Extended Abstract

Motivation. Pure text-based reasoning is brittle, as small models can arithmetic slips and lose track of which numbers remain. **Tool-Integrated Reasoning (TIR)** lets the policy invoke external tools (e.g. a calculator) mid-generation. Building upon tool-augmented RL techniques introduced in research such as Search-R1 [1] and Tool-Star [2], our goal is to identify the interventions that help a tool-using policy better adapt to numerical tasks such as Countdown for small LLM modals such as Qwen2.5-0.5B.

Method. We RL-fine-tune a small language model Qwen2.5-0.5B to solve the Countdown arithmetic task: given a set of numbers and a target, produce an equation using each number exactly once. A rule-based verifier gives a sparse reward, making this an ideal testbed for RL fine-tuning of LLMs.

Implementation. We develop three deterministic tools: a calculator, a number tracker, and a running total. These tools are invocable using structured tags with `<tool_use><tool_use>` designation. We first build a dataset of $\sim$1,700 correct tool-using trajectories from a DeepSeek-chat teacher model for SFT fine-tuning. Moreover, we created a dataset with 500K problems that includes difficulty labels for use in curriculum training of online reinforcement learning algorithms. Both training datasets were modeled after the datasets used in the default parts of the project. These datasets were used to train published RL models for baseline comparison as well as a novel RL model to further train the policy to use assistive tools. Our runs use both standard dataset training as well as a curriculum training on a five different algorithms:

1. **SFT**: Supervised fine-tuning (SFT) on teacher trajectories with tool-integration and tool-result token masking.
2. **IPO Tool-Contrastive**: Offline preference optimization contrasting correct answer with tool-using vs. incorrect rollouts.
3. **RLOO + Reward Shaping**: Online policy gradient with a reward structure that penalizes incorrect outputs and encourages correct tool use.
4. **RLOO + Self-Critic**: Incorporates periodic GRPO updates over self-sampled groups.
5. **RLOO + Self-Critic + Curriculum Training**: Incorporates curriculum training with RLOO + Self-Critic.
6. **Hindsight RLOO**: Injects an oracle solution for every failed trajectory as if it had intended to reach that state and labels it as a high reward trajectory.
7. **Rejection-Sampling Distillation (ReST)**: Iteratively samples correct tool-using trajectories from the current policy and fine-tunes on them.

Results. The ReST model performed best from all our models. From our RL algorithms, RLOO + SC + Curriculum performed best. Pass@K was used to evaluate the models.

Discussion.

- **Multi-turn evaluation matters.** Gives the LLM flexibility to decide how best to leveraege tool during inference and raises pass@1 by up to 10.8%.
- **ReST outperforms all online RL variants**, ReST improves reliability (i.e. pass@1) by almost 6% .However, it does not significantly improve capability (i.e. pass@16).
- **Intra-trajectory tool volume hurts.** Methods that produce frequent multi-tool calls within a single trajectory show lower pass@1. This indicates that unnecessary tool calling interferes with good reasoning capabilities.
- **Inter-trajectory breadth helps.** The top-performing algorithms have the highest single tool-calling rates ($\geq$68%) and lowest multi-tool ratges ($\leq$5%), indicating that targeted and relevant tool calling (calculator) is beneficial.
- **Tool-integrated Reasoning Is Hard on Smaller Models.** Training our Qwen2.5-0.5B model to use tools is challenging because it can easily overfit to the training data or learn reward-hacking shortcuts.

Conclusion. Reliable tool-integrated reasoning at small scale requires targeted, verified tool use at inference time.

Tool-Integrated Reasoning for Countdown

Mahmood Alhusseini
Department of Management Science Engineering
Stanford University
mih@stanford.edu

Frank D'Agostino
Department of Computer Science
Stanford University
frankdag@stanford.edu

Sebastian Fisher
Department of Computer Science
Stanford University
sbfisher@stanford.edu

Abstract

We study Tool-Integrated Reasoning (TIR) for the Countdown arithmetic task on Qwen2.5-0.5B, comparing supervised fine-tuning, offline preference optimization, iterative rejection-sampling distillation, and three RLOO variants including a novel Hingsight RLOO with oracle augmentation and an oracle-mixed baseline. Multi-turn tool execution at inference time provides a large performance gain (+10%). Among training approaches, ReST outperforms online RL variants. Hindsight RLOO achieves a poor pass@1 despite perfect format compliance, revealing an oracle format lock-in failure where the model learns the output template but not the conditional arithmetic reasoning. We also develop a augmented dataset for difficulty-based curriculum learning. We then run initial tests of three various methods of curriculum learning with this augmented dataset. Our failure mode analysis shows targeted tool invocation is a stronger predictor of success than invocation frequency.

1 Introduction

We RL-fine-tune a small language model (**Qwen2.5-0.5B** [3]) to solve the **Countdown** arithmetic task: given a set of numbers and a target, produce an equation using each number exactly once. A rule-based verifier gives a sparse reward, making this an ideal testbed for RL fine-tuning of LLMs.

Countdown captures the core difficulty of a broad class of reasoning tasks. It demands multi-step planning, exact arithmetic, and explicit constraint tracking (using each number exactly once), the same competencies that limit performance in formal math, coding, and other long-horizon sequential problems. Solutions are verifiable and difficulty is tuneable, we can isolate the effect of specific training interventions without the confounding complexity of open-ended benchmarks.

Pure text-based reasoning is brittle, where small models make arithmetic slips and lose track of which numbers remain. **Tool-Integrated Reasoning (TIR)** lets the policy invoke external tools (e.g. a calculator) mid-generation. Building upon tool-augmented RL techniques introduced in research such as Search-R1 [1] and Tool-Star [2], our goal is to identify the interventions that help a tool-using policy better adapt to numerical tasks such as Countdown.

2 Related Work

RLHF [4] established the paradigm of fine-tuning LLMs by using a learned reward model. To remove reward model bias, subsequent work moved towards outcome-based verifiable rewards. DeepSeek-R1 [5] trains long train-of-thought reasoning using GRPO [6] with binary correctness rewards on math/coding tasks. RLOO [7] offers a simpler leave-one-out baseline estimator as an alternative to PPO, reducing variance.

There is a growing line of research working to train LLMs to interleave tools within its trajectories.

Toolformer [8] showed LLMs can learn to call APIs mid-generation through self-supervised training on inserted tool-calls. ReAct [] interleaves tool-calls during inference time with no fine-tuning. Tool-Star [2] trains LLMs to interleave reasoning with tool calls. It shapes reward hiearchically across tool invocations rather than rewarding only the final answer. It also incorporates a self-critic, where the model critiques its own reasoning.

ReST [9] fine-tunes on reward-filtered self generated data. It iterates between generation and SFT on high-reward outputs. However, this is sample-inefficient since early on in training it rarely produces correct outputs. IPO [10] optimizes pairwise preferences to enable preference alignment while reducing chances of policy collapse. It does not use a learned reward model.

3 Method

3.1 Multi-turn Evaluation

To allow the model to use tools, we use multi-turn sampling from the generating model. Specifically, we register two stop strings, `"</use_tool>"` and `"</answer>"`. If the engine (vLLM) sees the closing answer tag, it knows generation is done. If a `"</use_tool>"` tag is encountered, the corresponding tool is executed deterministically, and the output is appended to the model output in between `<tool_result></tool_result>` tags. Then, the model continues auto-regressive generation with this updated context. Generation continues in this manner until the max output token count is reached (1024 in our case) or an answer tag is output by the model.
As an experiment, in addition to multi-turn evaluation, we also evaluate some of our models using standard single-turn evaluation, where we have the model generate a solution in one generation step. In this form of evaluation, the model may generate tool calls and tool responses on its own.

3.2 Tool Design

We define three tools we believed could be relevant to the Countdown task: a calculator, number tracker, and running total tool tracker. Usage of these is described in the "Tool Usage Examples" figure below.

Tool Usage Examples

Calculator

Evaluates arithmetic expressions.

```
<use_tool>
calculator: 50 + 14
</use_tool>
<tool_result>
64
</tool_result>
```

Number Tracker

Reports unused numbers.

```
<use_tool>
number_tracker:
available: 5 59 3 76
| used: 3 76
</use_tool>
<tool_result>
remaining: 5 59
</tool_result>
```

Running Total

Tracks cumulative sums.

```
<use_tool>
running_total:
20, 15, -3
</use_tool>
<tool_result>
total: 32
steps: 20->35->32
</tool_result>
```

3.3 Datasets

3.3.1 SFT dataset with tool trajectories

For the supervised fine-tuning phase, we first generate a dataset of almost 1700 examples with full-trajectories using deepseek-chat as a teacher model. Using the default training examples from the default project (`asingh15/countdown_tasks_3to4`), the teacher model was prompted to solve the problem while using one of the three tools described in section 3.2. From a total of 2,000 examples run, deepseek-chat was able to solve 85% (i.e. 1700 examples) correctly. This dataset was used as training trajectories for SFT. We purposefully used a very strong model for this dataset generation to ensure that we got a good expert example of problem solutions to fine-tune the model with and give it the ability to use tools. Figure 1 shows an example trajectory generated for SFT on the left.

3.3.2 Difficulty-labeled dataset for curriculum training

In addition to the dataset with trajectories for SFT, we generated an augmented version of the `asingh15/countdown_tasks_3to4` dataset for curriculum online reinforcement learning. In order to get difficulty labels, we prompted a larger model (Qwen/Qwen3-32B [11]) to solve the problems. We used an open-source model rather than calling an API so that we could scale generation throughput across multiple modal instances. It is first prompted to solve the problem with tools. Then, it is separately prompted to solve the problem without tools. For both of these samples, the model is also prompted to output a difficulty score in `<difficulty></difficulty>` tags as a single integer from 1 to 10. After this process runs, we give three different types of difficulty labels to each of the roughly 500k examples in the countdown dataset, as follows:

1. 3-number problems are medium, 4-number problems are hard. Call this column `numcount_difficulty`.
2. Problems that the teacher model could solve without tools are easy. Problems that it could solve with tools, but not without tools, are medium. Problems that it couldn't solve at all are hard. Call this column `solvability`.
3. The difficulty assigned by the teacher model gets its own column. Call this `difficulty_score`. If the difficulty label was not parseable, fallback to 10 (alternate design choices for the fallback are left as future work).

The final dataset is the same as the original countdown dataset, but with three new columns: `numcount_difficulty, solvability, difficulty_score` [1]. Figure 1 shows an example entry in the augmented dataset, with its three new difficulty labels by the different metrics we defined. We later ran some initial attempts at curriculum reinforcement learning as described in section 3.10, but due to time constraints did not fully flesh these out.

3.4 Supervised Fine-Tuning (SFT)

We fine-tune the base model (after the initial vanilla SFT phase) on 1700 trajectories from the teacher model described in section 3.3.1. We call this a "warm-start" since we start from the already fine-tuned Tokens inside executed `<tool_result>` tags are masked from the loss, so the policy is trained only on tokens it generates. We can formalize the loss function for this supervised fine-tuning by Equation (1).

$$\mathcal{L}_{\text{TIR-SFT}}(\theta) = -\frac{1}{|\mathcal{R} \setminus \mathcal{T}|} \sum_{t \in \mathcal{R} \setminus \mathcal{T}} \log \pi_\theta(x_t \mid x_{<t}) \tag{1}$$

where $\mathcal{R}$ are response token positions and $\mathcal{T}$ are `<tool_result>` token positions.

3.5 IPO Tool-Contrastive

As an offline method, we apply preference optimization. For each training prompt, we sample completions from SFT with tool execution. We then label correctly tool-using rollouts as positive entries, and incorrect rollouts as negative entries.

[1] The augmented dataset can be found in this link: `https://huggingface.co/datasets/sbfisher/countdown_curriculum`

Examples of Teacher-generated Trajectory vs. Datasets for online RL Algorithms

Teacher-Generated Trajectory Example

Prompt: Using the numbers [3, 23, 20], create an equation that equals 83.

```
Completion:
<think>
  We need to combine 3, 23, and 20
  to get 83.
  Try: 23 * 3 = 69, 69 + 20 = 89
  (too high).
  Try: 20 * 3 = 60, 60 + 23 = 83.
  Yes! Verify with calculator.
</think>
<use_tool>
  calculator: 20 * 3 + 23
</use_tool>
<tool_result>
  83
</tool_result>
The result matches the target.
<answer>
  20 * 3 + 23
</answer>
```

score: 1.0 *tool calls:* 1

Online RL Training Example (Hard Difficulty)

Prompt: Using the numbers [43, 8, 47, 20], create an equation that equals 72.

```
(no completion needed for online
 rollouts)

ground_truth:
  { target: 100,
    numbers: [5, 59, 3, 76] }

difficulty_score: 7 (teacher rating)
solvability: medium
    (only solved with tools)
numcount_difficulty: hard (4 numbers)

Reference solution:
  5 * ((3 - 59) + 76) = 100
```

Figure 1: Left: teacher-generated trajectory used for SFT warm-up, including tool calls and a verified answer. Right: an RL data online training example used for online algorithms that generate their own rollouts (e.g. RLOO), containing only the numbers and target

$$\mathcal{L}^{\theta}_{\text{IPO}}(\pi_\theta;\pi_{\text{ref}}) = \left\| h^{\pi_\theta}_{y_w,y_l} - (2\beta)^{-1} \right\|_2^2, \quad h^{\pi_\theta}_{y_w,y_l} = \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \tag{2}$$

3.6 RLOO Reward Shaping

We optimize the policy online with RLOO, with a leave-one out baseline and KL penalty to anchor to the SFT reference policy. We replace the binary verifier signal with a hierarchical reward adapted from Tool-Star [2]:

$$R = \begin{cases} \max(\text{Acc} + r_M, \text{Acc}) & \text{format good, Acc} > 0 \\ 0 & \text{format good, Acc} = 0 \\ -1 & \text{otherwise,} \end{cases}$$

The penalty targets the dominant failure mode, which is malformed outputs, while r_M encourages tool-use.

$$\mathcal{L}_{\text{TIR-RLOO}}(\pi_\theta) = -\mathbb{E}\left[\hat{A}_i \cdot \sum_{t \in \mathcal{R} \setminus \mathcal{T}} \log \pi_\theta(x_t \mid x_{<t}) \right] \tag{2}$$

where $\hat{A}_i$ is the leave-one-out advantage.

3.7 RLOO Curriculum (different 'curriculum' method from 3.10)

We found that the multi-tool use is sometimes too complex early on in training, and the policy reward hacks by calling tools over giving correct responses. To address this, for the first half of training the

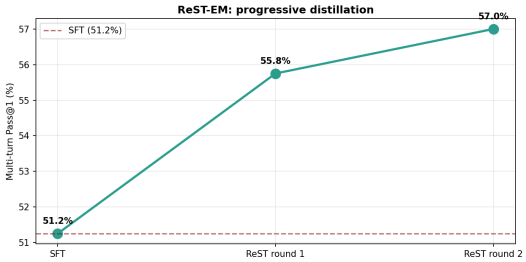

Figure 2: Pass@1 performance comparing SFT, ReST round 1, and ReST round 2. Each round is a subsequent iteration of sampling reward-filtered trajectories and training the policy on only those trajectories.

bonus is given for just a single-tool, then in the second half requires 2 or more tool-calls. Analyzing tool-usage, strong answers tend to just rely on the calculator tool, so rewarding multiple tools may conflict with the optimum.

3.8 Rejection-Sampling Distillation (ReST)

Rather than using online methods, we then tried to distill the policies own correct behavior. We sample many completions which use tool calls, and keep only the correct ones and fine-tune them. Iterating this loop, inspired by the ReST EM paper [9], we iterate to draw each round's data from a strong model. Formally, we build a reward-filtered set of self-generated solutions and fit via MLE:

$$\left[\mathcal{D}_{t+1} = \left\{(q,o) : q \in \mathcal{Q},\ o \sim \pi_t(\cdot \mid q),\ R(o,q) = 1\right\}\right] \tag{3}$$

$$\left[\pi_{t+1} = \arg\max_{\theta}\ \mathbb{E}_{(q,o)\sim\mathcal{D}_t}\big[\log \pi_\theta(o \mid q)\big]\right] \tag{4}$$

Where $\mathcal{Q}$ are the training prompts. Sampling many candidates per prompt captures solutions that appear only at large k in pass@k so this transfers capabilities into pass@1.

3.9 Hindsight RLOO

In the standard RLOO algorithm, if all sampled examples have zero reward, then we cannot improve the model as a reward difference in the group is needed to improve learning. Inspired by Hindsight Experience Replay [12], in our hindsight RLOO algorithm, we create a novel training paradigm that improves on RLOO using hindsight fixes. We first sample a number of examples, get their rollouts, and score each them with a tool-aware reward. Then, for each rollout, we first solve the Countdown instance using a brute-force solver, build a synthetic tool-integrated reasoning trajectory with the correct tool and answer, and then replace one rollout slot with the oracle response. This process allows us to get a positive advantage from an initial all zero reward trajectory samples, allowing us to provide a usable gradient signal for training. Moreover, we use the following reward for hindsight RLOO.

$$r_{b,i} = \mathrm{clip}\big(r^{\text{base}}_{b,i} + \alpha \min(c^{\text{rel}}_{b,i}, C_{\max}) - \beta\, c^{\text{irr}}_{b,i},\ 0,\ 1\big)\,, \tag{5}$$

where $r^{\text{base}}_{b,i} \in \{0,1\}$ is correctness, $c^{\text{rel}}_{b,i}$, $c^{\text{irr}}_{b,i}$ are relevant and irrelevant tool call counts, $\alpha = \beta = 0.1$, and $C_{\max} = 3$.

Beyond the data-augmentation step, we introduce a modification to the RLOO advantage estimator itself. In standard RLOO the leave-one-out baseline for rollout i in group g is $b_i = \frac{1}{G-1}\sum_{j\neq i} r_j$. For an all 0 zero the advantage is thus 0.

We replace the RLOO baseline with an oracle-mixed baseline:

$$b_i^{mix} = (1-\lambda_t)\frac{1}{G-1}\sum_{j\neq i}(r_j + \lambda_t\, r^*)$$

Where $r^* = 1$ is the oracle reward and λ_t is the mixing weight.

The resulting advantage for a rollout that is oracle augmented is $\hat{A}_i = r_i - b_i^{mix}$ which is negative for failed rollouts and positive for injected oracle rollout.

3.10 Difficulty-Labeled Curriculum Training

Regular RLOO samples training examples uniformly. However, an alternate approach is to use curriculum training and introduce the model to problems with increasing complexity as it learns. As a separate experiment from our modifications to the RL algorithm, we ran experiments with varying curriculum strategies using the augmented Countdown dataset ($\sim$490K problems) with difficulty labels from a Qwen3-32B [11] teacher model (see Section 3.3.2). We compare no curriculum (uniform sampling, the baseline (Tool-Star hard-only method), and three other curriculum strategies we developed ourselves.

1. **No curriculum**: Uniform sampling over all problems (baseline).
2. **Hard only**: Train exclusively on problems the teacher could not solve, this was the curriculum used in Tool-Star [2]).
3. **Numcount**: Linear ramp from 100% 3-number problems to 100% 4-number problems.
4. **Solvability**: Ramp from tool-solvable (medium) to unsolved (hard) problems. Slightly different than the tool-star algorithm, which didn't use solved problems during RL.
5. **Score**: Gradually expand the eligible training pool from the easiest 20% to 100%, sorted by teacher-rated difficulty.

All five runs use RLOO with a hierarchcial reward (described in section 3.6) and self-critic phase adapted from the Tool-Star paper [2], starting from the same TIR SFT checkpoint. The curriculum ramp completes at step 60 of 150 total training steps.

4 Experimental Setup

4.1 Infrastructure, Datasets, and Training Pipeline

For all training and evaluation, we use bfloat16 precision on H100 GPUs on Modal. We use vLLM as the inference engine. We use the Qwen2.5-0.5B [3] model in all of our experiments.

Our training pipeline can be thought of in three stages:

1. **Vanilla SFT**: Fine-tuning on the countdown trajectories dataset mentioned above. **We don't change this phase**. We start from the same resulting Qwen model from this phase for all the remaining training. [2]
2. **TIR SFT**: Fine-tune on $\sim$1,700 tool-using trajectories generated by DeepSeek-chat, with tool-result tokens masked from the loss. We experimented a bit with this phase, but not much past the 1700 trajectory dataset.
3. **TIR Reinforcement Learning**: Perform a tool-integrated reasoning RL method on the model from the previous step. The is where most of our experimentation occurred. More details on each specific method is included in the methods and results sections for those methods.

We use a few datasets throughout the training pipeline:

- **Vanilla SFT dataset** CS224R-provided dataset on HuggingFace (Asap7772/cog_behav_all_strategies) of 1,000 correct trajectories of countdown problem solutions provided by CS224R.
- **SFT with trajectories dataset**: 1,707 correct tool-using solutions generated by DeepSeek-chat.
- **RL Dataset**: `asingh15/countdown_tasks_3to4` dataset of around ($\sim$490K problems)
- **Curriculum RL dataset**: The full `asingh15/countdown_tasks_3to4` training split ($\sim$490K problems), augmented with difficulty labels from Qwen3-32B [11], described in section 3.3.2.

[2]The model checkpoint can be found here `https://huggingface.co/sbfisher/cs224r-finalproj-checkpoint1-sft`

- **Evaluation set**: The test split of `asingh15/countdown_tasks_3to4` (50 problems).

4.2 Metrics

We evaluate model checkpoints by sampling $K{=}16$ responses per problem using multi-turn tool execution (Section 3.1) with vLLM settings of temperature=0.6, top-p=0.95, and top-k=20. We report the unbiased pass@k estimator:

$$\text{pass@}k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

where n is the number of samples and c is the number of correct samples for a given problem. Our primary metric is pass@1, which measures accuracy of solving the problem in one try. Comparing this with pass@k for other values of k allows us to qualitatively evaluate the models.

4.3 Baselines

Our main comparison baselines for performance are the results we obtained using vanilla SFT and vanilla RLOO, as these were our best performing models with single-shot evaluation.

- **Vanilla SFT**: Standard SFT without TIR (pass@1 $\approx$ 40%).
- **Vanilla RLOO**: Standard RLOO without TIR (pass@1 $\approx$ 55%).

In addition, we train IPO, RLOO, RLOO + Self-Critic, and ReST with tool-integrated reasoning and use them to compare against novel extensions we trained (HindSight RLOO, RLOO + Self-critic + Curriculum Training).

5 Results

5.1 Supervised Fine-Tuning

The model was initially fine-tuned on a dataset of around 1000 example trajectories not using tools.

As shown in Figure 3, under multi-turn evaluation, the SFT model solves 51.2%, compared to 40.5% for single-turn tool use. Interestingly, the single-turn result is still better than vanilla SFT (which achieved 35%), likely because the model learns more reasoning capabilities from the extra trajectories that include tools.

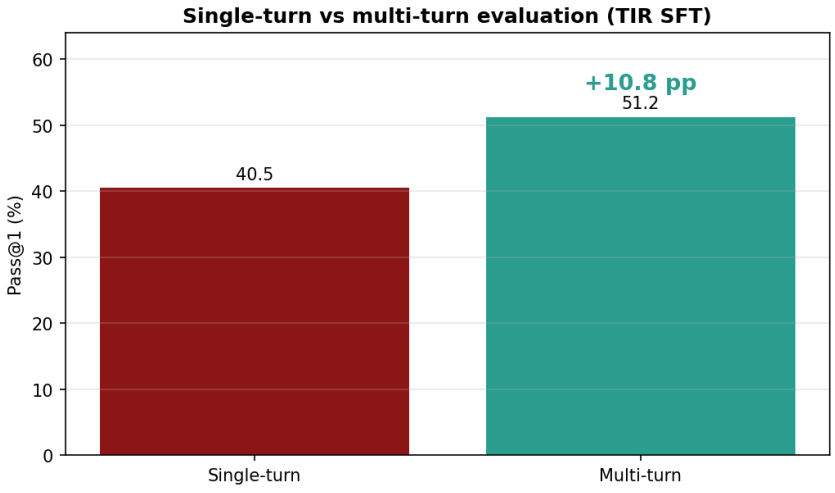

Figure 3: Single-turn vs. multi-turn evaluation.

5.2 Baseline models: IPO, RLOO, RLOO + Self-Critic, and ReST

We've analyzed the results for the different baseline models. From figure **??**, we can see that the best performing model was ReST round 2, achieving a pass@1 of 57.0%, followed by ReST round 1. From the RL family algorithms, RLOO + Self-Critic performed best with a pass@1 of 52.6%. The results of IPO and simple RLOO both achieved a pass@1 rate of 50.4%. Overall, the results of the different RL algorithms are similar and can be due to random fluctuations during the training process. However, we see that all RL models failed policy performance did worse than SFT and ReST models.

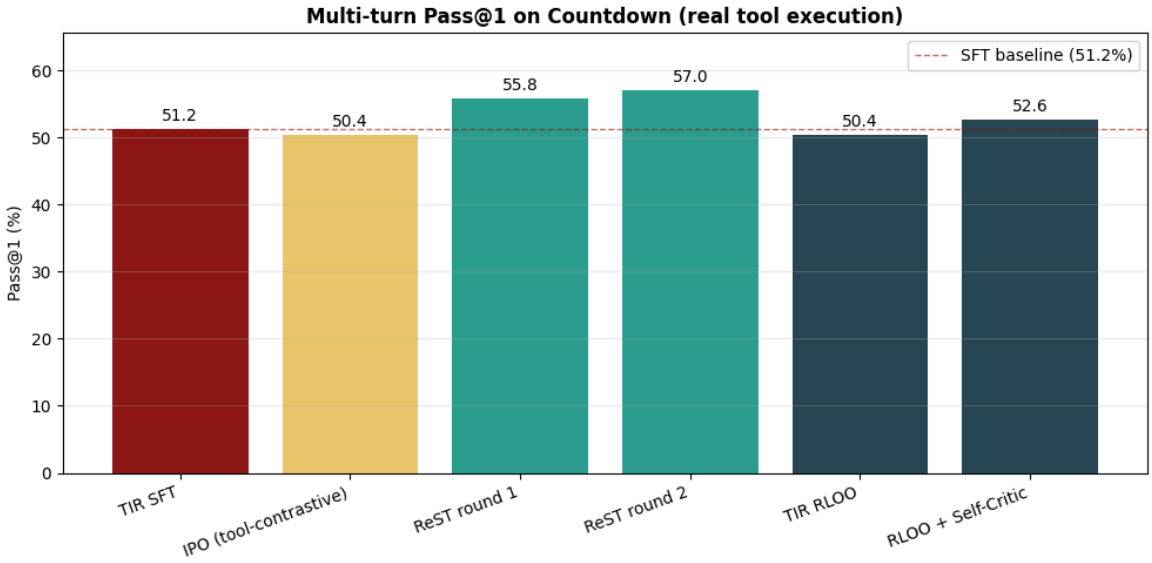

Figure 4: Comparison of pass@1 for the baseline models (IPO, RLOO, RLOO + SC, and ReST). Overall, ReST round 2 performed best, followed by RLOO + Self-Critic. Our proposed Hindsight TIR model performed worst.

5.3 HindSight RLOO

Our hindsight RLOO model achieved a pass@1 rate of 21.62% and a pass@16 of about 66%. We show a comparison of the the different models in 5. HindSight RLOO achieved the worst result but gradually improved from 21% to 66% from pass@1 to pass@16.

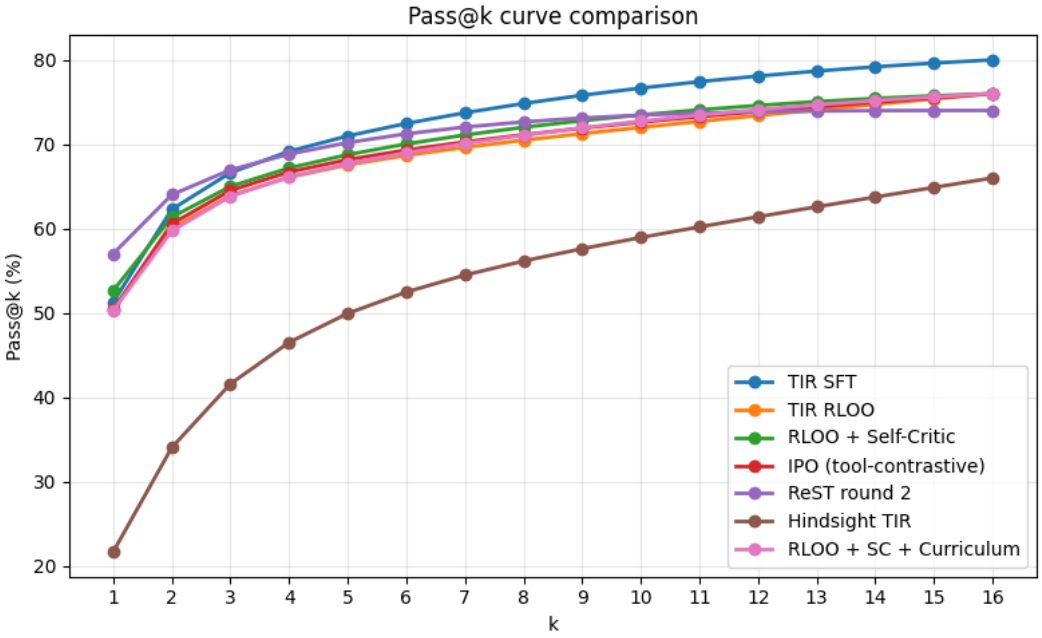

Figure 5: Comparison of pass@k for the SFT, baseline models, and hindsight RLOO. Overall, ReST round 2 achieved the best pass@1 while TIR SFT achieved the best pass@16. HindSight RLOO achieved the worst result but gradually improved from 21% to 66% from pass@1 to pass@16.

5.4 Quantitative Analysis

Method	**Pass@1**	**Δ SFT%**	**Fmt%**	**Tool%**
TIR SFT	51.2	–	60	65
IPO (tool-contrastive)	50.4	-0.9	58	64
ReST round 1	55.8	+4.5	65	69
ReST round 2	57.0	+5.8	65	68
TIR RLOO	50.4	-0.9	62	66
RLOO + Self-Critic	52.6	+1.4	63	67
RLOO + SC + Curriculum	50.2	-1.0	61	66
Hindsight RLOO	21.6	-29.6	92	100

Table 1: Pass@1, Format, and Tool Metrics across all methods.

5.5 Qualitative Analysis

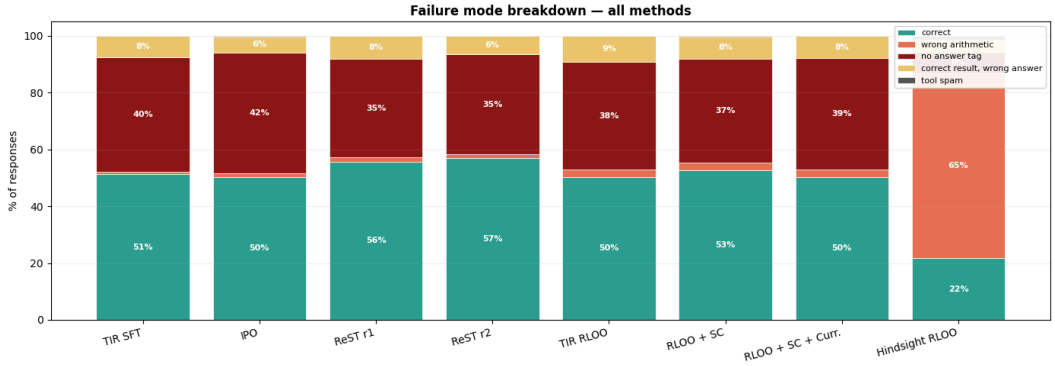

Figure 6: Breakdown of failure mode prevalence, split by categories such as correct, wrong arithmetic, no answer tag, or correct result but wrong answer.

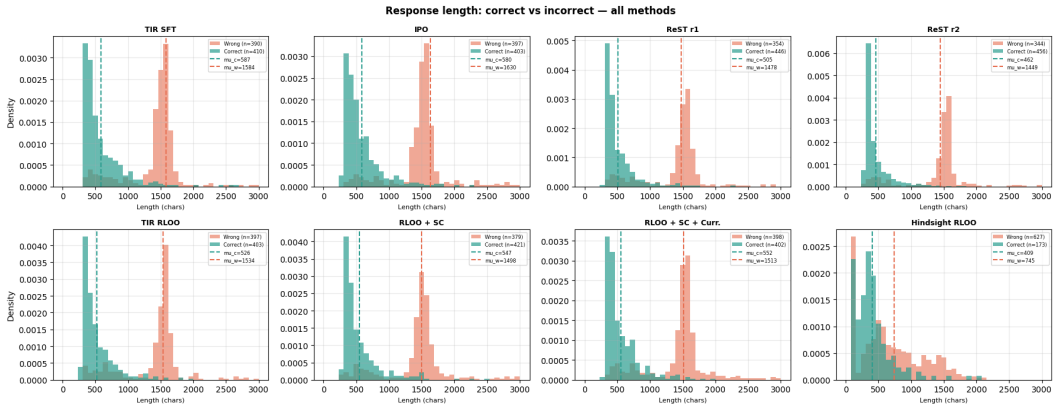

Figure 7: Comparisons of response lengths for correct vs. incorrect responses. We can see that problems the models get wrong often require longer responses, indicating it is a more difficult problem and requires more reasoning. Hindsight RLOO breaks this mold, since it generally relies on shorter responses since it is trained on the oracle.

5.6 Difficulty-Labeled Curriculum Training (incomplete evaluation)

Figure 8 reports pass@k on the evaluation set for each curriculum strategy described in section 3.10, evaluated at the latest available checkpoint (step 60). All curriculum runs start from the same TIR SFT checkpoint and use identical hyperparameters. Due to a lack of time, the checkpoints are evaluated around step 60 of the learning process. As we can see in the plots, using the approach of gradually introducing less solvable problems to the model has the best results, and actually seems to outperform the approach from Tool-star, which only trains RL on problems that the teacher model couldn't solve.

However, these results are inconclusive due to the lack of training time. More iteration is needed in future work to confirm the effectiveness of these curriculum training methods.

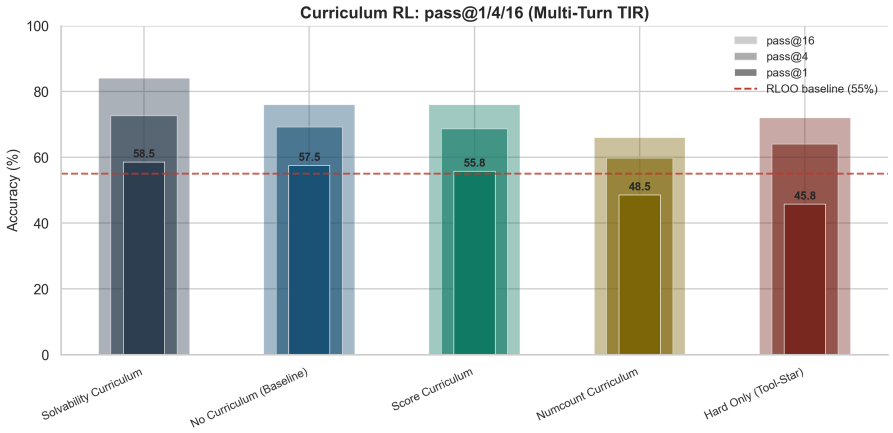

Figure 8: Pass@1/4/16 for each curriculum strategy evaluated at the latest checkpoint. The dashed line indicates the RLOO baseline (55% pass@1) from our best milestone 2 run. Widest bars show pass@16, then pass@4, then narrowest bar is pass@1.

6 Discussion

Our experiments and iterations on Qwen2.5-0.5B reveal several consistent patterns about what makes tool-integrated reasoning work at a small scale, and where it breaks down.

One of the largest performance gaps in our study is not within training, but rather in inference. The TIR SFT model scores 40.5% pass@1 under single-turn evaluation and 51.2% under multi-turn evaluation. This implies that chained tool calls enable the model to solve complex Countdown tasks more reliably.

ReST round 2 achieves the highest pass@1 (57.0%), outperforming every online RL variant and beating the SFT warm-start by 5.8 percentage points. It also has a strong format and tool-use rate.

On the other hand, online RLOO variants have to balance exploring a high combinatorial space while getting sparse binary rewards. The policy is prone to reward hacking rather than genuine improvement. This is because when there are nonzero groups, they dominate the gradient and cause large gradient updates that destabilize learned behaviors. Self-critic RLOO helps with one failure-mode (no-answer tag rate drops from 40% to 23%) but creates another by increasing wrong arithmetic rate. Adding curriculum scheduling on top made things slightly worse, which we attribute to the model overfitting on easier problems in the first training phase and not generalizing when harder ones are introduced.

Hindsight RLOO achieved a 21.6% pass@1, even though it was explored to address the zero gradient issue faced by other RLOO variants. We determined that the cause is oracle format lock-in. Since the oracle was able to generate a single-step response with a single-tool call, this fixed template dominates the training signal. We can see that it learns the format perfectly, with 100% tool-use rate and 92.5% valid answer tag, which is way higher than any other method we tried. However, it did not learn to perform multiple tool calls to test hypotheses and arrive at an answer, and it also did not always learn to properly grab the calculator result to put into the final answer brackets. We see the latent capability as we got up to pass@16, where its pass@K curve has a higher slope than other RLOO variants. In the future, we can adjust the oracle format structure and potentially split up the oracle into multiple tool calls in the future. We can also adjust the weights further of the oracle-mixed baseline to less heavily include the oracle to stabilize training and make it generalizable.

7 Conclusion

Reliable tool-integrated reasoning at small scale requires targeted, verified tool use at inference time.

- Multi-turn tool-usage gives the LLM flexibility to decide how best to leverage tool during inference.

- ReST improves reliability of the SFT on tool-trajectory performance but does not significantly improve capability.
- Efficiency is driven by targeted, narrow tool-use rather than frequency or diversity of tools within a single trajectory.
- The Qwen2.5-0.5B model may have weak baseline reasoning for math & logic, and can be harder to fine-tune with RL.

Moving forward, we think refining the Hindsight RLOO oracle generation is a promising direction with potential to outperform all existing methods if iterated on further.

8 Team Contributions

- **Mahmood:** Baseline algorithms, RL training for multiple algorithms, trajectory generation, report writing.
- **Sebastian:** Methods, Difficulty-based curriculum learning/dataset generation, sft TIR trajectory dataset generation, RL training for multiple algorithms
- **Frank:** Methods, Quantitative & Qualitative results, ReST, IPO Tool-contrastive, RLOO + Self-Critic, Hindsight RLOO

Changes from Proposal: The main changes are the introduction of more robust curriculum learning datasets and the implementation of the Hindsight RLOO method. We also experimented with other methods, such as Reward-Conditioned GRPO, but decided not to include the implementation for our final report. We also decided not to use the tool recommendation approach with DSPy as we found that we had enough to iterate on without working on that.

References

[1] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[2] Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.

[3] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[4] 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.

[5] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[7] 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. *arXiv preprint arXiv:2402.14740*, 2024.

[8] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2023.

[9] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

[10] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR, 2024.

[11] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[12] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.