Tool-Integrated RLOO and Pass@K: Testing the Invisible Leash

Isaiah Flores Mia Xiao Katherine Xu
Department of Computer Science, Stanford University
CS224R Final Project

Extended Abstract

Motivation and Problem Statement

Recent work on reinforcement learning with verifiable rewards (RLVR) [7] suggests that RL can improve how often a small language model samples solutions it already knows, but may not expand the model's underlying reasoning frontier. This phenomenon has been described as the **invisible leash** [7] : RL simply reshapes the output distribution without producing new reasoning capabilities beyond its known action space.

This project explores whether **tool use** can break that leash. Tools such as calculators, code interpreters, and retrieval systems expand the action space available to a language model. If a small model cannot internally perform some reasoning step, an external tool may offer a way around that limitation. We test this hypothesis in the controlled setting of Countdown arithmetic, where a model receives 3–4 integers and a target value and must produce an arithmetic expression that evaluates to the target. Our base model is Qwen 2.5-0.5B, initialized from a supervised Countdown model, and our tool is a calculator that evaluates model-generated arithmetic expressions.

Method and Novelty

We implement a tool-integrated RLOO pipeline for the Countdown task. The model emits a calculator call in the form `<calculator>EXPR</calculator>`. The rollout harness pauses generation, evaluates the expression `EXPR` using a safe arithmetic evaluator, injects either `<calc>RESULT</calc>` or `<calc_error>...</calc_error>`, and resumes generation. This creates a tool-integrated reasoning trajectory while keeping the calculator output environment-generated rather than model-generated.

Naive tool RL fails because a 0.5B model has little prior probability of emitting calculator tags, meaning that RL cannot reinforce useful tool behavior that is never sampled to begin with. We therefore introduce three interventions to address this issue. First, using Tool-SFT warm starts on correctly formatted calculator trajectories. Second, implementing multi-solution Tool-SFT, where each Countdown prompt is paired with multiple distinct correct expressions. And lastly, creating reward-shaped tool-integrated RLOO, with bonuses for tool format, executable expressions, valid number use, and matching the target number. We also implement a tool-aware pass@k evaluator, since a default text-only evaluator cannot correctly score tool trajectories without executing calculator calls.

Implementation and Headline Results

In the pure-text milestone, RLOO improves Pass@1 over SFT and IPO, but Pass@16 performance saturates near the underlying SFT/IPO frontier. This is consistent with the invisible leash hypothesis. In the tool-

integrated environment, Tool-SFT successfully teaches the model to call the calculator with tool-use rates reaching 1.00 and malformed calls disappearing, compared to the forced-prefix case where we relied on the model to make tool calls soley through modifying the prompt. Additionally, answer-calculator consistency reaches 1.00 in our main tool-aware evaluations, which represents the instances where the answer returned matches the expression sent to the calculator. However, tool correctness does not imply task correctness.

Our strongest positive result is that multi-solution Tool-SFT improves tool-aware Pass@1 from 0.12 to 0.22 and Pass@16 from 0.52 to 0.60, with valid-number use rising from 0.983 to 0.991. After fixing a seed-collapsed evaluator, we find that the model does sample diverse expressions: the 4-solution Tool-SFT model produces 12.60 unique expressions on average among 16 completions. However, a large gap remains between Pass@1 and Pass@16: 19/50 prompts are missed by the first sample but solved by one of the next 15 samples. This suggests that multi-solution Tool-SFT expands solution support, but the model still lacks reliable ranking or self-selection.

Discussion, Limitations, and Conclusion

Our results suggest that tool integration shifts the bottleneck rather than eliminating it. Tool-SFT and reward shaping teach the model *how* to use the calculator: it learns the tags, valid number use, executable expressions, and answer formatting. But the model still struggles with *what* to calculate: it often chooses a legal arithmetic expression that evaluates to the wrong target. Thus, calculator access partially expands the sampled solution support, but the remaining bottleneck is target-directed ranking and selection.

The main limitations are model scale, training length, and tool scope. Results on Qwen-0.5B may not transfer to larger models with better arithmetic priors. Our final RLOO comparisons use short 50-step runs, and our calculator tool is narrower than a code interpreter or search procedure. Future work should test larger models, longer RL, verifier-guided rejection sampling, explicit expression-diversity objectives, and richer tools. Overall, our takeaway is that tool use teaches the model how to calculate, but not reliably what to calculate.

Abstract

Reinforcement learning with verifiable rewards can improve small language models on reasoning tasks, but recent work suggests that such improvements are constrained by an "invisible leash," where RL increases the probability of already-accessible solutions without expanding the model's actual reasoning frontier. We test whether tool use can break this leash by giving a 0.5B model access to a calculator during RLOO training on Countdown arithmetic. We implement a tool-integrated rollout protocol, Tool-SFT warm starts, multi-solution Tool-SFT, reward-shaped TIR-RLOO, and a tool-aware pass@k evaluator that executes calculator calls during generation. Tool-SFT reliably teaches calculator use and valid number usage, and multi-solution Tool-SFT improves tool-aware Pass@1 from 0.12 to 0.22 and Pass@16 from 0.52 to 0.60. Seed-fixed evaluation reveals substantial expression diversity: the 4-solution model samples 12.60 unique expressions on average among 16 completions. However, the large Pass@1–Pass@16 gap shows that the model often samples a correct expression somewhere in its support without ranking it first. Reward shaping improves rollout reward but is less reliable than supervised expression diversity, introducing more tool errors and lower valid-number use. Overall, calculator access helps the model generate and execute diverse candidate programs, but target-directed selection remains the bottleneck.

1 Introduction

Small language models are increasingly important for cost-sensitive applications and inference where larger proprietary models are impractical or expensive. A real-world example of this is the deployment of on-device tutoring in mathematics (e.g. Alpha School, an AI powered private school in Austin, TX). Reinforcement learning with verifiable rewards (RLVR) is attractive in this regime because it can optimize models directly against task correctness without requiring human preference labels. However, recent work argues that RLVR on small models may be limited by an *invisible leash*: RL reshapes the sampling distribution over solutions already available to the model, but does not reliably produce new reasoning capabilities.

Tool use offers a natural way to test this limitation. If a model lacks internal arithmetic ability, a calculator appears to expand the action space since the model no longer needs to compute the answer itself, and only needs to call the tool correctly. This motivates our central question:

Can tool-integrated RLOO break the invisible leash for a 0.5B model on arithmetic reasoning?

We hypothesize that calculator access alone is insufficient to break the invisible leash because the underlying bottleneck is target-directed expression search, not arithmetic execution. We study this question on Countdown arithmetic. The task goes as follows: Given 3–4 integers and a target, the model must produce an arithmetic expression that evaluates to the target. We use Qwen 2.5-0.5B initialized from a supervised Countdown checkpoint and introduce a calculator tool into the rollout loop. Our core finding is that tool use and multi-solution supervision improve execution and sampled solution support, but not reliable first-sample selection. The model learns to call the calculator and use valid numbers but often chooses expressions that evaluate to the wrong target.

Our contributions are:

1. We implement a tool-integrated RLOO pipeline with calculator execution, tool-output injection, and masked environment tokens.
2. We introduce Tool-SFT and multi-solution Tool-SFT to create a prior over calculator use and increase expression diversity.
3. We design reward-shaped TIR-RLOO with components for format, executability, valid number use, and target matching.

4. We implement a tool-aware pass@k evaluator and diversity diagnostics for unique sampled expressions.

5. We provide a mixed diagnostic result: tool use and multi-solution SFT expand sampled solution support, but the model still struggles to rank or select the correct expression first.

2 Related Work

RL with verifiable rewards. RLVR [7] trains language models using automatically checkable rewards, such as exact mathematical correctness or unit-test pass rates. The invisible leash hypothesis argues that RLVR may improve sampling from an existing set of solutions without expanding the set of problems a small model can solve.

RLOO and policy-gradient fine-tuning. RLOO [1] is a REINFORCE-style method that samples multiple completions per prompt and uses a leave-one-out baseline to reduce variance. It is simple, scalable, and well-suited to verifiable reward settings because each sampled completion can be scored independently.

Preference optimization. IPO [2] and related preference-optimization methods provide an alternative to online policy gradient fine-tuning. In our project, IPO serves as a pure-text baseline rather than the main tool-integrated method.

Tool-integrated reasoning. Tool-using language models [5, 4, 3] can call external systems such as calculators, code interpreters, search engines, or retrieval databases. Prior work shows that tools can improve reasoning when the model learns when and how to invoke them. Our work focuses on a small-model setting, in contrast to [4, 3] which operate at 7B+ scale, where the model lacks a strong prior over tool syntax, requiring Tool-SFT before RL can reinforce tool use.

Positioning relative to the invisible leash. Wu et al. [7] establish the invisible-leash claim in the pure-text RLVR setting and show that RL on small models reshapes the existing solution distribution without necessarily expanding it. We extend this test to the tool-integrated regime, asking whether an external action space, a calculator, changes this constraint. Our result is mixed: multi-solution Tool-SFT expands sampled solution support, but the model still struggles to rank or select correct expressions first.

3 Methods

Novelty. Our pipeline differs from prior tool-integrated RL work in three respects. First, we operate in the small-model regime (0.5B parameters), where the base model has essentially no probability mass on calculator tags, in contrast to [4, 3] which study 7B+ models with established tool priors. This regime makes a Tool-SFT warm-start a prerequisite rather than an optimization, since RL alone cannot bootstrap tool use from zero. Second, we introduce multi-solution Tool-SFT, which trains on multiple distinct correct expressions per problem to widen the model's sampled solution distribution rather than concentrating it on a single canonical form. Third, we design a number-use-aware reward shaping function with components that directly target our dominant observed failure mode, namely syntactically valid calculator calls that use numbers absent from the prompt, an issue not addressed by execution-only reward signals in prior work.

Assumptions. Our pipeline assumes (i) the calculator returns exact arithmetic results with no rounding error, since all Countdown operations are integer or rational; (ii) the safe AST evaluator correctly rejects malformed or unsafe expressions and does not produce side effects; (iii) the model has access only to the calculator tool, not to a code interpreter, retrieval system, or other external resource, isolating the contribution of arithmetic execution from richer tool affordances; and (iv) injected tool-output tokens are environment-generated and therefore correctly excluded from the policy loss via masking, so the model is not trained to predict calculator outputs.

3.1 RLOO Objective

For each prompt x, RLOO samples a group of k completions $\{y_1, \ldots, y_k\}$ from the current policy. Each completion receives a scalar reward $R(y_i, x)$. The leave-one-out advantage for completion i is

$$\hat{A}_i = R(y_i, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_j, x).$$

The policy is updated to increase the probability of completions with positive advantage and decrease the probability of completions with negative advantage. We use KL regularization to a reference model and entropy regularization to stabilize training. Because sampling is performed through vLLM while updates are performed with a Hugging Face model, we also use sequence-level importance weighting to account for sampler and update mismatch.

3.2 Tool-Integrated Rollout Protocol

In the tool-integrated setting, model generation is no longer a single uninterrupted text completion. The model can emit a calculator call:

`<calculator>EXPR</calculator>`.

The rollout harness detects a completed calculator call, pauses generation, extracts `EXPR`, and evaluates it using an arithmetic evaluator. If evaluation succeeds, the environment injects

`<calc>RESULT</calc>`.

If evaluation fails, the environment injects

`<calc_error>ERROR</calc_error>`.

Generation then resumes, and the model is expected to output a final answer of the form

`<answer>EXPR</answer>`.

Injected tool-output tokens are masked from the policy loss because they are produced by the environment rather than the model. This prevents the policy from being trained to predict calculator outputs directly.

Algorithm. Algorithm 1 summarizes one tool-integrated rollout.

For each prompt, RLOO samples k such rollouts independently, scores each with $R(s, x)$ and computes the leave-one-out advantage $\hat{A}_i$.

Algorithm 1 Tool-integrated RLOO rollout for one prompt x

Require: Policy π_θ, prompt x, calculator $\mathcal{C}$, max tool calls T, max tokens L

$s \leftarrow x$ ▷ running context
$t \leftarrow 0$ ▷ tool-call counter
while $|s| < L$ and not terminated **do**
  Generate tokens from $\pi_\theta(\cdot \mid s)$ until `</calculator>` or end-of-answer
  Append generated tokens to s
  if generation ended at `</calculator>` and $t < T$ **then**
    Extract expression `EXPR` from latest `<calculator>...</calculator>`
    $r \leftarrow \mathcal{C}(\texttt{EXPR})$ ▷ safe AST arithmetic evaluator
    if r is a numeric result **then**
      Append `<calc>`r`</calc>` to s ▷ environment-generated, masked from loss
    else
      Append `<calc_error>`r`</calc_error>` to s
    end if
    $t \leftarrow t + 1$
  else if generation ended at `</answer>` **then**
    return s ▷ terminate rollout
  end if
end while
return s

3.3 Tool-SFT and Multi-Solution Tool-SFT

Naive tool-RL fails because the base 0.5B model has little probability mass on calculator tags. If the model never samples `<calculator>` tokens, RL cannot reinforce useful tool use. We therefore first apply Tool-SFT.

Each Tool-SFT trajectory contains a prompt, a calculator call, an injected result, and a final answer. For example:

```
Target: 95, Numbers: [19, 25, 89]

<calculator>(89 + (25 - 19))</calculator>
<calc>95</calc>
<answer>(89 + (25 - 19))</answer>
```

Single-solution Tool-SFT. For each Countdown problem, we generate a single correct arithmetic expression by enumerating expressions over the prompt's numbers under the allowed operator set $\{+, -, \times, \div\}$ and selecting the first whose evaluation equals the target. This produces one $(\text{prompt}, \text{trajectory})$ pair per problem.

Multi-solution Tool-SFT. To widen the model's distribution over arithmetic expressions, we generate up to m distinct correct expressions per problem. We enumerate candidate expressions over the prompt numbers and the operator set in a fixed canonical order, keeping each expression whose evaluation equals the target. Expressions are deduplicated by their string representation after normalization (parentheses-stripped, operand-sorted within commutative operators). Enumeration continues until either m distinct expressions are found or the search space is exhausted. Each retained expression becomes its own SFT trajectory with

the same prompt but a different calculator call and answer. The resulting training set contains up to m rows per problem. In our main experiments we use $m = 4$, which produced 15,713 SFT rows from 5,000 problems, an average of 3.14 distinct correct expressions per problem.

This is intended to push probability mass onto multiple correct expressions per prompt rather than concentrating it on a single form, with the goal of improving expression diversity.

3.4 Reward Shaping

Final task correctness is sparse in the tool-integrated setting, meaning that a rollout can fail because of malformed tags, invalid syntax, hallucinated numbers, tool errors, or simply choosing an expression that does not evaluate to the target. To provide dense intermediate signal we decomposed the per-rollout reward into a base verifier score and an additive shaping layer:

$$R(s, x) \;=\; R_{\text{base}}(s, x) \;+\; \sum_{c \in \mathcal{C}} \lambda_c \, \phi_c(s, x).$$

The base verifier score is the standard Countdown reward used in our pure-text milestone as 1.0 if the `<answer>` expression uses valid numbers and evaluates to the target, 0.1 if an answer tag is present but incorrect, or 0 otherwise. The shaping components ϕ_c and their coefficients λ_c are listed in Table 1. Coefficients were chosen so that no single shaping bonus exceeds the base verifier reward, ensuring that correct answers remain the dominant signal.

Component c	Coefficient λ_c	Definition $\phi_c(s, x)$
Calculator target match	$+0.30$	Indicator: $\mathcal{C}(\texttt{EXPR}) =$ target
Valid number use	$+0.05$	Indicator: all numbers in `EXPR` come from prompt
Valid tool format	$+0.01$	Indicator: `<calculator>...</calculator>` parses
Invalid number-use penalty	-0.05	Indicator: `EXPR` contains a number not in the prompt
Tool-error penalty	-0.02	Indicator: $\mathcal{C}(\texttt{EXPR})$ raises an error
Answer-mismatch penalty	-0.02	Indicator: `<answer>` expression $\neq$ calculator expression

Table 1: Shaping components and coefficients for TIR-RLOO. Bonuses (positive λ_c) and penalties (negative λ_c) are applied additively on top of the base verifier reward.

Each shaping component addresses a specific failure mode observed in earlier pilots. The calculator-target bonus directly rewards the model for proposing expressions that solve the problem, providing dense signal where the sparse base reward fires only on full success. The valid-number bonus and penalty address the dominant failure of tool-error penalties alone, where the model emits syntactically valid calculator calls using numbers absent from the prompt. The answer-mismatch penalty enforces protocol discipline, the final `<answer>` must match the expression the calculator already verified, preventing the model from "checking" one expression and answering with another.

3.5 Tool-Aware Pass@k Evaluation

Default text-only evaluation cannot correctly score tool-integrated trajectories because it does not execute calculator calls. We implement a tool-aware evaluator that mirrors the training rollout protocol. It allows the model to emit calculator calls, executes them, injects calculator outputs, resumes generation, and then scores the final `<answer>`.

We compute pass@k per prompt. For each prompt, pass@k is 1 if any of the first k sampled completions is correct and 0 otherwise. We also log tool diagnostics, including tool-use rate, malformed calls, tool

errors, valid-number use, calculator target match, answer-calculator consistency, and expression diversity. Expression diversity is measured by the number of unique expressions among the k rollouts for each prompt.

4 Experimental Setup

Task and dataset. We evaluate on `asingh15/countdown_tasks_3to4`. Each prompt contains 3–4 integers and a target value. A solution is correct if the extracted arithmetic expression uses valid numbers and evaluates to the target.

Model. We use Qwen 2.5-0.5B initialized from `asingh15/qwen-sft-countdown-defaultproj`.

Baselines. We compare:

- text-only SFT,
- IPO,
- pure-text RLOO,
- Tool-SFT with one solution per problem,
- Tool-SFT with four solutions per problem,
- Tool-SFT plus number-use RLOO.

Baseline justification. Each baseline isolates one design choice in the tool-integrated pipeline. The text-only SFT base measures starting capability without RL. IPO and pure-text RLOO measure what pure-text post-training can recover on top of that base, and serve as the reference point for the "invisible leash" claim. Tool-SFT with one solution per problem isolates the contribution of tool-protocol supervision, holding training-data diversity fixed. Tool-SFT with four solutions per problem adds expression diversity to the supervision signal, isolating the effect of supervised expression variety from RL-induced variety. Finally, Tool-SFT plus number-use RLOO measures whether reward-shaped RL on top of a tool-using warm-start exceeds either intervention alone. Together the six baselines decompose the pipeline into (i) warm-start, (ii) diversity supervision, and (iii) RL with shaped reward.

Metrics. We report Pass@1 and Pass@16, final rollout reward for RL-trained conditions, valid-number-use rate, calculator target-match rate, tool-error rate, malformed-call rate, and expression-diversity diagnostics.

Metrics justification. We report Pass@1 to measure greedy single-sample correctness, the metric most relevant to single-shot deployment, and Pass@16 to measure performance under a realistic best-of-k sampling budget. Their ratio diagnoses whether additional samples explore genuinely new solutions or duplicates of the same expression, which is the core test of the invisible leash hypothesis. The auxiliary metrics (valid-number-use rate, calculator target-match rate, tool-error rate, malformed-call rate, and unique-expression count at $k = 16$) decompose pass@k failures into specific failure modes (protocol, execution, or search), allowing us to localize the bottleneck.

Training details. Pure-text milestone RLOO uses 100 training steps. Final tool-integrated RLOO comparisons use 50 training steps. Tool-SFT is trained on either 5000 examples with one epoch or 1000 examples with six epochs in the milestone-aligned setting. Multi-solution Tool-SFT uses up to four expressions per prompt.

5 Results

All reported numbers are from single-seed runs. Multi-seed sweeps were not feasible within our compute budget given the per-run cost of TIR-RLOO with vLLM rollouts; we acknowledge this as a key limitation and discuss it further in Section 6.

5.1 Pure-Text RLOO Milestone

Model	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16
SFT (base)	0.30	0.40	0.52	0.62	0.68
IPO	0.36	0.46	0.64	0.70	0.76
RLOO	0.50	0.62	0.66	0.72	0.72

Table 2: Pure-text milestone. RLOO improves Pass@1 over IPO, but Pass@16 saturates near the SFT/IPO frontier.

The pure-text milestone shows that RLOO improves single-sample performance, but does not clearly expand the high-k frontier. This motivates the tool-integrated setting. If a calculator expands the action space, it may allow the model to exceed this frontier.

5.2 Tool-SFT and Multi-Solution Tool-SFT

Condition	Pass@1	Pass@16	Valid #s	Target Match	Tool Errors	Unique Expr@16
SFT base, text-only eval	0.285	0.76	–	–	–	–
Tool-SFT, 1 solution/problem	0.12	0.52	0.983	0.120	3/800	12.24
Tool-SFT, 4 solutions/problem	**0.22**	**0.60**	**0.991**	**0.220**	**3/800**	**12.60**

Table 3: Seed-fixed Tool-SFT comparison with 16 tool-aware completions per prompt. The text-only SFT base is included only as a reference point, since it is evaluated without calculator execution. Under tool-aware evaluation, multi-solution Tool-SFT improves over single-solution Tool-SFT on Pass@1, Pass@16, valid-number use, and target match. Both Tool-SFT variants now show substantial expression diversity once sampling uses distinct rollout seeds, but multi-solution supervision produces the strongest and most reliable tool-aware performance.

Tool-SFT reliably teaches the model to use the calculator interface which can be seen where both Tool-SFT variants achieve near-perfect tool-use rates, high answer-calculator consistency, and very few tool errors under the seed-fixed tool-aware evaluator. Multi-solution Tool-SFT further improves the quality of the sampled solution set, compared with one-solution Tool-SFT, it raises Pass@1 from 0.12 to 0.22 and Pass@16 from 0.52 to 0.60, while improving valid-number use from 0.983 to 0.991. The key change is not merely better formatting, but better expression diversity. The 4-solution model samples an average of 12.60 unique

expressions among 16 completions. Thus, supervised expression diversity expands the model's sampled solution set more reliably than short-horizon reward shaping.

5.3 Reward-Shaped Tool-Integrated RLOO

Condition	Final Rollout Reward	Pass@1	Pass@16	Valid #s	Tool Errors	Unique Expr@16
Pure-text RLOO 50-step baseline	0.2125	0.04	0.34	–	–	–
Tool-SFT + number-use RLOO	**0.685**	**0.06**	**0.54**	0.879	83/800	13.08

Table 4: Seed-fixed 50-step RLOO comparison. Number-use shaping increases final-batch shaped rollout reward and improves Pass@16 over the exact 50-step pure-text baseline checkpoint. However, the resulting tool trajectories are noisier than Tool-SFT alone, valid-number use drops to 0.879 and 83/800 rollouts have tool errors. This suggests that reward shaping increases expression diversity, but short-horizon RLOO does not improve tool reliability as cleanly as multi-solution Tool-SFT.

Reward shaping improves final-batch rollout reward and increases held-out Pass@16 under the seed-fixed evaluator. However, the improvement comes with a reliability tradeoff, where compared with multi-solution Tool-SFT, the RLOO-shaped checkpoint had lower Pass@1, lower valid-number use, and more tool errors. Thus, RLOO shaping helps diversify sampled expressions, but the most stable improvement in our experiments comes from supervised expression diversity rather than reward-shaped RL.

One possible explanation is policy drift away from the calculator-expression distribution learned during Tool-SFT. Tool-SFT provides dense supervision over a narrow protocol: correctly formatted calculator calls, valid arithmetic expressions, legal number usage, and consistent answer formatting. In contrast, RLOO optimizes only the scalar reward assigned to sampled trajectories. Because the reward function is an imperfect proxy for the full calculator protocol, policy-gradient updates can increase the probability of trajectories that receive partial reward while gradually reducing probability mass on the clean tool-use patterns established during Tool-SFT. This effect is amplified by the small-batch, high-variance setting used in our experiments. The resulting behavior is consistent with the observed drop in valid-number use and increase in tool errors despite higher rollout reward, suggesting that short-horizon RLOO may trade protocol reliability for reward optimization unless sufficiently constrained by supervision or stronger regularization.

In an additional RLOO attempt initialized from the 4-solution Tool-SFT checkpoint, the saved interrupted checkpoint degraded sharply: seed-fixed evaluation gave Pass@1/16 of 0.00/0.02, valid-number use of 0.05, 745/800 tool errors, and 104/800 malformed tool calls. Although expression diversity remained high, the expressions were largely malformed or invalid, suggesting that naive short-horizon shaped RLOO from a strong tool prior can destabilize the calculator protocol.

5.4 Diversity Diagnostics

A key diagnostic is expression diversity: whether pass@k improves because additional samples contain genuinely different arithmetic programs rather than duplicates of the same expression. For the 5k multi-solution Tool-SFT model, seed-fixed tool-aware evaluation gives Pass@1/2/4/8/16 of 0.22/0.26/0.38/0.46/0.60. The model maintains reliable tool use, valid-number-use rate is 0.991, answer-calculator consistency is 1.00, and only 3 out of 800 rollouts produce tool errors. Crucially, the model now samples a diverse set of expressions: the mean number of unique expressions among 16 completions is 12.6, and the median 14.0.

This reveals a more nuanced bottleneck. Multi-solution Tool-SFT does expand the model's sampled solution support where 30/50 prompts are solved by at least one of 16 completions, compared with only 11/50 solved by the first completion. Thus, 19/50 prompts, or 38%, are missed at Pass@1 but solved by

Pass@16. The remaining failure is therefore not simply lack of expression diversity, it is also a ranking or self-selection problem. The model often samples a correct expression somewhere in its support, but does not reliably place that expression first.

5.5 Qualitative Rollouts

Successful tool trajectory.

```
Target: 95, Numbers: [19, 25, 89]

<calculator>(89 + (25 - 19))</calculator>
<calc>95</calc>
<answer>(89 + (25 - 19))</answer>
```

Remaining failure mode.

```
Target: 62, Numbers: [48, 28, 42]

<calculator>(28 - (42 - 48))</calculator>
<calc>34</calc>
<answer>(28 - (42 - 48))</answer>
```

The failed example is valid in the tool-use sense since the calculator call is executable, the numbers are legal, and the final answer matches the calculator expression. But it is wrong in the task sense because the expression evaluates to 34 rather than 62. This illustrates the remaining failure mode: the model can execute calculator calls and often samples diverse candidate expressions, but it does not reliably select a target-matching expression first.

No-shaping ablation: syntactic collapse. To isolate the contribution of number-use reward shaping, we ran a 50-step TIR-RLOO ablation with all shaping coefficients λ_c set to zero. Final rollout reward fell to 0.05 and the model syntactically collapsed, we observed rollouts containing unmatched parentheses, hallucinated tokens (`nullptr`, `gap`), and broken output structure (`</44>`). Tool errors reached 100%. This confirms that the shaping reward is load-bearing and without it, the sparse base reward provides insufficient signal to maintain even the tool-protocol behavior established by Tool-SFT.

6 Discussion

Our results suggest that tool integration changes the failure type rather than eliminating it. Before tool use, the model must solve arithmetic internally. After Tool-SFT, the model can call an external calculator, but it must still generate the expression to be evaluated. The bottleneck therefore shifts from computation to target search.

This also explains the mismatch between shaped rollout reward and held-out Pass@k. Reward shaping can improve intermediate behaviors such as legal number use and executable tool calls. But these behaviors are not sufficient for solving Countdown and the model must still find an expression whose value equals the target. More broadly, our results suggest a tension between supervised tool learning and reward-based optimization. Tool-SFT teaches a highly structured calculator protocol, whereas RLOO only observes scalar rewards. When the reward does not perfectly encode protocol correctness, RL updates may push the policy away from the distribution learned during Tool-SFT even while improving the optimized reward. This may

explain why some of our shaped-RLOO checkpoints exhibited lower valid-number use, more malformed tool calls, and higher tool-error rates than the corresponding Tool-SFT warm starts.

Broader impacts. Mixed results on small-model TIR have implications for cost-sensitive deployment. If tool access expands sampled support but does not reliably solve selection, scaling the base model may be necessary even when tools are available, which is a more expensive solution than the tools-as-equalizer narrative suggests. This suggests practitioners targeting on-device or edge deployment should not assume tool integration alone substitutes for model scale on reasoning-heavy tasks.

Limitations. The main limitations are model size and scale, training length, and tool scope. Qwen-0.5B may not have enough capacity to represent a diverse distribution over arithmetic programs. Our final TIR-RLOO comparisons use 50 training steps, and longer training could change the trajectory. Finally, a calculator is a narrow execution tool and richer tools such as code interpreters or verifiers may produce different results.

Difficulties encountered. We encountered a couple of engineering difficulties that shaped our final pipeline. First, our initial tool-aware pass@k evaluator silently used a fixed sampling seed across the k rollouts per prompt, so every completion was identical and pass@1 trivially equaled pass@16. Diagnosing this bug required adding the diversity diagnostics reported in Section 4. And second, sparse-reward TIR rollouts caused sequence-level importance weights to collapse to $\sim 10^{-41}$ within six steps, freezing gradients. We diagnosed this by comparing TIR and pure-text RLOO training curves and stabilized training by raising the KL coefficient from 10^{-3} to 10^{-2}.

7 Conclusion

We tested whether calculator access can break the invisible leash for tool-integrated RLOO in a 0.5B language model. Our answer is mixed and diagnostic. Multi-solution Tool-SFT expands the model's sampled solution support, improving Pass@1 from 0.12 to 0.22 and Pass@16 from 0.52 to 0.60 while producing 12.60 unique expressions on average among 16 completions. However, the much lower Pass@1 of 0.22 compared with Pass@16 of 0.60 shows that the model often fails to rank or select the correct expression first. Reward-shaped RLOO improves final-batch rollout reward and can increase sample diversity, but it is less reliable than Tool-SFT, producing lower valid-number use and more tool errors. Overall, tool use helps the model generate and execute diverse candidate programs, but target-directed selection remains the bottleneck. Future work should test whether process-reward shaping over intermediate calculator calls can teach target-directed search, or whether verifier-guided sampling at inference time can solve the selection bottleneck.

8 Team Contributions

- **Isaiah Flores:** Implemented and ran pure-text RLOO baselines, contributed to RLOO training infrastructure, analyzed rollout reward and pass@k comparisons, helped debug training and evaluation scripts, and poster/report synthesis.
- **Mia Xiao:** Implemented and ran Tool-SFT warm-start experiments and Tool-SFT integrated RLOO experiments, aligned Tool-SFT hyperparameters with the original SFT milestone setting, contributed to dataset generation and tool-prior analysis, and ran additional evaluation comparisons.

- **Katherine Xu:** Implemented tool-integrated rollout/evaluation components, multi-solution Tool-SFT support, reward-shaping diagnostics, tool-aware pass@k evaluation, diversity diagnostics, and poster/report synthesis.

Changes from Proposal. The original proposal focused on tool-integrated RLOO as the main extension. During implementation, we found that naive tool-RL could not bootstrap calculator use because the small model rarely emitted tool tokens. We therefore added Tool-SFT warm starts, multi-solution Tool-SFT, reward shaping, and a tool-aware evaluator. This changed the project from simply testing whether tool-RL improves performance to diagnosing which subproblem fails among tool invocation, legal number use, or target expression search.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024.

[2] Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences, 2023.

[3] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025.

[4] Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025.

[5] Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning, 2025.

[6] 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. Training language models to follow instructions with human feedback, 2022.

[7] Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin, 2025.