Extended Abstract

Motivation Large language models (LLMs) trained with reinforcement learning are highly sensitive to the difficulty and ordering of training problems. Problems that are too easy provide little learning signal. Problems that are too hard yield sparse, unstable rewards. Standard RL training pipelines (SFT, DPO, RLOO) treat curriculum design as a static, human-specified process, typically ordering problems by operand count or other proxies. This can fail to capture subtler sources of difficulty and cannot adapt to the model's evolving capabilities during training. We ask whether a curriculum policy can improve RLOO fine-tuning on Countdown by selecting prompts near the Solver's current empirical ability threshold using data that tracks the solver's current state.

Method We introduce an Adaptive Curriculum Assigner, a learned MLP policy trained with REINFORCE that dynamically selects which difficulty bucket of problems to present to the Solver LLM at each training step. The Assigner is rewarded when the Solver's success rate on sampled problems falls near 50% (the "Zone of Proximal Development") and penalized otherwise. The Assigner is an RL-trained MLP curriculum policy that conditions bucket selection on live Solver training statistics, including KL divergence, entropy, per-bucket accuracies, and a dynamic exponential moving average (EMA) of per-prompt success rates.

Implementation We evaluated three variants of the Assigner. V1 used 6 fixed success-rate buckets with EMA $\alpha = 0.5$, but suffered from bucket collapse and reward hacking as the Assigner learned to select buckets the Solver could already solve, since reward was computed after rollouts. V2 addressed these issues by lowering EMA to $\alpha = 0.15$, reordering all prompts into percentile-based buckets every 10 steps to maintain equal bucket populations, and computing the Assigner reward from a pre-rollout accuracy snapshot to eliminate the circularity in V1. V3 added entropy regularization to the Assigner objective and reduced the number of buckets from 6 to 4, aiming to force broader exploration across difficulty levels.

Results V2 was the strongest-performing variant, achieving Pass@1 = 0.653 compared to the RLOO baseline of 0.635, with the advantage growing at higher K: +3.0 points at Pass@8 (0.775 vs. 0.745) and +2.0 points at Pass@16 (0.800 vs. 0.780). This 2.7% relative improvement is more than half of the improvement in Self-Evolving Curriculum (SEC), which we achieve despite using a 6 times smaller model. V1 showed a marginal gain at Pass@1 (0.645) but degraded below baseline at higher K, reflecting the reward-hacking and bucket collapse that caused the Assigner to over-exploit already-solved problems. V3 underperformed the baseline across all metrics, with Pass@1 dropping to 0.574, likely because entropy regularization discouraged the medium-difficulty concentration that produces useful training signal. Together, these results suggest that adaptive curriculum selection can meaningfully improve reasoning performance, but only when the Assigner is properly incentivized and its training dynamics are stable.

Discussion This work empirically documents two distinct curriculum learning failure modes: easy-bucket exploitation in V1, where the Assigner reward-hacked by selecting already-solved buckets, and hard-bucket over-sampling in V3, where entropy regularization penalized the desired concentration on medium-difficulty problems. A central limitation is that standard REINFORCE tools (like leave-one-out baselines and entropy regularization) may either strengthen the issues relating to bucket collapse, or cause the Assigner to fail to exploit medium-difficulty problems. Both prevent the Assigner from sustaining useful updates.

Conclusion The Adaptive Curriculum Assigner demonstrates that a small RL-trained MLP can meaningfully improve LLM reasoning performance by dynamically selecting training problems near the Solver's current ability threshold. V2's gains over RLOO are promising given the resource constraints, but the fragility of the approach under bucket collapse points to a clear bottleneck. Future work should investigate curriculum architectures that better balance exploitation of medium-difficulty problems with exploration of the broader problem space.

Learning to Teach, Teaching to Learn

Sam Lustgarten
Department of Computer Science
Stanford University
srl1885@stanford.edu

Isaac Park
Department of Computer Science
Stanford University
iwpark@stanford.edu

Abstract

Training large language models with reinforcement learning is highly sensitive to the difficulty and ordering of problems presented during training, yet standard pipelines rely on static, human-specified curricula that cannot adapt to a model's evolving capabilities. We propose an Adaptive Curriculum Assigner, a small MLP trained with REINFORCE that dynamically selects difficulty buckets of problems for the Solver LLM and is rewarded for targeting problems near the model's current ability threshold (the "Zone of Proximal Development"). Unlike prior multi-agent curriculum approaches, our Assigner conditions on live Solver training statistics including KL divergence, entropy, and per-bucket exponential moving average accuracies. Across three design variants, our best-performing configuration (V2) achieves Pass@1 = 0.653 versus an RLOO baseline of 0.635, with the gap widening to +3.0 points at Pass@8, which is more than half of Self-Evolving Curriculum's improvement despite using a 6× smaller model. We additionally document two empirical failure modes: easy-bucket exploitation and hard-bucket over-sampling, which demonstrate that the Zone is a real but fragile target, and that standard REINFORCE tools such as entropy regularization can prevent concentration on medium-difficulty problems.

1 Introduction

Large language models can solve reasoning tasks after supervised fine-tuning, but their performance depends strongly on the difficulty and ordering of training problems. Problems that are too easy provide little useful learning signal; problems that are too hard yield sparse, unstable rewards. The ideal training problem sits just at the edge of the model's current ability: challenging enough to be informative, tractable enough to occasionally succeed. This principle, known in the human learning literature as the "Zone of Proximal Development," is difficult to implement in LLM training because problem difficulty is not fixed as it depends on the current state of the model. Specifically, a problem that was hard at step 10 may be trivial by step 100, and surface-level difficulty signals like operand count fail to capture subtler sources of difficulty, such as a target answer being a prime number.

Thus, our question is whether a curriculum policy can improve RLOO fine-tuning on Countdown by selecting prompts near the Solver's current empirical ability threshold, using data that tracks the solver's current state. Our extension answers this by introducing an adaptive curriculum policy for the Countdown arithmetic reasoning task. We train an Adaptive Curriculum Assigner, a small MLP with its own RL objective, that observes the Solver LLM's current training state and dynamically selects which difficulty bucket of problems to present next. The Assigner is rewarded for targeting problems near the Solver's current ability threshold and penalized for presenting problems that are too easy or too hard. This approach could improve upon standard SFT, DPO, and RLOO pipelines by replacing static, human-defined curricula with an adaptive policy that responds to the model's evolving capabilities throughout training.

2 Related Work

A key challenge in RL fine-tuning of LLMs is that the learning signal is highly sensitive to problem difficulty. Problems that are too easy provide no gradient; problems that are too hard yield sparse rewards. Curriculum learning, or presenting training problems in a carefully chosen order, has been studied as a solution to this in machine learning (Bengio et al., 2009). Its core idea is motivated by educational psychology, particularly the Zone of Proximal Development (ZPD), a concept that suggests students learn more when they try to accomplish tasks just beyond their current abilities (Chaiklin, 2003). Curriculum learning's motivation is well documented in LLMs: DeepSeek-AI (2025) showed that while RL can create strong reasoning in LLMs, rewards can be sparse or unstable for problems that are "too hard." Meanwhile, other approaches, like Sundaram et al. (2026), have already attempted to give LLMs problems at their "edge of learnability."

Parashar et al. (2026) showed that scheduling tasks from easy to hard improves LLM reasoning substantially over vanilla RL training, particularly for smaller models in the 1.5B–3B parameter range that otherwise struggle to extract signal from hard problems. However, this work establishes difficulty primarily through human-aligned signals such as the number of operands or problem annotations. Thus, this scale does not adapt to the model's evolving capabilities during training.

Chen et al. (2025b)'s Self-Evolving Curriculum (SEC) addresses this by formulating curriculum selection as a non-stationary multi-armed bandit problem. Each problem category (defined by difficulty level or problem type) is treated as a bandit arm, with arms selected to maximize learning gain. Across planning, inductive reasoning, and mathematics tasks, SEC substantially improves generalization to out-of-distribution harder problems. Other work uses multiple interacting agents to create self-improving training loops. Chen et al. (2025a) uses Multi-Agent Evolve, which creates a Proposer, Solver, and Judge from a single shared LLM "backbone" and uses RL to optimize all three simultaneously. The Proposer generates challenging questions, the Solver attempts solutions, and the Judge evaluates both. On Qwen2.5-3B-Instruct, MAE achieves an average improvement of 4.54% across multiple reasoning benchmarks.

Our work is closely related to SEC: both learn a curriculum policy concurrently with RL fine-tuning. The key differences are that our Assigner is an RL-trained MLP rather than a bandit, and it conditions on richer Solver training statistics including KL divergence, entropy, and per-bucket EMA success rates. Moreover, unlike the MAE, an LLM, our Assigner does not generate new problems, but selects from a fixed prompt pool. Finally, through the design of our reward function, our Assigner explicitly optimizes for problems near the Solver's current ability threshold via the Zone of Proximal Development reward.

3 Method

3.1 Method Overview

We propose an adaptive curriculum-learning extension for reinforcement learning on the Countdown reasoning task. The base model, which we call the *Solver*, is a supervised fine-tuned language model that is further trained using the same RLOO reinforcement-learning objective as the baseline. Our extension does not modify the Solver architecture, the rollout format, or the correctness function defined in the earlier milestones of our project. Instead, it modifies the distribution of training prompts seen by the Solver.

The extension adds a second learned policy, called the *Assigner*. The Assigner chooses which difficulty region the Solver should train on at each step. Specifically, the Assigner observes the Solver's current training state, and then outputs a probability distribution over difficulty buckets. A bucket is sampled from this distribution, and the Solver then trains on prompts sampled uniformly from that bucket.

The main (and best) version of our method, V2, uses six dynamically rebalanced difficulty buckets, exponential moving average prompt-level success estimates with $\alpha = 0.15$, and percentile-based rebucketing every 10 training steps. The Assigner is trained with a REINFORCE-style policy-gradient update. Its reward is highest when the selected bucket has estimated Solver success near 50%, encouraging the Assigner to choose prompts that are neither too easy nor too hard.

3.2 Dynamic Difficulty Estimation

We estimate prompt difficulty dynamically using the Solver's own performance. This is necessary for both theoretical and practical reasons. Theoretically, fixed characteristics of a problem such as the number of operands in a Countdown problem may not fully capture the true difficulty of a prompt (as an example, two prompts with the same number of operands may differ in difficulty due to all sorts of other factors, including the target number or the arithmetic operations it may require). And practically, a dynamic system is better because the prompt-dataset we use only has 3 or 4 operands per problem, which gives little information about their true difficulty.

For each prompt i in the training set, we maintain an exponential moving average of Solver success, denoted $\hat{s}_{i,t}$. After the Solver attempts prompt i at training step t, the estimate is updated as

$$\hat{s}_{i,t} = \alpha s_{i,t} + (1 - \alpha)\hat{s}_{i,t-1},$$

where $s_{i,t} \in [0, 1]$ is the observed Solver success rate on prompt i at step t, and α is the EMA update rate. In our main method, we use $\alpha = 0.15$. Lower values of $\hat{s}_{i,t}$ indicate harder prompts, while higher values indicate easier prompts. As a prompt that is difficult early in training can become easy later, prompt difficulty must be (and is) updated throughout training.

3.3 Difficulty Buckets

The training set is partitioned into B difficulty buckets. In the main method, we use $B = 6$. Each bucket contains prompts grouped according to their current EMA success estimates. The lowest-success buckets contain prompts the Solver rarely solves, while the highest-success buckets contain prompts the Solver solves frequently.

In V2 and V3, to prevent the buckets from becoming highly imbalanced as the Solver improves, we rebucket prompts every $R = 10$ training steps. Rebucketing is done by sorting prompts according to their current $\hat{s}_{i,t}$ values and splitting them into percentile-based buckets of approximately equal size. In V2, after each rebucketing step, bucket-level accuracy estimates are reset to 0.5 so that stale bucket statistics do not dominate the Assigner's future decisions.

3.4 Assigner Policy and State Representation

The Assigner is a small MLP parameterized by ϕ. At training step t, it receives a state vector x_t summarizing the current Solver and bucket state. It outputs a categorical distribution over the B difficulty buckets:

$$\pi_\phi(b_t \mid x_t),$$

where $b_t \in \{1, \ldots, B\}$ is the bucket selected at step t.

The Assigner state vector includes the following features:

$$x_t = [A_{1,t}, \ldots, A_{B,t}, C_{1,t}, \ldots, C_{B,t}, A_t^{global}, H_t, D_t^{KL}, t/T],$$

where $A_{b,t}$ is the estimated accuracy of bucket b, $C_{b,t}$ is the normalized count of prompts in bucket b, A_t^{global} is the global Solver accuracy, H_t is the Solver entropy, D_t^{KL} is the Solver KL divergence, and t/T is the normalized training step. The Assigner therefore observes both local information about each difficulty bucket and global information about the Solver's training dynamics.

At each training step, the Assigner samples a bucket

$$b_t \sim \pi_\phi(\cdot \mid x_t).$$

The Solver batch is then constructed by sampling prompts uniformly from the selected bucket.

3.5 Assigner Reward

The Assigner is rewarded for selecting buckets that are close to the Solver's current learning boundary, which we define as the region of problems the Solver correctly answers half of the time. Let S_t denote the estimated Solver success rate. The Assigner per-sample reward is

$$r_t = 1 - |S_t - 0.5|.$$

This reward is maximized when $S_t = 0.5$. It decreases when the selected bucket is either too easy (when S_t is closer to 1) or too hard (when the S_t is closer to 0).

In all versions we implemented, S_t is the success rate of the specific bucket from which the assigner chose its new prompt. In the methods V2 and V3, S_t is computed from a snapshot of the selected bucket's accuracy before the Solver performs the current rollout, which avoids rewarding the Assigner based on outcomes from the same prompts it just selected, which would create a circular reward signal.

3.6 Assigner Optimization

The Assigner is trained using a REINFORCE-style policy-gradient objective. For a batch of n sampled prompts, each prompt t receives a per-sample reward $r_t = 1 - |S_t - 0.5|$, where S_t is the pre-rollout accuracy of its sampled bucket. To reduce gradient variance in V2 and V3, we subtract a leave-one-out baseline from each reward to form advantages:

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

The Assigner loss is then

$$\mathcal{L}_{\text{assigner}} = -\frac{1}{n} \sum_{i=1}^{n} A_i \log \pi_\phi(b_i \mid x_t) - \lambda H(\pi_\phi(\cdot \mid x_t))$$

where $H(\pi_\phi(\cdot \mid x_t))$ is the entropy of the Assigner's bucket distribution and λ is an entropy regularization coefficient. The Solver and Assigner are updated in the same training loop: the Assigner chooses a bucket, the Solver trains on prompts from that bucket, prompt-level EMA estimates are updated using Solver correctness, and the Assigner is updated according to the curriculum reward.

3.7 Algorithm

Algorithm 1 Adaptive Curriculum Assigner

Require: Solver policy π_θ, Assigner policy π_ϕ, prompt set $\mathcal{D}$, number of buckets $B = 6$, EMA rate $\alpha = 0.15$, rebucketing interval $R = 10$

Initialize prompt success estimates $\hat{s}_{i,0} = 0.5$ for all prompts $i \in \mathcal{D}$
Partition $\mathcal{D}$ into B difficulty buckets using current $\hat{s}_{i,0}$ values
Initialize bucket accuracy estimates $A_{b,0} = 0.5$ for all buckets b
for training step $t = 1, \ldots, T$ **do**
    Compute Assigner state x_t from bucket accuracies, bucket counts, global Solver accuracy, entropy, KL divergence, and normalized training step
    Sample bucket $b_t \sim \pi_\phi(\cdot \mid x_t)$
    Set $S_t \leftarrow A_{b_t,t}$ using the pre-rollout bucket accuracy snapshot
    Compute Assigner reward for each prompt (as above) $r_t = 1 - |S_t - 0.5|$
    Sample a batch of prompts uniformly from bucket b_t
    Generate Solver rollouts using π_θ
    Compute rollout correctness for each sampled prompt
    Update Solver policy π_θ using the RLOO objective
    Compute LOO advantages $A_i = r_i - \frac{1}{n-1} \sum_{j \neq i} r_j$
    Update Assigner policy π_ϕ using $\mathcal{L}_{\text{assigner}} = -\frac{1}{n} \sum_i A_i \log \pi_\phi(b_i \mid x_t) - \lambda H(\pi_\phi(\cdot \mid x_t))$
    Update prompt-level EMA values $\hat{s}_{i,t} = \alpha s_{i,t} + (1 - \alpha)\hat{s}_{i,t-1}$ for sampled prompts
    Update bucket-level accuracy estimates using the new sampled-prompt outcomes
    if $t \bmod R = 0$ **then**
        Repartition prompts into B approximately equal-sized buckets using percentiles of $\hat{s}_{i,t}$
        V2 only: Reset bucket accuracy estimates to 0.5
    end if
end for

3.8 Ablation Variants

We implement three variants of the adaptive curriculum extension.

V1: Fixed Bucket Curriculum. V1 uses six fixed buckets defined by success-rate ranges. It uses an EMA update rate of $\alpha = 0.5$, making prompt difficulty estimates more reactive to recent Solver outcomes. The Assigner reward is computed after the Solver attempts the selected prompts. This variant tests whether a simple success-range bucket system is sufficient for adaptive curriculum learning.

V2 (main method): Percentile-Rebalanced Curriculum. V2 uses six buckets, an EMA update rate of $\alpha = 0.15$, pre-rollout bucket-accuracy snapshots for the Assigner reward, and percentile-based rebucketing every 10 training steps. This variant is designed to reduce noisy difficulty estimates, avoid circular reward assignment, and maintain balanced bucket populations throughout training.

V3: Entropy-Regularized Curriculum. V3 tests whether stronger exploration improves the Assigner. It reduces the number of buckets from six to four, adds entropy regularization to the Assigner objective, allows the Assigner to observe actual bucket accuracies rather than reset-only values, and uses an exponential Assigner learning-rate schedule. This variant tests whether encouraging broader bucket exploration improves or harms adaptive curriculum selection.

3.9 Assumptions and Novelty

Our method relies on three main assumptions: that recent empirical Solver success is a useful proxy for prompt difficulty; that prompts near the Solver's current learning boundary provide the most useful training signal; and finally, that this boundary can be calculated as selected-bucket success near 50%. The novelty of the method is that the curriculum is not fixed in advance, nor based on the characteristics of our task dataset like operands. Instead, difficulty is estimated dynamically from Solver behavior, and a learned Assigner policy is trained to sample from the most useful region of

the difficulty distribution. The Assigner adapts its sampling policy throughout training using Solver performance, bucket statistics, entropy, KL divergence, and training progress.

4 Experimental Setup

4.1 Task and Dataset

We evaluate our method on the Countdown arithmetic reasoning task. Each prompt gives the model a target number and a set of allowed input numbers. The model must produce an arithmetic expression that uses the allowed numbers to equal the target. A rollout is counted as correct only if the generated expression evaluates to the target and obeys the constraint that the provided numbers are used validly. We use the `asingh15/countdown_tasks_3to4` dataset, which contains Countdown problems with three to four input numbers. This task is well suited for evaluating reinforcement learning and curriculum learning because correctness can be checked automatically, while difficulty varies across prompts in ways that are not always captured by simple prompt-features like number of operands.

4.2 Models and Baselines

The base Solver is initialized from the supervised fine-tuned language model for Countdown reasoning from the default project milestone. For context, SEC reports its main Countdown results using Qwen2.5-3B and Qwen2.5-7B models, whereas our Solver uses the required Qwen2.5-0.5B Base model. Relative to SEC's smaller 3B setting, our model is 6× smaller. We use the same Solver architecture and rollout format across all experiments, so differences in performance relative to our RLOO baseline can be attributed to the curriculum sampling strategy rather than changes in model size or prompt format.

Our primary baseline is the default RLOO reinforcement-learning setup. This is the most direct comparison because our extension keeps the Solver's RLOO objective fixed and only changes how training prompts are selected. We compare the RLOO baseline against three adaptive curriculum variants:

- **RLOO Baseline:** the original reinforcement-learning training loop without an adaptive curriculum assigner.
- **V1: Fixed Bucket Curriculum:** an adaptive assigner with six fixed success-rate buckets and EMA rate $\alpha = 0.5$.
- **V2: Percentile-Rebalanced Curriculum:** our main method, using six percentile-rebalanced buckets, EMA rate $\alpha = 0.15$, pre-rollout bucket-accuracy rewards, and rebucketing every 10 steps.
- **V3: Entropy-Regularized Curriculum:** an exploratory variant using four buckets, entropy regularization, actual bucket-accuracy observations, and an exponential assigner learning-rate schedule.

The RLOO baseline is the key baseline because it isolates the effect of curriculum learning. V1 and V3 serve as ablations that test whether fixed bucket boundaries, noisier EMA updates, fewer buckets, and entropy-driven exploration improve or harm the adaptive curriculum.

4.3 Training Details

All methods use the same underlying Solver training objective and correctness function. The Solver is updated using RLOO, while the adaptive variants additionally train the Assigner policy. The Assigner is a small MLP that outputs a categorical distribution over difficulty buckets.

Table 1 summarizes the key hyperparameters used for our main V2 adaptive curriculum run. We keep the Solver objective fixed to RLOO and change only the training prompt distribution through the learned Assigner policy. For reproducibility, Appendix A reports the full set of training, sampling, curriculum, checkpointing, and hardware settings used in the V2 run. The Python trainer contains fallback defaults, but the values reported here are the launch-time values passed by our V2 training script.

Hyperparameter	Value
Initial Solver policy	SFT Countdown checkpoint passed through `MODEL_NAME`
Reference policy	Same as initial Solver policy
Dataset	`asingh15/countdown_tasks_3to4`
Task	Countdown arithmetic reasoning with 3–4 numbers
Solver objective	RLOO
Solver learning rate	1×10^{-5}
Batch size	128 prompts
Rollouts per prompt	8
Generated completions per update	1024
Training steps	100
Solver Entropy coefficient	0.001
Sampling settings	temperature $= 1.0$, top-$p = 1.0$, top-$k = -1$
Curriculum buckets	6
Prompt difficulty estimate	EMA of Solver success
Prompt-success EMA rate	$\alpha = 0.15$
Rebucketing interval	Every 10 training steps
Assigner reward target	50% selected-bucket Solver success
Assigner objective	REINFORCE policy-gradient loss
Assigner learning rate	1×10^{-2}
Assigner scheduler	LinearLR, start factor 0.1, end factor 1.0, total iterations 20
Assigner architecture	MLP: $16 \to 64 \to 32 \to 6$

Table 1: Key hyperparameters for the main V2 adaptive curriculum run.

For V1, we use six fixed buckets and a larger EMA rate of $\alpha = 0.5$. For V3, we use 4 buckets and add entropy regularization to encourage bucket exploration. Aside from the curriculum-specific changes, all variants keep the Solver training setup as close as possible to the RLOO baseline.

4.4 Evaluation Metrics

We evaluate models using Pass@K on held-out Countdown prompts. For each prompt, the model samples K independent completions. The prompt is counted as solved if at least one of the K completions is correct. In total, Pass@K is the fraction of prompts from the evaluation set where at least one of the model's K attempts to solve the prompt are correct. We report Pass@1, Pass@4, Pass@8, and Pass@16.

Pass@K is appropriate for Countdown because language models often find multiple valid ways to reason about one problem, and a model's usefulness can depend on whether it can find a correct solution within a small number of attempts. Pass@1 measures single-sample reliability, while larger values of K measure whether the model has learned a useful solution distribution even when its first sample is incorrect. Most importantly, Pass@K is used in most related papers (including almost all the above papers in Related Work), so Pass@K allows us to use a similar metric to compare our model.

4.5 Quantitative and Qualitative Evaluation

The quantitative evaluation compares the RLOO baseline against V1, V2, and V3 using Pass@K. This tests whether adaptive curriculum selection improves final reasoning performance over standard RLOO training.

In addition to aggregate Pass@K, we analyze qualitative training behavior. Specifically, we examine bucket-selection trajectories, bucket collapse, over-exploration, and example rollouts. These qualitative analyses can help explain why a curriculum variant succeeds or fails, rather than only reporting final accuracy numbers.

5 Results

5.1 Quantitative Evaluation

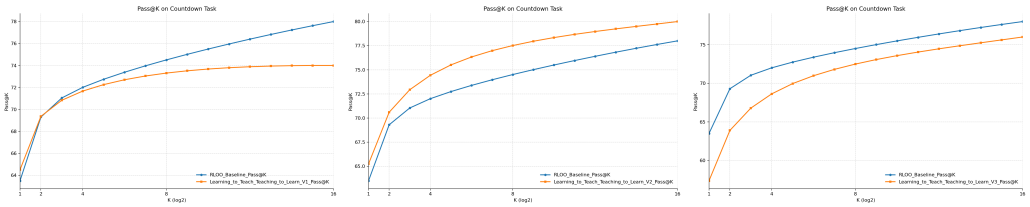
Table 2 reports held-out Countdown performance using Pass@K for $K \in \{1, 4, 8, 16\}$. Each method was evaluated from a single training run due to compute constraints. The main curriculum variant, V2, is the only method that consistently improves over the RLOO baseline across all values of K.

Table 2: Pass@K comparison on held-out Countdown tasks. Higher is better.

Method	Pass@1	Pass@4	Pass@8	Pass@16
RLOO Baseline	0.635	0.720	0.745	0.780
Extension V1	0.645	0.717	0.733	0.740
Extension V2	**0.653**	**0.744**	**0.775**	**0.800**
Extension V3	0.574	0.686	0.725	0.760

V2 improves the baseline by $+0.018$ Pass@1, $+0.024$ Pass@4, $+0.030$ Pass@8, and $+0.020$ Pass@16. The improvement is largest at Pass@8, suggesting that the learned curriculum does not merely increase the probability of the first sample being correct, but also improves the model's ability to produce at least one valid solution across multiple rollouts.

V1 produces only a small Pass@1 gain over the baseline and underperforms at larger K. This pattern suggests that V1 helped the model solve only a narrow subset of prompts more confidently. V3 performs worse than the baseline at every value of K, indicating that the additional entropy regularization and coarser four-bucket curriculum overcorrected toward exploration rather than a more-refined learning trajectory.



(a) V1 vs. baseline (b) V2 vs. baseline (c) V3 vs. baseline

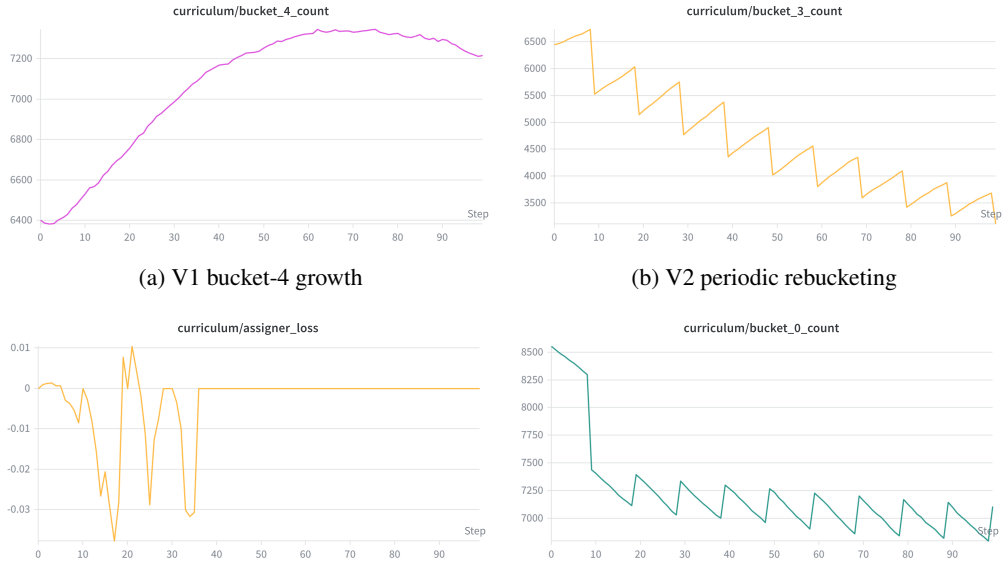
Figure 1: Pass@K curves for each curriculum variant against the RLOO baseline. RLOO baseline performance is shown in blue, while the respective Vn performance is shown in orange. V2 is the only variant that consistently dominates the baseline across all tested values of K.

5.2 Qualitative Analysis

The quantitative results are explained by the training dynamics of the learned assigner. Figure 2 shows representative W&B diagnostics for the three curriculum variants. V1 used fixed difficulty buckets, a relatively large EMA update rate, and a reward computed after the solver attempted the assigned examples. This likely created a feedback loop: once the solver became good at a bucket, the assigner received high reward for selecting it, even if the bucket no longer provided useful learning signal. The growth of bucket 4 in Figure 2(a) is consistent with this collapse toward high-reward, already-learned examples.

V2 addresses this failure mode by using a pre-rollout reward snapshot, lowering the EMA coefficient, and rebucketing examples by current solver success percentiles every 10 steps. The sawtooth pattern in Figure 2(b) reflects this periodic rebucketing: as examples become easier or harder relative to the current solver, they are reassigned rather than remaining in stale fixed bins. This makes the curriculum more adaptive and explains why V2 improves both Pass@1 and higher-K performance. However, Figure 2(c) also shows that the assigner loss goes to approximately zero around step 37, and the inclusion of LOO means that this is the result of V2 collapsing to choosing a single bucket.

V3 attempted to avoid collapse by reducing the number of buckets to four and adding entropy regularization. This produced more exploration, but the evaluation results suggest that the exploration was not targeted enough. Using only 4 buckets meant that comparisons of difficulty across problems were much coarser than with 6 buckets, and the entropy term penalized the useful concentration on medium-difficulty prompts that V2 had discovered. As a result, V3 likely sampled more broadly but provided weaker training signal to the solver.



(a) V1 bucket-4 growth

(b) V2 periodic rebucketing

(c) V2 assigner-loss plateau

(d) V3 hard-bucket dynamics

Figure 2: Curriculum diagnostics. V1 exhibits bucket collapse, V2 exhibits the intended rebucketing dynamics but later loses assigner signal, and V3 explores more broadly without improving final Pass@K.

Overall, the results support the hypothesis that learned curricula can improve RL fine-tuning for reasoning tasks, but only when the curriculum reward is aligned with the solver's perceived difficulty before updating and the bucket structure remains calibrated to the solver's current ability. V2 is the strongest variant because it makes the difficulty labels dynamic and prevents the assigner from being rewarded for examples that only became easy after the current solver update. At the same time, the plateau in assigner loss shows that the method remains sensitive to advantage estimation and bucket collapse. Future versions should evaluate multiple random seeds, use a denser dataset and model, and add an explicit ablation for rebucketing frequency and entropy weight.

6 Discussion

V2's improvement over the RLOO baseline confirms that adaptive curriculum selection can provide a meaningful training signal. Small design decisions determine whether the Assigner concentrates on useful medium-difficulty problems or drifts toward failure modes at either extreme.

V1 demonstrates easy-bucket exploitation. Because the Assigner reward was computed after rollouts using global Solver accuracy, the Assigner learned to select buckets it already knew the Solver could solve. The fixed bucket boundaries collapsed as the Solver improved, leading the Assigner to select bucket 4 throughout all of its training.

V3 demonstrates the opposite failure. Entropy regularization likely over-penalized the Assigner for concentrating on medium-difficulty buckets, which is actually the behavior the reward function is designed to encourage. Reducing the number of buckets from six to four also reduced the Assigner's resolution over task difficulty, so the hardest bucket spanned a wider range of actual problem difficulties. Its Pass@1 of 0.574, which is well below the RLOO baseline, shows that forced

exploration can actively harm curriculum quality when the goal is to stay near a specific difficulty region rather than explore broadly.

V2's primary limitation is bucket collapse. If the policy forces all prompts in a batch to be drawn from the same bucket, all per-sample rewards within a step are identical, causing the leave-one-out baseline to produce zero advantage estimates and collapse the Assigner's gradient signal. In particular, because the V2 loss uses standard advantage-weighted log-likelihood, a constant zero requires the leave-one-out advantages to remain 0 across steps. This rules out the possibility that our rebalancing of all buckets to 0.5 forced the loss to remain at 0 because we would expect a "sawtooth" pattern to emerge as prompt-accuracies diverge after rebucketing. Furthermore, because the Assigner's loss never moved from 0 after step 37, this indicates the Assigner's policy has collapsed onto a single bucket. Addressing this incompatibility is the most pressing direction for future work.

More broadly, adaptive curriculum learning could improve the efficiency of RL fine-tuning by focusing the model on examples that are most useful for the current model, potentially making reasoning-model training more efficient with each iteration. However, we also demonstrated that this method introduces new risks: our V1 shows how poor model specifications may result in reward-hacking, while V3 shows how encouraging more exploration may actually degrade the model's ability to focus on those medium-difficulty problems we want it to consistently train on. Future evaluations should therefore test adaptive curricula across multiple tasks and include diagnostics for curriculum collapse, reward hacking, and exploration.

7 Conclusion

We introduced the Adaptive Curriculum Assigner, a small MLP trained with REINFORCE that dynamically selects difficulty buckets of training problems for a Solver LLM during RL fine-tuning on the Countdown task. Our best-performing variant, V2, achieves Pass@1 = 0.653 against an RLOO baseline of 0.635, with the gap widening to 3.0 points at Pass@8, which is a $\sim$2.7% relative improvement. While lower than Self-Evolving Curriculum's reported improvement, we note that we accomplished this despite using a 6$\times$ smaller model.

Beyond the positive result, we empirically document two curriculum learning failure modes. These findings suggest that the Zone of Proximal Development is a real training phenomenon but requires careful Assigner design to exploit. Future work should explore gradient estimators robust to low within-batch reward variance, multi-bucket sampling strategies that restore reward diversity within each step, and curriculum architectures that can balance staying near the Zone of Proximal Development with broader exploration of the problem space.

8 Team Contributions

- **Sam Lustgarten** designed the adaptive curriculum structure, defined difficulty buckets for Countdown problems, and ran ablation experiments comparing fixed versus adaptive curricula.
- **Isaac Park** implemented the Assigner model, fine-tuned the Assigner reward function, and compared our curriculum extension against standard RLOO.

Changes from Proposal: No changes from the original proposal.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th International Conference on Machine Learning (ACM International Conference Proceeding Series, Vol. 382)*. ACM, 41–48. doi:10.1145/1553374.1553380

Seth Chaiklin. 2003. The Zone of Proximal Development in Vygotsky's Analysis of Learning and Instruction. In *Vygotsky's Educational Theory in Cultural Context*. Cambridge University Press, 39–64.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025b. Self-Evolving Curriculum for LLM Reasoning. *arXiv preprint arXiv:2505.14970* (2025).

Yixing Chen et al. 2025a. Multi-Agent Evolve: LLM Self-Improve through Co-Evolution. *arXiv preprint arXiv:2510.23595* (2025).

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025). `https://arxiv.org/abs/2501.12948`

Shubham Parashar et al. 2026. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. *arXiv preprint arXiv:2506.06632* (2026).

Shobhita Sundaram, John Quan, Ariel Kwiatkowski, Kartik Ahuja, Yann Ollivier, and Julia Kempe. 2026. Teaching Models to Teach Themselves: Reasoning at the Edge of Learnability. *arXiv preprint arXiv:2601.18778* (2026). `https://arxiv.org/abs/2601.18778`

A Full Hyperparameter Table

Table 3: Full training and curriculum hyperparameters for the V2 adaptive curriculum run.

Category	**Hyperparameter**	**Value**
Model	Initial Solver policy	SFT Countdown checkpoint passed through `MODEL_NAME`
Model	Reference policy	Same as initial Solver policy
Tokenizer	Tokenizer	`Qwen/Qwen2.5-0.5B`
Dataset	Dataset	`asingh15/countdown_tasks_3to4`
Dataset	Splits	`train`, `test`
Solver training	Solver objective	RLOO
Solver training	Optimizer	AdamW
Solver training	Learning rate	1×10^{-5}
Solver training	Learning-rate schedule	Constant
Solver training	Warmup ratio	0.0
Solver training	Weight decay	1×10^{-4}
Solver training	Batch size	128 prompts
Solver training	Rollouts per prompt / group size	8
Solver training	Generated completions per update	$128 \times 8 = 1024$
Solver training	Gradient accumulation steps	128
Solver training	Gradient clipping	1.0
Solver training	Number of training steps	100
RLOO regularization	KL coefficient	0.001
RLOO regularization	Entropy coefficient	0.001
RLOO update	Leave-one-out baseline	Enabled within each response group
RLOO update	Importance weights	Enabled when sample log-probs are available
RLOO update	Importance-weight clamp	$[0.8, 1.2]$
Sampling	Temperature	1.0
Sampling	Top-p	1.0
Sampling	Top-k	-1
Sampling	Min-p	0.0
Sampling	Max response tokens	1024
Sampling	Max prompt length	512
Sampling	Max model length	2048
Curriculum	Curriculum enabled	True
Curriculum	Number of buckets	6

Category	Hyperparameter	Value
Curriculum	Prompt difficulty estimate	EMA of Solver success
Curriculum	Prompt-success EMA rate	$\alpha = 0.15$
Curriculum	Initial prompt success estimate	0.5
Curriculum	Initial bucket accuracy	0.5 for all buckets
Curriculum	Initial bucket assignment	Random bucket assignment
Curriculum	Bucket construction after rebucketing	Percentiles of prompt EMA success
Curriculum	Rebucketing interval	Every 10 training steps
Curriculum	Assigner reward target	Selected-bucket success near 50%
Curriculum	Assigner objective	REINFORCE policy-gradient loss
Curriculum	Assigner baseline	Leave-one-out baseline over sampled bucket rewards
Assigner	Architecture	MLP: $16 \to 64 \to 32 \to 6$
Assigner	Input dimension	$2B + 4 = 16$, where $B = 6$ buckets
Assigner	Hidden sizes	64, 32
Assigner	Activation	ReLU
Assigner	Output layer	Softmax over difficulty buckets
Assigner	Optimizer	Adam
Assigner	Learning rate	1×10^{-2}
Assigner	Scheduler	LinearLR
Assigner	Scheduler start factor	0.1
Assigner	Scheduler end factor	1.0
Assigner	Scheduler total iterations	20