

# Extended Abstract

**Motivation** Reinforcement learning (RL) has emerged as a powerful paradigm for improving the reasoning capabilities of large language models (LLMs), enabling problem-solving behaviors on tasks such as mathematical reasoning and code generation. However, RL suffers from a cold-start problem, where sparse reward signals early in training produce weak gradient updates that slow or destabilize learning. Curriculum Learning (CL) addresses this by progressively exposing models to harder problems, ensuring more informative reward signals early in training. Building upon the static Gaussian scheduling approach of Parashar et al. (2026), this project investigates whether an adaptive approach driven by direct empirical model performance can improve upon a fixed schedule.

**Method** I propose an adaptive Gaussian scheduler that advances the curriculum based on the model’s empirical per-bucket pass rate. The training set is augmented and partitioned into three difficulty buckets by input count: easy (2-input), medium (3-input), and hard (4-input). At each training step, sampling weights are computed via a Gaussian centered at a scalar cursor  $x_t \in [0, 2]$ . The cursor advances by increment  $\delta$  when the model’s EMA pass rate on bucket  $k^* = \lfloor x_t \rfloor$  exceeds mastery threshold  $R$ , with a timeout fallback after  $M$  steps to guarantee forward progress. I compare this against a static Gaussian scheduler (Parashar et al., 2026), where the cursor advances on a fixed power-law clock, and a random-sampling RLOO baseline.

**Implementation** All models are initialized from the same Qwen2.5-0.5B SFT checkpoint (asingh15/qwen-sft-countdown-defaultproj), trained with RLOO optimization for 100 steps on an augmented Countdown dataset, and evaluated on the pass@ $k$  performance on a 50-problem test set. For the adaptive scheduler, I experiment with six configurations across pass-rate threshold  $R \in \{0.4, 0.6, 0.8\}$  and EMA smoothing factor  $\alpha \in \{0.1, 0.3\}$ . Other hyperparameters (e.g.,  $\sigma$  and  $\delta$ ) are held constant.

**Results** The static Gaussian scheduler achieves the strongest performance, with pass@1 of 52.12%, compared to 47.38% for the random baseline and 50.12% for the best adaptive variant ( $R=0.4, \alpha=0.3$ ). Among adaptive variants, faster-progressing configurations (lower  $R$ , higher  $\alpha$ ) tend to achieve higher pass@1, suggesting that greater exposure to harder problems improves overall performance. Per-bucket analysis confirms this. 4-input pass rates increase monotonically with hard-bucket exposure time, while 2- and 3-input pass rates remain stable across all variants regardless of exposure, suggesting that easier problems saturate quickly and yield diminishing returns from additional training time. Qualitatively, the static scheduler produces more structured reasoning and generalizable search strategies, while the baseline and adaptive variants share similar failure modes, including repetitive “Let me try a different approach” loops and misaligned <answer> tags that contradict the derived solution. I hypothesize that the smooth, continuous progression of the static cursor function enables more stable gradient updates, whereas the adaptive scheduler’s step-like, threshold-driven motion introduces abrupt distribution shifts that hinder deeper reasoning learning.

**Discussion** The limited training and testing scale (100 training steps and 50 test problems) act as a primary constraint on the reliability of the results, as models may not have fully converged and small accuracy differences span only one or two problems. Longer training runs with multiple random seeds and a larger evaluation set are needed to reliably distinguish true effects from noise. Nonetheless, the findings of this work, such as the relationship between hard-problem exposure and reasoning performance, are likely to generalize to other reasoning tasks with structured difficulty.

**Conclusion** Ultimately, this work demonstrates that curriculum learning meaningfully improves over random sampling for RLOO post-training on Countdown, and identifies exposure to hard problems as a key driver of overall reasoning performance. While the adaptive scheduler does not outperform the static baseline, the findings motivate future work on smoother adaptive advancement mechanisms, larger training and evaluation scale, and broader search over the hyperparameters in the presented adaptive method.

---

# Curriculum Learning with Success-Rate-Driven Adaptive Gaussian Scheduler

---

**Andrew T. Lee**

Department of Computer Science  
Stanford University  
adtlee@stanford.edu

## Abstract

Reinforcement learning (RL) fine-tuning of language models suffers from sparse reward signals early in training, hindering learning on difficult reasoning tasks. Curriculum Learning (CL) addresses this by progressively exposing models to harder problems. This project builds upon the work of Parashar et al. (2026) and proposes an adaptive Gaussian curriculum scheduler driven by a per-bucket success-rate threshold. Experiments show that both static and adaptive curriculum scheduling improve over random sampling, and that greater exposure to hard problems acts as the primary driver for improved performance within the adaptive variants. However, the adaptive scheduler does not outperform the static scheduler, suggesting that smooth, continuous curriculum progression may be more conducive to stable RL fine-tuning than step-like, threshold-driven motion.

## 1 Introduction

Reinforcement learning (RL) has emerged as a powerful post-training paradigm, enabling large language models (LLMs) to develop complex problem-solving behaviors (Guo et al., 2025). These skills are increasingly relevant to real-world applications such as mathematical reasoning, code generation, and scientific problem solving, where models must plan over long horizons and verify their own outputs. However, RL fine-tuning suffers from a fundamental cold-start problem. When a model cannot yet solve most problems it encounters, the majority of rollouts yield near-zero reward, producing weak gradient updates that slow or destabilize learning.

Curriculum Learning (CL) has shown promise in addressing this issue, and prior work has explored both static and adaptive scheduling strategies to guide difficulty progression during RL fine-tuning (Parashar et al., 2026; Chen et al., 2025). However, existing approaches either advance on a fixed clock regardless of model progress, or rely on indirect proxy signals to measure mastery, leaving open the question of whether a simpler, direct empirical performance signal can match or improve upon these approaches.

To address this, I propose an adaptive CL method that combines Gaussian scheduling with a direct success-rate threshold mechanism, where the curriculum advances when the model’s empirical per-bucket pass rate exceeds a mastery threshold  $R$ . I hypothesize that governing difficulty progression by demonstrated model competence will yield improved final performance over a fixed schedule by preventing premature exposure to harder problems before the model is ready.

## 2 Related Work

RL-based post-training methods have been shown to significantly enhance the reasoning capabilities of LLMs. Guo et al. (2025) demonstrated that pure RL fine-tuning can incentivize emergent reasoning

patterns in LLMs, such as self-reflection and verification, without needing human-labeled reasoning trajectories, surpassing supervised approaches like supervised fine-tuning (SFT) and Chain-of-Thought (CoT) prompting. However, because reward signals rely solely on output correctness, RL fine-tuning often struggles on difficult tasks where pre-trained models have low performance, producing sparse and weak learning signals that inhibit training.

CL serves as a viable solution to this cold-start problem by progressively exposing the model to problems of increasing difficulty, ensuring dense reward signal early in training before harder problems are introduced. Parashar et al. (2026) developed Easy-to-Hard (E2H) Reasoner, which partitions training data into difficulty buckets and transitions between them using fixed probabilistic Gaussian and Cosine schedulers. These schedulers produce smooth transitions between difficulty levels, mitigating catastrophic forgetting and overfitting to easy tasks. Their results outperformed traditional step-based CL schedulers. However, their schedulers advance on a fixed training-step clock, progressing through difficulty levels regardless of the model’s learning progress.

Adaptive schedulers address this limitation by dynamically modifying the curriculum in response to the model’s learning state during training. Chen et al. (2025) developed the Self-Evolving Curriculum (SEC), which formulates curriculum selection as a non-stationary Multi-Armed Bandit problem. At each fine-tuning step, SEC samples a training batch by estimating each difficulty bucket’s expected learning gain, using advantage values from the RL algorithm as a proxy signal. SEC achieved a 13% relative improvement over a standard random curriculum on Countdown. However, SEC relies on a gradient-level advantage proxy signal to measure task mastery, which does not directly measure whether the model can actually solve problems in a given bucket. It also introduces additional hyperparameters (e.g., MAB temperature and learning rate) that require careful tuning.

### 3 Method

This project builds upon the work of Parashar et al. (2026) by introducing adaptability into the Gaussian scheduler. Rather than advancing the curriculum on a fixed training-step clock, the adaptive scheduler advances based on the model’s empirical per-bucket success rate. The curriculum progresses to the next difficulty level only when the model demonstrates sufficient mastery on the current one, as measured by a pass-rate threshold.

#### 3.1 Data

To assign difficulty levels, I partition the training set into buckets by input count, following Parashar et al. (2026). Since `as_ingh15/countdown_tasks_3to4` only contains 3- and 4-input problems, the native difficulty structure—only two levels to transition between—is too coarse to support a meaningful curriculum. To address this, I augment the training set with 2-input Countdown problems. I generate these additional problems programmatically by iterating through pairings of numbers from 1 to 99. The combined training set is then partitioned into three buckets: 14,828 *easy* (2-input) problems, 240,608 *medium* (3-input) problems, and 249,706 *hard* (4-input) problems. Table 1 provides an example data entry from the augmented dataset.

prompt	ground_truth	num_operands
< im_start >system You are a helpful...	{"numbers": [61, 41, 82], "target": 20}	3

Table 1: Example row from `adlee238/cs224r-countdown_tasks_augmented`.

#### 3.2 Static Gaussian Scheduler

The two main scheduling methods implemented in this project are the Static Gaussian Scheduler and the Adaptive Gaussian Scheduler. The static scheduler refers to the one proposed by Parashar et al. (2026). In this framework, each difficulty level is treated as a position in a 1D space, and sampling weights are assigned based on the position of a cursor  $x_t$  that sweeps from easy to hard problems over the course of training. For  $K$  difficulty categories (0-indexed), each category  $k$  is assigned a mean position  $\mu_k = k$  (e.g., the easiest category  $k = 0$  is assigned  $\mu_0 = 0$ ). For training step  $t$  out of

$T$  total steps, the cursor position is defined as:

$$x_t = \left(\frac{t}{T}\right)^\beta (K - 1).$$

The sampling weight for difficulty category  $k$  at step  $t$  is then

$$S_{\text{Gaussian}}(t, k) = \exp\left(-\frac{(x_t - \mu_k)^2}{2\sigma^2}\right),$$

as visualized in Figure 1. These weights are normalized across all  $K$  categories before sampling.

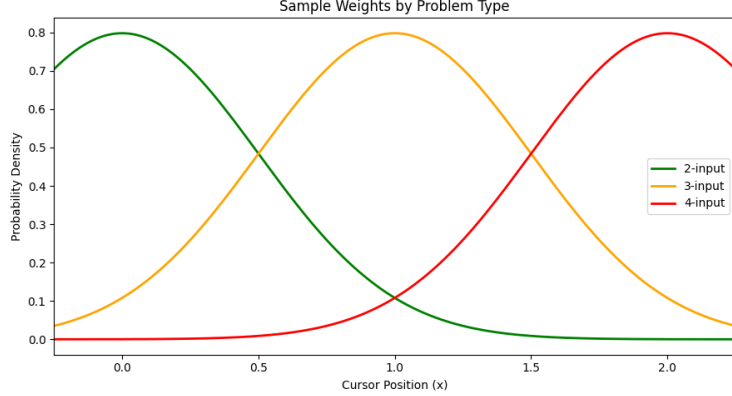


Figure 1: Sample weights distribution by cursor position.

### 3.3 Adaptive Gaussian Scheduler

This scheduler modifies the static scheduler above by making the cursor position a function of the model’s empirical per-bucket success rate. I retain the sampling weight function  $S_{\text{Gaussian}}(t, k)$  defined above, but redefine the cursor function.

Let  $x_0 = 0$ . The cursor  $x_t$  advances according to:

$$x_{t+1} = \min\left(K - 1, \begin{cases} x_t + \delta & \text{if } r_{t,k^*} \geq R \\ x_t + \delta & \text{if } t - t_{\text{last advance}} \geq M \\ x_t & \text{otherwise} \end{cases}\right)$$

where:

- $\delta$  is the fixed cursor increment per advancement.
- $r_{t,k^*}$  is the model’s estimated pass rate on bucket  $k^*$  at training step  $t$ , with  $k^* = \lfloor x_t \rfloor$  being the difficulty level that needs to be cleared.
- $R$  is the pass-rate threshold.
- $M$  is a timeout window in training steps.

The first case advances the cursor when the model has achieved sufficient mastery on the  $k^*$  bucket. The second case is a timeout fallback that advances the cursor, regardless of success rate, if no advancement has occurred in past  $M$  steps, ensuring the curriculum makes forward progress even if the model plateaus. In other cases, where the model has not yet passed the mastery threshold, the cursor remains in place.

The estimated pass rate  $r_{t,k^*}$  is measured as an exponential moving average (EMA) of per-sample scores within bucket  $k^*$ , updated at each training step according to:

$$r_{t,k^*} = (1 - \alpha) \cdot r_{t-1,k^*} + \alpha \cdot \frac{1}{|\mathcal{B}_{k^*}^{(t)}|} \sum_{i \in \mathcal{B}_{k^*}^{(t)}} \text{score}_i$$

where:

- $\mathcal{B}_{k^*}^{(t)}$  is the set of samples drawn from bucket  $k^*$  in the batch at step  $t$ .
- $\alpha \in (0, 1]$  is the EMA smoothing factor controlling how quickly the estimate responds to recent performance
- $\text{score}_i \in \{0.0, 0.1, 1.0\}$  is the output of `compute_score` for sample  $i$ .

## 4 Experimental Setup

I evaluate three models: RLOO (baseline), RLOO with Static Gaussian Scheduler, and RLOO with Adaptive Gaussian Scheduler. All three share the same base architecture: a Qwen2.5-0.5B model warm-started from the same SFT checkpoint (`asingh15/qwen-sft-countdown-defaultproj`), trained with RLOO optimization on the augmented training set for 100 steps. All other hyperparameters (e.g., learning rate, batch size, and RLOO sampling parameters) are held constant across runs. The baseline RLOO model samples training batches uniformly at random from the augmented dataset, with no curriculum scheduling.

For RLOO with Static Gaussian Scheduler, I follow Parashar et al. (2026) and set standard deviation  $\sigma = 0.5$ , which concentrates sampling weight on one difficulty bucket at a time while maintaining non-zero weight on neighboring buckets, and pacing exponent  $\beta = 0.5$ , which causes the cursor to advance quickly early in training and slow near the end, spending more time on harder problems. These values demonstrated the best results on Countdown in their work.

For RLOO with Adaptive Gaussian Scheduler, I use the same Gaussian sampling distribution with  $\sigma = 0.5$ , and set cursor increment  $\delta = 0.1$  and timeout window  $M = 10$  steps. I experiment with pass-rate threshold  $R \in \{0.4, 0.6, 0.8\}$  and EMA smoothing factor  $\alpha \in \{0.1, 0.3\}$ , yielding 6 configurations in total. A lower  $R$  promotes the cursor more aggressively, while a higher  $R$  makes promotion more difficult. For  $\alpha$ , while a higher value makes the pass-rate estimate more reactive to recent rollouts.

The primary evaluation metric is  $\text{pass}@k$  on the 50-problem test set, measuring functional correctness of generated solutions. I additionally report per-bucket success rate  $r_{t,k^*}$  for the adaptive scheduler over training steps to assess curriculum behavior and training effectiveness across runs.

## 5 Results

The  $\text{pass}@k$  results for the three models are shown in Figure 2. For adaptive, I show only the best performing configuration. A full graph with all variants’ results is available in Appendix A. The static scheduler is the clear winner of the three approaches. The best-performing adaptive scheduler beats the baseline slightly at  $\text{pass}@1$ , but eventually converges to similar performance. It does not surpass the static scheduler’s performance.

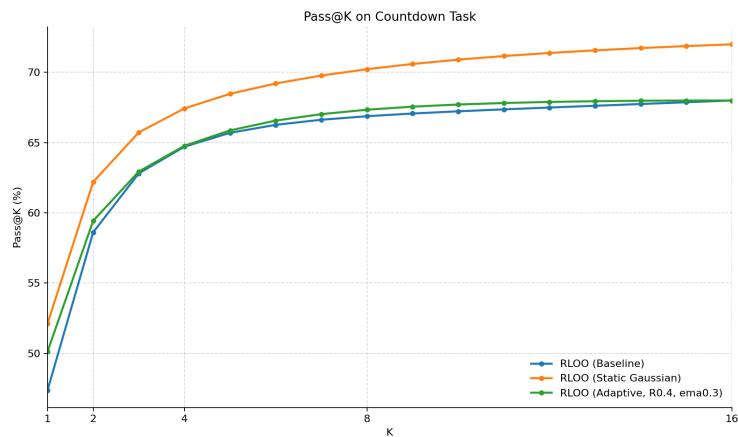


Figure 2:  $\text{Pass}@k$  on test set for baseline, static scheduler, and best-performing adaptive scheduler.

## 5.1 Quantitative Evaluation

Table 2 provides more detailed pass@ $k$  metrics for all methods and variants. As shown, the static scheduler delivers a consistent gain of around 4% over the baseline at any  $k$ , reinforcing the effectiveness of curriculum learning than random sampling. However, for the best-performing adaptive scheduler ( $R = 0.4, \alpha = 0.3$ ), it beats the baseline only slightly at Pass@1 (+2.7%) but ties with it at Pass@16 (68.0% both). The variant ( $R = 0.8, \alpha = 0.3$ ) is also a close contender, with a higher Pass@16 rate. Regardless, it is clear that the adaptive scheduler is unable to outperform the static scheduler.

Method	pass@1 (%)	pass@4 (%)	pass@16 (%)
RLOO (Baseline)	47.38	64.71	68.0
RLOO (Static Gaussian)	<b>52.12</b>	<b>67.44</b>	<b>72.0</b>
RLOO (Adaptive, $R=0.4, \alpha=0.1$ )	48.88	65.47	70.0
RLOO (Adaptive, $R=0.4, \alpha=0.3$ )	50.12	64.77	68.0
RLOO (Adaptive, $R=0.6, \alpha=0.1$ )	46.00	65.68	70.0
RLOO (Adaptive, $R=0.6, \alpha=0.3$ )	47.00	64.64	70.0
RLOO (Adaptive, $R=0.8, \alpha=0.1$ )	45.88	66.03	72.0
RLOO (Adaptive, $R=0.8, \alpha=0.3$ )	49.12	65.76	72.0

Table 2: Pass@ $k$  on test set for all approaches.

Focusing on the adaptive variants, I analyze the effects of pass-rate threshold  $R$  and EMA smoothing factor  $\alpha$  on curriculum behavior and pass@ $k$  performance.

- Effect of  $R$ .** A lower threshold causes the cursor to advance more aggressively into harder buckets, while a higher threshold enforces longer exposure to easier problems before progression, as reflected in the cursor trajectories in Figure 3. In terms of performance, the  $\alpha = 0.1$  variants highlight a visible pattern: pass@1 increases monotonically as  $R$  decreases, from 45.88% at  $R = 0.8$  to 46.00% at  $R = 0.6$  to 48.88% at  $R = 0.4$ , suggesting a potential correlation of faster curriculum progression with better final accuracy. However, this pattern does not hold cleanly for  $\alpha = 0.3$ , where  $R = 0.6$  (47.00%) underperforms  $R = 0.8$  (49.12%) despite advancing the cursor more aggressively. I attribute this discrepancy to noise given the limited number of training steps and small test set size, rather than a systematic reversal of the trend.
- Effect of  $\alpha$ .** A higher  $\alpha$  makes the EMA pass-rate estimate more reactive to recent rollout rewards, causing mastery thresholds to be triggered more quickly and producing faster cursor progression, as visible in Figure 3. In terms of performance, higher  $\alpha = 0.3$  variants consistently outperform their  $\alpha = 0.1$  counterparts at pass@1 across all values of  $R$  (50.12% vs. 48.88% at  $R = 0.4$ ; 47.00% vs. 46.00% at  $R = 0.6$ ; 49.12% vs. 45.88% at  $R = 0.8$ ).

Taken together, a lower  $R$  and a higher  $\alpha$  both have the effect of accelerating cursor progression, increasing the model’s exposure to harder 4-input problems during training. The pass@1 results from Figure 2 then show a general trend where faster-progressing variants tend to achieve higher pass@1: ( $R=0.4, \alpha=0.3$ ) achieving 50.12%, followed by ( $R=0.4, \alpha=0.1$ ) at 48.88%, followed by ( $R=0.6, \alpha=0.1$ ) at 47.00%, and slower variants generally trailing behind. This suggests that prolonged exposure to harder problems is the primary driver of pass@1 performance among the adaptive variants, though the relationship is noisy given the limited number of training steps and small test set. Notably, this effect diminishes at higher  $k$ : pass@16 is tied across several variants, regardless of progression speed, suggesting that when the model is given multiple attempts, curriculum speed and hard-problem exposure matters.

However, this relationship does not extend to a comparison with the static scheduler. Despite several adaptive variants having greater exposure to 4-input problems than the static scheduler, none were able to match its pass@1 performance of 52.12%. Additionally, the adaptive variants whose cursor trajectories most closely resemble the static scheduler, ( $R=0.6, \alpha=0.1$ ) and ( $R=0.8, \alpha=0.3$ ), still underperform substantially, suggesting that the gap is not merely due to exposure time. I hypothesize that the continuously smooth progression of the static scheduler may be more conducive to stable

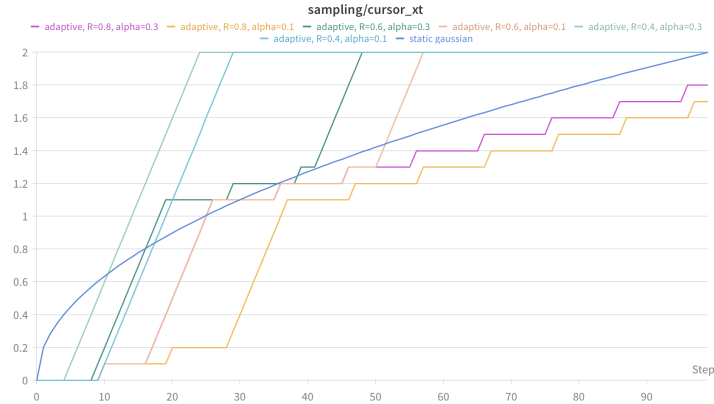


Figure 3: Cursor  $x_t$  position across training steps for static scheduler and adaptive variants.

gradient updates than the step-like advancement of the adaptive cursor. In other words, the *manner* of curriculum progression—smooth vs. abrupt—may matter as much as the *amount* of exposure to hard problems.

Investigating the effect of curriculum progression on per-bucket performance, Figure 4 shows the pass rate estimates for 2-input and 3-input problems across all adaptive variants. Despite large differences in exposure to these easier buckets across variants, pass rates remain remarkably consistent: 2-input pass rates cluster between 0.8–0.9 and 3-input pass rates between 0.65–0.75 across all variants, with the exception of some noise from the  $R=0.6$ ,  $\alpha=0.3$  variant. This suggests that once the model achieves a baseline level of competence on easier problems, additional exposure to them yields diminishing returns: performance on 2-input and 3-input problems appears to saturate quickly regardless of how much curriculum time is spent there.

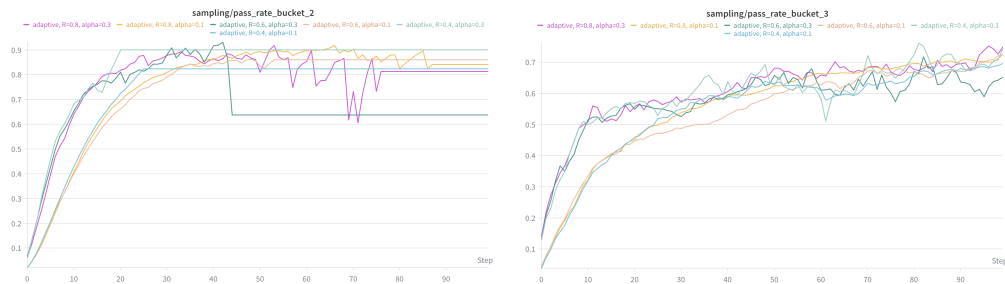


Figure 4: Per-bucket pass rate estimates for 2-input (left) and 3-input (right) problems across all adaptive variants.

In contrast, Figure 5 reveals a clear ordering for 4-input problems: variants that spend more time in the hard bucket achieve consistently higher pass rates, with  $(R=0.4, \alpha=0.3)$  performing best and  $(R=0.8, \alpha=0.1)$  performing worst. Crucially, this improvement does not appear to transfer from easier buckets; increased exposure to 2-input and 3-input problems does not improve 4-input performance. Together, these findings suggest that direct exposure to hard problems is necessary and sufficient for improving hard-problem performance, while easier problems serve primarily as a warm-up to stabilize early training.

## 5.2 Qualitative Analysis

I examine individual model outputs on test problems to identify behavioral differences across the three approaches. Review Appendix B to see sample rollouts from each of the three methods.

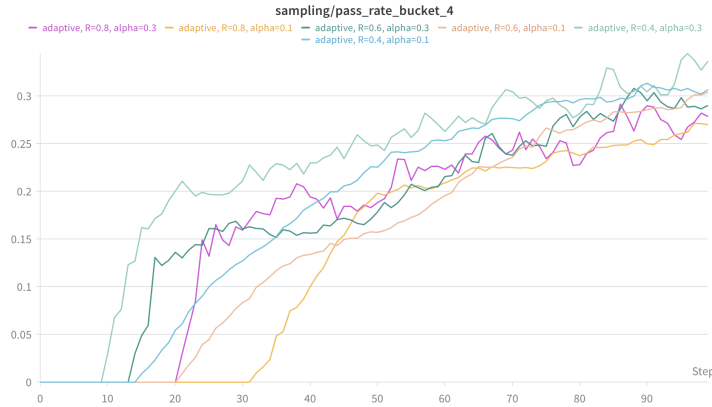


Figure 5: Per-bucket pass rate estimates for 4-input problems across all adaptive variants.

- Baseline:** The baseline RLOO model exhibits a pattern of generating plausible-looking reasoning steps without verifying their validity. For example, on the problem  $[49, 41, 73] \rightarrow 17$ , the model correctly identifies  $73 - 41 = 32$  and  $49 - 32 = 17$  in its chain-of-thought, but then outputs  $49 - (49 - (73 - 41))$ , using 49 twice in violation of the problem constraints and ignoring the correct answer it deduced earlier. This suggests the baseline learns to approximate the *structure* of valid solutions without truly internalizing its reasoning and the constraints of the task.
- Static scheduler:** The static scheduler produces noticeably more organized reasoning. For problems  $[74, 5, 20, 88] \rightarrow 50$  and  $[21, 14, 56] \rightarrow 84$ , the model begins by listing factors of the target before exploring arithmetic operations, a sensible heuristic for Countdown-style tasks. In some other cases, the model adopts a systematic search strategy, first exhausting subtraction combinations across input pairs before moving to addition, progressing methodically rather than sampling operations at random. This structured behavior likely emerged from training on a smoother difficulty distribution. By being gradually exposed to problems of increasing complexity, the model was incentivized to develop generalizable search heuristics rather than memorizing shallow patterns that happen to work on easy problems.
- Adaptive scheduler:** The adaptive scheduler shares similar failure modes as with the baseline. The adaptive model frequently enters repetitive “Let me try a different approach” loops without meaningfully changing or explaining its strategy, suggesting it has not internalized a systematic search procedure beyond shallow pattern-matching. The adaptive model also correctly identifies the solution path inside the `<think>` block at times but then produce malformed or duplicated `<answer>` tags in the final output. These behaviors suggest that the step-like, threshold-driven curriculum advancement of the adaptive scheduler may impede the development of generalizable reasoning and output formatting strategies. Unlike the static scheduler’s smooth progression, abrupt shifts in training batch composition may prevent the model from consolidating structured reasoning habits, keeping it anchored to superficial solution patterns rather than developing the deeper search heuristics observed in the static scheduler.

Across the 50-problem test set, the three models show different pass rates in 34 of the problems. The static scheduler’s advantage concentrates on medium-hard problems. For example, on problem 04 the static scheduler achieves 16/16 correct rollouts compared to the baseline’s 9/16, and on problem 15 it achieves 9/16 compared to the adaptive scheduler’s 2/16. However, on the two of the harder problems in the test set (problem 08 and problem 18), both requiring a non-obvious division step (e.g.,  $56 \div 14 = 4$ , then  $21 \times 4 = 84$ ), all three models fail equally, achieving at most 2/16 correct rollouts. This suggests that curriculum pacing is effective in the middle of the difficulty spectrum, helping the model consolidate competence on moderately hard problems. However, it does not help learn new, more complex solution strategies needed to solve the hardest of problems.

## 6 Discussion

The primary limitation of this project is the scale of experimentation. With only 100 training steps, not all models have fully converged, which undermines the reliability of performance comparisons across approaches. In particular, the slower-progressing adaptive variants may have benefited from additional training. With more steps, they may receive more useful exposure with harder problems, potentially narrowing the gap with the static scheduler and revealing new trends across different  $R$  and  $\alpha$  configurations. More broadly, performance differences between approaches may be understated at the current scale. Longer training could amplify true effects, making it easier to distinguish meaningful patterns from noise. Another limitation is the reliability of the  $\text{pass}@k$  comparisons due to the relatively small 50-problem test set. Small differences in  $\text{pass}@1$  (e.g., 47.00% vs. 48.88%) span only one or two problems and fall well within the range of noise, making statistically confident conclusions difficult to draw.

Despite the performance gap between the adaptive and static schedulers, the findings of this work carry broader implications for curriculum-based RL of language models. The trend across adaptive variants, where faster progression and longer exposure to harder problems improves  $\text{pass}@1$ , suggests that the amount of time a model spends on difficult problems during RL training is an important lever for improving final reasoning performance. Notably, long exposure to harder problems does not appear to hurt performance on easier ones, so practitioners could potentially bias their training distributions toward harder problems without sacrificing breadth. This insight is likely generalizable beyond Countdown to other verifier-based reasoning tasks, where training distributions are similarly skewed toward problems of varying difficulty and the cold-start problem is equally acute.

## 7 Conclusion

Ultimately, this work investigates whether an adaptive, success-rate-driven curriculum can improve upon static Gaussian scheduling for RLOO fine-tuning of language models on the Countdown reasoning task. Both static and adaptive curriculum scheduling provide meaningful improvements over random sampling, confirming that intentionally progressing from easy to hard problems during RL training is an effective strategy for improving model learning.

While the adaptive scheduler did not outperform the static baseline in this setting, the performance gap is in part attributable to the limited training scale. Future work should explore this approach at a greater scale, such as training for 200+ steps with multiple random seeds, ablating timeout threshold  $M$  and cursor step size  $\delta$ , and evaluating on a larger test set. Additionally, future adaptive curriculum methods should explore smoother, continuous advancement mechanisms, such as reformulating the discrete cursor update rule as a differentiable function of training step and success rate, or replacing the binary mastery signal with a richer performance signal.

## 8 Team Contributions

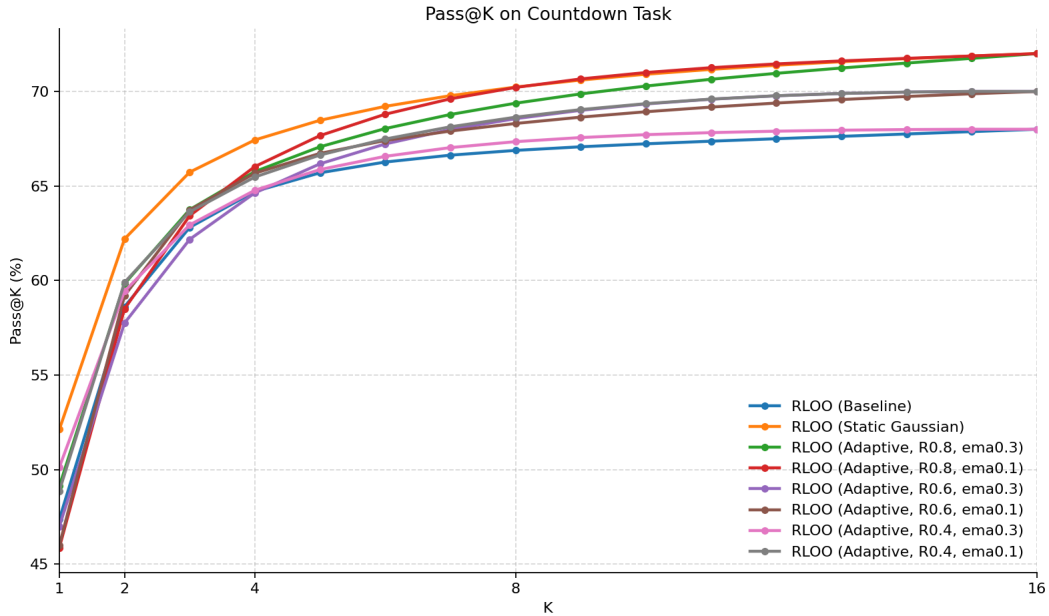
I completed this project individually. For the extension, I used Claude to assist with implementation, including synthetic 2-input problem generation, the custom dataset and scheduler classes, and modifications to the RLOO training loop. Claude also assisted with the qualitative analysis of model rollouts. All experimental design, result interpretation, and core ideas are my own.

## References

- Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamaloo. 2025. Self-Evolving Curriculum for LLM Reasoning. arXiv:2505.14970 [cs.AI] <https://arxiv.org/abs/2505.14970>
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638. doi:10.1038/s41586-025-09422-z
- Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2026. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. arXiv:2506.06632 [cs.LG] <https://arxiv.org/abs/2506.06632>

## A Pass@k Graph for All Methods

This figure details the pass@k trends for the baseline, the static scheduler, and all adaptive variants.



## B Sample Rollouts

The following listing is a sample rollout from the Basic RLOO (Baseline) model.

```
[PROMPT]
Target: 17
Allowed numbers: [49, 41, 73]

[BASIC RLOO ROLLOUT]
<think>
Let me try to find a way to get to 17.
First, let me try to get close to 17 with some operations:
49 - 41 = 8, still too far
73 - 49 = 24, too far
73 - 41 = 32, too far

Let me try division since we have large numbers:
73 / 49 = 1.59, not helpful
49 / 73 = 0.65, not helpful

Let me try a different approach:
41 + 49 = 90
73 - 49 = 24
90 - 24 = 66, too far

Let me try:
73 - 49 = 24
41 + 24 = 65
65 - 41 = 24, not what we want

One more attempt:
49 - 41 = 8
73 - 8 = 65
```

```

65 - 4 = 61, not what we want

Let me try:
73 - 41 = 32
49 - 32 = 17!

I found it! Let me verify:
73 - 41 = 32
49 - 32 = 17
</think>
<answer> 49 - (49 - (73 - 41)) </answer>

```

Listing 1: Sample Test-Set Rollout (Countdown) using Basic RLOO

The following listing is a sample rollout from the RLOO model trained with the static Gaussian scheduler.

```

[PROMPT]
Target: 50
Allowed numbers: [74, 5, 20, 88]

[STATIC SCHEDULER RLOO ROLLOUT]
<think>
Let me analyze this step by step:

1. First, let's look at factors of 50: 1, 2, 5, 10, 25, 50
None of our numbers are factors of 50, so we'll need to create 50 through addition/
  subtraction.

2. Let's try working with the larger numbers:
88 - 74 = 14 (too small)
88 - 20 = 68 (too large)

3. Let's try working backwards:
50 could be reached by adding a small number to 45
45 could be reached by subtracting numbers...

4. Looking at the numbers:
74 is close to 50 (difference of 24)
20 could be useful to get to 50

5. Let's try:
74 + 20 = 94 (too large)
74 - 20 = 54 (getting closer)
88 - 20 = 68 (too large)

6. What if we try:
88 - 74 = 14
14 + 5 = 19
19 + 20 = 39 (getting closer)

7. Let's try:
74 + 20 = 94
94 - 88 = 6
6 * 5 = 30 (not helpful)

8. What about:
88 - 74 = 14
14 + 20 = 34
34 + 5 = 40 (not helpful)

9. Let's try:
74 - 20 = 54
54 + 5 = 60
60 - 88 (not helpful)

```

```

10. One more attempt:
88 - 74 = 14
14 + 5 = 19
19 + 20 = 39 (not helpful)

11. Let's try:
74 + 5 = 79
79 - 20 = 59
59 - 88 (not helpful)

12. Final attempt:
88 - 74 = 14
14 + 20 = 34
34 + 5 = 49 (close but not quite)

13. Let's try:
74 + 20 = 94
94 - 88 = 6
6 + 5 = 11 (not helpful)

14. One last try:
88 - 74 = 14
14 * 5 = 70
70 - 20 = 50 (This works!)
</think>

<answer>(88 - 74) * 5 - 20</answer>

```

Listing 2: Sample Test-Set Rollout (Countdown) using RLOO with static scheduler

The following listing is a sample rollout from RLOO model trained with the adaptive Gaussian scheduler ( $R = 0.4, \alpha = 0.3$ ).

```

[PROMPT]
Target: 17
Allowed numbers: [49, 41, 73]

[ADAPTIVE SCHEDULER RLOO ROLLOUT]
<think>
Let me try to find a path to 17:
1. First, let's try to get closer to 17 by reducing the larger numbers:
49 - 41 = 8
This gets us closer to 17, but still too high.

Let's try a different approach:
73 - 41 = 32
49 - 32 = 17!

Let me verify:
73 - 41 = 32
49 - 32 = 17
This works! Let's write it in the correct format.
</think>
<answer>49 - (49 - 41)</answer>
<answer>73 - (73 - 49)</answer>
<answer>73 - (49 - 41)</answer>

```

Listing 3: Sample Test-Set Rollout (Countdown) using RLOO with adaptive scheduler