Extended Abstract

Progress-Aware Prompt Sampling for Verifier-Based RL Fine-Tuning

Motivation and problem statement. Verifier-based reinforcement learning fits reasoning tasks where correctness can be checked automatically. In Countdown, a model combines a small set of numbers with arithmetic operators to reach a target, and a rule-based verifier scores the final expression. Exact rewards still do not guarantee useful policy-gradient signal. Some prompts are already easy and most sampled completions are correct; others are too hard and most completions are wrong. In both cases, a REINFORCE leave-one-out (RLOO) update has little within-prompt reward contrast. This project asks whether online RL fine-tuning improves when training samples prompts more selectively.

Method and novelty. I implemented a coverage-preserving progress-aware prompt sampling extension for the default CS224R Countdown pipeline. Prompts were grouped into difficulty buckets using number count and target magnitude. During RLOO, the sampler tracked bucket-level short- and long-horizon verifier success, estimated learning progress from the short-minus-long success gap, combined it with within-prompt reward variance, and used the resulting utility to choose buckets. I tested bucket-uniform, empirical-prior, conservative empirical, and density-ratio variants. I also added two late follow-ups: random bucket partitions and a prompt-level utility sampler that tracks success, reward variance, degeneration, format validity, and sampling count. Because the trained policy often solved prompts under pass@16 but not pass@1, I also tested verifier-filtered self-training (RFT) to distill verified successful rollouts into pass@1 behavior.

Implementation and headline results. All methods used Qwen2.5-0.5B, the same verifier, the same RLOO objective, and matched held-out evaluation. On the main 200-prompt evaluation with 16 responses per prompt, uniform RLOO achieved average verifier score 0.5092, pass@1 0.4694, and pass@16 0.6650. The adaptive variants did not beat this baseline: bucket-uniform progress reached average score 0.4402 and pass@1 0.3853; empirical-prior progress reached 0.4696 and 0.4153; and the best adaptive variant, conservative empirical progress, reached 0.4847 and 0.4356. Paired bootstrap intervals supported the same conclusion: conservative progress had average-score difference -0.0245 with 95% CI $[-0.0446, -0.0049]$ and pass@1 difference -0.0338 with 95% CI $[-0.0553, -0.0125]$. Diagnostics showed that adaptive sampling increased degenerate reward-batch rate from 0.2703 under uniform to 0.3076–0.3545.

Discussion, limitations, and conclusion. The main result is negative but useful. Bucket-level adaptive sampling can change the effective training distribution without improving the learning signal. Preserving the empirical prompt prior repaired part of this shift, but was not sufficient. Late 50-prompt follow-ups told the same story: prompt-level utility reached pass@1 0.36 and random-bucket progress with 16 buckets reached 0.42, both below the uniform baseline's 0.56. The RFT follow-up was mixed as well: naive RFT slightly improved average score (0.5110 vs. 0.5092) and pass@1 (0.4813 vs. 0.4694), but copied repeated `<think>` and `<answer>` artifacts; clean filtering kept only 10 rows from 62,636 verifier-correct completions and did not improve accuracy. These results suggest that verifier feedback is useful, but the utility estimates and self-training filters tested here are too coarse. Future work should use lower-variance prompt- or completion-level utility estimates, preserve density ratios, and explicitly filter formatting validity when recycling rollouts.

Progress-Aware Prompt Sampling for Verifier-Based RL Fine-Tuning

Jason Yan
Stanford University
jasonyan@stanford.edu

Abstract

This project tests whether verifier-based RL fine-tuning for mathematical reasoning improves when online rollouts use more informative prompts. In Countdown, a rule-based verifier gives exact correctness rewards, but many prompts still produce degenerate reward batches where all sampled completions are correct or all are wrong. Those batches give weak RLOO advantages. I implemented a progress-aware prompt sampler that tracks learning progress and reward variance under several bucket priors, then tested random-bucket and prompt-level utility variants. On a 200-prompt robustness evaluation, adaptive samplers did not outperform uniform RLOO: the uniform baseline reached average score 0.5092 and pass@1 0.4694, while the best adaptive variant reached 0.4847 and 0.4356. Additional 50-prompt follow-ups with prompt-level utility and random buckets also underperformed uniform. Diagnostics showed that adaptive sampling increased the degenerate reward-batch rate and often worsened format repetition. A verifier-filtered self-training follow-up showed that the RLOO policy can generate many correct completions, but naive distillation copied formatting artifacts and clean filtering removed almost all data. Verifier feedback helps, but the prompt-utility estimates and self-training filters tested here are too coarse; stronger methods need more stable utility estimates and stricter format-aware filtering.

1 Introduction

Reinforcement learning from feedback is now common in language-model post-training. In instruction following, feedback is often a learned reward model or a human preference signal Ouyang et al. (2022). In mathematical and programmatic reasoning tasks, rewards can sometimes be computed exactly. A rule-based verifier can check whether a generated answer satisfies a target condition, so the model can train from online samples without new human labels.

Countdown is a compact version of this setting. Given a target number and a set of input numbers, the model must produce an arithmetic expression that uses the numbers correctly and evaluates to the target. The verifier is simple, but the task is still hard enough for a small language model to produce incorrect or malformed solutions.

The default CS224R pipeline fine-tunes a supervised model with methods such as preference optimization and RLOO. My extension focuses on a different part of the pipeline: the prompt distribution used during online RLOO rollouts. A prompt only provides informative policy-gradient signal if the sampled completions produce reward contrast. If all completions are correct, the prompt says little about which completion should become more likely. If all completions are wrong, the same issue appears in the opposite direction. In RLOO, the leave-one-out advantage is computed within a group of samples from the same prompt, so all-correct and all-wrong groups are nearly degenerate.

This project asks three research questions:

1. Can progress-aware prompt sampling improve verifier-based RLOO relative to uniform prompt sampling?
2. How important is preserving the empirical prompt distribution when adaptively sampling prompts?
3. Can verified successful rollouts be distilled into better pass@1 performance through self-training?

The first question came out mostly negative. Adaptive bucket-level samplers did not improve expected pass@k or average verifier reward. The second answer was clearer: preserving the empirical prompt prior matters. A naive bucket-uniform prior shifted probability mass toward rare buckets and substantially hurt low-k performance. Empirical-prior variants repaired part of the damage, but still did not beat uniform sampling. Late follow-ups with random bucket partitions and prompt-level utility sampling also failed to beat uniform on the official 50-prompt evaluation. Verifier-filtered self-training produced a small apparent gain under naive filtering, but diagnostics showed serious formatting drift and the clean filtered version did not improve performance.

This report does not claim a new state-of-the-art Countdown solver. It studies a plausible design choice in verifier-based RL and finds a concrete failure mode: exact rewards are not enough when the sampling utility is too coarse and the prompt distribution shifts.

2 Related Work

RL fine-tuning for language models. RLHF trains language models to align with human preferences by optimizing against feedback signals Ouyang et al. (2022). Preference-optimization methods such as DPO and IPO avoid online rollouts by transforming preference data into supervised-style objectives Rafailov et al. (2023); Gheshlaghi Azar et al. (2024). These methods are stable and data-efficient when good comparisons are available, but they do not directly answer which prompts should be sampled when a verifier can score new rollouts.

Policy gradients and verifier rewards. REINFORCE-style estimators optimize expected reward by increasing the probability of sampled actions with positive advantage Williams (1992). Recent work revisits such estimators for language-model fine-tuning and finds that simple policy-gradient methods can be competitive when implemented carefully Ahmadian et al. (2024). Mathematical reasoning systems such as DeepSeekMath and DeepSeek-R1 use automatically checkable rewards in reasoning domains Shao et al. (2024); DeepSeek-AI (2025). This project keeps the verifier reward fixed and changes the sampling distribution over prompts.

Curriculum and self-paced learning. Curriculum learning orders training examples from easier to harder Bengio et al. (2009). Self-paced learning adapts this idea to the current learner, selecting examples based on what the model can currently learn from Kumar et al. (2010). My progress sampler uses the same intuition with online verifier outcomes instead of supervised loss. A bucket is treated as useful when it shows recent learning progress and non-degenerate reward variance. The experiments show that this signal is too coarse in this setting.

Verifier-filtered self-training. The RFT follow-up is related to self-training and rejection-sampling approaches for reasoning. STaR fine-tunes on model-generated rationales that lead to correct answers, iterating this process to bootstrap reasoning ability Zelikman et al. (2022). Rejection sampling fine-tuning similarly generates multiple mathematical reasoning paths, keeps verifier-correct samples, and uses them as additional supervised data Yuan et al. (2023). My RFT experiment is smaller and diagnostic: it tests whether verified rollouts from the RLOO policy can close the pass@16-to-pass@1 gap, and it measures format drift when correctness is the only filtering criterion.

3 Method

3.1 Base Pipeline

The base system follows the default Countdown fine-tuning setup. A supervised fine-tuned model generates reasoning traces wrapped in `<think>` tags and a final arithmetic expression wrapped in `<answer>` tags. A rule-based verifier extracts the answer expression, checks that it uses the provided numbers legally, and evaluates whether it reaches the target. The RLOO trainer samples K

completions for each prompt, scores them with the verifier, and computes a leave-one-out baseline from the other completions for the same prompt.

For completion y_i sampled for prompt x, with reward r_i, the leave-one-out advantage is

$$A_i = r_i - \frac{1}{K-1}\sum_{j \neq i} r_j. \tag{1}$$

The policy-gradient update minimizes

$$\mathcal{L}_{\mathrm{RLOO}}(\theta) = -\frac{1}{K}\sum_{i=1}^{K} A_i \, w_i \log \pi_\theta(y_i \mid x), \tag{2}$$

where w_i is the implementation's importance weight between the sampled policy and update policy. The extension leaves the verifier, model, and RLOO loss fixed, and only changes how prompts are selected during online training.

3.2 Bucket-Level Progress-Aware Sampler

Each training prompt is assigned to a combined bucket based on two observable features: the number of input numbers and the target magnitude. The final runs use eight buckets: three-number and four-number prompts crossed with four target bins, < 25, 25–49, 50–74, and ≥ 75. Let $b(x)$ denote the bucket for prompt x. For each bucket b, the sampler tracks short- and long-horizon exponential moving averages of verifier success:

$$p_b^{\mathrm{short}} \leftarrow (1-\alpha_s)p_b^{\mathrm{short}} + \alpha_s \hat{p}_b, \tag{3}$$

$$p_b^{\mathrm{long}} \leftarrow (1-\alpha_l)p_b^{\mathrm{long}} + \alpha_l \hat{p}_b, \tag{4}$$

where $\hat{p}_b$ is the observed mean verifier score in the current update and $\alpha_s > \alpha_l$. In all progress-sampler runs, $\alpha_s = 0.3$ and $\alpha_l = 0.03$. Statistics are updated once per training batch using the prompts sampled in that batch; unseen buckets retain their prior probability until they receive data. Recent learning progress is estimated as

$$LP_b = \max(0, p_b^{\mathrm{short}} - p_b^{\mathrm{long}}). \tag{5}$$

The sampler also records within-prompt reward variance,

$$I_b = \mathrm{Var}(r_{b,1}, \ldots, r_{b,K}), \tag{6}$$

which is meant to identify prompts that produce non-degenerate RLOO comparisons. The verifier score is 1.0 for a valid correct expression, 0.1 for a parsed but incorrect or invalid expression, and 0.0 when no usable answer can be extracted; this is why some qualitative failures have score 0.1 rather than exactly zero. The bucket utility is

$$U_b = \lambda LP_b + (1-\lambda)I_b, \tag{7}$$

with $\lambda = 0.5$ in the final experiments.

3.3 Coverage Priors and Stabilization

The first adaptive sampler mixed the utility softmax with a bucket-uniform coverage prior:

$$P(b) = (1-\epsilon)\mathrm{softmax}(U_b/\tau) + \epsilon P_{\mathrm{uniform}}(b). \tag{8}$$

This preserves coverage over buckets, but not necessarily over the original data distribution. The main bucket-uniform progress run used $\epsilon = 0.2$, $\tau = 0.2$, and capped any bucket at probability 0.5. Because some buckets are rare in the empirical training set, a bucket-uniform prior can over-sample them. I then implemented empirical-prior variants,

$$P(b) = (1-\epsilon)\mathrm{softmax}(U_b/\tau) + \epsilon P_{\mathrm{data}}(b), \tag{9}$$

where $P_{\mathrm{data}}(b)$ is the bucket frequency in the original training data. I tested an empirical-prior run with $\epsilon = 0.5$, $\tau = 0.2$, and maximum bucket probability 0.25; a conservative version with $\epsilon = 0.8$,

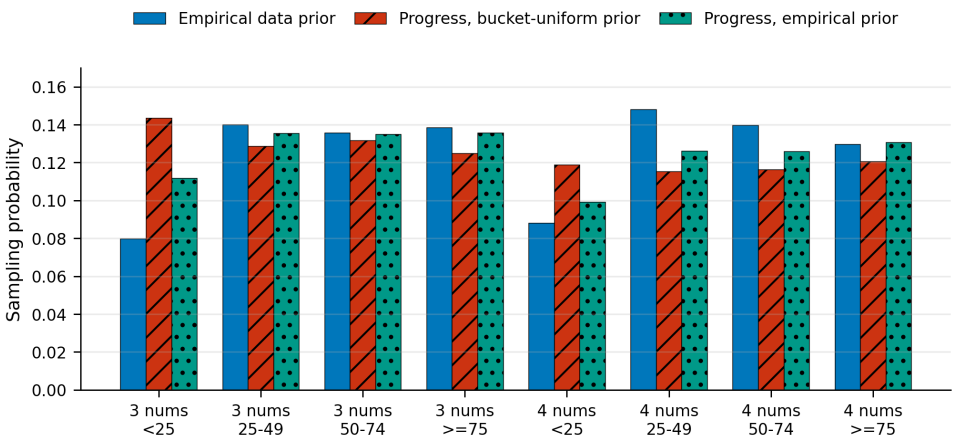

Figure 1: Mean bucket sampling probabilities during training compared with the empirical data prior. Bucket-uniform progress sampling assigns too much probability to rare buckets, especially small-target three-number prompts. Empirical-prior variants reduce this shift but still do not improve held-out performance.

$\tau = 0.5$, and maximum bucket probability 0.18; and a density-ratio variant that clips $P(b)/P_{\text{data}}(b)$ to $[0.5, 1.5]$.

I also ran two late sampler follow-ups to test whether the negative result was caused by the particular hand-designed buckets. The first replaces the target-magnitude buckets with random partitions of prompts and uses the same progress utility with an empirical prior. The successful follow-up used 16 random buckets; the 8-bucket version was unstable and did not complete a full 100-step run. The second follow-up moves from bucket-level statistics to prompt-level utility. For each prompt, it tracks short- and long-horizon success, reward variance, degenerate-batch rate, format-valid rate, and number of times sampled. Prompt utility uses the same progress-plus-variance intuition, with penalties for degeneration, invalid formatting, and over-sampling. This version is a direct test of the hypothesis that the bucket average was too coarse.

3.4 Verifier-Filtered Self-Training Follow-Up

The RLOO evaluation showed a gap between low-k and high-k success: the uniform checkpoint solved substantially more prompts by pass@16 than by pass@1. The policy could sometimes sample correct solutions, but did not consistently place them at high probability. I then implemented a verifier-filtered self-training follow-up. The uniform RLOO checkpoint generated 32 completions per training prompt. Correct completions were kept by the verifier, capped per prompt, mixed with original SFT examples, and used for one additional supervised fine-tuning epoch. A second clean version kept only completions with exactly one answer tag, no repeated reasoning tags, a short length, and one shortest correct solution per prompt.

4 Experimental Setup

Task and model. All experiments use the `asingh15/countdown_tasks_3to4` Countdown dataset, which contains prompts with three or four input numbers. The base model is the default Qwen2.5-0.5B Countdown SFT checkpoint used in the course pipeline. The verifier gives a task reward based on answer extraction, legal number use, and arithmetic correctness. Uniform RLOO is the main baseline because it keeps the online training prompt distribution matched to the data distribution while using the same optimizer, model, and verifier as the adaptive variants.

Training runs. The main comparison includes uniform RLOO, progress sampling with a bucket-uniform prior, progress sampling with an empirical prior, a conservative empirical-prior variant, and a density-ratio correction. Each RLOO run used 100 training steps, batch size 128, group size 8,

Table 1: Main 200-prompt evaluation. All methods are evaluated on the same prompts with 16 responses per prompt. Values are mean ± standard error over prompts.

Method	Avg.	Pass@1	Pass@4	Pass@8	Pass@16	Zero-success
Uniform RLOO	**0.509±0.027**	**0.469±0.029**	**0.598±0.033**	**0.630±0.033**	**0.665±0.034**	**0.335±0.034**
Progress, bucket-uniform	0.440±0.023	0.385±0.025	0.572±0.033	0.607±0.034	0.630±0.034	0.370±0.034
Progress, empirical	0.470±0.025	0.415±0.027	0.564±0.034	0.588±0.034	0.605±0.035	0.395±0.035
Progress, conservative	0.485±0.026	0.436±0.029	0.572±0.034	0.598±0.034	0.620±0.034	0.380±0.034
Progress, density ratio	0.426±0.023	0.364±0.026	0.538±0.033	0.577±0.034	0.610±0.035	0.390±0.035

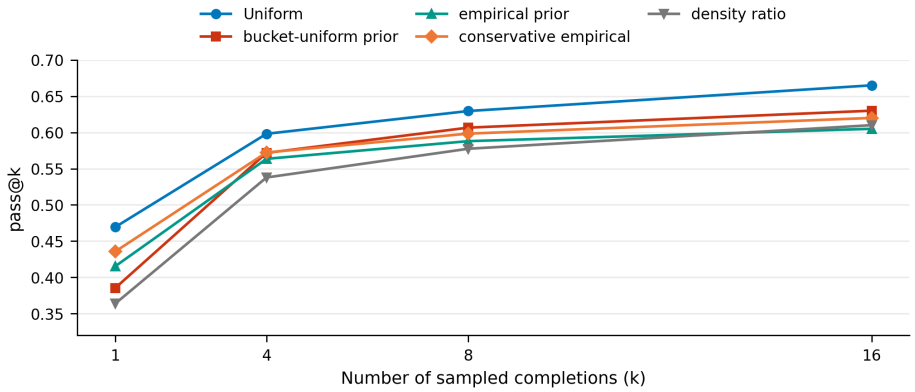

Figure 2: Order-averaged pass@k curves for the main 200-prompt evaluation. Adaptive bucket-level sampling does not improve high-k or low-k performance relative to uniform RLOO.

and learning rate 10^{-5}. Late follow-up sampler runs used a more conservative vLLM configuration for stability but the same optimizer scale and training length: prompt-level utility sampling and random-bucket progress with 16 buckets completed 100 steps, while random-bucket progress with 8 buckets failed before completion. The self-training follow-up used the uniform RLOO checkpoint as the generator and base checkpoint for RFT. Naive RFT used $K = 32$ rollout samples, cap 2 correct completions per prompt, 30% original SFT data mix, and learning rate 10^{-5}. Clean RFT used cap 1 shortest clean completion per prompt, 50% original SFT mix, and learning rate $5 \cdot 10^{-6}$.

Evaluation. The main robustness evaluation uses the same 200 held-out prompts and the same decoding settings for all methods: 16 responses per prompt, temperature 0.6, top-$p = 0.95$, and top-$k = 20$. I report average verifier score, order-averaged pass@1, pass@4, pass@8, pass@16, zero-success prompt rate, and all-success prompt rate. Because the compute budget did not allow multiple complete training seeds, I use paired prompt-level comparisons against the uniform baseline as the main uncertainty analysis: every method is evaluated on the same prompts with the same decoding protocol, so wins, losses, ties, and mean paired differences estimate whether aggregate changes are consistent across prompts. I also report 95% bootstrap confidence intervals over prompts for the paired differences, using 20,000 bootstrap resamples. Official 50-prompt results are included in the appendix.

5 Results

5.1 Main Quantitative Results

Table 1 reports the main 200-prompt robustness evaluation with standard errors over prompts. Uniform RLOO is the strongest method across average score and every pass@k value. The bucket-uniform progress sampler performs worst among the main adaptive samplers, especially at pass@1. The empirical-prior correction repairs much of that damage, and the conservative empirical variant is the best adaptive sampler, but neither beats uniform.

The paired comparison in Table 2 serves as the main variance-control analysis. For average score, the conservative empirical variant wins on 100 prompts but loses on 81 and has a negative mean

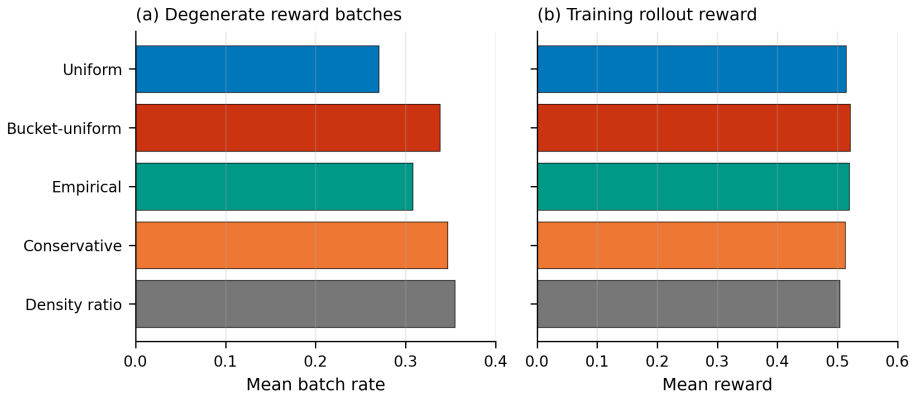

Figure 3: Training diagnostics. Adaptive samplers did not reduce degenerate reward batches; they generally increased them. This undermines the central mechanism that motivated progress-aware sampling.

difference. Its paired bootstrap 95% CI is also below zero for both average score and pass@1. At pass@1, all adaptive variants lose more prompts than they win. This paired view makes the aggregate result harder to dismiss as a small evaluation artifact.

Table 2: Paired prompt-level comparison against uniform RLOO on the 200-prompt evaluation. Confidence intervals are 95% paired bootstrap intervals over prompts for method minus uniform.

Method	Metric	Wins	Losses	Ties	Mean diff.	95% CI
Bucket-uniform	Pass@1	25	100	75	-0.0841	$[-0.1066, -0.0619]$
Bucket-uniform	Avg. score	73	106	21	-0.0690	$[-0.0900, -0.0484]$
Empirical	Pass@1	31	86	83	-0.0541	$[-0.0753, -0.0334]$
Empirical	Avg. score	83	89	28	-0.0396	$[-0.0595, -0.0206]$
Conservative	Pass@1	50	76	74	-0.0338	$[-0.0553, -0.0125]$
Conservative	Avg. score	100	81	19	-0.0245	$[-0.0446, -0.0049]$
Density ratio	Pass@1	18	101	81	-0.1056	$[-0.1300, -0.0822]$
Density ratio	Avg. score	77	99	24	-0.0832	$[-0.1063, -0.0612]$

5.2 Why Bucket-Uniform Progress Sampling Fails

Figure 1 shows the distribution shift caused by the bucket-uniform progress sampler. The empirical data prior is not uniform over buckets: for example, small-target three-number prompts are relatively rare. Bucket-uniform coverage treats each bucket equally and increases probability mass on rare buckets. The first failure mode follows from that mismatch: maintaining bucket coverage is not the same as preserving the prompt distribution. The empirical-prior variants reduce this shift, which is why their results are less damaging, but they still rely on a coarse utility estimate.

5.3 Training Diagnostics

The intended mechanism of progress-aware prompt sampling was to reduce rollout waste by selecting prompts with more informative reward variation. The diagnostics show the opposite. Uniform RLOO had a degenerate reward-batch rate of 0.2703 ± 0.0079 over training steps. All adaptive variants had higher degenerate rates, with the density-ratio variant reaching 0.3545 ± 0.0115. Mean training reward was similar across methods, so the adaptive samplers were not just selecting harder prompts. The bucket-level signal failed to identify prompts with better within-group reward contrast.

5.4 Late Sampler Follow-Ups

After the main experiments, I tested two additional sampler variants on the official 50-prompt evaluation: prompt-level utility sampling and random-bucket progress sampling. These were meant

Table 3: Verifier-filtered self-training follow-up on the same 200-prompt evaluation.

Method	Avg.	Pass@1	Pass@4	Pass@8	Pass@16	Zero-success
Uniform RLOO	0.5092	0.4694	0.5981	0.6295	**0.6650**	**0.3350**
Naive verified RFT	**0.5110**	**0.4813**	**0.6042**	**0.6325**	0.6600	0.3400
Clean shortest RFT	0.5016	0.4600	0.5971	0.6268	0.6550	0.3450

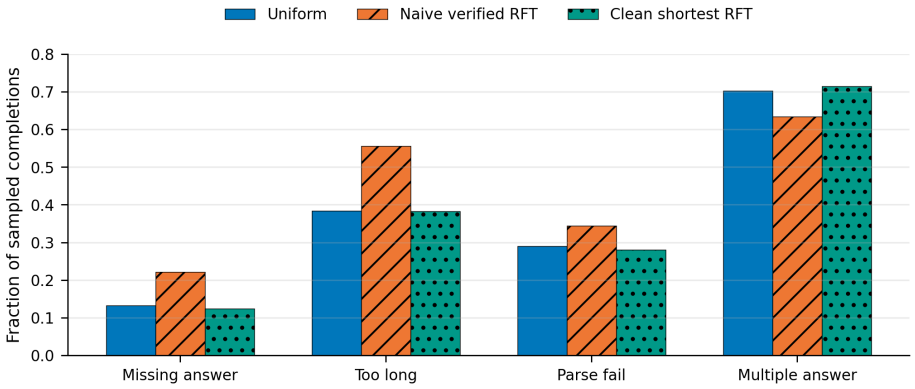

Figure 4: Format diagnostics for the RFT follow-up. Naive RFT copies formatting artifacts from verified rollouts, while clean filtering reduces some artifacts but removes almost all supervised signal.

to check whether the earlier negative result was specifically caused by the hand-designed target-magnitude buckets. The answer was no. Table 6 in the appendix shows that uniform RLOO reached pass@1 0.56 and pass@16 0.74, while prompt-level utility reached pass@1 0.36 and pass@16 0.70, and random-bucket progress with 16 buckets reached pass@1 0.42 and pass@16 0.68. These results do not replace the 200-prompt robustness table, but they reinforce the same conclusion: the simple adaptive utility estimates used here did not improve held-out performance. The prompt-level run also showed strong formatting drift, with multiple-answer tags in 99.4% of sampled completions and repeated-think tags in 98.6%, compared with 75.6% and 88.8% for the corresponding uniform 50-prompt baseline.

5.5 Verifier-Filtered RFT Follow-Up

Table 3 reports the RFT follow-up. Naive verified RFT is close to uniform RLOO and slightly higher on average score and pass@1, but it does not improve pass@16 and the margin is small. Clean shortest RFT performs below uniform. These results suggest that verified self-training has a real signal, but the naive implementation is not a robust improvement.

The rollout examples show where the extra verified data goes wrong. Naive RFT often learns completions that are verifier-correct but poorly formatted, including repeated `<think>` tags and multiple `<answer>` tags. Figure 4 shows that naive RFT increases missing-answer and too-long rates relative to uniform. Clean filtering improves some format statistics, but the filter is too strict: from 62,636 verifier-correct rollout completions, only 10 clean rows covering 10 prompts remained after enforcing exactly one answer tag, no repeated think tags, short length, and cap-1 shortest selection.

5.6 Qualitative Examples

The qualitative examples match the quantitative diagnosis. On one prompt with target 61 and numbers $[66, 71, 49, 51]$, the uniform RLOO policy produced a correct completion while the bucket-uniform progress model cycled through repeated unsuccessful attempts. This matches the paired pass@1 losses in Table 2. In another prompt with target 41 and numbers $[16, 21, 46]$, naive RFT produced a verifier-correct equation, $(46 - 21) + 16$, but then continued with another `<think>` block and extra text after the answer tag. Such completions show that the model can solve the arithmetic problem, but they are poor supervised targets if copied directly.

Table 4: Representative qualitative failures. Excerpts are shortened to show the diagnostic pattern rather than the full generation.

Case	Prompt / method	Short excerpt and interpretation
Uniform succeeds, progress fails	Target 61, nums $[66, 71, 49, 51]$; bucket-uniform progress	The progress model repeats attempts such as "$71 - 66 = 5$," "$5 + 51 = 56$," and "$56 + 49 = 105$" without reaching a valid final expression. This is a concrete paired loss for adaptive sampling.
Correct but poor RFT target	Target 41, nums $[16, 21, 46]$; naive RFT	The model emits `<answer>(46 - 21) + 16</answer>` and then continues into another `<think>` block. The verifier accepts the arithmetic, but the completion is a bad imitation target.
Clean filtering removes signal	Clean RFT data generation	The uniform RLOO checkpoint produced 62,636 verifier-correct rollout completions before filtering, but strict format and cap-1 filtering kept only 10 rows across 10 prompts.

A third example shows a missing-answer failure: the naive RFT model generated a long reasoning trace for the target-61 prompt but never produced a parseable final answer. Clean RFT reduced some format failures but failed on prompts where naive RFT was correct, which is consistent with the filter keeping only 10 examples. These cases clarify the self-training result. Verifier-correct completions do contain signal, but correctness alone is a weak filter for imitation learning.

6 Discussion

Adaptive sampling changed the training distribution, and that shift dominated the intended benefit. The first progress sampler was designed to preserve coverage, but it preserved coverage over buckets rather than over prompts. The distinction mattered: rare buckets were over-sampled, and performance dropped sharply. Empirical-prior sampling fixed the most obvious distribution shift, but not the deeper problem that bucket-level progress and variance are too coarse.

The degenerate-batch diagnostic is the key check on the method. Uniform RLOO wastes rollouts on all-correct or all-wrong groups. If the sampler were working, it should reduce that rate. Instead, every adaptive variant increased it. Bucket-level success statistics were a weak proxy for prompt-level informativeness. Prompts in the same bucket can differ substantially, and a bucket can have a reasonable average success rate while still containing many degenerate prompts.

The late prompt-level result makes the failure more specific. Moving from buckets to individual prompts was a reasonable next test, but the simple prompt utility used here was still not stable enough. It underperformed uniform on the 50-prompt evaluation and produced even more repeated answer and reasoning tags. The missing signal is not just a finer prompt index. The sampler also needs a more reliable estimate of which prompt-completion pairs produce useful, well-formed learning updates.

The RFT follow-up shows a second limitation of verifier-only feedback. A verifier can determine whether a final expression is correct, but it may not penalize all undesirable generation behavior. If a completion contains one correct answer and then repeats tags or continues generating, a verifier may still mark it correct. Self-training on those completions can amplify format drift. Strict format filtering has the opposite problem: it can remove too much data. A better method would filter for correctness, uniqueness of the final answer, concise reasoning, and prompt diversity.

The experiments use one model size, one task family, and a limited training budget. I did not run multiple random seeds because the project compute budget went to method variants and 200-prompt robustness evaluation. The empirical conclusions are strongest for this controlled Countdown setup, not for all verifier-based RL. Still, the failure modes are likely relevant beyond Countdown: coarse adaptive sampling and naive verifier-filtered imitation are common design temptations in reasoning fine-tuning.

7 Conclusion

This project tested whether progress-aware prompt sampling can improve verifier-based RLOO for Countdown. The result is a useful negative finding. Naive bucket-level progress sampling hurts because it changes the effective prompt distribution. Preserving the empirical prompt prior is necessary, but still not sufficient, because bucket-level progress and reward variance do not reliably

identify informative prompts. Verifier-filtered self-training shows that the policy can sample many correct completions, but correctness alone does not produce clean supervised targets.

The late random-bucket and prompt-level utility follow-ups did not reverse this conclusion. Randomizing the bucket partition did not help, and a first prompt-level utility sampler also underperformed uniform. Future work should estimate utility with lower-variance prompt-level or completion-level signals, use density-ratio constraints to preserve the data distribution, and filter generated solutions for both correctness and format validity before self-training. Verifier-based reasoning tasks already give us rewards. The harder question is which reward observations should shape the policy.

8 Team Contributions

This is a single-member project. Jason Yan implemented the default fine-tuning and evaluation pipeline used for this report, including SFT integration, IPO and RLOO training, verifier-based evaluation, progress-aware prompt sampling, empirical-prior and density-ratio variants, random-bucket and prompt-level sampler follow-ups, training diagnostics, verifier-filtered RFT follow-up experiments, clean filtering, result aggregation, plotting, and report writing.

Changes from proposal. The original proposal focused on progress-aware prompt sampling and planned to compare uniform, static curriculum, success-rate adaptive, and progress-aware samplers. During experimentation, the project shifted toward a deeper analysis of progress-aware variants because the early bucket-uniform sampler revealed a strong distribution-shift failure mode. I added empirical-prior, conservative empirical, and density-ratio corrections to test that failure mode directly. Near the end, I added random-bucket and prompt-level utility follow-ups to test whether the hand-designed bucket partition itself was the main bottleneck. I also added the verifier-filtered RFT follow-up after observing a persistent gap between pass@1 and pass@16. These changes preserved the proposal's central question about useful verifier feedback while making the final study more diagnostic.

Data Availability, Ethics, Funding, Conflicts, and AI Disclosure

All experiments used the public Countdown dataset and locally generated model rollouts. No human-subject data or private user data were used. The project received no external funding beyond the use of Modal credits for running experiments, and the author declares no conflicts of interest. AI assistance (ChatGPT and Codex) was used for code review, experiment organization, and figure/table preparation.

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*. Association for Computational Linguistics, 12248–12267.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 41–48.

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 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.

M. Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems*, Vol. 23.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems*, Vol. 35. 27730–27744.

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 preprint arXiv:2305.18290* (2023).

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024).

Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8, 3-4 (1992), 229–256.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models. *arXiv preprint arXiv:2308.01825* (2023). `https://doi.org/10.48550/arXiv.2308.01825`

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems*, Vol. 35. 15476–15488. `https://doi.org/10.48550/arXiv.2203.14465`

A Run Configuration

Table 5: Summary of final run configurations.

Run	Role	Sampler / data	Key controls	Evaluation
Uniform	RLOO baseline	Uniform prompt sampling	100 steps, batch 128, group 8, lr 10^{-5}	200 prompts, 16 responses, temp 0.6
Progress bucket-uniform	Main adaptive ablation	Progress utility, uniform bucket prior	$\epsilon = 0.2$, $\tau = 0.2$, max bucket prob 0.5	Same protocol
Progress empirical	Prior correction	Progress utility, empirical bucket prior	$\epsilon = 0.5$, $\tau = 0.2$, max bucket prob 0.25	Same protocol
Progress conservative	Conservative adaptive variant	Progress utility, empirical bucket prior	$\epsilon = 0.8$, $\tau = 0.5$, max bucket prob 0.18	Same protocol
Density ratio	Distribution correction	Empirical prior with clipped density ratio	ratio clipped to $[0.5, 1.5]$	Same protocol
Random buckets, 16	Late sampler diagnostic	Progress utility over random prompt partitions	empirical prior, 16 buckets, stable vLLM config	50 prompts, 16 responses
Prompt utility	Late sampler diagnostic	Per-prompt success, variance, format, and coverage utility	penalties for degeneration, invalid format, and oversampling	50 prompts, 16 responses
Naive RFT	Verified self-training	K=32 rollouts from uniform RLOO, cap 2 correct completions	30% original SFT mix, 1 epoch, lr 10^{-5}	Same protocol
Clean RFT	Clean verified self-training	Shortest clean correct completion, cap 1	50% original SFT mix, 1 epoch, lr $5 \cdot 10^{-6}$	Same protocol

B Official 50-Prompt Evaluation

Table 6: Official 50-prompt evaluation. The early bucket results and late sampler follow-ups show the same pattern: adaptive sampling did not clearly beat uniform.

Method	Avg.	Pass@1	Pass@4	Pass@8	Pass@16
Uniform	0.583	0.56	0.70	0.72	0.74
Progress, bucket-uniform	0.507	0.32	0.66	0.72	0.74
Progress, empirical	0.530	0.54	0.68	0.72	0.76
Prompt utility	0.451	0.36	0.62	0.68	0.70
Random buckets, 16	0.497	0.42	0.68	0.68	0.68

C Additional Diagnostics

Table 7: Training diagnostics for RLOO runs. Values are mean ± standard error over 100 training steps where applicable. Degenerate batch rate is the fraction of rollout groups where rewards provide little or no within-prompt contrast.

Method	Mean reward	Final reward	Degenerate rate	Reward variance
Uniform	0.514±0.0065	0.490	**0.270±0.0079**	0.0637±0.0014
Progress, bucket-uniform	0.521±0.0054	0.522	0.338±0.0105	0.0648±0.0012
Progress, empirical	0.520±0.0059	0.517	0.308±0.0105	0.0613±0.0013
Progress, conservative	0.513±0.0058	0.544	0.346±0.0113	0.0601±0.0011
Progress, density ratio	0.504±0.0056	0.477	0.354±0.0115	0.0620±0.0011

Table 8: Format diagnostics for late 50-prompt sampler follow-ups. The prompt-level utility run underperformed uniform and also had the highest repetition rates.

Method	Missing answer	Multiple answer	Multiple think	Too long
Uniform	0.0988	0.7563	0.8875	1.0000
Prompt utility	0.0000	0.9938	0.9863	1.0000
Random buckets, 16	0.0000	0.9863	0.9575	1.0000

Table 9: Format diagnostics for the RFT follow-up. The rates are computed over generated evaluation completions.

Method	Missing answer	Too long	Parse fail	Multiple answer
Uniform RLOO	0.1325	0.3837	0.2897	0.7031
Naive RFT	0.2216	0.5563	0.3441	0.6338
Clean RFT	0.1241	0.3825	0.2797	0.7147