high confidence paths, which improves Pass@1 but often leads to the same solution across samples. On the other hand, AST $\tau=2$ has noisier intermediate reasoning in exchange for broader coverage across solutions. Across sampled responses it gets more distinct correct solutions (Figure 3), which is exactly what Pass@K rewards. The effect is not caused by longer reasoning alone, as AST $\tau=2$ turns its broader search into more distinct *correct* solutions, not just more equations. The recovery statistic in Table 4 supports the same interpretation as AST $\tau=2$ more often reaches a correct final answer after making at least one wrong intermediate step (Appendix C.5 shows such a rollout),

indicating a less brittle but also less precise search process. Since equation extraction is regex-based and most probably is imperfect, we consider these measurements reflective of a trend rather than exact measurements.

Table 4: Reasoning arithmetic statistics, obtained by parsing equations inside `<think>`. “Recovery” is the fraction of correct answers whose trace contains ≥ 1 wrong intermediate equation.

Method	Eqs/resp	correct (%)	Recovery (%)
SFT	20.1	94.2	23.4
RLOO (sparse)	25.9	97.8	16.0
AST $\tau=2$	29.7	93.2	30.2

6 Discussion

Limitations. Our reward depends on a symbolic solver that enumerates all valid solution expressions for a prompt, which is only feasible because Countdown’s search space is small. On tasks where the set of correct solutions cannot be enumerated exactly, the AST distance would not be feasible. Our comparisons also used a single run for each experiment due to compute budget constraints. We constructed a larger test set to get a more reliable estimate of the model’s performance, 50-prompt Pass@1 gap between RLOO and AST shrank the 400-prompt set. The reasoning arithmetic precision and recovery statistics come from a regex parser and are indicative of trends, not exact rates. Finally, training was only done for 100 steps, and both Pass@1 and Pass@ K were still improving.

Difficulties encountered. The dense reward initially encouraged reward hacking, to address this we gated AST credit on using the prompt’s exact numbers, the policy learned to emit trivially parseable expressions such as `<answer>1+1</answer>`.

Broader impact. Where a task has enough structure to verify partial progress exactly, a verifiable dense reward is a cheap alternative to a learned process reward model, avoiding both its annotation cost and its exploitability. In general, other mathematical tasks and coding could benefit from this were their rewards are often constructed to be binary. Additionally, we found that using partial credit can encourage exploration, which can be used as a lever on many tasks to encourage diversity.

7 Conclusion

In this project we explored an AST-shaped dense reward for RLVR on Countdown that grades a model’s answer by its commutativity-aware tree-edit distance to the nearest exact solution, giving partial credit to near misses. The reward is a strict generalization of the sparse baseline and trades a small single-shot cost (-3.81% Pass@1) for a consistent multi-sample gain, AST $\tau=2$ overtakes sparse RLOO from $K = 4$ and leads by $+4.77\%$ at $K = 16$, with $\tau = 2$ performing best. We also uncovered that it is not simply the case that AST writes more equations per response, but it also has more distinct correct solutions per prompt. For any pipeline that draws several samples, dense AST shaping is therefore preferred over sparse RLOO.

Because intermediate arithmetic is itself exactly checkable, a feasible next step is to add an a reasoning arithmetic reward that scores the reasoning steps and not only the final answer. Similar to process supervision as in PRM Lightman et al. (2023); Wang et al. (2024), but exact and model-free. This would directly target the recovery behavior we observed (correct answers reached through incorrect intermediate steps) and could close AST’s precision gap with sparse RLOO while preserving its exploration. We also fixed the reward weights and varied only τ , in particular λ_{AST} was kept constant at 0.5, and a fuller sweep of the weights and their interaction with τ may be beneficial.

Changes from Proposal Our proposal additionally suggested weighting the policy-gradient update per token, down-weighting the `<think>` reasoning tokens by a factor α relative to the `<answer>` tokens. Early experiments with $\alpha = 0.1$ underperformed uniform weighting ($\alpha = 1$), so we did not explore that path further and instead invested our effort in more analytical experiments to understand the observed behaviour of AST dense rewards with RLOO.

References

- Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. 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>
- Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. arXiv:2404.03683 [cs.LG] <https://arxiv.org/abs/2404.03683>
- 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, Yuduan 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
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let’s Verify Step by Step. arXiv:2305.20050 [cs.LG] <https://arxiv.org/abs/2305.20050>
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL] <https://arxiv.org/abs/2203.02155>
- Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. TinyZero. <https://github.com/Jiayi-Pan/TinyZero>. Accessed: 2025-01-24.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] <https://arxiv.org/abs/2305.18290>

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 <http://arxiv.org/abs/1707.06347>

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300 [cs.CL] <https://arxiv.org/abs/2402.03300>

Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. <https://qwenlm.github.io/blog/qwen2.5/>

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2024. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. arXiv:2312.08935 [cs.AI] <https://arxiv.org/abs/2312.08935>

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671* (2024).

A Reward Ladder

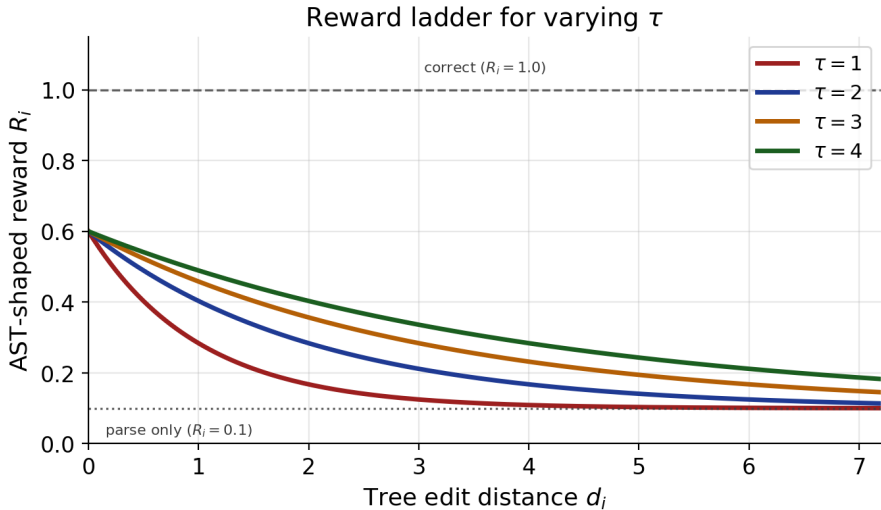


Figure 4: Reward ladder for the AST-shaped reward. The partial-credit branch $\lambda_p + \lambda_a e^{-d_i/\tau}$ is plotted against tree-edit distance d_i for $\tau \in \{1, 2, 3, 4\}$, with $\lambda_c = 1.0$, $\lambda_p = 0.1$, and $\lambda_a = 0.5$. Smaller τ concentrates credit on near misses, while larger τ spreads it over more distant solutions.

B Datasets

Table 5 lists all datasets used in our experiments. We used the original datasets provided for 3-4-numbers and supplemented with additional ones to test different aspects of AST based RLOO.

Table 5: Datasets used for training and evaluation. “(ours)” marks the additional datasets that we constructed.

Dataset	Split	# Examples	Use
cog_behav_all_strategies	train	1,000	SFT warm-start
countdown_tasks_3to4	train	490,314	RLOO training (3-4 num)
countdown_tasks_3to4	test	50	Pass@ K evaluation (3-4 num)
countdown_3to4_held_out_400 (ours)	test	400	Pass@ K verification (3-4 num)
countdown_5num_tasks (ours)	train	14,950	RLOO training (5 num)
countdown_5num_tasks (ours)	test	50	Pass@ K evaluation (5 num)
countdown_6num_tasks (ours)	test	50	OOD evaluation (6 num)

C Additional Experiments

C.1 Difficulty breakdown within 3-4 numbers

Table 6: Results split by difficulty, $k \leq 16$.

Method	3-number		4-number	
	Pass@1	mean P@ K	Pass@1	mean P@ K
SFT	45.25	87.26	21.27	47.09
RLOO (sparse)	79.98	88.69	37.35	52.47
AST $\tau=2$ (ours)	76.20	90.70	33.50	55.30

C.2 Verification on a larger held-out set (400 prompts)

To rule out uncertainty in the presented results caused by using a small test of 50 samples, we repeat the headline Pass@ K comparison on an independent set of 400 held-out 3-4-numbers (Figure 5).

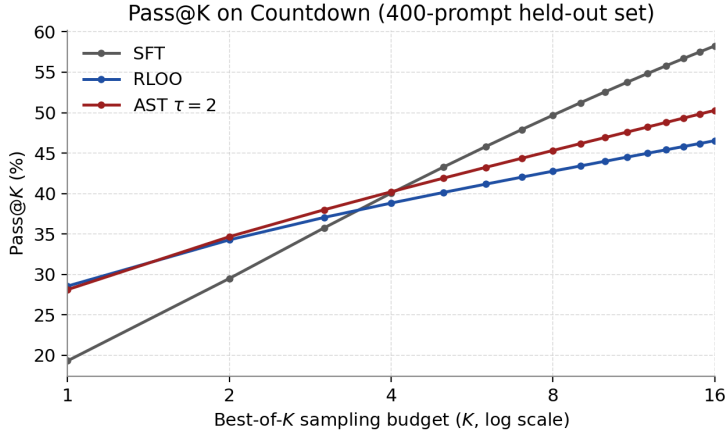


Figure 5: Pass@ K ($k \leq 16$) on the independent 400-prompt held-out set.

C.3 Training on the 5-number task

Figure 6 plots the mean per-sample reward on the held-out evaluation set, logged over the course of the 5-number run, for sparse RLOO and AST $\tau=2$. Recall that the sparse reward returns 0.1 for a well-formed but incorrect answer and 1.0 only for a correct one, so a curve sitting at 0.1 means the policy produces parseable answers that are not correct. Sparse RLOO stays exactly there for the whole run, which is indicative of mode collapse. AST training becomes unstable and reward drops at around step 60 but starts recovering.

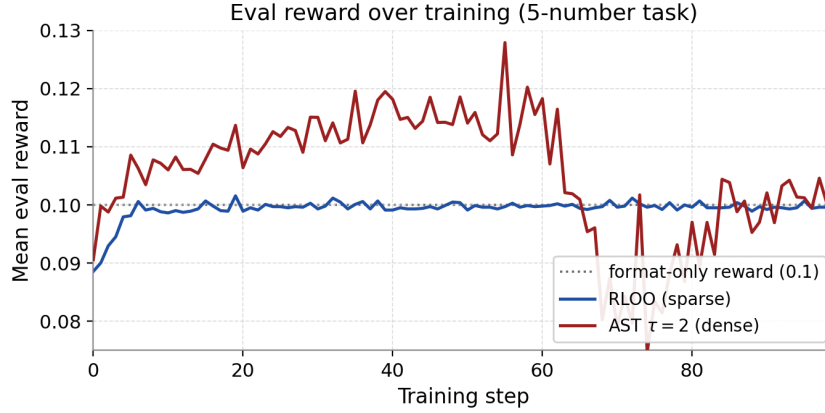


Figure 6: Mean eval reward over training on the 5-number task.

C.4 Generalization to 6-number problems

We also evaluated, zero-shot, on a held-out 6-number set (Appendix B). All methods score exactly 0% at both Pass@1 and Pass@K. At six operands a valid derivation requires composing five operations over a much larger search space, which exceeds what a 0.5B base model can execute reliably, especially when testing on OOD samples.

C.5 A correct AST $\tau=2$ answer despite a wrong intermediate step

An AST $\tau=2$ rollout on numbers $\{81, 8, 3\}$ with target 35. The reasoning contains an incorrect intermediate computation, it writes $105 - 7 = 108$, which is wrong, but the model discards that path and still reaches a correct final answer. Probably the model was trying to compute $105 + 3 = 108$ but 3 was already used in that reasoning path.

```

AST  $\tau=2$ : numbers  $\{81, 8, 3\}$ , target 35

<think>
Let me try to find a way to get to 35 using these numbers.

First, let me try working with the larger numbers:
81 - 8 = 73
73 - 3 = 70
70 is too far from 35

Let me try a different approach:
8 * 3 = 24
24 + 81 = 105
105 - 7 = 108 (too far)

Let me try division:
81 / 3 = 27
27 + 8 = 35

This works! Let me verify:
81 / 3 = 27
27 + 8 = 35
</think>

<answer> (81 / 3 + 8) </answer>

```