

Extended Abstract

Motivation. Tool-integrated reasoning (TIR) allows language models (LMs) to offload some computation to external tools, and RL post-training has been found to reliably induce this behavior. However, outcome-only RL with tools has a frequent failure mode: the policy learns to fully dispatch computation to the tool rather than internalizing the underlying skill, so capability collapses once the tool is not available (Zeng et al., 2026). This is, perhaps, not a concern when the tool is always available, but problematic when robustness to tool absence matters, or when the explicit goal is for the model to learn the underlying skill itself. This phenomenon is most evident in small models, where the capability prior is often weak and so there is correspondingly more to gain from offloading; even so, present remedies operate at a billion-parameter scale. We study whether a small policy can be made to internalize arithmetic on the Countdown task instead of offloading it.

Method. We propose *annealed tool access*, a curriculum over the policy’s action space during RLOO training. The model begins with full calculator tool access; the tool availability is then progressively withdrawn, which incentivizes the policy to reproduce successful (tool-aided) rollouts internally. We use a linear schedule, which leaves the final third of training completely tool-free. Unlike reward-focused remedies, our annealing method does not require auxiliary supervision or adjusted reward redesign.

Implementation. We extend an RLOO rollout with an inline calculator. The policy can write `<calc>expr</calc>` and the vLLM worker interrupts generation, evaluates the expression, injects `<result>value</result>`, and resumes; the injected tool-output tokens are masked from the gradient. From an SFT warm-start, distilled from a Qwen2.5-7B-Instruct teacher to instill a strong tool-dispatch prior, we train four RLOO conditions on the Countdown task with Qwen2.5-0.5B: (a) no-tool, (b) fixed-tool, (c) annealed, and (d) fixed-tool evaluated without the tool.

Results We evaluate four RLOO conditions on a 50-prompt held-out Countdown split, reporting $\text{pass}@k$ under both no-tool and tool-enabled evaluation. From the SFT-B initialization, fixed-tool RLOO exhibits the predicted cognitive-offloading collapse: when the calculator is withdrawn at eval time, $\text{pass}@1$ falls to 0.036 ± 0.02 , an order of magnitude below the no-tool RLOO baseline (0.368 ± 0.09), with roughly two-thirds of rollouts still emitting `<calc>` tokens into the no-tool eval environment. Our annealing curriculum eliminates this collapse: the annealed checkpoint matches no-tool RLOO on $\text{pass}@1$ (0.368), edges it on $\text{pass}@8$ (0.650 vs. 0.602), and retains most of the fixed-tool’s tool-eval gain ($\text{pass}@8 = 0.643$ vs. 0.721). Training dynamics give the cleanest evidence of internalization: mean reward continues to climb from 0.420 at full tool withdrawal (step 67) to 0.485 at step 99, a +0.065 gain that is necessarily tool-free. We scope the claim with a third finding: all SFT-B conditions trail the no-SFT reference baseline by roughly 0.18 on $\text{pass}@1$, which we attribute to catastrophic forgetting during the 7B-distillation SFT pass rather than to the RL recipe.

Discussion. The three findings described above give a clean account of when annealing helps. Fixed-tool RL on a strong prior reproduces the cognitive offloading collapse (as predicted), and annealing removes it without sacrificing tool-evaluation performance. The tool scaffolds high-reward rollouts early, and withdrawing it forces the policy to carry the rollout on its own, weight-based arithmetic prior. The continued post-withdrawal reward climb is evidence of this. Annealing protects against offloading within a given SFT prior, but cannot recover capability that a prior SFT pass overwrote, pointing to lighter-weight distillation (e.g., LoRA or mixed-trace SFT) as the natural next step (we leave this for future work).

Conclusion. An action-space curriculum that gradually withdraws the calculator tool is enough to prevent a small policy from the tool dispatch collapse, otherwise experienced by a conventional fixed-tool RL, recovering no-tool robustness while retaining most of the gain from the tool. Getting a small model to use a tool it never learned in pretraining is itself a challenging task, since few-shot prompting is often insufficient, and SFT can trigger catastrophic forgetting and hence confound the benefits of annealing. We see careful tool bootstrapping combined with availability annealing as a promising recipe for tool internalization at small scale.

Improving Tool Internalization for Small Models: Annealed Tool Access for RLOO on Countdown

Maty Bohacek
Department of Computer Science
Stanford University
maty@stanford.edu

Jason Zhang
Department of Computer Science
Stanford University
jasonbz@stanford.edu

Abstract

Tool-integrated reasoning (TIR), i.e., language model (LM) reasoning that relies on external tools, augments conventional LM inference with external computation, which unlocks paradigms for which their architectures might not be suitable or where external interaction is needed. However, outcome-only reinforcement learning (RL), a form of RL where the agent only receives feedback on the final result and not on individual steps, with tools, is prone to cognitive offloading: the policy learns to dispatch work to the tool rather than internalize the underlying skill, and capability fully collapses once the tool is withdrawn. This pressure is strongest for small models, yet the remedies currently available target models with billion-scale parameters. We introduce *annealed tool access*, a simple intervention on the RL side that requires no reward shaping or auxiliary supervision: a Qwen2.5-0.5B policy is trained with reinforcement learning with outcome optimization (RLOO) on the Countdown task under a calculator whose per-step availability is linearly withdrawn, starting with full access and ending completely tool-free. We find that (i) training RL with continual tool access and a strong prior for tool use reproduces the predicted collapse—no-tool pass@1 falls to 0.036, an order of magnitude below the no-tool RLOO baseline at 0.368, with two thirds of rollouts still emitting calculator calls into a disabled tool environment; (ii) annealing eliminates this collapse, matching no-tool RLOO on pass@1 at 0.368, slightly outperforming on pass@8 0.650 vs. 0.602, and retaining most of the fixed-tool model’s advantage on tool-available evaluations, with mean reward continuing to improve after the tool is fully withdrawn, indicating that arithmetic is being internalized rather than re-routed; and (iii) all of our fine-tuned conditions trail a no supervised fine-tuning (no-SFT) reference by around 0.18 pass@1, which we attribute to catastrophic forgetting during the distillation SFT pass rather than the RL method used, thus scoping the annealing claim. Annealed tool access hence yields the tool-trained model’s tool-time performance and the no-tool model’s soundness in a single run.

1 Introduction

Tool-integrated reasoning (TIR) augments language models (LMs) with external computation. This allows them to execute commands and invoke tools rather than relying solely on their own architecture and weights. Reinforcement learning (RL) is essential for inducing this behavior. Across calculators, code interpreters, and search engines, RL post-training reliably teaches models when and how to call a tool without explicit demonstrations (Li et al., 2025; Singh et al., 2025; Mai et al., 2025; Jin et al., 2025). However, a persistent failure mode shadows this success. Outcome-only RL with tools tends to induce cognitive offloading, where the model learns to fully delegate computation to the tool instead of internalizing the underlying skill, and its capability collapses fully once the tool is removed (Zeng et al., 2026). If tools are guaranteed at deployment, this does not pose an issue.

Problems arise, however, when resilience to tool absence is needed, or when the training goal is to abstract the tool away, i.e., using it temporarily so the model learns the skill on its own.

This tension is perhaps most acute for small models. The smaller the policy, the weaker its latent capability prior, and the stronger the incentive to offload to a tool that is reliably correct (Gandhi et al., 2025). Yet the existing remedies for tool overuse—AdaTIR’s difficulty-aware efficiency rewards (Fang and Sun, 2026), OTC’s penalty-based shaping (Wang et al., 2025), and TInR’s bidirectional knowledge alignment (Xu et al., 2026)—all operate on billion-parameter scales, and require auxiliary trace generation or other forms of non-trivial reward redesign. None of these, to the best of our knowledge, target small models, where the offloading pressure is strongest, and the question of whether the skill can be internalized at all is open.

We propose a simple RL intervention for small models: *annealed tool access* during RLOO training. The policy starts with full access to the tool (in our case, calculator), where sparse outcome rewards would otherwise provide practically no learning signal. Tool availability is then gradually withdrawn during training, forcing the policy to reproduce its successful, tool-aided rollouts internally. We study this on the Countdown task using the Qwen2.5-0.5B (Pan et al., 2025) model, aiming to improve no-tool evaluation over both a standard no-tool RLOO baseline and a fixed-tool TIR baseline. We expect this to invoke the amplifier-of-priors dynamic, as described by Gandhi et al. (2025): by the time the tool is removed, the policy has gathered enough successful rollouts to build an arithmetic prior that pure no-tool RLOO may never reach. Unlike prior remedies, annealing requires no reward shaping or additional supervision; it is simply a curriculum over the action space available to the policy.

Our contributions are threefold:

- **Annealed Tool Access as a Tool-internalization Curriculum.** We introduce a tool-availability schedule for RLOO that withdraws a tool over training and integrates an inline calculator into the vLLM rollout with proper tool-output masking (Section 3). To our knowledge, progressively withdrawing access to tools has not been studied in the existing TIR literature.
- **Evidence that Annealing Prevents Dispatch Collapse at Small Scale.** We present evidence that progressive tool removal prevents the usual performance collapse when the model is evaluated without access to tools, especially for small models. On the Countdown arithmetic task with Qwen2.5-0.5B, training with constant tool access results in the model performing poorly when the tool is removed. With our annealing approach, most gains from tool use are retained and this drop is prevented. Reward continues to increase after tool access ends, suggesting that the model is internalizing the skill rather than just finding new ways to invoke the tool.
- **Scoping the Claim via a Forgetting Control.** To clarify what our approach can and cannot solve, we compare against a control in which the model is only fine-tuned via SFT and not further improved with RL. We observe slightly lower performance (approx. 0.18 in pass@1) relative to this baseline, which we attribute to catastrophic forgetting during SFT. This highlights that annealing addresses tool dependence but does not resolve forgetting from earlier training steps.

2 Related Work

TIR and the Dispatch Failure Mode. RL-based TIR has been adopted for calculators, coding, and search (Li et al., 2025; Singh et al., 2025; Jin et al., 2025; Dong et al., 2025). ZeroTIR (Mai et al., 2025) shows that tool use can emerge from RL alone, without explicit demonstrations. Zeng et al. (2026) characterize tool overuse as a side effect of outcome-only RLVR: they observe monotonically increasing tool calling even when the correct answer could be easily reached internally by the model; this holds across various tested configurations. This dispatch tendency (and, notably, its consequence, capability collapse when the tool is removed) is the failure mode our intervention targets.

Tool Internalization. Huang et al. (2025) use SFT on back-translated traces to train competitive small models for math without tools. AdaTIR (Fang and Sun, 2026) introduces difficulty-aware rewards that tighten tool access; they report a 4.8% no-tool accuracy gain on AIME 2024 with

Qwen-2.5-1.5B. OTC (Wang et al., 2025) minimizes dependency on tools via penalty-based reward shaping, and TInR (Xu et al., 2026) aligns tool-using and tool-free reasoning bidirectionally. As noted previously, these methods operate at a billion-parameter scale and require auxiliary trace generation or a significantly redesigned reward. Our annealing approach is complementary and strictly on the side of the RL: it requires only a curriculum over the action space available to the policy, i.e., no reward shaping or additional supervision.

Curriculum and Access-schedule Analogs. Action-space curricula (Narvekar et al., 2020) have appeared sparingly in LM post-training (Parashar et al., 2025). Tool-availability schedules in particular, i.e., progressively *withdrawing* rather than introducing a tool, do not appear, to the best of our knowledge, in the TIR literature. The closest analog is OTC’s penalty on tool calls, but there the tool remains available throughout training. We, instead, modulate its availability directly.

Countdown and the Amplifier Hypothesis. Gandhi et al. (2025) argue that RL amplifies priors latent in the base model rather than inducing genuinely new capabilities. This framing makes our 0.5B regime demanding: with a weak prior, we can expect that RL with a calculator has little internal computation to amplify. Annealed tool access is designed to address this asymmetry: using the tool to scaffold high-reward rollouts early in training, when outcome-only signals would otherwise be sparse, and then progressively withdrawing it so the policy must reproduce those successful runs on its own.

3 Method

Task. We study the Countdown task, where the goal is for the policy to connect provided integers with arithmetic operations (+, −, ×, ÷), using each number once, to reach a target value. We optimize Qwen2.5-0.5B with RLOO (Ahmadian et al., 2024). The reward function is set up as follows: 1.0 for a correctly formatted `<answer>` block that satisfies the task, 0.1 for a correctly formatted `<answer>` block but incorrect answer, and 0 otherwise. The RLOO configuration is consistent across all conditions; we vary whether and when the calculator is available/annealed (see Appendix B).

Inline Calculator Integration. We extend the basic RLOO rollout with an inline calculator tool. The policy can invoke the tool by writing `<calc>expression</calc>`. The vLLM worker effectively treats `</calc>` as a stop string. At this point, it evaluates the expression, and injects `<result>value</result>` back into the context before resuming LLM generation. Per standard TIR practice (Jin et al., 2025; Li et al., 2025), the tokens of the injected `<result>` are treated as tool output, and are hence masked out of the policy gradient. The importance weighting, leave-one-out advantage, and KL term are left unchanged (see Appendix D).

Annealed Tool Access. Our intervention is effectively a probability $p_{\text{tool}}(t)$ that the calculator tool is enabled for a given rollout at training step t . When the tool is disabled for a rollout, any `<calc>` tags are passed through as regular tokens without the vLLM execution feedback described above, and so the policy receives reward only for answers it can produce on its own, directly from its weights. We adopt a linear schedule,

$$p_{\text{tool}}(t) = \max\left(0, 1 - \frac{t}{T_{\text{anneal}}}\right), \quad T_{\text{anneal}} = \frac{2}{3}T,$$

such that the calculator is available at the start, decays linearly, and becomes completely withdrawn for the final third of the training run. The intuition for this design is the following: early rollouts lean on the tool to reach high-reward states that sparse outcome rewards alone could not discover, and the no-tool tail forces the policy to internalize whatever arithmetic carried those rollouts. We also implement step and logarithmic schedules (see Appendix E), but leave their comparison to future work.

Conditions and Hypotheses. From a shared SFT warm start (Section 4), we train four RLOO conditions: (a) a no-tool baseline ($p_{\text{tool}} \equiv 0$), (b) fixed tool access ($p_{\text{tool}} \equiv 1$, the standard TIR baseline), (c) annealed access, and (d) the fixed-tool checkpoint evaluated without the tool. The primary comparison we concern ourselves with is (c) vs. (a), under no-tool evaluation. We hypothesize that (d) < (a), i.e., fixed-tool training collapses when the tool is withdrawn (**H1**); that (c)

> (a) on no-tool evaluation (**H2**); and that (c) approaches but does not match (b) under tool-enabled evaluation (**H3**).

4 Experimental Setup

Model, Data, and Training Procedure. All policies in our experiments are Qwen2.5-0.5B models, trained and evaluated on the Countdown 3-4 task split (asingsh15/countdown_tasks_3to4). The RLOO runs share the same configuration: group size $G=8$, 100 training steps, KL against a fixed reference, and the tool-output mask of Section 3; the conditions differ only in the tool availability (constant_off for no-tool, constant_on for fixed-tool, and linear for annealed). For the complete set of SFT and RLOO hyperparameters, see Appendix B; the reward and dispatch protocols are outlined in Appendix D. Each run was executed on a single NVIDIA H100 (via Modal) approx 4.5 hours.

Evaluation Protocol. We report pass@ k for $k \in \{1, 8\}$ on a 50-prompt held-out Countdown split, sampling $n=16$ rollouts per prompt at $T=0.6$, top- $p=0.95$. We evaluate each condition under two regimes: a *no-tool* regime, where the calculator tool is disabled, and a *tool* regime, where the calculator tool is available via inline `<calc>` dispatch, allowing us to distinguish behavioral collapse from capability collapse. Reported \pm values are 95% bootstrap CI half-widths over the 50 evaluation prompts and capture evaluation-prompt variance only. Each condition is a single RL training seed. We additionally track the tool-reliance rate (calls per rollout and the fraction of rollouts emitting `<calc>`) throughout training and at evaluation time.

Bootstrapping Tool Use via SFT. A prerequisite for any of the four RL conditions to be meaningful is that the base policy actually emits well-formed `<calc>` dispatches at non-trivial rates — if it never calls the tool, fixed-tool and annealed RLOO degenerate into the no-tool baseline. One problem we ran into early into experimentation was the provided Qwen2.5-0.5B SFT checkpoint (asingsh15/qwen-sft-countdown-defaultproj) did not emit any `<calc>` dispatches under both the zero-shot and 2-shot calculator preambles (§D). To get around prompt-only induction being insufficient, we employed an additional SFT pass on the initial Countdown checkpoint (asingsh15/qwen-sft-countdown-defaultproj), built by a rejection-sampling pipeline. We generated two different datasets - SFT-A and SFT-B. *SFT-A* is self-distilled from the provided 0.5B SFT checkpoint. However, because that 0.5B model itself invokes the calculator on only $\sim 1.5\%$ of rollouts, the resulting 883-example dataset bakes in tool *syntax* but not strong tool-call *habits*, and at eval time the average calls per rollout stays below 1 across all SFT-A RL conditions even when the calculator is offered. We found this to be ultimately ineffective in providing enough structure for the model to reliably call the `<calc>` tool in the subsequent RL environment, so we decided to use SFT-B for our experiments instead. *SFT-B*, distilled from the $14\times$ -larger Qwen2.5-7B-Instruct teacher ($\sim 47\%$ tool-use rate, 5,000 accepted rollouts), carries a strong enough tool-dispatch prior into RL for the cognitive offloading phenomenon to be observable. The rejection-sampling pipeline and more elaboration on both datasets is described in full in Appendix C.

5 Results

From our experiments, we find support for three key takeaways. First, as predicted by Zeng et al. (2026), fixed-tool RL on a strong tool-dispatch prior results in a cognitive offloading failure mode when the calculator is withdrawn at evaluation time (Section 5.1.1). Second, our annealing RL curriculum eliminates this cognitive offloading collapse while retaining most of the knowledge on tool-use in calculator-enabled evaluation environments. The training dynamics curves show reward continuing to climb even *after* the calculator tool is fully withdrawn, implying that the model has internalized core arithmetic principles (Section 5.1.2). Third, we find that the entire SFT-B finetuned family of models substantially underperform their no-SFT reference baselines, which we attribute to catastrophic forgetting from the provided Countdown SFT prior (asingsh15/qwen-sft-countdown-defaultproj) during our distillation finetuning pass, which helps us better scope our claims with the impact of annealing (Section 5.1.3).

Table 1: **Pass@k by condition (SFT-B)**. Bold marks the best value per column among conditions (a)–(c). Condition (e) is a no-SFT reference baseline.

Condition	No-tool eval		Tool eval	
	pass@1	pass@8	pass@1	pass@8
(a) No-tool RLOO	0.368 \pm 0.09	0.602 \pm 0.13	0.484 \pm 0.12	0.615 \pm 0.13
(b) Fixed-tool (TIR)	0.036 \pm 0.02	0.212 \pm 0.09	0.543 \pm 0.12	0.721 \pm 0.11
(c) Annealed	0.368 \pm 0.09	0.650 \pm 0.12	0.488 \pm 0.12	0.643 \pm 0.12
(e) No SFT, no-tool RLOO	0.553 \pm 0.11	0.729 \pm 0.12	0.588 \pm 0.12	0.724 \pm 0.12

50 eval prompts, $n=16$ rollouts each; \pm = 95% bootstrap CI half-width over prompts. Single seed per condition.

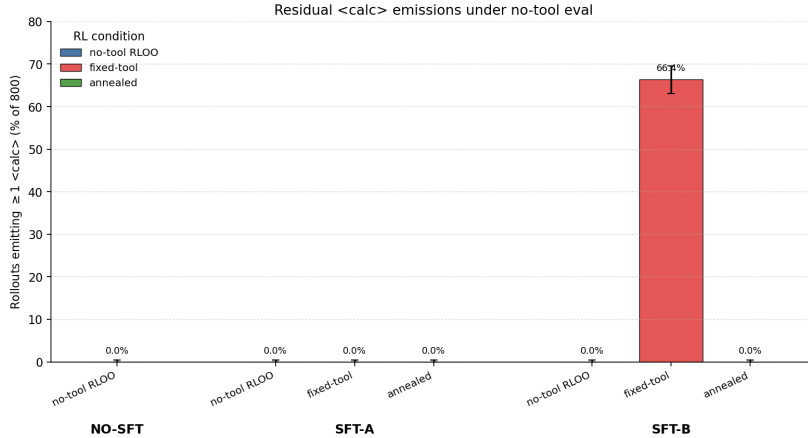


Figure 1: **Residual <calc> emission under no-tool eval (SFT-B)**. Fraction of rollouts (of 800 per condition: 50 prompts \times 16 samples) that emit at least one <calc> token when the calculator server is disabled. Error bars are 95% Wilson intervals. Only the fixed-tool RLOO checkpoint exhibits the excessive & inappropriate dispatch behavior; the no-tool and annealed checkpoints are at 0/800.

5.1 Quantitative Evaluation

For our quantitative evaluation, we report $pass@k$ at $k \in \{1, 8\}$ across 50 held-out Countdown prompts ($n=16$ rollouts per prompt, $T=0.6$, top- $p=0.95$). Each condition gets evaluated under two regimes: *no-tool* (verifier only, calculator server disabled) and *tool* (calculator available via inline <calc> dispatch). Reported \pm values are 95% Confidence Interval (CI) half-widths over the 50 Countdown Eval prompts, and capture eval-prompt variance only. each condition is a single RL training seed.

5.1.1 Finding 1: Fixed-tool TIR collapses without the tool

Row (b) under the no-tool eval column in Table 1 paints a clear picture: the fixed-tool checkpoint scores $p@1 = 0.036 \pm 0.02$, an *order of magnitude* below the no-tool and annealing RLOO conditions at 0.368 ± 0.09 . The fixed-tool checkpoint also scores 0.543 ± 0.12 on $p@1$ when the calculator is *available*, showing that the fixed-tool trained model has not forgotten how to solve Countdown, but instead has forgotten how to solve Countdown without dispatching to a tool, clearly exhibiting the cognitive offloading failure mode predicted by Zeng et al. (2026).

Figure 1 confirms this is a behavioral collapse, not a capability one. Under the no-tool eval, the SFT-B fixed-tool checkpoint *still tries to emit* at least one <calc> token in roughly two-thirds of rollouts; every other condition (including SFT-B annealed and SFT-B no-tool RLOO) emits zero. The fixed-tool policy is continuously trying to call a tool that the eval harness has explicitly disabled, waiting for a <result> injection that never arrives. §5.2 walks through the trajectory-level structure of this failure.

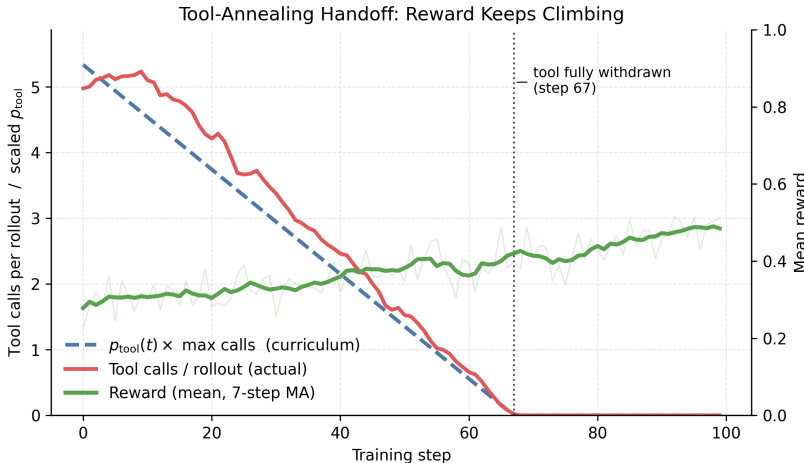


Figure 2: **Tool-annealing handoff (SFT-B annealed, single seed)**. Dashed blue: scheduled $p_{\text{tool}}(t) \cdot \text{max_calls}$. Red: empirical tool calls per rollout. Green: 7-step moving-average mean reward (right axis). The calculator is fully withdrawn at step 67; mean reward continues to climb through and past withdrawal (0.420 \rightarrow 0.485), a gain that is necessarily tool-free.

5.1.2 Finding 2: Annealing internalizes arithmetic as the tool is withdrawn

The annealing schedule was designed to give the policy enough successful tool-aided rollouts early in training to build an arithmetic prior, then withdraw the tool gradually so that prior is forced to carry the rollout on its own. Figure 2 is the cleanest evidence we have that this mechanism actually helps the model internalize its own arithmetic prior. This is most clearly demonstrated by the model’s ability to continue accruing more reward even after tool withdrawal at step 67 of 100. Mean reward at withdrawal is 0.420; by step 99, it has continued to climb to 0.485, a gain of **+0.065 even after the tool is gone**. Because the tool is unavailable across this entire window, the gain is necessarily tool-free: the policy is learning to solve Countdown internally during exactly the phase the curriculum was designed to force it to.

Table 1 shows the effects of this handoff at eval-time. Annealing matches no-tool RLOO on p@1 (0.368) and edges it on p@8 (0.650 vs. 0.602). Since the p@8 gap between no-tool RLOO and annealing-tool RLOO has overlapping CIs at $n=50$, so we frame it as suggestive of internalization rather than significant. Importantly, annealing retains most of the fixed-tool’s tool-eval gain (p@8 = 0.643 vs. 0.721) while completely eliminating the no-tool collapse (0.650 vs. 0.212). The behavioral signal in Figure 1 also provides strong evidence for internalization: the annealed checkpoint emits zero residual `<calc>` tokens under no-tool eval, meaning it has not only learned to score well without the tool, it has also stopped *reaching* for the tool. In sum, the annealing curriculum gets us the tool-trained model’s tool-eval performance and the no-tool-trained model’s robustness in one run.

5.1.3 Finding 3: SFT-B trails the no-SFT reference

Every SFT-B condition in Table 1 trails the no-SFT reference baseline (e) by roughly 0.18 on p@1 — a gap larger than any annealing effect within the SFT-B arm. We posit this to catastrophic forgetting during the SFT distillation phase, rather than the annealing curriculum. The provided checkpoint (asinh15/qwen-sft-countdown-defaultproj) was already SFT-tuned on Countdown; our SFT-B initialization applies a second 6-epoch pass over 4,900 examples distilled from Qwen2.5-7B-Instruct, a 14 \times -larger teacher whose reasoning style diverges substantially (longer `<think>` chains, multi-turn `<calc>` tool calls) from the base model’s distribution. We read the resulting gap as catastrophic forgetting: the distillation pass likely overwrote much of the Countdown skill the base checkpoint already had, and the 100 steps of RLOO were not enough to recover it. Figure 2 is consistent with this — the SFT-B annealed reward is still climbing at step 99, indicating the arm entered RL from a worse starting point and ran out of training steps rather than hitting a performance ceiling.

```

Prompt: nums=[49, 41, 73], target=17 Score: 0.0 <calc> openings: 135
<think>First, let's consider the numbers 49, 41, and 73. We need to use these numbers to get 17.
Let's start by trying to subtract or divide to get a smaller number. </think>
<think>Let's try 73 - 49 - 41. </think>
<calc>73 - 49 - 41</calc>
<calc>17</calc> <calc>17</calc> <calc>17</calc> <calc>17</calc> <calc>17</calc> <calc>17</calc>
... [<calc>17</calc> repeats ≈130 more times until max_tokens] ...

No <answer> block is ever emitted. The first <calc> call identifies the correct value (17), but the
policy never closes the trajectory with an <answer> tag, so the verifier scores it 0.

```

Figure 3: **Dispatch collapse (SFT-B fixed-tool, no-tool eval).** The policy loops the same fragment waiting for a <result> injection that the disabled tool server never produces. This is the trajectory-level structure of the 66% residual-emission rate in Figure 1 and the p@1=0.036 entry in Table 1 row (b).

This allows us to scope the annealing claim: annealing reliably protects against the cognitive offloading failure mode within the SFT-B finetuned models (Sections 5.1.1 and §5.1.2), but cannot recover the Countdown skill lost during a prior SFT pass. This provides an interesting avenue for future work: a useful next step would be experimenting with other (perhaps more lightweight) finetuning configurations like LoRA or mixed trace schedules that mixes Qwen-7B traces with the original base-checkpoint completions, preserving the prior while still seeding tool use.

Limitations: All RL conditions are single-seed; the reported CIs reflect eval-prompt variance only. We test only the linear p_{tool} schedule; step and logarithmic variants are noted as future work. The 50-prompt test split limits CI tightness; the p@8 annealing gain over no-tool RLOO is suggestive but not significant at this n .

5.2 Qualitative Analysis

The quantitative section established that fixed-tool RLOO collapses under no-tool eval and that annealing helps eliminates the collapse. This section shows what each regime looks like at the trajectory level.

Dispatch collapse (Figure 3). This figure clearly shows the failure mode of tool dispatch collapse in the SFT-B fixed-tool / no-tool eval condition. Given (nums=[49, 41, 73], target=17), the policy tries to offload the computation with the <calc> tool, and when the environment never emits a <result> block, the policy collapses into continuously trying to offload the task into <calc> tool calls 135 times until max_tokens is reached. The verifier requires a well-formed <answer> tag, so the rollout scores 0. This is a clear instantiation of the cognitive offloading failure mode: the policy has internalized the tool-dispatch behavior so deeply that it cannot complete the task when the tool is taken away at eval time. The 66% residual-emission rate in Figure 1 reflects this same behavior extrapolated across the rest of the evaluation.

Internalized Arithmetic Control (Figure 4). The same SFT-B initialization, trained the Annealed RL Curriculum, solves the same Countdown prompt (nums=[44, 19, 35], target=98) in plain text with zero <calc> openings and a verifier score of 1.0. The model reasons inside <think> tags, explores three different paths of reasoning, and is able to land on the expression $44 + 19 + 35 = 98$, emitting a valid <answer> block. Holding the SFT prior and the no-tool eval regime fixed and varying only the RL recipe isolates the calculator dispatch failure to fixed-tool RL itself: the SFT-B prior is not the problem, the no-tool eval regime is not the problem, the fixed-tool RL recipe is.

6 Discussion

The three findings described above give a coherent account of where annealed tool access is suitable. By contrasting the (b) and (c) conditions, we find that holding the SFT prior plus the no-tool eval regime fixed, while varying only the RL recipe, brings about dispatch collapse (full tool dependence) only in the fixed-tool RL setting. Fixed-tool RLOO amplifies the SFT-B dispatch prior into a policy that cannot complete the Countdown task without the calculator (Section 5.1.1), as suggested by Zeng

```

Prompt: nums=[44, 19, 35], target=98 Score: 1.0 <calc> openings: 0
<think>First, I will try to use multiplication or division to get close to 98. Let's start with 44 * 19. </think>
<think>44 * 19 = 876. This is too large, so let's try 44 * 35. </think>
<think>44 * 35 = 1540, which is also too large. Let's try 19 * 35. </think>
<think>19 * 35 = 665, which is still too large. Now, let's try 44 + 19 + 35. </think>
<think>44 + 19 + 35 = 98. This equals 98, so the equation is correct. </think>
<answer> 44 + 19 + 35 </answer>

```

Figure 4: **Internalized arithmetic under annealing (SFT-B annealed RLOO, no-tool eval)**. Same SFT prior and same prompt as Figure 3, different RL recipe with tool linearly annealed. the model explores three multiplicative paths, rejects each as too large, and lands on $44 + 19 + 35 = 98$ without emitting a single `<calc>` token.

et al. (2026). Annealing on the same prior prevents the collapse while keeping most of the tool evaluation gain (Section 5.1.2). Put differently, annealing helps the model use the tool as a scaffold to be used early and later discarded, as opposed to a consistent crutch.

To understand whether the observed behavior can be attributed to genuine internalization or re-routing of the same behavior, we look to the training dynamics and note that the mean reward keeps climbing even after the calculator is fully withdrawn at step 67 ($0.42 \rightarrow 0.49$). This gain is realized in a necessarily tool-free setting and, what’s more, the annealed checkpoint produces zero residual `<calc>` tokens under no-tool evaluation, meaning it has stopped reaching for the tool. This is the phenomenon described by Gandhi et al. (2025), but induced deliberately: the tool manufactures the high-reward rollouts that a weaker prior simply could not reach on its own, and the no-tool tail of the rollout forces the policy to amplify whatever arithmetic those rollouts contained.

Finally, we turn to the observed catastrophic forgetting result (see Section 5.1.3). All fine-tuned conditions trail the no-SFT baseline by approx. 0.18 (pass@1). Annealing protects against offloading within the given prior, but it cannot recover or reinforce capability that a prior SFT pass destroyed through catastrophic forgetting. The practical lesson here is that the same SFT pass that instills the tool-dispatch prior our annealing builds on can simultaneously erode the base skill that annealing seeks to surface. Lighter bootstrapping (e.g., LoRA) would be the main avenue for future work.

Limitations. Our results are based on experiments conducted with a single seed on a single 50-prompt split with only the linear schedule.

7 Conclusion

We asked whether a small policy can be made to internalize arithmetic rather than offload it to a calculator tool, and find that a simple annealing curriculum is enough. Annealing the calculator’s availability over the course of RLOO training prevents the model from the cognitive offloading failure that fixed-tool RL otherwise induces (see Figure 3), recovering no-tool robustness while retaining most of the tool-enabled gain. Through our exploration, we find that forcing a small model to use a tool it didn’t acquire in pretraining is nontrivial. Few-shot prompting on its own is insufficient, and while SFT is effective for inducing the tool calling behavior, it must be done carefully, since it can also cause catastrophic forgetting that confounds the benefits of annealing. We see prior-preserving tool bootstrapping combined with availability annealing as a promising and lightweight recipe for tool internalization at small scale.

8 Team Contributions

- **Group Member 1:** Jason created the SFT A/B datasets, finetuned Qwen 2.5 0.5B on both A + B and ran RLOO post-training on all RL Curriculums (No-Tool, Fixed-Tool, Annealing-Tool) on both checkpoints, evaluated both models on countdown eval, and put together much of the Results section in the Final paper.
- **Group Member 2:** Maty scoped out the experimental design, motivation, wrote the project proposal, implemented the calculator tooling and curriculums for calculator annealing,

created and filled out much of the poster. He also wrote the Introduction, Related Work, Methods, and Discussion sections.

Changes from Proposal The biggest change from the Proposal was the necessity of adding SFT from a Larger Teacher Model (Qwen 2.5 7B) in order to instill proper tool-calling functionality within the Qwen 2.5 0.5B model.

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 12248–12267.
- Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-star: Empowering LLM-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410* (2025).
- Zhaiyu Fang and Ruipeng Sun. 2026. AdaTIR: Adaptive Tool-Integrated Reasoning via Difficulty-Aware Policy Optimization. *arXiv preprint arXiv:2601.14696* (2026).
- Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307* (2025).
- Xingyue Huang, Xianglong Hu, Zifeng Ding, Yuan He, Waleed Alzarooni, Ziyu Ye, Wendong Fan, Bailan He, Haige Bo, Changran Hu, et al. 2025. Distilling Tool Knowledge into Language Models via Back-Translated Traces. *arXiv preprint arXiv:2506.19171* (2025).
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Serkan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516* (2025).
- Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. ToRL: Scaling tool-integrated RL. *arXiv preprint arXiv:2503.23383* (2025).
- Xinji Mai, Haotian Xu, Zhong-Zhi Li, Weinong Wang, Jian Hu, Yingying Zhang, Wenqiang Zhang, et al. 2025. Agent RL scaling law: Agent RL with spontaneous code execution for mathematical problem solving. *arXiv preprint arXiv:2505.07773* (2025).
- Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research* 21, 181 (2020), 1–50.
- 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.
- Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, et al. 2025. Curriculum reinforcement learning from easy to hard tasks improves LLM reasoning. *arXiv preprint arXiv:2506.06632* (2025).
- Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. Agentic reasoning and tool integration for LLMs via reinforcement learning. *arXiv preprint arXiv:2505.01441* (2025).
- Hongru Wang, Cheng Qian, Wanjun Zhong, Xiushi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. OTC: Optimal tool calls via reinforcement learning. *arXiv e-prints* (2025), arXiv–2504.
- Qiancheng Xu, Yongqi Li, Fan Liu, Hongru Wang, Min Yang, and Wenjie Li. 2026. TInR: Exploring Tool-Internalized Reasoning in Large Language Models. *arXiv preprint arXiv:2604.10788* (2026).
- Yirong Zeng, Shen You, Yufei Liu, Qun Yao Du, Xiao Ding, Yutai Hou, Yuxian Wang, Wu Ning, Haonan Song, Dandan Tu, et al. 2026. The Tool-Overuse Illusion: Why Does LLM Prefer External Tools over Internal Knowledge? *arXiv preprint arXiv:2604.19749* (2026).

A Full Pass@k (SFT A + B) by Condition

Table 2 reports pass@k for every condition across both SFT arms (SFT-A and SFT-B) and the no-SFT reference, under both no-tool and tool-enabled evaluation.

We uncover two additional findings to note beyond what was covered in the SFT-B results in §5.1. First, the SFT-A row is statistically flat across the three RL conditions: the 95% bootstrap CIs overlap completely on all four metrics, and the mean calls per rollout for SFT-A under tool eval never exceeds 0.22 (§C). This adds quantitative evidence that the SFT-A self-distillation approach that yielded ~1.5% tool-use rate at distillation time produced an SFT prior too weak for dispatch collapse to occur, so annealing had nothing to fix. Second, the no-SFT reference baseline (e) outperforms the SFT-B condition in absolute terms by ≈ 0.18 on pass@1. We attribute this to catastrophic forgetting during the SFT distillation pass and discuss it as a scoping condition for the annealing claim in §5.1.3.

Table 2: **Pass@k by condition across all SFT arms.** For each arm, bold marks the best value per column among the three RL conditions. The no-SFT row is shown as a reference baseline.

Condition	No-tool eval		Tool eval	
	p@1	p@8	p@1	p@8
<i>SFT-A initialization (self-distilled from 0.5B SFT teacher)</i>				
(a) No-tool RLOO	0.551 ± 0.11	0.723 ± 0.11	0.576 ± 0.12	0.701 ± 0.12
(b) Fixed-tool (TIR)	0.579 ± 0.11	0.718 ± 0.12	0.599 ± 0.12	0.705 ± 0.12
(c) Annealed	0.560 ± 0.11	0.723 ± 0.12	0.603 ± 0.11	0.721 ± 0.12
<i>SFT-B initialization (distilled from Qwen2.5-7B-Instruct)</i>				
(a) No-tool RLOO	0.368 ± 0.09	0.602 ± 0.13	0.484 ± 0.12	0.615 ± 0.13
(b) Fixed-tool (TIR)	0.036 ± 0.02	0.212 ± 0.09	0.543 ± 0.12	0.721 ± 0.11
(c) Annealed	0.368 ± 0.09	0.650 ± 0.12	0.488 ± 0.12	0.643 ± 0.12
<i>(e) No SFT, no-tool RLOO</i>	0.553 ± 0.11	0.729 ± 0.12	0.588 ± 0.12	0.724 ± 0.12

Sampled across 50 eval prompts in `countdown_eval` with default sampling hyperparameters ($T=0.6$, $top-p=0.95$, $top-k=20$, $n=16$ rollouts/prompt). Each condition is a single RL training seed; \pm shows the 95% bootstrap CI half-width over the 50 eval prompts (2000 resamples), capturing eval-prompt variance only, not training-run variance.

B Hyperparameters and Implementation Details

This section lists the exact hyperparameters used for all training stages. Every condition shares the same RLOO configuration except for tool-related flags; the only differences across conditions (a)/(b)/(c) are the `tool_anneal_schedule` and `calculator_prompt_mode` settings (Table 3, bottom block).

B.1 Supervised Fine-Tuning (SFT-A and SFT-B)

Both SFT checkpoints are trained from the Qwen2.5-0.5B base model with identical hyperparameters; the only difference between SFT-A and SFT-B is the training dataset (see §C).

Hyperparameter	Value
Base model	Qwen2.5-0.5B
Learning rate	5×10^{-5}
Batch size	64 (gradient accumulation 8)
Epochs	6
Weight decay	0.01
Warmup ratio	0.05
Gradient clip	1.0
Max prompt length	512 tokens
Max response length	1024 tokens
Launcher	sft_trainer/train_sft_modal.sh

B.2 RLOO

All RL runs in this paper (SFT-A \times {a,b,c}, SFT-B \times {a,b,c}, no-SFT \times {a}) use the hyperparameter configuration in Table 3. Group-size $G=8$ gives the LOO baseline, KL is computed against a fixed reference (the initial policy at step 0), and the tool output (i.e., text inside server-injected `<result>` tags) is masked out of the policy gradient via the tool-output mask described in §D.

B.3 Evaluation

All $\text{pass}@k$ numbers reported in §5.1 and Table 2 use the same vLLM decoding configuration: $T=0.6$, $\text{top-}p=0.95$, $\text{top-}k=20$, $\text{min-}p=0$, $n=16$ rollouts per prompt, evaluated on the 50-prompt held-out Countdown test split. The tool-enabled regime adds `-use_calculator` and `-calculator_prompt_mode when_available` to the eval command (see §E); the no-tool regime disables the calculator server entirely.

C SFT Dataset Construction

Both SFT datasets are built by the same rejection-sampling pipeline. For each prompt in `asingh15/countdown_tasks_3to4`, we sample $G=8$ rollouts from a teacher policy using the inline-calculator tool loop with the 2-shot calculator preamble (Section D) prepended in `calculator_prompt_mode = "always"`. A rollout is accepted if it contains at least one well-formed `<calc>...</calc>` span paired with an environment-injected `<result>...</result>`, has no `<result>ERR</result>`, and is scored 1.0 by the verifier `evaluation/countdown.py::compute_score`. Teacher decoding uses $T=0.6$, $\text{top-}p=0.95$, $\text{top-}k=20$, max tokens 1024, and `max_tool_calls = 8`.

SFT-A is self-distilled from the provided Countdown SFT checkpoint (`asingh15/qwen-sft-countdown-defaultproj`). We ran rejection-sampling on 12,800 source prompts, which yielded 883 accepted rollouts (drawn from 780 unique source prompts; the per-prompt acceptance cap was 3), split 865/18 into train and test. We find, however, that the 0.5B SFT teacher was not very effective at embedding proper tool calling behavior into its rollouts. The teacher invoked the calculator on only $\sim 1.5\%$ of rollouts even under 2-shot prompting, and at eval time, the mean calls per rollout stays below 1 across all SFT-A RL conditions even when the calculator is offered. This underpronounced behavior led us to pursue expert teacher distillation instead of self distillation for our experiments.

SFT-B was distilled from the much larger Qwen/Qwen2.5-7B-Instruct under the same pipeline and the same decoding hyperparameters. The run saw 10,080 source prompts before halting at its global cap of 5,000 accepted rollouts (one accepted rollout per source prompt), split 4,900/100 into train and test. The 7B teacher invoked the calculator on $\sim 47\%$ of rollouts under the same 2-shot preamble (around $32\times$ SFT-A’s tool-use rate). This gap made SFT-B viable for the dispatch-collapse experiments. SFT-B finetuned RLOO models averaged around 5-6 `<calc>` calls when given access to the tool, allowing the models to carry a strong tool-dispatch prior, allowing us to test the effect of tool-internalization under different RLOO circumstances much more clearly.

Table 3: **RLOO hyperparameters**. Shared across all conditions except the bottom (annealing) block, which is condition-specific.

Hyperparameter	Value
<i>Optimization</i>	
Learning rate	1×10^{-5} (constant, no warmup)
Weight decay	1×10^{-4}
Gradient clip	none
Batch size	128 prompts (gradient accumulation 128)
Group size G	8 (RLOO leave-one-out baseline)
KL coefficient	1×10^{-3} (vs. fixed reference)
Entropy coefficient	1×10^{-3}
Training steps	100 (checkpoint every 10)
<i>Training-time sampling (vLLM)</i>	
Temperature	1.0
top- p / top- k	1.0 / -1 (disabled)
min- p	0.0 (disabled)
Max model length	2048
Max tokens / rollout	1024
GPU memory utilization	0.9
Other	chunked prefill enabled
<i>Tool integration (conditions (b) and (c))</i>	
max_tool_calls	8
calculator_prompt_mode	when_available
tool_anneal_schedule	linear (condition (c) only)
tool_anneal_ratio	0.667
<i>Reward</i>	
Verifier	evaluation/countdown.py::compute_score
Reward values	0.0 (no answer) / 0.1 (format only) / 1.0 (correct)
<i>Hardware</i>	
GPU	1 \times NVIDIA H100 (Modal)
Wallclock	\sim 4–4.5h per RLOO run
Launcher	rloo_trainer/train_rloo_modal.sh

D Calculator Prompts and Tool-Dispatch Protocol

This section documents the exact chat-template-formatted prompts used for both training and evaluation, the three calculator-prompt modes, and the inline `<calc>/<result>` dispatch contract.

D.1 No-Calculator Prompt

```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
A conversation between User and Assistant. The user asks a question, and
the Assistant solves it. The assistant first thinks about the reasoning
process in the mind and then provides the user with the answer.
User: Using the numbers [ 4 73 97 18], create an equation that equals 10.
You can use basic arithmetic operations (+, -, *, /) and each number can
only be used once. Show your work in <think> </think> tags. And return the
final answer in <answer> </answer> tags, for example
<answer> (1 + 2) / 3 </answer>.
Assistant: Let me solve this step by step.<|im_end|>
```

```
<|im_start|>assistant
```

The verifier (`evaluation/countdown.py::compute_score`) parses the contents of the final `<answer>...</answer>` and assigns 1.0 if the expression uses each input number exactly once and it evaluates to the target, 0.1 for parse-only success, and 0.0 otherwise.

D.2 Calculator Prompt (2-Shot)

When the calculator is available, the system message is replaced with the 2-shot prompt. The user turn is identical to the no-calculator prompt.

```
<|im_start|>system
You are a helpful assistant.
```

You may use an inline calculator for arithmetic. To call it, write `<calc>arithmetic expression</calc>`. When the calculator is available, the environment will append `<result>value</result>`, and you should continue solving from that result. Use only the numbers from the problem and basic arithmetic operations (+, -, *, /).

Here are two examples of using the calculator:

Example 1:

```
Using the numbers [4, 5, 2], create an equation that equals 22.
<think> Let me compute 4 * 5 + 2. <calc>4 * 5 + 2</calc><result>22</result>
That equals 22. </think>
<answer> 4 * 5 + 2 </answer>
```

Example 2:

```
Using the numbers [6, 3, 7], create an equation that equals 25.
<think> First compute 6 * 3. <calc>6 * 3</calc><result>18</result> Now add
7: <calc>18 + 7</calc><result>25</result> That equals 25. </think>
<answer> 6 * 3 + 7 </answer><|im_end|>
```

```
<|im_start|>user
```

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer.

```
User: Using the numbers [ 4 73 97 18], create an equation that equals 10.
You can use basic arithmetic operations (+, -, *, /) and each number can
only be used once. Show your work in <think> </think> tags. And return the
final answer in <answer> </answer> tags, for example
<answer> (1 + 2) / 3 </answer>.
```

```
Assistant: Let me solve this step by step.<|im_end|>
```

```
<|im_start|>assistant
```

E Reproducibility Pointers

Code

- RL orchestration: `rloo_trainer/rloo.py`
- Sampler / tool loop: `rloo_trainer/sampling_worker.py`, `rloo_trainer/tool_loop.py`, `rloo_trainer/calculator.py`
- Annealing schedules: `rloo_trainer/anneal.py` (linear, step, log, constant_on, constant_off)
- Result masking: `rloo_trainer/tokenize_utils.py::tool_output_mask`
- SFT training: `sft_trainer/train_sft.py`
- SFT dataset generation: `dataset_generation/rejection_sample_sft_a.py`

- Tool-aware eval: `evaluation/countdown_eval.py` (flags: `-use_calculator`, `-calculator_prompt_mode when_available`)
- Verifier: `evaluation/countdown.py::compute_score`
- Pass@ k + bootstrap CI analysis: `evaluation/view_passk.py` (CLI, writes `pass@k_summary.txt`), `evaluation/view_passk.ipynb` (interactive, writes `pass@k.png`)
- Modal launchers: `rloo_trainer/train_rloo_modal.sh`, `sft_trainer/train_sft_modal.sh`, `modal_train.py`

Models and data. The base model is Qwen2.5-0.5B. The SFT-B distillation teacher is Qwen2.5-7B-Instruct. The source Countdown task split, the provided SFT checkpoint used as the no-SFT reference baseline, and our two distilled SFT datasets (SFT-A and SFT-B) are all hosted on the HuggingFace Hub at (`minisounds/countdown-sft-b-distill-7b`, `minisounds/countdown-sft-a-tool-rs`).

Hardware. All RLOO runs executed on a single NVIDIA H100 via Modal; ~ 4 – 4.5 h per run.