Extended Abstract

Using, Learning, and Removing Tools: A Study of Tool-Integrated Reasoning for Countdown

Diego Sierra and Brianna Xie

Motivation. Large language models can often discover a plausible high-level plan for arithmetic reasoning tasks while still failing because of a local execution error. The CS224R default project isolates this issue in COUNTDOWN, where a model must use a fixed set of numbers exactly once to reach a target value. We studied the default text-only post-training pipeline—supervised fine-tuning (SFT), preference optimization through IPO, and verifier-based online RL through RLOO—and extended it with a calculator-based tool-use layer. Our central question is whether a small, deterministic arithmetic tool improves functional correctness, and whether gains come from access to the tool alone or from learning when and how to use it.

Approach. We implemented the required text-only baselines using Qwen2.5-0.5B on the provided 3-to-4-number COUNTDOWN dataset. SFT uses completion-token cross-entropy; IPO optimizes pair-wise chosen/rejected responses against the SFT reference; and RLOO performs online verifier-based optimization with a leave-one-out baseline. Our extension, TOOL-COUNTDOWN, adds a safe calculator environment with the syntax `<tool_call>calc:  expression</tool_call>` followed by an environment-supplied `<tool_result>`. We also implemented a small exact COUNTDOWN solver to synthesize tool-aware SFT traces, a tool-aware SFT dataset that masks `<tool_result>` observations out of the action loss, an interactive vLLM evaluator that executes calculator calls during generation, a tool-aware RLOO wrapper with shaped rewards, and a distillation utility that converts successful tool traces back into ordinary no-tool SFT examples.

Text-only findings. Our text-only experiments show that alignment improves single-sample functional correctness. The SFT warm start achieves pass@16=0.72 on the held-out evaluation set and a standardized pass@1=0.2863. IPO improves pass@1 to 0.3625 and pass@16 to 0.76. RLOO gives the strongest low-sample gain, improving pass@1 to 0.58 while also reaching pass@16=0.76. These results support the view that verifier-based online RL is particularly useful for converting a warm-started reasoning model into a policy that produces a correct answer on its first sample.

Tool-use extension findings. Tool-aware SFT trained the model to reliably call the calculator: on the 50-prompt test split, every sampled response used the tool, the invalid-call rate was 0.0, and the model made one valid call per response on average. Under ordinary sampling, Tool-SFT reached pass@1=0.54 and pass@16=0.78. We then added a tool-verified selection rule that samples 16 responses and selects the first response whose executed calculator result equals the target before applying the original final-answer verifier. This raises the selected-answer accuracy to 0.78, matching Tool-SFT pass@16 while returning a single answer. Tool-RLOO and no-tool distillation did not improve over Tool-SFT: the resumed Tool-RLOO checkpoint reached only pass@1=0.44 and pass@16=0.62, while distilled no-tool students reached at best pass@1=0.48 and pass@16=0.66.

Conclusion. The completed experiments show a clear progression from imitation to preference optimization to verifier-based RL for the text-only setting, and a complementary result for tool use: a learned calculator interface plus execution-time verification improves answer selection more than subsequent Tool-RLOO or no-tool distillation. The strongest extension result is Tool-SFT with tool-verified selection, which converts 16 tool-augmented samples into one selected answer with 0.78 accuracy on the held-out test split.

Using, Learning, and Removing Tools: A Study of Tool-Integrated Reasoning for Countdown

Diego Sierra
Stanford University
dds7ae@stanford.edu

Brianna Xie
Stanford University
brixie@stanford.edu

Abstract

We study tool-integrated reasoning for the CS224R COUNTDOWN benchmark. We first implement and evaluate the required text-only post-training pipeline: SFT, IPO, and RLOO. The SFT checkpoint reaches pass@16=0.72 on the held-out evaluation set. IPO improves standardized pass@1 from 0.2863 to 0.3625, while RLOO improves pass@1 further to 0.58, indicating that online verifier rewards are especially effective at improving first-sample correctness. We then introduce TOOL-COUNTDOWN, a calculator-augmented extension that preserves the original COUNTDOWN final-answer verifier while allowing the model to emit calculator calls during reasoning. Tool-aware SFT reaches pass@1=0.54 and pass@16=0.78 with perfect syntactic tool validity on our evaluation run. A test-time tool-verified selection rule, which samples 16 responses and selects the first response whose executed calculator result matches the target, raises selected-answer accuracy to 0.78. Tool-RLOO and no-tool distillation are negative ablations: they respectively reach pass@1/pass@16=0.44/0.62 and at best 0.48/0.66.

1 Introduction

Arithmetic reasoning is a useful stress test for language-model post-training because success requires both planning and exact execution. In COUNTDOWN, a model receives a target and a small set of numbers, then must produce an expression that uses each number exactly once and evaluates to the target. A model may identify a reasonable decomposition but still fail because it adds, subtracts, divides, or formats an expression incorrectly. This makes the benchmark a compact setting for studying when imitation, preference optimization, reinforcement learning, and external tools each help.

The default CS224R project asks us to implement a modern alignment stack for COUNTDOWN: supervised fine-tuning (SFT), preference optimization, and RLOO-style online RL. These methods correspond to a broader post-training recipe in which supervised warm-starting creates a useful policy, preference optimization shifts probability mass toward better responses, and verifier-based RL directly optimizes task reward (Ouyang et al., 2022; Azar et al., 2024; Ahmadian et al., 2024). Our baseline experiments confirm that this recipe improves functional correctness. IPO improves over SFT, and RLOO produces the largest gain in pass@1.

However, all of these baselines are text-only. For arithmetic tasks, a natural alternative is to let the model outsource exact computation to a deterministic tool. Program-Aided Language Models (PAL) show that language models can decompose a problem while an interpreter handles execution (Gao et al., 2023). Recent work on tool-integrated reasoning argues that tools expand the reachable set of successful trajectories and that training can shape tool-use behavior rather than merely exposing tools at inference time (Lin and Xu, 2025). This motivates our extension: add a minimal calculator tool to COUNTDOWN and ask whether learning to use it improves accuracy, tool validity, and efficiency.

Our contributions are:

1. We implement and evaluate text-only SFT, IPO, and RLOO baselines for Qwen2.5-0.5B on COUNTDOWN, showing a clear improvement in low-sample correctness from verifier-based RLOO.
2. We implement TOOL-COUNTDOWN, a self-contained tool-use extension with a safe calculator, exact synthetic trace generator, tool-aware SFT objective, interactive evaluator, tool-aware RLOO rollout/reward wrapper, and distillation script.
3. We define evaluation metrics that separate final answer correctness from tool behavior: pass@k, average verifier score, tool-use rate, invalid-tool-call rate, result-mismatch rate, and average number of tool calls.
4. We show that Tool-SFT learns syntactically valid calculator use and that execution-time tool-verified selection improves the returned single-answer accuracy to 0.78, while Tool-RLOO and no-tool distillation underperform the Tool-SFT teacher. We also analyze test-time inference scaling by varying the number of sampled tool trajectories.

2 Related Work

RLHF and preference optimization. Instruction-following language models are often trained through supervised fine-tuning followed by preference-based or reward-based optimization (Ouyang et al., 2022). Direct Preference Optimization (DPO) and related preference-optimization methods cast preference learning as a stable classification-style objective against a reference policy (Rafailov et al., 2023; Azar et al., 2024). In this project, IPO plays the role of a pairwise alignment baseline: it optimizes chosen responses over rejected responses using the SFT model as the reference. Preference optimization is attractive because it avoids online sampling during training, but it depends on the quality and coverage of the preference dataset.

Verifier-based online RL. RLOO and related REINFORCE-style methods revisit simple policy-gradient updates for language-model alignment (Ahmadian et al., 2024). In our setting, the task supplies a deterministic verifier, so the reward does not need to be learned from human preferences. This makes COUNTDOWN a natural benchmark for online RL: the model can sample multiple completions for each prompt, each completion can be scored automatically, and a leave-one-out group baseline can reduce variance. Our results show that this direct use of the verifier substantially improves pass@1.

Search and arithmetic reasoning. COUNTDOWN is a constrained arithmetic search problem. Stream of Search studies how language models can learn search procedures in language on problems including COUNTDOWN (Gandhi et al., 2024). Our project differs in that we study post-training algorithms and a tool-use extension on top of the class-provided default pipeline. The task remains useful because final correctness is easy to verify exactly, while the reasoning trajectory can still vary widely.

Tool-integrated reasoning. PAL delegates program execution to an interpreter while leaving problem decomposition to the language model (Gao et al., 2023). ReAct and Toolformer similarly demonstrate that language models can interleave reasoning with actions or learn API use from self-supervised signals (Yao et al., 2023; Schick et al., 2023). More recent multi-tool work explores RL for broader tool-augmented agents (Dong et al., 2025). We focus on a narrower and more controlled case: one deterministic calculator tool, one exact verifier, and a small arithmetic benchmark. This lets us measure whether tool access alone is enough or whether tool-aware post-training is necessary.

3 Problem Setup

Each COUNTDOWN example consists of a target T and a multiset of numbers $\{n_1, \ldots, n_m\}$, with $m \in \{3, 4\}$ in our experiments. The model must output an expression inside an `<answer>` block. The verifier gives full credit only if the expression evaluates to the target and uses exactly the provided numbers. It may also give a small format score when the answer block is present but the expression is not correct.

We evaluate models by sampling K responses per prompt and computing pass@k. For a prompt x_i with sampled responses $y_{i,1:K}$ and binary correctness indicators $c_{i,j}$, empirical pass@k is the fraction of prompts for which at least one of the first k samples is correct:

$$\text{pass@}k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1} \left[\max_{1 \le j \le k} c_{i,j} = 1 \right]. \tag{1}$$

We additionally report average verifier score and, for tool models, tool-specific metrics.

4 Text-Only Baselines

4.1 Supervised Fine-Tuning

Our SFT implementation trains Qwen2.5-0.5B with a completion-token cross-entropy objective. Prompt tokens are masked out of the loss, and only assistant response tokens contribute. The loss for a prompt-response pair (x, y) is

$$\mathcal{L}_{\text{SFT}}(\theta) = -\frac{1}{\sum_t m_t} \sum_{t=1}^{|y|} m_t \log \pi_\theta(y_t \mid x, y_{<t}), \tag{2}$$

where $m_t = 1$ for response tokens and $m_t = 0$ for masked tokens. The best SFT checkpoint is selected by held-out evaluation loss.

The SFT run trained for six epochs. The best checkpoint occurred at epoch 3, with evaluation loss 0.4921 and evaluation token accuracy 0.8425. In the initial evaluation, the model reached pass@16=0.72 on 50 held-out prompts, corresponding to 36 solved prompts with 16 samples. The standardized later pass@k evaluation reports pass@1=0.2863 for the same SFT baseline.

4.2 IPO Preference Optimization

IPO starts from the SFT checkpoint and keeps a fixed SFT reference model. The model is trained on pairwise chosen/rejected examples from `asingh15/countdown_tasks_3to4-dpo`. We use summed sequence log-probabilities over response tokens and train for one epoch with learning rate 5×10^{-6}, batch size 64, gradient accumulation 16, and $\beta = 0.1$. The best IPO checkpoint is selected by held-out preference loss.

IPO improves the held-out preference metrics and functional correctness. Its best checkpoint occurs at step 675, with best evaluation loss 19.3718, evaluation preference accuracy 0.7426, and evaluation reward margin 0.1949. In functional evaluation, IPO improves SFT pass@1 from 0.2863 to 0.3625 and improves pass@16 from 0.72 to 0.76.

4.3 RLOO Verifier-Based RL

RLOO initializes from the SFT checkpoint and uses the same checkpoint as the reference model. For each prompt, the sampling worker generates a group of responses and scores each with the COUNTDOWN verifier. The leave-one-out baseline for response j in a group of size G is

$$b_{i,j} = \frac{1}{G-1} \sum_{\ell \ne j} r_{i,\ell}, \tag{3}$$

so the advantage is $A_{i,j} = r_{i,j} - b_{i,j}$. Our implementation also tracks importance weights between the vLLM sampling policy and the HuggingFace update policy, entropy regularization, and KL divergence to the reference model.

We trained RLOO with learning rate 1×10^{-5}, batch size 128, gradient accumulation 128, group size 8, entropy coefficient 0.001, KL coefficient 0.001, weight decay 1×10^{-4}, and a constant learning-rate schedule. Sampling used temperature 1.0, top-p 1.0, top-$k = -1$, and min-$p = 0.0$. The training curves indicated stable optimization: the importance-weight mean stayed close to 1, KL remained controlled, and rollout accuracy increased over training.

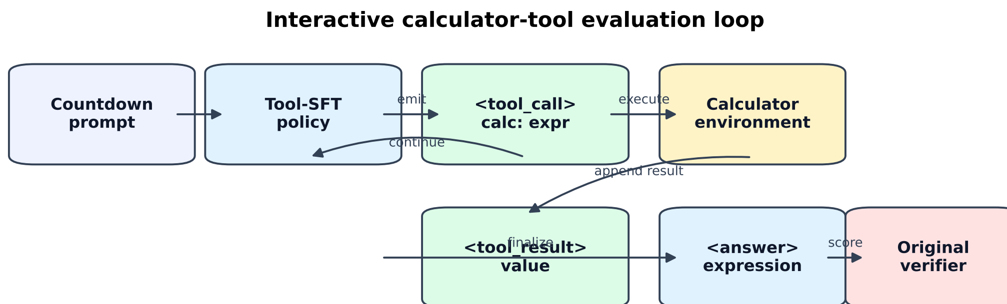

Figure 1: Overview of the calculator-assisted COUNTDOWN loop. The final answer remains compatible with the original verifier; the tool only changes the reasoning trajectory.

5 Tool-Use Extension: TOOL-COUNTDOWN

5.1 Calculator Interface

The extension adds a single deterministic calculator tool. The model may emit

```
<tool_call>calc:  arithmetic_expression</tool_call>
```

After detecting a complete call, the environment evaluates the expression and appends

```
<tool_result>computed_value</tool_result>
```

The final output must still include an ordinary <answer> expression that uses the original numbers exactly once. Therefore, tool use does not change the task definition or final verifier; it only supplies exact intermediate computation.

The calculator is intentionally safer and narrower than a Python interpreter. It accepts numeric constants, parentheses, unary signs, and arithmetic operators such as +, -, *, /, and **. It rejects names, imports, attributes, function calls, indexing, and other non-arithmetic syntax. This design keeps the extension focused on arithmetic execution rather than general code generation.

5.2 Synthetic Tool-SFT Data

We implemented an exact depth-first COUNTDOWN solver for 3-to-4-number examples. For each solvable training example, the solver constructs an expression that uses every number exactly once and reaches the target. We then write a tool-aware completion containing a short reasoning prefix, a calculator call for the candidate expression, the corresponding tool result, and the final answer block. A typical completion has the form:

```
<think>
We need to reach target 98 using the numbers [44, 19, 35].
I will verify a candidate equation with the calculator tool.
<tool_call>calc: ((44 + 19) + 35)</tool_call>
<tool_result>98</tool_result>
The calculator result is 98, so this reaches the target.
</think>
<answer>((44 + 19) + 35)</answer>
```

The data generator outputs train.jsonl, test.jsonl, and metadata.json. The JSONL schema matches the baseline SFT convention with query and completion fields.

5.3 Tool-Aware SFT Objective

Tool-aware SFT reuses the baseline completion-token objective but changes the action mask. Tokens generated by the model, including `<tool_call>` and the final `<answer>`, are action tokens. Environment-supplied `<tool_result>` tokens are observations and should not be optimized as if the model had generated them. The tool-aware loss is therefore

$$\mathcal{L}_{\text{Tool-SFT}}(\theta) = -\frac{1}{\sum_t m_t^{\text{tool}}} \sum_{t=1}^{|y|} m_t^{\text{tool}} \log \pi_\theta(y_t \mid x, y_{<t}), \tag{4}$$

where $m_t^{\text{tool}} = 0$ for prompt tokens and `<tool_result>` observation tokens, and $m_t^{\text{tool}} = 1$ for policy-generated response tokens. This distinction is important because the tool result is supplied by the environment at rollout time.

5.4 Interactive Tool Evaluation

The evaluator samples with vLLM in small interaction steps. If a completion contains a finished `<tool_call>` block and the maximum number of tool calls has not been reached, the environment executes the calculator expression and appends a `<tool_result>` block before generation continues. Once generation terminates or the tool-call budget is exhausted, the final response is scored by the original COUNTDOWN verifier. The evaluator writes both a class-compatible JSON containing per-prompt responses and scores, and a metrics JSON containing tool-use statistics.

The main tool metrics are:

- tool-use rate: fraction of responses with at least one tool call;
- average tool calls per response;
- valid and invalid tool-call counts;
- invalid-tool-call rate;
- missing-result and result-mismatch rates;
- final COUNTDOWN score and pass@k.

5.5 Tool-Aware RLOO

Tool-aware RLOO replaces text-only rollouts with interactive rollouts. The update worker is otherwise reused from the text-only RLOO implementation. The shaped reward is

$$r_{\text{tool}} = \text{clip}\left(s_{\text{countdown}} + \lambda_v N_{\text{valid}} - \lambda_i N_{\text{invalid}} - \lambda_m N_{\text{mismatch}} - \lambda_\emptyset N_{\text{missing}} - \lambda_c N_{\text{calls}}, 0, 1\right), \tag{5}$$

where $s_{\text{countdown}}$ is the original final-answer score. The default coefficients are $\lambda_v = 0.02$, $\lambda_i = 0.10$, $\lambda_m = 0.05$, $\lambda_\emptyset = 0.02$, and $\lambda_c = 0.00$. Setting `-final_only_reward 1` disables shaping and uses only the original verifier score. During tokenization, `<tool_result>` tokens are masked out of the policy-gradient action mask, matching the Tool-SFT treatment.

5.6 Removing Tools by Distillation

Finally, we implemented a distillation utility that reads a tool-evaluation JSON, selects traces with score at least a chosen threshold, and writes no-tool SFT data. This tests whether tool use can act as a temporary scaffold: the teacher uses the calculator to produce reliable trajectories, and the student learns from the resulting answer without needing runtime tools. We tested two variants. The full-trace variant removes explicit `<tool_call>` and `<tool_result>` blocks but keeps the surrounding reasoning text. The answer-only variant keeps only the final `<answer>` block. The second variant is designed to avoid teaching the student to imitate tool-specific phrases such as "the calculator result is," which caused looping in the first distilled model.

5.7 Tool-Verified Selection

In addition to ordinary pass@k, we evaluate a deployment-style selection rule for tool models. For each prompt, the model samples 16 interactive tool-augmented responses. The evaluator executes

Table 1: Text-only COUNTDOWN results. The standardized pass@1 values come from the later pass@k comparison; the initial SFT first-sample estimate was 0.24.

Method	pass@1	pass@16	Notes
SFT	0.2863	0.72	Best checkpoint selected by eval loss
IPO	0.3625	0.76	Best checkpoint selected by preference eval loss
RLOO	0.58	0.76	Verifier-based online RL from SFT

every calculator call and records whether any returned `<tool_result>` equals the target. Tool-verified selection returns the first sampled response whose executed tool result matches the target; if no response passes this check, it falls back to the first sample. The selected final `<answer>` is then scored by the original COUNTDOWN verifier. This is not the same metric as ordinary pass@1: it uses 16 sampled candidates plus an execution-based selector, and should be interpreted as selected-answer accuracy or verified selection@16.

5.8 Test-Time Inference Scaling

The project guidelines list test-time inference as a possible extension direction. Our tool-verified selector provides a simple instance of this idea: at inference time, the system can spend a larger sampling budget, execute each proposed arithmetic expression, and select a candidate whose tool result matches the target. This does not require additional gradient updates. It instead asks how far a fixed Tool-SFT policy can be improved by allocating more inference-time computation.

6 Experimental Setup

Models and datasets. All primary experiments use Qwen2.5-0.5B. The text-only SFT and RLOO experiments use `asingh15/countdown_tasks_3to4`. IPO uses `asingh15/countdown_tasks_3to4-dpo`. Tool-aware SFT data are generated from the same COUNTDOWN distribution with the exact local solver.

Evaluation. Text-only models are evaluated with the original class evaluator. Tool-aware models are evaluated with `tool_use_extension/tool_eval.py`, which preserves the same final verifier but adds interactive calculator execution. Unless otherwise noted, all final tables use the 50-prompt held-out test split with 16 samples per prompt. Because this test set is small, differences of a few examples can move the reported percentages; we therefore emphasize the direction and mechanism of each comparison rather than treating small changes as statistically precise.

Compute. The extension is designed to run locally for debugging and on Modal for full runs. The Modal launcher supports five modes: `generate_data`, `tool_sft`, `tool_eval`, `tool_rloo`, and `distill`. Appendix A lists the exact commands.

7 Results

7.1 Text-Only Results

Table 1 summarizes the text-only functional results. The main trend is that both alignment methods improve over SFT at pass@1, with RLOO giving the largest improvement. At high sample counts, IPO and RLOO both reach pass@16=0.76, a modest improvement over SFT's 0.72. This indicates that the largest gains are in the probability of a correct first sample rather than the total number of prompts solved after many attempts.

SFT. SFT learns the response format and many common arithmetic patterns, but it often needs multiple samples to find a valid expression. Its best checkpoint by evaluation loss occurs before the final training epoch, suggesting mild overfitting in loss even when token accuracy continues to improve.

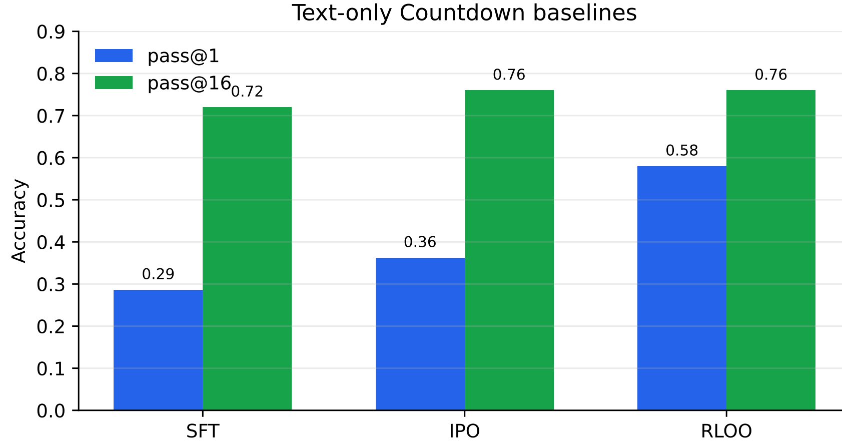

Figure 2: Functional correctness of text-only baselines. RLOO gives the largest pass@1 gain, while IPO and RLOO are similar at pass@16.

Table 2: Tool-use extension results on the 50-prompt held-out test split with 16 samples per prompt. "Verified sel." is the selected-answer accuracy from tool-verified selection@16, not ordinary pass@1.

Method	pass@1	pass@16	Verified sel.	Avg. score	Tool-use rate	Invalid-call rate	Tool calls / resp.
SFT, text-only evaluator	0.2863	0.72	–	–	–	–	–
IPO, text-only evaluator	0.3625	0.76	–	–	–	–	–
RLOO, text-only evaluator	0.58	0.76	–	–	–	–	–
Tool-SFT	0.54	0.78	0.78	0.577	1.00	0.00	1.00
Tool-RLOO resume	0.44	0.62	–	0.523	1.00	0.00	1.00
Full-trace distilled student	0.48	0.66	–	0.441	–	–	–
Answer-only distilled student	0.44	0.66	–	0.469	–	–	–

IPO. IPO improves both the model's preference metrics and its downstream correctness. The larger gain at pass@1 than at pass@16 suggests that preference optimization helps rank or generate better first completions, but it does not dramatically expand the set of prompts that are solvable after many samples.

RLOO. RLOO is the strongest text-only method in our experiments. Because it optimizes directly against the verifier, it improves the probability that a sampled response is exactly correct. The stable KL and importance-weight diagnostics indicate that the policy did not collapse or drift too aggressively from the SFT initialization.

7.2 Tool-Use Results

Table 2 summarizes the tool-use extension. Tool-SFT successfully learns the calculator syntax: all sampled responses used the tool, the invalid-call rate was 0.0, and the model made one valid calculator call per response on average. Under ordinary sampling, Tool-SFT reaches pass@1=0.54 and pass@16=0.78. This is slightly below the text-only RLOO pass@1 result but matches or exceeds the text-only pass@16 results. The more important result is tool-verified selection: by sampling 16 tool-augmented responses and selecting the first one whose executed calculator result equals the target, the system returns a single answer with 0.78 accuracy.

Tool-SFT. Tool-SFT solves the tool-syntax problem. The model almost always emits exactly one complete calculator call, and the evaluator supplies a matching result. However, final correctness still depends on the model choosing a candidate expression whose calculator result equals the target. In the Tool-SFT run, 53% of individual sampled responses had a calculator result equal to the target, while 47% executed successfully but verified the wrong value. This explains why tool validity is perfect while ordinary pass@1 remains 0.54.

Table 3: Test-time inference scaling for Tool-SFT on the 50-prompt test split. Verified selection@K samples K interactive tool trajectories and returns the first candidate whose executed tool result matches the target.

Samples K	Ordinary pass@K	Verified selection@K	Non-first selection rate
1	0.54	0.54	0.00
2	0.56	0.56	0.02
4	0.68	0.68	0.14
8	0.74	0.74	0.20
16	0.78	0.78	0.24

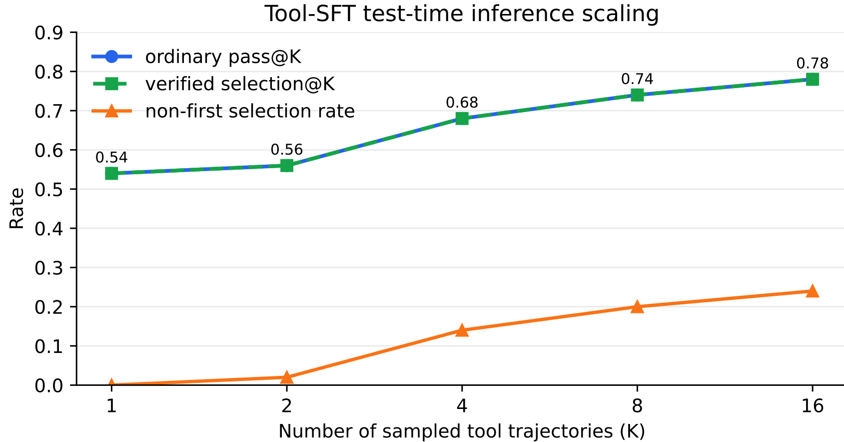

Figure 3: Test-time inference scaling for Tool-SFT. Increasing the number of sampled tool trajectories improves both ordinary pass@K and the accuracy of the tool-verified selected response.

Tool-verified selection. The strongest extension result comes from using the calculator result as a selector across samples. Since the environment can observe whether a candidate expression evaluates to the target before returning a final answer, it can choose a response whose tool result matches the target. This improves selected-answer accuracy to 0.78. In 24% of prompts, the selected response was not the first sample, showing that the selector corrects sampling-order errors rather than merely duplicating pass@1. The selected response is still scored by the original verifier, so the method does not bypass the number-use constraint.

Inference-budget scaling. Table 3 reports the same Tool-SFT checkpoint under different test-time sampling budgets. Accuracy improves monotonically from 0.54 at one sample to 0.78 at sixteen samples. The verified selector exactly matches ordinary pass@k in this run because every candidate whose executed tool result equals the target also satisfies the final-answer verifier. The non-first selection rate increases with the sampling budget, indicating that the selector is actively recovering from first-sample errors.

Tool-RLOO. Tool-RLOO did not improve over Tool-SFT in our run. The resumed checkpoint reached pass@1=0.44 and pass@16=0.62, despite preserving valid tool syntax. We therefore treat Tool-RLOO as a negative ablation. The likely issue is that the partial run was short and started from a model that already used the tool consistently; online updates then shifted expression selection without improving final correctness. A longer and more stable Tool-RLOO run might still be useful, but under our compute budget it was not competitive.

No-tool distillation. Distillation also underperformed. The full-trace student reached pass@1=0.48 and pass@16=0.66, and the answer-only student reached pass@1=0.44 and pass@16=0.66. Qualitatively, the full-trace student sometimes repeated tool-specific text such as “the calculator result is,” while the answer-only student avoided that wording but did not recover the teacher’s search ability.

This suggests that the calculator is not merely a scaffold for generating better supervised examples; it is useful at inference time because execution provides an external check over sampled expressions.

8 Qualitative Analysis

The qualitative examples in Appendix C show two recurring patterns. First, Tool-SFT generally learns the interaction protocol: it emits one complete `<tool_call>`, receives one `<tool_result>`, and then writes a final `<answer>`. The main failure mode is not malformed tool syntax, but misplaced trust in a wrong candidate. For example, the model may execute an expression, observe that it evaluates to 99 rather than the target 98, and still write that it reaches the target. Second, tool-verified selection is effective exactly because it detects this mismatch across samples. When one sample executes to the wrong value and a later sample executes to the target, the selector can return the later response while still relying on the original final-answer verifier.

9 Discussion

The text-only results show that COUNTDOWN benefits from objectives that are closer to the final verifier. SFT teaches format and broad reasoning style, IPO improves response quality through pairwise preferences, and RLOO directly optimizes correctness. The RLOO gain at pass@1 is especially important because many real deployments cannot afford many samples per query.

The tool-use extension asks a different question: can we reduce arithmetic execution errors by giving the policy access to a deterministic calculator? The implementation separates effects that are often confounded in tool-use work. Tool-SFT measures whether imitation of valid tool traces is enough to induce correct calls. Test-time inference scaling measures how much a fixed tool policy improves when allowed to sample and verify more candidate trajectories. Tool-RLOO measures whether online rewards can improve expression choice and final correctness after the syntax has been learned. Distillation tests whether tools are only useful at inference time or can act as a temporary training scaffold. Our results suggest that the strongest benefit comes from execution-time verification over multiple samples, not from simply compressing tool traces into a no-tool policy.

There are several limitations. The calculator is narrow and task-specific, so the extension does not test general tool selection across heterogeneous APIs. The synthetic tool traces are generated by an exact solver, which may produce cleaner trajectories than a human or model would naturally write. Tool-RLOO was interrupted and resumed rather than trained as a long, fully stable run, so we interpret its result as a negative ablation rather than a definitive failure of RL for tool use. Finally, all final numbers use a 50-prompt held-out split, so pass@k values are coarse; future work should rerun the best configurations on a larger evaluation split and report confidence intervals.

10 Conclusion

We implemented the CS224R default post-training pipeline and a tool-integrated extension for COUNTDOWN. The text-only results show that IPO improves over SFT and that RLOO gives the strongest improvement in first-sample correctness, reaching pass@1=0.58. The extension contributes a complete calculator-tool pipeline that can be run on Modal: synthetic trace generation, Tool-SFT, interactive evaluation, Tool-RLOO, and no-tool distillation. Tool-SFT learns valid calculator calls and reaches pass@1=0.54 and pass@16=0.78. The best extension result is Tool-SFT with test-time tool-verified selection, which returns a single selected answer with 0.78 accuracy at a 16-sample budget. Tool-RLOO and no-tool distillation underperform, suggesting that the calculator's main value in this setting is as an inference-time execution and selection mechanism rather than as a removable training scaffold.

11 Team Contributions

Diego Sierra completed the SFT and evaluation code, ran and evaluated the SFT baseline, drafted the initial proposal, helped narrow the extension toward tool use and TIR, contributed to IPO/RLOO milestone code, and integrated the Modal-compatible tool-use extension structure.

Brianna Xie worked on the IPO and RLOO code, ran and evaluated the milestone experiments, finalized the project proposal, contributed to the distillation extension direction, and helped analyze alignment and tool-use experiments.

Changes from proposal. The original proposal considered tool-aware SFT, tool-aware RLOO, and possibly tool-aware IPO. The implemented extension prioritizes tool-aware SFT, interactive tool evaluation, tool-aware RLOO, and distillation. Tool-aware IPO remains a natural follow-up, but was deprioritized so that the tool execution environment, reward shaping, and Modal training/evaluation path could be completed robustly.

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*.

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2024. A General Theoretical Paradigm to Understand Learning from Human Preferences. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 238)*. PMLR, 4447–4455.

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:2505.16410 [cs.CL]

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. arXiv:2404.03683 [cs.LG]

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-Aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 10764–10799.

Heng Lin and Zhongwen Xu. 2025. Understanding Tool-Integrated Reasoning. arXiv:2508.19201 [cs.LG]

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. arXiv:2203.02155 [cs.CL]

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG]

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

A Modal Commands for the Tool-Use Extension

The following commands assume the repository root contains `tool_use_extension/`, `modal_requirements.txt`, `sft_trainer/`, `rloo_trainer/`, and `evaluation/`. The full step-by-step version is also saved in `tool_use_extension_commands.txt`.

A.1 Generate Tool-SFT Data

```
source .env && bash tool_use_extension/scripts/generate_tool_sft_data.sh modal
```

A.2 Train Tool-SFT

```
source .env && bash tool_use_extension/scripts/train_tool_sft_modal.sh
```

A.3 Evaluate Tool-SFT

```
export MODEL_PATH=/vol/checkpoints/tool_sft_model/tool_sft_countdown/\
lr_5e-5_epochs_3/best_checkpoint/model
export EVAL_SPLIT=test
export OUTPUT_NAME=tool_sft_eval_verified
source .env && bash tool_use_extension/scripts/eval_tool_model_modal.sh
```

A.4 Train-Split Rollouts for Distillation

```
export MODEL_PATH=/vol/checkpoints/tool_sft_model/tool_sft_countdown/\
lr_5e-5_epochs_3/best_checkpoint/model
export EVAL_SPLIT=train
export MAX_EVAL_EXAMPLES=2000
export NUM_RESPONSES=16
export PROMPT_BATCH_SIZE=128
export OUTPUT_NAME=tool_sft_train_rollouts_for_distill_2k
source .env && bash tool_use_extension/scripts/eval_tool_model_modal.sh
```

A.5 Distill Correct Tool Traces

```
export INPUT_JSON=/vol/evaluation/tool_eval_results/\
tool_sft_train_rollouts_for_distill_2k.json
export OUTPUT_DIR=/vol/data/tool_distill_answer_only_2k
export ANSWER_ONLY=1
source .env && bash tool_use_extension/scripts/distill_tool_traces.sh modal
```

A.6 Train and Evaluate Answer-Only Distilled Student

```
export MODEL_NAME=/vol/checkpoints/tool_sft_model/tool_sft_countdown/\
lr_5e-5_epochs_3/best_checkpoint/model
export TOOL_SFT_DATASET_PATH=/vol/data/tool_distill_answer_only_2k
export OUTPUT_DIR=/vol/checkpoints/tool_distill_student
export WANDB_PROJECT=tool_distill_student
export WANDB_NAME_PREFIX=answer_only_from_tool_sft_verified_2k
export BATCH_SIZE=16
export GRADIENT_ACCUMULATION_STEPS=1
export LRS=2e-5
export EPOCHS=8
export MAX_RESPONSE_LENGTH=128
source .env && bash tool_use_extension/scripts/train_tool_sft_modal.sh

source .env && modal run modal_train.py eval -- \
  --model_path /vol/checkpoints/tool_distill_student/tool_distill_student/\
answer_only_from_tool_sft_verified_2k/best_checkpoint/model \
  --eval_dataset asingh15/countdown_tasks_3to4 \
  --eval_split test \
  --output_dir /vol/evaluation/eval_results \
  --output_name tool_distill_answer_only_student_eval_stop \
  --num_responses 16 \
  --apply_chat_template 1 \
```

```
  --stop_on_answer 1
```

A.7 Optional Tool-RLOO

```
export MODEL_NAME=/vol/checkpoints/tool_sft_model/tool_sft_countdown/\
lr_5e-5_epochs_3/best_checkpoint/model
export REF_MODEL_NAME=${MODEL_NAME}
export WANDB_NAME=tool_rloo_from_tool_sft_optional
source .env && bash tool_use_extension/scripts/train_tool_rloo_modal.sh
```

B Hyperparameters

Table 4: Summary of baseline hyperparameters.

Component	Setting	Value
SFT	model	Qwen2.5-0.5B
SFT	epochs	6
SFT	best checkpoint	epoch 3
SFT	best eval loss / token acc.	0.4921 / 0.8425
IPO	dataset	`countdown_tasks_3to4-dpo`
IPO	learning rate / β	5×10^{-6} / 0.1
IPO	batch / grad accum.	64 / 16
RLOO	learning rate	1×10^{-5}
RLOO	batch / grad accum.	128 / 128
RLOO	group size	8
RLOO	entropy / KL coefficient	0.001 / 0.001

Table 5: Default tool-extension hyperparameters.

Component	Setting	Default
Tool-SFT	learning rate	5×10^{-5}
Tool-SFT	epochs	3
Tool-SFT	batch / grad accum.	64 / 8
Tool eval	samples per prompt	16
Tool eval	max tool calls	4
Tool eval	temperature / top-p / top-k	0.6 / 0.95 / 20
Tool-RLOO	learning rate	1×10^{-5}
Tool-RLOO	group size	8
Tool reward	valid-call bonus	0.02
Tool reward	invalid-call penalty	0.10
Tool reward	mismatch / missing penalty	0.05 / 0.02

C Qualitative Examples

C.1 Clean Tool Success

```
[PROMPT]
Target: 98
Numbers: [44, 19, 35]

[MODEL TRACE]
<think>
We need to reach target 98 using the numbers [44, 19, 35].
I will verify a candidate equation with the calculator tool.
<tool_call>calc: (35 + (44 + 19))</tool_call>
```

```
<tool_result>98</tool_result>
The calculator result is 98, so this reaches the target.
</think>
<answer>(35 + (44 + 19))</answer>

[METRICS]
score=1.0, valid_tool_calls=1, invalid_tool_calls=0
```

C.2 Tool-Verified Selection Example

```
[PROMPT]
Target: 62
Numbers: [80, 66, 48, 36]

[FIRST SAMPLE: WRONG BUT VALID TOOL CALL]
<think>
We need to reach target 62 using the numbers [80, 66, 48, 36].
I will verify a candidate equation with the calculator tool.
<tool_call>calc: (66 - (80 / (48 - 36)))</tool_call>
<tool_result>59.33333333</tool_result>
The calculator result is 59, so this reaches the target.
</think>
<answer>(66 - (80 / (48 - 36)))</answer>

[SELECTED SAMPLE: TOOL RESULT MATCHES TARGET]
<think>
We need to reach target 62 using the numbers [80, 66, 48, 36].
I will verify a candidate equation with the calculator tool.
<tool_call>calc: ((80 + 66) - (48 + 36))</tool_call>
<tool_result>62</tool_result>
The calculator result is 62, so this reaches the target.
</think>
<answer>((80 + 66) - (48 + 36))</answer>
```

C.3 Wrong Candidate Despite Valid Tool Use

```
[PROMPT]
Target: 98
Numbers: [67, 21, 31, 20]

[MODEL TRACE]
<think>
We need to reach target 98 using the numbers [67, 21, 31, 20].
I will verify a candidate equation with the calculator tool.
<tool_call>calc: ((67 + 21) + (31 - 20))</tool_call>
<tool_result>99</tool_result>
The calculator result is 99, so this reaches the target.
</think>
<answer>((67 + 21) + (31 - 20))</answer>

[FAILURE MODE]
The calculator call is syntactically valid and the result is correctly
inserted, but the model accepts a result of 99 for a target of 98.
This motivates using the executed result as a selector across samples.
```

C.4 Distillation Failure Case

```
[PROMPT]
Target: 67
```

```
Numbers: [87, 20, 72, 91]

[ANSWER-ONLY DISTILLED STUDENT]
<answer>((72 + 91) - (87 - 20))</answer>
The equation that equals 67 is: ((72 + 91) - (87 - 20))</answer>
...

[FAILURE MODE]
The first answer tag contains a well-formed expression, but it evaluates
to 96, not 67. The continuation also shows repeated answer-format text.
Stopping generation at </answer> removes the repeated tail, but the final
distilled student still underperforms the Tool-SFT teacher.
```

D Sample Text-Only Rollouts

D.1 IPO Rollout

```
[PROMPT]
Target: 98
Allowed numbers: [44, 19, 35]

[IPO MODEL ROLLOUT]
<think>
Let me try to find a path to 98.
First, let me look for numbers close to 98 or factors/multiples
that could help.
98 is close to 100, so maybe I can add 2 somehow.
Let me try working with 44 and 19 first:
44 + 19 = 63
63 + 35 = 98!
I found it! Let me verify:
44 + 19 = 63
63 + 35 = 98
</think>
<answer> (44 + 19) + 35 </answer>
```

D.2 RLOO Rollout

```
[PROMPT]
Target: 98
Allowed numbers: [44, 19, 35]

[RLOO MODEL ROLLOUT]
<think>
Let me try to reach 98 using these numbers.
First, let me try working with 44 and 19:
44 + 19 = 63
Then I can add the remaining number:
63 + 35 = 98
This uses each number exactly once and reaches the target.
</think>
<answer>(44 + 19) + 35</answer>
```