Static vs. Adaptive Curriculum Learning for RLOO Fine-Tuning of Language Models

Norah Asemota

One-Page Extended Abstract

Reinforcement learning (RL) fine-tuning has become central to improving the reasoning ability of large language models (LLMs), but online RL methods such as RLOO suffer from sparse, high-variance reward early in training: when a small model fails on most hard problems, the majority of sampled rollouts return no usable learning signal. Curriculum learning is a natural remedy — order the training data by difficulty so the model first learns from problems it can actually solve, raising reward density early, before progressing to harder ones. This project asks a focused question: *does ordering training data by difficulty make RLOO fine-tuning more effective on a reasoning task, and does a performance-adaptive curriculum outperform a fixed one?*

I study the Countdown arithmetic task with a Qwen2.5-0.5B policy (initialized from supervised fine-tuning and IPO). I define task difficulty by the number of input operands (three operands "easy," four operands "hard") and compare three difficulty-sampling strategies under an otherwise identical RLOO pipeline: (i) random sampling, (ii) a *static* curriculum that ramps from easy to hard on a fixed step schedule, and (iii) an *adaptive* curriculum that advances difficulty only after the model's measured success rate on easier problems clears a mastery threshold. I implement all three through a single scalar "front" that maps to a sampling distribution over difficulty levels, so the only thing that varies across conditions is the difficulty mix — the sampling machinery, model, reward, and hyperparameters are held constant.

Main findings.

(1) **RLOO with a curriculum roughly doubles single-shot accuracy.** Both the static and adaptive runs improve pass@1 on the held-out test set from ≈ 0.30 (the SFT initialization) to 0.60–0.64, and pass@16 from ≈ 0.72 to 0.82, while training stably to ≈ 0.62 mean reward.

(2) **The choice of difficulty strategy made little difference to final capability.** Static and adaptive differ by at most 0.04 on every pass@k metric, including the by-difficulty breakdown — within the noise of the test subset.

(3) **Easy→hard skill transfer explains this.** The adaptive controller, due to its threshold gate, never advanced and sampled *zero* hard problems for the entire run. Despite this, it matched (indeed slightly exceeded) the static curriculum on hard four-operand problems at test time. Mastering three-operand problems appears to teach a transferable procedure — search, arithmetic, and answer formatting — that generalizes to four-operand problems without explicit training on them. On this task, the "hard" portion of the curriculum was therefore largely redundant.

(4) **Adaptive curricula are brittle.** A controller that appears reasonable can silently fail to advance, erasing the intended curriculum without raising any error. This is a practical cautionary result for performance-gated scheduling.

Relation to the proposal. The proposal hypothesized that adaptive curricula would improve final reasoning performance over static ones. My results do not support this: the adaptive controller failed to advance, and even setting that aside, difficulty ordering had little effect because of skill transfer. I treat the negative result as the central finding and discuss its implications for when curricula should and should not help. The single-author contribution breakdown and changes from the proposal plan appear in Section 7.

Abstract

I investigate whether curriculum learning over task difficulty improves RLOO fine-tuning of a small language model on the Countdown arithmetic-reasoning task. I define difficulty by operand count and compare random, static, and adaptive difficulty-sampling strategies through a unified "front" mechanism that varies only the difficulty distribution. RLOO with a curriculum roughly doubles single-shot test accuracy over the supervised initialization ($0.30 \rightarrow 0.60$–0.64 pass@1). However, the static and adaptive strategies are statistically indistinguishable on final capability (≤ 0.04 on all metrics). My adaptive controller failed to advance difficulty, training only on easy problems, yet matched the static curriculum even on hard problems it never saw. I attribute this to easy→hard skill transfer on Countdown and conclude that, at this scale, getting the RL pipeline and reward signal right matters more than the specific difficulty ordering. I discuss the brittleness of performance-gated adaptive curricula and the conditions under which difficulty ordering would be expected to help.

1 Introduction

Reinforcement learning from verifiable rewards has driven recent improvements in LLM reasoning, typically by sampling candidate solutions and rewarding correct ones [1]. A recurring difficulty in this regime is that reward is *sparse and high-variance* early in training: a small, weakly-initialized policy fails on most challenging problems, so a large fraction of sampled rollouts receive the minimum reward and contribute little gradient signal. For group-baseline methods like RLOO [2], this is compounded by the fact that prompts on which all sampled responses receive the same reward produce *zero* advantage and hence no learning signal at all.

Curriculum learning offers a principled response: if I present easier problems first — problems the model can actually solve — I increase the density of informative, non-zero-reward rollouts early in training, when the policy is weakest. Prior work has shown that easy-to-hard ordering improves reasoning and sample efficiency in RL fine-tuning [3], and that difficulty schedules can be made adaptive to the model's own learning progress [4]. An open practical question is whether the added complexity of an *adaptive* schedule pays off relative to a simple *fixed* one in a standard RLOO pipeline.

I study this question on the Countdown arithmetic task using a Qwen2.5-0.5B policy. My contributions are:

- A unified curriculum mechanism — a scalar "front" that maps to a sampling distribution over difficulty levels — that expresses random, static, and adaptive strategies through a single code path, so that only the difficulty distribution differs across conditions (Section 3).
- A controlled comparison of the three strategies showing that RLOO with a curriculum roughly doubles pass@1 over the SFT initialization, but that the choice of difficulty strategy has little effect on final capability (Section 5).
- An analysis attributing this null effect to easy→hard skill transfer, supported by the observation that a model trained only on easy problems matches one trained on a balanced mix even on hard problems (Section 6).
- A practical cautionary finding on the brittleness of performance-gated adaptive curricula.

2 Related Work

Online RL for LLM reasoning. Online RL methods directly optimize task reward and have produced strong gains on mathematical and multi-step reasoning [1]. RLOO [2] is a lightweight REINFORCE-style estimator that replaces a learned value function with a leave-one-out baseline computed within a group of sampled responses, reducing variance without a critic. These methods

optimize for task-specific rewards but do not, by themselves, prescribe *which* problems to train on or in what order; this project studies precisely that axis on top of an RLOO pipeline.

Curriculum learning for RL fine-tuning. Parashar et al. [3] show that ordering training problems from easy to hard improves LLM reasoning and sample efficiency under sparse reward, but use a fixed (static) curriculum and do not study adaptive difficulty scheduling. Chen et al. [4] propose a self-evolving curriculum that adjusts task difficulty during training based on learning progress, and Sundaram et al. [5] use a teacher–student framework that generates intermediate "stepping-stone" problems to overcome plateaus; both add significant machinery (a learned scheduler or a separate teacher model). My contribution is narrower and complementary: I ask whether a *simple* performance-gated adaptive schedule improves over a fixed one within a standard RLOO pipeline, holding everything else constant, and I report a negative result together with a mechanistic explanation (skill transfer) for why the ordering had little effect on this task.

3 Method

3.1 RLOO objective

For each prompt x I sample a group of G responses $\{y_1, \ldots, y_G\}$ from the current policy π_θ and score each with a task reward $R_i = R(x, y_i)$. RLOO computes a leave-one-out baseline for each response from the *other* members of its group, yielding the advantage

$$A_i \;=\; R_i \;-\; \frac{1}{G-1}\sum_{j\neq i} R_j. \tag{1}$$

The policy-gradient loss is taken over response tokens only, with an entropy bonus and a KL penalty to a fixed reference policy π_{ref} for stability. Because I sample with the inference engine and take a single gradient step from the same checkpoint each round, training is on-policy and the sequence-level importance weight is identically one.

3.2 Difficulty and the curriculum front

I define task difficulty by the number of input operands in a Countdown problem: three operands is "easy" (d=3), four is "hard" (d=4). Let the sorted difficulty levels be indexed $i = 0, \ldots, K-1$ (here K=2). I represent curriculum state by a single scalar *front* $f \in [1, K]$ and define per-level sampling weights and probabilities

$$w_i \;=\; \text{clip}(f - i,\; 0,\; 1), \qquad p_i \;=\; \frac{w_i}{\sum_j w_j}. \tag{2}$$

Intuitively (Figure 1), the easiest level always retains full weight while harder levels phase in as f grows: at f=1 only easy problems are sampled; at f=2 the mix is balanced. This is a *cumulative* easy→hard schedule that adds hard problems without abandoning easy ones, preserving reward density throughout.

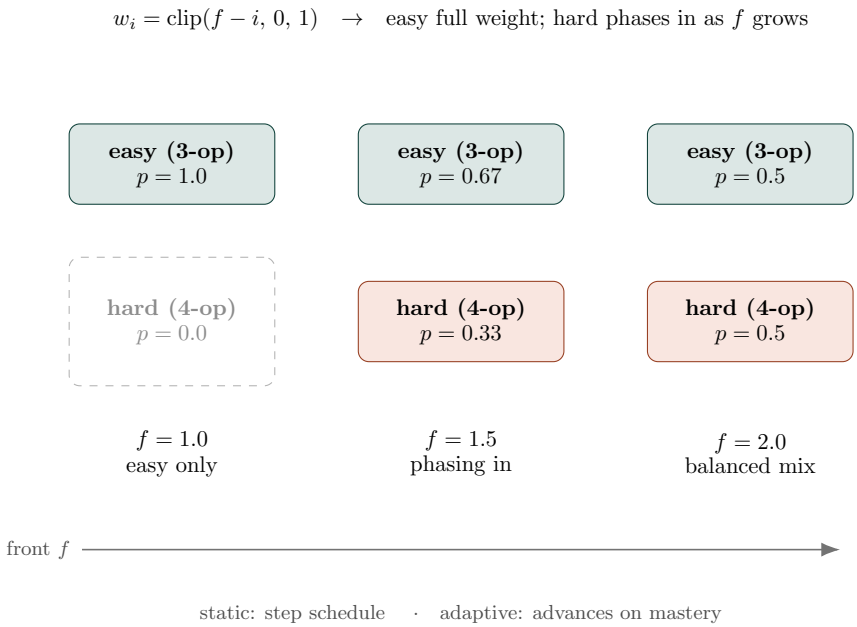


Figure 1: The scalar front f maps to a sampling distribution over difficulty. At f=1.0 only easy (d=3) problems are sampled; as f rises to 2.0, hard (d=4) problems phase in to a balanced mix.

3.3 Three strategies

The three conditions differ only in how f (and thus p_i) is set each step:

- **Random (no curriculum):** p_i is fixed to the dataset's natural difficulty proportions; equivalently, examples are drawn uniformly.
- **Static:** f is a deterministic function of the training step — a linear ramp from 1 to K (optionally after an easy-only warmup). Difficulty tracks the clock.
- **Adaptive:** f advances only when an exponential moving average (EMA) of the solve rate on the current frontier level exceeds a mastery threshold τ; otherwise it is held. Difficulty tracks the model's measured learning progress.

Crucially, all three share one sampling pipeline, the same RLOO update, the same model, and identical hyperparameters; only the difficulty distribution p_i differs. This isolates the effect of difficulty ordering.

4 Experimental Setup

Task and data. I use the `countdown_tasks_3to4` dataset. Each problem provides a set of numbers and a target; the model must produce an arithmetic expression using each number once that evaluates to the target. Prompts elicit a `<think>` reasoning trace followed by an `<answer>` expression. Roughly 60% of problems are three-operand (easy) and 40% are four-operand (hard).

Reward. The reward is 1.0 if the extracted expression evaluates exactly to the target, 0.1 for a well-formed expression that uses the numbers but yields the wrong value, and 0 otherwise. I define a problem as "solved" when reward ≥ 1.0 and report solve rate accordingly.

Model and pipeline. The policy is Qwen2.5-0.5B, initialized from a supervised fine-tuning (SFT) checkpoint followed by IPO preference optimization, consistent with the standard SFT → preference → online-RL pipeline. RLOO sampling uses a vLLM engine; gradient updates use a separate worker. Shared hyperparameters across all runs: learning rate 10^{-5} (constant), batch size 128 prompts, group size G=8, KL coefficient 10^{-3}, entropy coefficient 10^{-3}, and 100 training steps. Sampling for training uses temperature 1.0. All experiments run on a single H100 GPU.

Curriculum settings. The static schedule ramps f linearly over training. The adaptive schedule uses EMA decay 0.9, mastery threshold τ=0.5 on the per-prompt solve rate, and a small promotion increment with a short cooldown between promotions.

Implementation correctness. Obtaining a learning signal at all required two corrections to the base RLOO update that I note for reproducibility: (i) the per-token policy-gradient loss must be masked to *real* response tokens (the response indicator alone includes right-padding), and (ii) because training is on-policy, the sequence-level importance weight should be exactly one; an empirical re-derivation of the ratio from re-tokenized text introduced large multiplicative variance. With these in place, mean reward rose from a flat ≈0.30 random walk to a stable climb (Section 5); these are properties of the RLOO implementation rather than of the curriculum, and I report them only to contextualize the training curves.

Evaluation. I evaluate each final checkpoint with pass@k on the held-out test split, sampling 16 responses per problem at temperature 0.6 and counting a problem as passed at k if any of the first k samples is exactly correct. I report both aggregate pass@k and pass@k split by difficulty.

5 Results

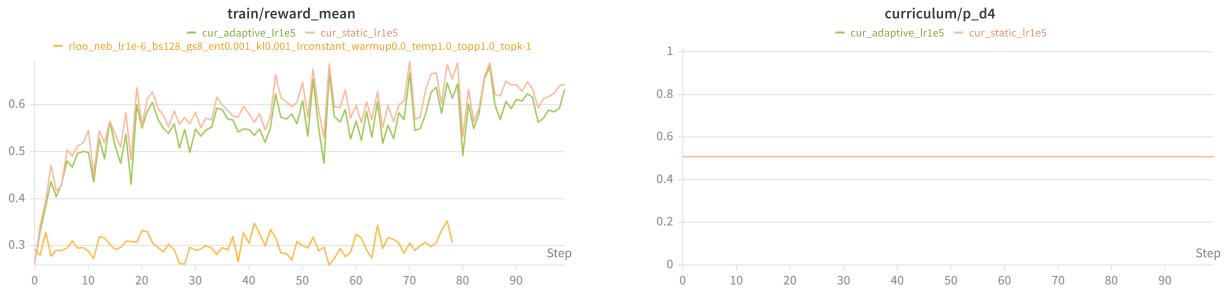
5.1 Quantitative Evaluation

Training dynamics. Both curricula train stably, with mean reward climbing from ≈ 0.29 to ≈ 0.62 over 100 steps and no collapse (Figure 2a). As a reference point, an early configuration without the implementation corrections noted above remained flat near 0.30 for the entire run, indicating that the corrected pipeline is what enables learning.

The realized difficulty mix (Figure 2b) reveals an important asymmetry between the two curricula. The static schedule held a balanced ≈ 50/50 easy/hard mix as designed. The adaptive schedule, however, *never advanced its front*: despite the easy-problem solve rate reaching ≈0.85 — well above the mastery threshold — the controller's front stayed pinned at f=1.0, and it sampled *zero* hard (four-operand) problems for all 100 steps. The adaptive run therefore trained on a strict easy-only subset of the data, so the static and adaptive conditions are not a clean like-for-like comparison of difficulty schedules; I return to this in Section 6.

Final capability. Table 1 reports aggregate pass@k. Both curricula roughly *double* the single-shot accuracy of the SFT initialization, lifting pass@1 from ≈0.30 to 0.60–0.64 and pass@16 from ≈0.72 to 0.82. Adaptive holds a slight aggregate edge (0.64 vs. 0.60 pass@1), but this is consistent with the test set's easy-majority composition and the fact that adaptive concentrated all of its training on easy problems.

The by-difficulty breakdown (Table 2) is the most informative result. On easy problems both models reach 0.875–0.917 pass@1; on hard problems both land at 0.346–0.385 pass@1. The two



(a) Mean reward over training.

(b) Share of hard (d=4) problems per batch.

Figure 2: Training dynamics. Left: both curricula learn stably. Right: the static curriculum maintains a balanced difficulty mix, while the adaptive controller never advances and trains only on easy problems.

Table 1: Aggregate pass@k on the held-out test set (16 samples/problem). Both curricula roughly double SFT single-shot accuracy.

Model	**pass@1**	**pass@16**
SFT (initialization)	≈ 0.30	≈ 0.72
RLOO + static	0.60	0.82
RLOO + adaptive	0.64	0.82

strategies differ by at most 0.04 on every cell — within the sampling noise of the test subset. Most strikingly, the adaptive model, which never trained on a single four-operand problem, *matches and slightly exceeds* the static model on four-operand problems (pass@1 0.385 vs. 0.346; pass@16 0.692 vs. 0.654).

5.2 Qualitative Analysis

Inspecting sampled completions clarifies both the difficulty gap and a common failure mode. On easy three-operand problems, successful traces typically reach a correct expression after a short, focused search (e.g. for target 98 from $\{44, 19, 35\}$: "$44 + 19 = 63$; $63 + 35 = 98$"), and the model reliably emits a well-formed `<answer>`. Four-operand problems require a substantially larger search and the model more often exhausts its reasoning budget before finding a valid expression.

A recurring failure on hard problems, especially those with duplicated numbers, is the production of *degenerate but well-formed* expressions: the model constructs an expression that uses all numbers and parses correctly but does not equal the target, for example exploiting $(x - x) = 0$ to manufacture syntactic validity when no genuine solution is found. Because such expressions earn the 0.1 format reward rather than 0, they are locally reinforced relative to malformed output, which likely contributes to the plateau in four-operand accuracy. This pattern is consistent across both curricula and indicates that the four-operand bottleneck is one of search and verification capability rather than of output formatting.

Table 2: pass@k split by difficulty. Differences between strategies are small (≤ 0.04), within test-set noise. Adaptive matches static on hard (d=4) problems despite never training on one.

	pass@1		**pass@16**	
Difficulty	static	adaptive	static	adaptive
d=3 (easy)	0.875	0.917	1.000	0.958
d=4 (hard)	0.346	0.385	0.654	0.692

6 Discussion

Why did difficulty ordering matter so little? My central result is that the difficulty strategy had essentially no effect on final capability, and the cleanest evidence is the adaptive run: a model that trained *only* on easy problems matched a model that trained on a balanced mix, even on hard problems it never saw. The most plausible explanation is *skill transfer*. On Countdown, easy and hard problems are not distinct skills; they are the same underlying procedure — searching over combinations, performing the arithmetic, and formatting the answer — applied at different scales. A four-operand problem is a larger search over that same procedure, not a qualitatively new capability. Once the policy masters the procedure on easy problems, where reward is dense and learning is fast, the competence transfers upward, and explicit training on hard problems adds little.

This reframes what a curriculum buys on this task. The benefit of front-loading easy problems is not that it eventually teaches the hard ones — transfer already does that — but that easy problems supply denser early reward, helping the model acquire the core procedure quickly and stably. The "hard" portion of the schedule is largely redundant when transfer is strong, which is exactly why random, static, and (accidentally easy-only) adaptive all converge to similar final performance.

Brittleness of adaptive curricula. The adaptive controller failed silently: it never advanced its front despite the easy solve rate clearly exceeding the mastery threshold, and produced no error to signal the failure. In effect, a controller that looked reasonable collapsed the experiment into easy-only training. This is a concrete cautionary result: performance-gated schedules introduce a control-logic surface that can fail without any crash, and they should be validated by directly logging the schedule variable (here, the front and the realized hard-problem share) rather than assumed to behave as designed. A fixed schedule, while less principled, is robust by construction.

When would a curriculum help? The natural boundary condition is a task where hard instances require something qualitatively new that easy instances never expose — a new sub-skill, a longer horizon, or a different solution structure — so that transfer from easy to hard is weak. In that regime, training on hard problems is not redundant, and the order in which difficulty is introduced should matter more. Countdown, parameterized only by operand count, does not appear to be such a task at this model scale.

Limitations. Three limitations bound my conclusions. First, the intended matched random-sampling baseline did not complete (the no-curriculum run terminated early during reward computation on a hard-problem generation), so my cleanest contrast is static vs. (easy-only) adaptive rather than curriculum vs. random; I treat the random arm as future work. Second, the adaptive controller's failure to advance means I did not obtain a faithful adaptive curriculum, only its

degenerate easy-only limit; a corrected controller is needed to test the adaptive hypothesis properly. Third, all results use a single 0.5B model, a two-level difficulty axis, and 100 training steps; transfer and curriculum effects may differ with finer difficulty, longer training, or larger models. The reported static-vs-adaptive differences (≤ 0.04) are within test-set noise and should not be over-interpreted.

7 Conclusion

I studied whether ordering training data by difficulty improves RLOO fine-tuning of a small LLM on Countdown, comparing random, static, and adaptive difficulty schedules through a single front-based mechanism. RLOO with a curriculum roughly doubled single-shot test accuracy over the SFT initialization, but the choice of difficulty strategy had little effect on final capability: static and adaptive were statistically indistinguishable, and a model trained only on easy problems matched one trained on a balanced mix even on hard problems. I attribute this to strong easy→hard skill transfer on Countdown, and I additionally document the silent failure of a performance-gated adaptive controller as a practical cautionary result. The broader lesson is that, at this scale and on this task, getting the RL pipeline and reward signal correct mattered more than the specific difficulty ordering. Future work includes a corrected adaptive controller, a matched random baseline, and finer-grained difficulty axes and longer horizons where transfer is weaker and a curriculum is more likely to pay off.

Contributions and Changes from the Proposal

This was a single-author project. Norah Asemota was responsible for all components: implementing the SFT, IPO, and RLOO pipeline; designing and implementing the curriculum extension (the front mechanism and the static and adaptive schedules); running all experiments and evaluations; and writing this report. Relative to the proposal, the experimental plan was unchanged in intent (compare random, static, and adaptive curricula on Countdown), but two deviations arose in practice: (i) the adaptive controller failed to advance and thus realized an easy-only schedule rather than a true adaptive one, and (ii) the random-sampling baseline run did not complete and is reported as a limitation and future work. The proposal hypothesized that adaptive curricula would improve performance; the results instead support a skill-transfer explanation under which difficulty ordering has little effect, which I adopt as the report's main finding.

References

[1] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[2] A. Ahmadian, C. Cremer, M. Gallé, et al. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of ACL*, 2024.

[3] S. Parashar, S. Gui, X. Li, et al. Curriculum reinforcement learning from easy to hard tasks improves LLM reasoning. In *International Conference on Learning Representations (ICLR)*, 2026.

[4] X. Chen, J. Lu, M. Kim, et al. Self-evolving curriculum for LLM reasoning. *arXiv preprint arXiv:2505.14970*, 2025.

[5] S. Sundaram, J. Quan, A. Kwiatkowski, et al. Teaching models to teach themselves: Reasoning at the edge of learnability. *arXiv preprint arXiv:2601.18778*, 2026.