Extended Abstract

Motivation Sub-1B-parameter language models are fast, lightweight, and well-suited for resource-constrained settings, but they often struggle with mathematical reasoning tasks that require exact arithmetic, search, and constraint satisfaction. We studied this challenge on the classic Countdown task. Initial experiments with SFT, IPO, and RLOO on Qwen-2.5-0.5B showed that even the best baseline achieved only around 50% pass@1, leaving substantial room for improvement. Error inspection suggested that the model especially struggled with problems involving division, factoring, and multi-step search, motivating our extension ideas.

Method Our extensions focus on two approaches: curriculum learning and inference-time iterative feedback. For curriculum learning, we divided training into three phases and categorized each problem into three difficulty levels based on whether division appears in the ground-truth equation and the number of operands (three or four). We relabeled the `cog_behav_all_strategies` dataset according to these criteria for SFT training and generated a new dataset from scratch for RLOO. In addition to the standard progressive setup, where each difficulty level is introduced sequentially across training phases, RLOO is also evaluated under a mixed setup in which the sampler draws examples from all three categories according to a probability distribution that gradually favors harder examples as the training progresses. For inference-time iterative feedback, a Qwen-2.5-7B-Instruct model is used as a critic to provide corrective advice on previous round's incorrect answers, allowing the model to learn from its own mistakes and make revisions.

Implementation For curriculum learning, we defined training phase boundaries by epochs for SFT and by optimization steps for RLOO. For SFT, we increased training from 6 to 10 epochs to ensure a comparable amount of data exposure during curriculum learning as in the non-curriculum baseline. For inference-time feedback, the pipeline is limited to a maximum of three solver attempts. To improve compute efficiency, we also incorporated verifier-gated early stopping, terminating immediately when a valid solution is found, and dynamic batching. We also refined the critic and revision prompts and tuned vLLM sampling temperatures to encourage stable, structured outputs and reduce formatting failures.

Results Our experiments demonstrated that SFT curriculum learning exceeded the baseline pass@1 by 3.375%, with gains in pass@k becoming larger as k increases. Conversely, RLOO curriculum learning with the progressive setup negatively affected overall pass@k, and the mixed setup yielded only modest gains for $k \geq 4$ when evaluated using the `countdown_eval.py` script. However, within the iterative feedback framework, all checkpoints exhibited consistent improvements in accuracy, with RLOO trained under progressive curriculum achieving the highest solve rate across all three attempts.

Discussion Our findings suggest that curriculum learning and inference-time feedback are useful but limited tools for improving reasoning in small language models. Curriculum learning is sensitive to the design of curriculum and the sampling schedule. The feedback loop is limited by the solver's ability to translate natural-language feedback into new arithmetic search behavior. We also observed failure modes such as repeated incorrect reasoning, prompt-template copying, and instability from stochastic vLLM generation on a small evaluation set.

Conclusion Our work provides evidence for the effectiveness of these two methods, while also highlighting the need for a more robust evaluation approach. Future work should include expanding evaluation sets, testing different sampling probabilities across curriculum learning phases, and adopting a more instruction-capable solver model to better understand and utilize feedback.

Improving Mathematical Reasoning in Small Language Models via Curriculum Learning and Iterative Execution Feedback

Jiayu Sui
Department of Statistics
Stanford University
echosui@stanford.edu

Xinyi Ai
Department of Statistics
Stanford University
xiai@stanford.edu

Abstract

In this project, we implemented three baseline RL methods: SFT, IPO, and RLOO to adapt the Qwen-2.5-0.5B base model for the Countdown task. After observing that the model could only attain a pass@1 of at most 52.75%, we adopted two extension paradigms: three-phase progressive and mixed curriculum learning, where training examples are dynamically sampled for each phase based on their difficulty level, and an iterative feedback loop where a Qwen-2.5-7B-Instruct model serves as a critic to provide corrective advice for previously incorrect answers. Iterative feedback showed consistent improvements in accuracy, whereas curriculum learning produced mixed results, suggesting the need for a more deterministic evaluation method to enable more robust comparisons.

1 Introduction

With the emergence of techniques like reinforcement learning and Chain of Thought (CoT) prompting Wei et al. (2023), the reasoning abilities of Large Language Models (LLMs) have significantly improved over the past few years. However, when it comes to math-related tasks, LLMs may suffer from hallucinations or inherent arithmetic limitations that prevent them from generating correct answers. This issue is particularly severe in resource-constrained settings, where latency or compute budgets necessitate the use of smaller, less capable language models.

After conducting experiments on the Countdown task using vanilla Supervised Fine-Tuning (SFT), Identity Preference Optimization (IPO), and REINFORCE Leave-One-Out (RLOO) with the Qwen-2.5-0.5B base model, we found that the results align with our hypothesis: the model could at best achieve around 50% pass@1 on the evaluation set, leaving ample room for future gains. Upon reviewing the sampled trajectories, we discovered that the model often struggles to formulate correct equations involving division or factoring. Therefore, we decided to adopt two distinct strategies: curriculum learning during training and an iterative feedback loop during inference, hoping that the model could better reason and solve this kind of problem with additional help.

2 Related Work

2.1 Curriculum Learning

The idea of curriculum learning was first introduced by Bengio et al. (2009), who suggested that, similar to how humans and animals learn, models may also benefit from training examples being presented in a meaningful order of increasing complexity rather than in a random sequence. This concept has been widely adopted in reinforcement learning to tackle a variety of problems, but how

to measure complexity and how to design an effective curriculum remains highly task-dependent even today. For instance, Gong et al. (2021) defined sample difficulty based on eigenvector density and leveraged an attention-weighted curriculum learning scheduler to enhance the model's intent detection capability. Likewise, Parashar et al. (2026) showed that scheduling mathematical tasks from easy to hard according to signals such as plan length, labeled difficulty, and operand count using cosine and Gaussian schedulers combined with the GRPO algorithm can effectively improve LLM reasoning. Moving away from schedulers, recent works by Zhang et al. (2025) and Wang et al. (2025) also explored the feasibility of dynamically adjusting the curriculum during training. The former re-orders data batches by estimated difficulty scores from low to high after each parameter update, while the latter increases the sampling probability of data from distributions with higher expected absolute advantage.

Since there are no absolute right or wrong answers, we would like to further investigate in this project whether other combinations of difficulty quantification and more easily constructed curriculum learning methods can work well with basic training approaches such as SFT, IPO, and RLOO.

2.2 Inference Time Iterative Feedback

Recent works have studied whether large language models can improve their outputs through iterative feedback and revision at inference time. Self-Refine Madaan et al. (2023) proposed a generate-feedback–refine loop in which a model first produces an initial answer, then generates feedback on its own output, and finally conditions on that feedback to produce an improved response. This framework showed that iterative refinement can improve outputs without additional supervised or reinforcement-learning updates. Similarly, Reflexion Shinn et al. (2023) introduced verbal reinforcement for language agents, where agents use linguistic feedback from previous failures to guide future attempts rather than updating model weights. Our feedback loop implementation followed this broad inference-time refinement paradigm: the solver's parameters remain fixed, and improvement is attempted by feeding prior failed attempts and natural-language feedback back into the model context.

A key limitation of pure self-correction is that language models may not reliably identify or repair their own reasoning errors without external grounding. Huang et al. (2024) found that intrinsic self-correction can fail on reasoning tasks and may even degrade performance when the model lacks external feedback. This motivated approaches that incorporate validation signals beyond the model's own judgment. CRITIC Gou et al. (2024) addressed this issue by allowing an LLM to interact with external tools to validate and progressively revise its outputs, emphasizing the importance of tool-grounded feedback for reliable correction. Our method was motivated by the same observation: for Countdown arithmetic, a deterministic verifier can reliably determine whether an answer uses the required numbers and evaluates to the target. We therefore used the verifier as the correctness gate and only invoke the critic when the solver's answer is incorrect.

Our work differs from prior refinement methods in two main ways. First, unlike Self-Refine, where the same model typically produces both feedback and refinement, we used a separate instruction-tuned critic model to diagnose errors in the solver's failed response. This creates a lightweight multi-agent setup in which the solver and critic play distinct roles. Second, unlike open-ended refinement tasks where correctness is difficult to define, Countdown provides an exact rule-based verifier. This let us terminate trajectories once a correct answer is found and evaluate each feedback round quantitatively through pass rate and attempts-to-success. Thus, our contribution is not a new training algorithm, but a verifier-gated, solver–critic inference-time correction framework for arithmetic reasoning with small language models.

3 Method

3.1 Countdown Task

In the countdown task, the model is given a list of numbers and is asked to combine them using the four basic arithmetic operations ($+, -, \times, /$) to form an expression that evaluates to a target value. Each number can be used at most once. The model is also required to place its reasoning process within `<think></think>` tags and return the final answer in `<answer></answer>` tags.

A verifier is then employed to assess the final answer. Expressions that match the target value receive a score of 1; those composed only of the allowed numbers but producing an incorrect result receive

a score of 0.1; and all other cases receive a score of 0. The resulting scores formed the basis for determining response correctness throughout the remainder of this work.

3.2 SFT

SFT is a training method that fine-tunes the model by optimizing the next-token prediction objective over given prompts x and expert completions y. Specifically, this is formulated as a cross-entropy loss computed on completion tokens:

$$\mathcal{L}^{\theta}_{\text{SFT}} = \mathbb{E}_{x,y\in\mathcal{D}}\left[-\sum_{t=1}^{|y|}\log \pi_\theta(y_t|x, y_{<t})\right]$$

SFT was performed on Qwen-2.5-0.5B base model using `Asap7772/cog_behav_all_strategies` dataset to provide a warm-start checkpoint for subsequent training.

3.3 IPO

IPO is a preference optimization algorithm that learns to maximize the reward margin $r_\theta(x, y_w) - r_\theta(x, y_l)$ between preferred and dispreferred answer pairs, where the reward r is implicitly defined by how much more likely the current policy is to produce a response compared to a fixed reference policy. The objective is defined as:

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

IPO was initialized from `asingh15/qwen-sft-countdown-defaultproj`, which is an SFT checkpoint shared with us, and trained on a preference dataset `asingh15/countdown_tasks_3to4-dpo`.

3.4 RLOO

RLOO is an online policy gradient estimator adapted from REINFORCE with a leave-one-out average reward baseline computed over multiple rollouts sampled per prompt to reduce gradient variance. Formally, it can be written as:

$$\nabla_\theta \mathcal{J} = \frac{1}{k}\sum_{i=1}^{k}\left[R(y_{(i)}, x) - \frac{1}{k-1}\sum_{j\neq i} R(y_{(j)}, x)\right]\nabla \log \pi(y_{(i)}|x) \text{ for } y_{(1)}, \ldots, y_{(k)} \overset{i.i.d}{\sim} \pi_\theta(\cdot|x)$$

Since trajectories were sampled from vLLM (viewed as a behavior policy) and gradient updates were performed using our fine-tuned model, we multiplied the advantage term by an importance weight ratio $w(y, x) = \exp(\log \pi_\theta(y|x) - \log \mu(y|x))$ to ensure the estimator remains unbiased.

Similar to IPO, RLOO was also trained on `asingh15/qwen-sft-countdown-defaultproj` with the dataset `asingh15/countdown_tasks_3to4`.

3.5 Curriculum Learning

The curriculum learning framework was built on top of two baseline methods, SFT and RLOO, with the same starting checkpoints. To incorporate the idea of gradually exposing the model to problems of increasing difficulty levels, we first defined the following rules to measure difficulty:

- If the equation constructed to reach the target contains no division: "level" = "easy"
- Else if the expression involves only three numbers: "level" = "medium"
- Otherwise: "level" = "hard"

For SFT, the training dataset `cog_behav_all_strategies` already includes a "completion" column that provides the final `<answer>`, so we can directly evaluate it to assign a "level" label to each row.

However, since the corresponding final equations are not available in `countdown_tasks_3to4` for RLOO, we synthetically generated a dataset by sampling 3 to 4 integers between 0 and 100 for each instance and combining them with randomly selected arithmetic operators to form candidate expressions (which can be used to determine difficulty later). We then retained only those examples whose evaluated results (targets) are integer values within [0, 100], ensuring that the data remain in-distribution relative to the original dataset. We manually controlled the dataset to consist of 30,000 examples for each difficulty level after the filtering process, and 50 of them were split off to form the test set.

Following this curriculum design, we divided the training into three equal-length phases based on the number of epochs for SFT and total training steps for RLOO. A sampler was introduced for each dataloader to select a fixed number of samples (set equal to the number of easy examples to avoid repeated sampling within a single iteration) from data pools depending on the current training phase.

The standard **progressive** setup is as follows: in phase one, the data pool contains only easy examples; phase two expands the pool to include medium examples, and phase three further incorporates all hard examples. Since the training set constructed for RLOO is more balanced than that used for SFT, we also conducted RLOO curriculum learning experiments under the second **mixed** setup in which the entire dataset is treated as a unified pool, and the sampler draws examples from each difficulty group according to a probability distribution that gradually favors harder examples as training progresses. By doing so, it prevents the model from experiencing abrupt distribution shifts and their potential adverse impact on training. The sampling probabilities were set to (0.75, 0.20, 0.05) in phase one, (0.40, 0.40, 0.20) in phase two, and (0.20, 0.30, 0.50) in phase three.

3.6 Inference Time Iterative Feedback

The core idea behind inference-time solver-critic feedback loop is to improve reasoning at test time by using a separate critic model to diagnose failed solver outputs and provide natural-language feedback for revision. This procedure is formalized in Algorithm 1.

Algorithm 1 Verifier-Gated Solver-Critic Feedback Loop

Require: Problem x, solver S, critic C, verifier V, maximum attempts $T = 3$
history $\leftarrow [\,]$
for $t = 1$ to T **do**
  if $t == 1$ **then**
    prompt $\leftarrow$ original problem prompt
  else
    prompt $\leftarrow$ problem + previous solver response + critic feedback
  end if
  $y_t \leftarrow S(\textit{prompt})$
  $\textit{score}_t \leftarrow V(x, y_t)$
  Append $(y_t, \textit{score}_t)$ to *history*
  if $\textit{score}_t == 1$ **then**
    Mark trajectory as solved at attempt t
    return *history*
  end if
  if $t < T$ **then**
    $\textit{feedback}_t \leftarrow C(x, y_t)$
    Append $\textit{feedback}_t$ to *history*
  end if
end for
return *history*

A key design choice is verifier-gated early stopping. The critic is only invoked after incorrect solver attempts, which avoids spending extra compute on already-correct examples and makes the number of feedback rounds interpretable as an efficiency metric. We also used dynamic batching: after each round, solved examples are removed from the active batch, and only unsolved examples proceed to the next critic and solver call.

The critic prompt is structured to avoid unconstrained commentary. It asks the critic to inspect the final proposed expression, check format, verify number usage, evaluate arithmetic, identify the main error, and provide one actionable correction suggestion. We explicitly instruct the critic not to provide a complete final expression.

The implementation logs every trajectory in full, including the problem, solver prompts, solver responses, verifier scores, critic prompts, critic feedback, whether the problem was solved, and the attempt at which it was solved. These logs support both quantitative evaluation and qualitative failure analysis.

4 Experimental Setup

All experiments were run on NVIDIA H100 GPU instances via Modal. The respective train and test set sizes and hyperparameter configurations for each method are listed in Tables 1 and 2.

	SFT	IPO	RLOO	SFT Curriculum Learning	RLOO Curriculum Learning
Train Size	1,000	48,300	490,000	1,000	89,950
Test Size	200	138	50	200	50
# Easy				665	29,983
# Medium				44	29,983
# Hard				287	29,984

Table 1: Train and test set sizes for all training methods

	batch size	grad acc steps	weight decay	warmup ratio	lr	epoch	extra params
SFT	64		0.01	0.05	5×10^{-5}	6	
IPO	64	16	0.01	0.05	5×10^{-6}	1	$\beta = 0.1$
RLOO	128	128	1×10^{-4}	0.0	1×10^{-5}		cont.

	train steps	group size	entropy coef	KL coef	tempature	top_k	top_p / min_p
RLOO	100	8	0.001	0.001	1.0	-1.0	1.0 / 0.0

Table 2: Hyperparameter configurations for all training methods

The main metric used to evaluate the performance of each method is pass@k, which measures the proportion of samples that contain at least one correct answer among k independently generated trials. The evaluation is carried out on the test split of dataset `countdown_tasks_3to4` that comprises 50 prompts in total.

4.1 Curriculum Learning Adjustments

Since we found that there are only 665 easy examples in the SFT training set after assigning labels, we increased the number of epochs from 6 to 10 to ensure that the model was exposed to a comparable amount of data during curriculum learning ($665 \times 10 = 6650$) as in the baseline setting ($1000 \times 6 = 6000$). Another minor implementation difference was the use of gradient accumulation with 8 accumulation steps during SFT curriculum learning. However, this change is not expected to materially affect the overall training dynamics or model behavior.

4.2 Inference Time Iterative Feedback Adjustments

The inference-time feedback-loop framework is compatible with any solver checkpoint from previous training. For the critic, we used Qwen-2.5-7B-Instruct, a more capable instruction-tuned model that can generate structured feedback from the diagnostic prompt without additional task-specific training. The solver and critic weights were fixed throughout evaluation, so any improvement comes only from test-time interaction among the solver, verifier, and critic.

All generations were performed with vLLM. For the initial solver attempt, we used temperature = 0.6, top_p = 0.95, top_k = 20, and a maximum generation length of 1024 tokens, matching our standard Countdown evaluation setting without feedback loops. Higher temperature setting encourages diverse problem-solving attempts. For revision attempts, we used temperature = 0.3, top_p = 0.95, top_k = 20, and 256 maximum tokens, so the solver is more focused when conditioning on critic feedback. For critic generation, we used temperature = 0.2, top_p = 0.95, top_k = 20, and 256 maximum tokens to encourage concise and stable diagnostic feedback.

We report pass rate after each solver attempt

$$\text{Pass@Attempt } t = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{problem } i \text{ is solved by attempt } t].$$

These metrics reflect whether verifier-gated feedback improve correctness over successive rounds. We also report the average number of solver attempts and critic calls per problem to quantify the extra test-time compute required by the method.

5 Results

5.1 Quantitative Evaluation

5.1.1 Baseline Methods

Plots 1 and 2 demonstrate the comparison of pass@k among these three methods. Both IPO and RLOO achieved higher pass@1 scores (40.125% and 52.75% respectively) than SFT (30%). Notably, although RLOO has significantly higher pass@1, IPO's pass@k surpasses RLOO when k exceeds 4, which highlights the complementary strengths of the two methods.

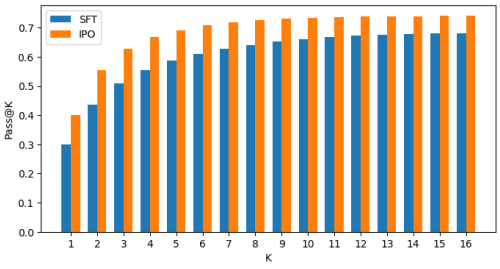

Figure 1: Pass@k: SFT vs IPO

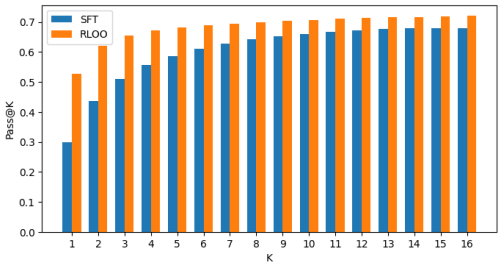

Figure 2: Pass@k: SFT vs RLOO

5.1.2 SFT Curriculum Learning

As we can see from Figure 3, SFT curriculum learning outperforms the baseline in terms of pass@k. Its pass@1 reaches 33.375%, and the performance gap between the two methods further widens as k increases. Eventually, for $k > 12$, the pass@k scores become even comparable to the IPO baseline.

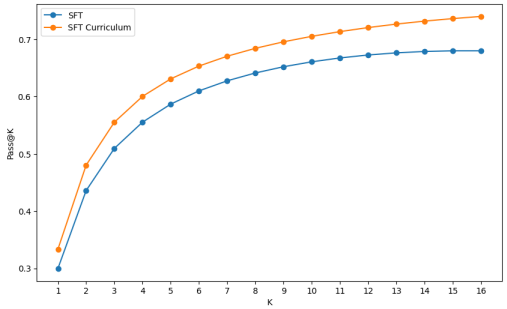

Figure 3: Pass@k: SFT vs SFT Curriculum Learning

5.1.3 RLOO Curriculum Learning

As illustrated in Figure 4, RLOO combined with the standard progressive curriculum learning exhibits the opposite behavior from what's observed with SFT: it under-performs the baseline across all pass@k values. In contrast, although pass@1 is slightly lower, RLOO with the curriculum that started with a mixed dataset rather than exclusively easy examples in the first phase achieves higher pass@k than the baseline for all $k \geq 4$.

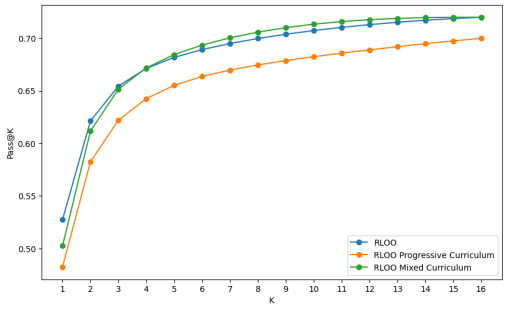

Figure 4: Pass@k: RLOO vs RLOO Curriculum Learning

To better understand the training dynamics of these three models, we also examined the rollout accuracy logged during training 5, defined as the average correctness of all rollouts generated across a batch of prompts. There are two sharp drops in rollout accuracy at steps 33 and 66, which correspond to the points where we either started introducing examples of the next difficulty level or increased the sampling probabilities of harder problems. This pattern suggests that mathematical reasoning involving division is indeed a difficult task for LLMs, and this may provide a potential direction for future data augmentation.

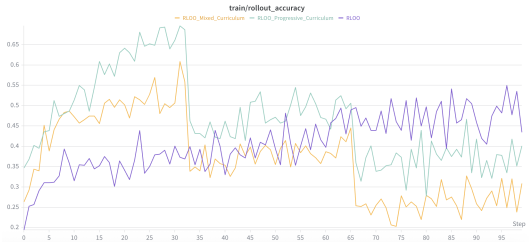

Figure 5: RLOO Rollout Accuracy

5.1.4 Inference Time Iterative Feedback

Solver checkpoint	Pass@Attempt 1 (%)	Pass@Attempt 2 (%)	Pass@Attempt 3 (%)	Avg. solver attempts
SFT Base	27.6 ± 2.6	29.2 ± 3.0	30.4 ± 3.0	2.432 ± 0.056
SFT Curriculum	29.6 ± 5.7	29.6 ± 5.7	31.6 ± 6.1	2.408 ± 0.115
RLOO Base	49.6 ± 3.3	51.6 ± 3.3	51.6 ± 3.3	1.988 ± 0.066
RLOO Progressive Curriculum	50.0 ± 4.9	52.8 ± 5.4	$\mathbf{53.2 \pm 5.8}$	$\mathbf{1.972 \pm 0.102}$
RLOO Mixed Curriculum	46.8 ± 5.4	51.2 ± 6.1	52.0 ± 6.0	2.020 ± 0.113
IPO Base	36.0 ± 2.4	39.2 ± 3.3	39.6 ± 3.3	2.248 ± 0.046

Table 3: Feedback-loop evaluation across 10 runs. We report mean $\pm$ standard deviation for pass rate after each attempt and the average number of solver attempts per problem.

According to Table 3, across all checkpoints, the feedback loop provides modest but consistent gains from Attempt 1 to Attempt 3. For example, SFT Base improves from 27.6% to 30.4%, IPO Base improves from 36.0% to 39.6%, and RLOO Progressive Curriculum improves from 50.0% to 53.2%. These gains indicate that critic-guided revision can recover some initially failed problems, but the improvement is relatively small.

The best overall performance comes from RLOO with progressive curriculum, which achieves the highest final solved rate, $53.2 \pm 5.8\%$, and the lowest average number of solver attempts, 1.972 ± 0.102. RLOO with mixed curriculum also performs strongly, reaching $52.0 \pm 6.0\%$, but it is slightly below the progressive curriculum checkpoint.

This observation appears to be inconsistent with our finding in section 5.1.3 where RLOO with progressive curriculum consistently under-performs compared to RLOO and RLOO with mixed curriculum when evaluated using pass@k via the `countdown_eval.py` script. This inconsistency is likely caused by the use of vLLM, as it introduces inherent randomness when generating responses. Due to the limited evaluation set size of 50, a single incorrectly sampled response can lead to a 2% decrease in pass@1, thereby amplifying vLLM-induced variability.

5.2 Qualitative Analysis

We manually inspected the saved feedback-loop trajectories to understand when critic feedback helps and when the loop fails. Overall, the feedback loop produced a small number of genuine corrections. Many other examples improved in format and legit number usage but not correctness, suggesting that feedback often helps the solver model to better understand Countdown task requirement but does not always fix the arithmetic reasoning.

Success Mode: Completing a Partially Correct Reasoning Path. One successful pattern occurs when the solver's initial reasoning contains useful intermediate steps but fails to produce a complete or properly formatted answer. For the problem with target 59 and numbers [37, 83, 21, 8], the first attempt started a promising calculation but did not finish with a valid answer. The critic identified the response as incomplete and encouraged the solver to complete the expression using the remaining numbers. On the second attempt, the solver found the correct reasoning path, computing 83 - 37 = 46, 46 + 21 = 67, and 67 - 8 = 59, but still failed to format the final answer correctly. After another feedback round, the third attempt produced the correct expression `<answer>83 - 37 + 21 - 8</answer>`. This shows that critic feedback can help transform partial reasoning into a correct solution, although multiple rounds may be needed when the solver's formatting remains unstable.

Failure Mode 1: Ignoring or Repeating Feedback. In some cases, the critic correctly identifies the issue, but the solver fails to use the feedback productively. For the problem with target 13 and numbers [61, 73, 63, 43], the critic pointed out that the solver's intermediate result did not reach the target and suggested recombining the differences differently. However, in later attempts, the solver largely repeated similar calculations and did not find a valid expression. This suggests that high-level natural-language feedback is not always enough for the small solver to change its search strategy.

Failure Mode 2: Template Copying Instead of Solving. Another common failure mode is that the solver copies the requested output template rather than replacing it with an actual solution. For the problem with target 56 and numbers [86, 11, 53, 59], after critic feedback the solver repeatedly generated literal placeholder text such as `<answer>expression</answer>` instead of producing an arithmetic expression. This indicates that the revision prompt itself can become a source of error for a small model: instead of treating the template as an instruction, the solver sometimes imitates the template text directly.

Overall, the feedback loop is most helpful when the solver is already near a valid solution or has a partially correct reasoning path that needs completion. It is less effective when the solver lacks a promising arithmetic plan, when the critic feedback is too vague, or when the revision prompt causes the solver to copy templates or critic-style text. These qualitative results support the quantitative finding that feedback improves final success rate modestly, but the main limitation is the small solver's ability to convert diagnostic feedback into a new, correct arithmetic expression.

6 Discussion

Our experimental results reveal several limitations of both proposed extensions. First, the results from curriculum learning confirmed that how examples are scheduled across training phases can indeed influence model performance. Due to limited compute budget, we only explored two curriculum designs instead of conducting grid search to find optimal proportions for mixed curriculum phases.

Second, qualitative analysis of inference-time feedback trajectories suggest that the loop is most useful when the solver is already close to a solution, but less effective when the solver lacks a promising arithmetic plan. We also observed failure cases where the solver ignored feedback, repeated similar incorrect calculations, or copied the prompt template instead of producing a valid expression. These results suggest that natural-language feedback is not always sufficient for a small 0.5B model to perform reliable arithmetic search. The feedback loop framework may require larger solver models to observe more learning behavior from critic feedback.

7 Conclusion

In this work, we extended baseline training methods along two directions: three-phase curriculum learning and iterative feedback during inference. Both extensions were shown to provide modest improvements in model performance, but no definitive conclusion can be drawn on whether progressive or mixed curriculum design is superior, as each may outperform the other under different evaluation methods. A more deterministic procedure is required for robust comparisons. Future work will focus on expanding evaluation sets, testing different sampling probabilities across curriculum learning phases, and adopting a more instruction-capable solver model to better understand and utilize feedback.

8 Team Contributions

- **Jiayu Sui:** Implemented inference-time iterative feedback pipeline and conducted relevant experiments.
- **Xinyi Ai:** Implemented curriculum learning models, built pipeline for RLOO curriculum learning data augmentation, and conducted relevant experiments.

The two team members share equal contribution on the SFT, IPO, RLOO baseline implementation and final report composition.

Changes from Proposal After the initial proposal, we decided not to pursue fine-tuning on inference-time feedback trajectories because it led to collapsed model performance. The Qwen-2.5-0.5B solver often failed to preserve its original Countdown response structure after training on long, noisy multi-turn trajectory data. We interpret this as a capacity limitation: the small model struggled to learn from correction traces while maintaining its role as a direct solver. We also found that TIR (Tool-Integrated Reasoning) was less useful for the Countdown task because a deterministic verifier already provides reliable correctness checking. We therefore kept the solver-critic loop as an inference-time method only and shifted more focus toward curriculum learning as a more stable way to improve training behavior and evaluation performance.

AI Tools Disclosure We used ChatGPT to support extension pipeline building, debugging, and final report grammar refinement.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada) *(ICML '09)*. Association for Computing Machinery, New York, NY, USA, 41–48. doi:10.1145/1553374.1553380

Yantao Gong, Cao Liu, Jiazhen Yuan, Fan Yang, Xunliang Cai, Guanglu Wan, Jiansong Chen, Ruiyao Niu, and Houfeng Wang. 2021. Density-Based Dynamic Curriculum Learning for Intent Detection. *CoRR* abs/2108.10674 (2021). arXiv:2108.10674 https://arxiv.org/abs/2108.10674

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. arXiv:2305.11738 [cs.CL] https://arxiv.org/abs/2305.11738

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. arXiv:2310.01798 [cs.CL] https://arxiv.org/abs/2310.01798

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL] https://arxiv.org/abs/2303.17651

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2026. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. arXiv:2506.06632 [cs.LG] https://arxiv.org/abs/2506.06632

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

Zhenting Wang, Guofeng Cui, Yu-Jhe Li, Kun Wan, and Wentian Zhao. 2025. DUMP: Automated Distribution-Level Curriculum Learning for RL-based LLM Post-training. arXiv:2504.09710 [cs.LG] https://arxiv.org/abs/2504.09710

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

Enci Zhang, Xingang Yan, Wei Lin, Tianxiang. Zhang, and Lu Qianchun. 2025. Learning Like Humans: Advancing LLM Reasoning Capabilities via Adaptive Difficulty Curriculum Learning and Expert-Guided Self-Reformulation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 6619–6633. doi:10.18653/v1/2025.emnlp-main.336