Extended Abstract

Motivation Reinforcement learning from verifier feedback improves LLM reasoning but suffers from sparse rewards on complex tasks, making exploration inefficient. Curriculum learning addresses this by structuring training from easy to hard, yet it remains unclear whether benefits come from task simplicity, structural relevance to the target, or the scheduling of easy and hard examples. We investigate these questions on the Countdown arithmetic task to clarify which properties of teacher-generated curricula matter most for RL-based reasoning.

Method We fine-tune `Qwen2.5-0.5B` from an SFT checkpoint on the Countdown task using RLOO. We construct three families of easy curricula: Gemini-2.5-Pro decomposed subproblems (correct only), SFT-teacher decomposed subproblems (including incorrect), and random 2-number and 3-number problems. Across five experiments we vary the curriculum source (Gemini vs. teacher, test- vs. train-derived), ordered easy-then-hard vs. shuffled training, curriculum relevance (teacher subproblems vs. random scaffolds), fixed schedules (alternating blocks, revisiting, interleaving, and hard-heavy cycles), and probabilistic schedules (Gaussian and cosine). All conditions are evaluated on a 500-problem held-out set using pass@k over $n = 16$ samples and compared against a hard-only RLOO baseline with paired permutation tests.

Implementation All methods fine-tune `Qwen2.5-0.5B` from a shared supervised checkpoint, against which we compare three alignment baselines (SFT, IPO, and RLOO), with RLOO serving as the baseline for our curriculum extension. RLOO samples G=8 responses per prompt with vLLM, computes leave-one-out advantages, and updates with a token-level policy gradient, training on "hard" 3-/4-number problems drawn from the same distribution as evaluation but disjoint from it. We evaluate with the pass@k estimator over n=16 samples per prompt (temperature 0.6, top-p 0.95, top-k 20, max 1024 tokens) for $k \in 1, 2, 4, 8, 16$ on a 500-problem set drawn from the training split (excluded from training) and on the original 50-problem test split.

Results Across five curriculum experiments, most conditions did not consistently beat the RLOO baseline at pass@1, but curriculum design still mattered. The best single-step gains came from task relevant subproblem curricula: training on teacher-generated subproblems derived from the actual hard problems improved pass@1 by up to 5.6 percentage points over hard only training, while unrelated easy problems provided smaller gains. Coverage across multiple attempts (pass@16) was most improved by smoother difficulty transitions. For instance, the gradual 2-number $\rightarrow$ 3-number $\rightarrow$ hard schedule and an expanded easy-hard-easy-hard revisit schedule each raised pass@16 by roughly 4 percentage points over RLOO. Probabilistic schedules (cosine and Gaussian decay) did not improve pass@1 but showed modest high-k coverage gains. Qualitative analysis revealed that many apparent failures were answer-extraction artifacts rather than reasoning failures, suggesting that a simple post-processing fix could recover additional correct solutions.

Discussion Subproblems structurally relevant to the target task outperformed unrelated easy problems, and smoother difficulty transitions improved coverage across multiple attempts. Concentrated warm-up phases outperformed continuous probabilistic mixing, and the augmented teacher dataset benefited most from a dedicated easy phase before hard training but showed little gain when interleaved stochastically. Dataset size and teacher reliability were the primary bottlenecks: small, noisy curriculum datasets provided insufficient signal to shift the learned policy.

Conclusion Curriculum learning did not solve Countdown reasoning, but revealed that the most useful easy examples are those that rehearse intermediate computations needed for the target task, and not simply the easiest problems available. Future work should focus on generating larger and cleaner curriculum datasets using symbolic solvers or verifier-guided pipelines, improving decoding protocols to reward the first valid answer rather than the last, and exploring adaptive schedules that introduce harder problems only after measurable improvement on simpler ones.

Can Teacher-Generated Curriculum Strategies Improve LLM Reasoning?

Amrita Malhotra
Department of Computer Science
Stanford University
amritam4@stanford.edu

Giselle Rivera
Department of Computer Science
Stanford University
giseller@stanford.edu

Zofia Dudek
Department of Biomedical Data Science
Stanford University
zodudek@stanford.edu

Abstract

Reinforcement learning from verifier feedback improves LLM reasoning, but sparse rewards make training on complex tasks difficult. Curriculum learning offers a solution by structuring training from easy to hard, yet it remains unclear whether benefits come from task simplicity, structural relevance to the target problem, or scheduling. We investigate these questions on the Countdown arithmetic task, training a 0.5B-parameter model with RLOO using teacher-generated curricula across five controlled experiments. We compare hard-only RLOO against curricula derived from Gemini and SFT teacher decompositions, random arithmetic scaffolds, and fixed and probabilistic scheduling strategies. Curriculum training did not consistently outperform the RLOO baseline, but curriculum design mattered: structurally relevant subproblems improved single-sample accuracy over unrelated easy problems, smoother difficulty transitions improved coverage across multiple attempts, and revisiting easy examples after hard training produced the largest pass@16 gains. Qualitative analysis reveals that many apparent failures reflect answer-extraction artifacts rather than reasoning failures. Our findings suggest that useful curricula must rehearse the intermediate computations needed for the target task rather than simply provide easier examples, and that dataset quality and scale remain major bottlenecks for curriculum-based RL in small reasoning models.

1 Introduction

Large language models are increasingly used in settings that require multi-step reasoning, such as mathematical problem solving, code generation, and decision support across many fields. Reinforcement learning (RL) has become a major approach for improving these reasoning abilities, however, standard RL often struggles on complex tasks due to sparse reward signals, where feedback is only provided upon reaching the correct final answer Parashar et al. (2025). This makes exploration inefficient and often prevents models from discovering effective intermediate strategies.

Curriculum learning addresses this limitation by structuring training data from easy to hard, enabling models to first master simpler subproblems before tackling more complex ones Bengio et al. (2009). In reasoning tasks, this approach is especially appealing because complex problems often contain easier intermediate computations. However, it remains unclear what kind of "easy" problems actually help. A curriculum might help because the problems are simpler, or because they are structurally related to the final task.

Stanford CS224R 2026 Final Report

To investigate this question, we train a student LLM with RL using different curricula generated by a teacher model on the Countdown task, a mathematical reasoning task in which the model is given a set of numbers and a target value and must construct an arithmetic expression that evaluates to the target. Countdown provides a useful testbed because it requires multi-step arithmetic reasoning, has clean and verifiable correctness criteria, and can be naturally decomposed into smaller subproblems.

We ask whether teacher-generated curricula improve RL training for LLM reasoning on Countdown, and whether any improvements come primarily from task simplicity, structural relevance to the target problem, or scheduling of easy and hard examples. Specifically, we compare hard-only RL training against several curriculum variants, including random easy problems, teacher-derived subproblems, and different scheduling strategies. We hypothesize that: (1) Curriculum training will improve model accuracy, measured by pass@k, compared to hard-only RL, and (2) Structurally relevant subproblems will be more beneficial to learning than easy but unrelated problems.

Together, these experiments aim to clarify which properties of teacher-generated curricula matter most for RL-based reasoning on the Countdown task, with broader implications for training LLMs on compositional reasoning problems.

2 Related Work

Prior work shows RL can improve LLM reasoning, as demonstrated by work such as DeepSeek R1 Guo et al. (2025), but sparse final-answer rewards remain a challenge. Parashar et al. (2025). Curriculum learning offers one approach to this challenge by shaping the order in which examples are presented during training. Foundational work by Benigo et al. Bengio et al. (2009) and a survey by Narvekar et al. Narvekar et al. (2020) show that the ordering of training examples can significantly impact learning and generalization. While effective, traditional curriculum learning relies heavily on manually designed datasets and predefined difficulty levels. In many situations, constructing such curricula is expensive, requires domain expertise, and may fail when the optimal learning trajectory is unknown Parashar et al. (2025).

To address these limitations, recent work such as SOAR Sundaram et al. (2026) proposes leveraging a model's latent knowledge to automatically generate curricula. In SOAR, a teacher model generates synthetic training problems for a student, and is optimized based on the student's improvement. This approach removes the need for manually designed curricula, but introduces additional training complexity and does not clarify whether improvements stem from easier tasks or from tasks that are more relevant to the final objective.

More recently, the E2H Reasoner (Parashar et al. (2025)) demonstrates that probabilistic scheduling of tasks from easy to hard improves reasoning and generalization compared to strictly sequential curricula. In particular, Gaussian-based scheduling mitigates issues such as overfitting to trivial tasks or forgetting them, which can arise when models are trained on only easy tasks before progressing to harder ones. However, this line of work assumes a predefined task pool and does not incorporate a dynamic teacher capable of adapting the curriculum based on the student's progress.

Our work builds on these directions by introducing a controlled teacher–student curriculum framework for the Countdown task. Unlike SOAR, which learns a teacher policy via meta-RL, we use a fixed heuristic teacher to generate curricula, allowing us to isolate the effect of curriculum structure without the added complexity of bilevel optimization. We explicitly compare two forms of teacher-generated curricula: (1) globally easy but unrelated problems, and (2) structurally relevant intermediate subproblems derived from the target task. This controlled comparison allows us to isolate whether curriculum benefits are driven primarily by task simplicity or by causal alignment with the final problem, which is an aspect that remains unexplored in prior work. By grounding this analysis in the Countdown task, which provides a clean and verifiable reasoning environment, our work offers a more principled understanding of how curriculum design impacts learning in RL-based reasoning systems.

3 Experimental Setup

3.1 Task and Dataset

We evaluate on the Countdown arithmetic task: given a set of N numbers, the model must produce an arithmetic expression using each number at most once that evaluates to a target value. We use the dataset `asingh15/countdown_tasks_3to4`, which contains problems with 3–4 numbers. The rule-based verifier assigns a reward of 1.0 for a syntactically valid and numerically correct solution, 0.1 for a valid but incorrect expression, and 0.0 otherwise. This sparse, structured reward signal makes Countdown a suitable benchmark for studying alignment and curriculum strategies: problems are objectively verifiable, difficulty is tunable, and early training is naturally sparse-reward.

All models are evaluated on a 500-problem held-out evaluation set drawn from the training split, with these examples excluded from curriculum training. This larger evaluation set provides more reliable pass@k estimates across conditions and enables direct comparison between the baseline and curriculum-trained models. We also evaluated models on the original 50-problem held-out test split used for the SFT, IPO, and RLOO checkpoints to maintain continuity with the default benchmark. Across the baseline models, the two evaluation sets showed the same qualitative trends, although pass@k values on the 500-problem set were generally lower by about 5–13 percentage points, suggesting that the larger set is a more challenging and stable estimate of performance.

3.2 Base Model

All methods fine-tune `Qwen2.5-0.5B` initialized from the supervised fine-tuning checkpoint `asingh15/qwen-sft-countdown-defaultproj`, which was trained on correct Countdown solution demonstrations. Using a shared SFT initialization ensures that differences between conditions reflect alignment and curriculum choices rather than initialization.

3.3 Baselines

We evaluate three alignment methods as baselines:

SFT serves as the primary reference. It requires no reward signal and reflects how well the model generalizes from demonstration data alone.

IPO Azar et al. (2023) fine-tunes from the SFT checkpoint using pairwise preference data (`asingh15/countdown_tasks_3to4-dpo`). We use learning rate 5×10^{-6}, $\beta = 0.1$, batch size 64, gradient accumulation over 16 steps, weight decay 0.01, warmup ratio 0.05, and one epoch. We include IPO to establish whether offline preference optimization improves over SFT before moving to online methods.

RLOO Kool et al. (2019) is the primary online RL baseline and the foundation for the curriculum extension. It samples $G = 8$ responses per prompt using vLLM, computes leave-one-out advantages, and updates the policy with a token-level policy gradient loss. We use learning rate 10^{-5}, AdamW with weight decay 10^{-4}, batch size 128, gradient accumulation over 128 steps, gradient clipping 1.0, KL divergence coefficient 10^{-3}, entropy regularization coefficient 10^{-3}, sampling temperature 1.0, and 120–250 training steps. RLOO is the strongest non-curriculum baseline and defines the upper bound we aim to match or exceed with curriculum training.

3.4 Evaluation Metric

We report **pass@k** using an unbiased estimator:

$$\text{pass@}k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

where $n = 16$ responses are sampled per prompt and c is the number of correct responses. We report pass@k for $k \in \{1, 2, 4, 8, 16\}$. Pass@1 is the primary metric as it measures single-sample correctness, which best reflects deployment performance. Pass@16 measures the ceiling of solvable problems. We use evaluation sampling parameters: temperature 0.6, top-p 0.95, top-k 20, max tokens 1024.

Every model is evaluated on the same problems, so we compare conditions within each experiment using paired tests: a two-sided sign-flip permutation test on the per-problem difference in pass@k, along with a 95% confidence interval on the mean difference from a paired bootstrap. As each experiment tests several conditions against a baseline, we apply a multiple-comparison correction (Holm or Bonferroni) and treat an effect as significant at corrected $p < 0.05$.

4 Method

We used a number of datasets and experimental conditions to assess whether structured curriculum learning, using unrelated or task-derived subproblems, as well as a variety of training schedules, can improve performance on the Countdown task as compared to the RLOO baseline.

Here is an example of a Countdown problem and a subproblem: Given the numbers $\{54, 37, 16, 2\}$ and target 25, a valid solution is $(54 - 37) + (16/2) = 25$. Decomposing this produces simpler subproblems, e.g. $16/2 = 8$ (numbers $\{16, 2\}$, target 8) or $54 - 37 = 17$ (numbers $\{54, 37\}$, target 17).

4.1 Datasets

Three datasets, derived from the provided RLOO dataset (`asingh15/countdown_tasks_3to4`), were used for training and testing:

- `asingh15/countdown_tasks_3to4`: The original 50 test problems used for RLOO evaluation. This testing set consisted of 3- and 4-number "hard" problems.
- `test_500`: 500 held-out test problems. This consists of the original 50 test problems from (`asingh15/countdown_tasks_3to4`) plus 450 more problems sampled from the training pool. This larger set was created in order to have more testing problems to generate subproblems from. This testing set also consisted of 3- and 4-number "hard" problems.
- `train_hard_25k`: $\sim$25000 hard 3-/4-number training problems sampled from `asingh15/countdown_tasks_3to4` and excluding the 500 evaluation problems. This set of training problems was used for all training runs to avoid data leakage between the training dataset and the larger testing set.

Additionally, we created a variety of datasets for training using AI generation and random number generation:

- **Gemini-derived easy subproblems (2-number).** Gemini-2.5-pro solved each problem from `countdown_test_500`, then decomposed its solution into binary subcomputations. Subproblems were only kept if Gemini solved the parent problem correctly. Gemini was chosen to act as a more proficient "expert" generator of subproblems.
  - `test_easy_gemini`: subproblems decomposed from Gemini's solutions to the 500 test problems ($\sim$958 entries).
  - `train_easy_gemini`: subproblems from a sample of $\sim$2,500 train problems ($\sim$596 entries). This dataset was created as a control to compare the effectiveness of testing dataset vs. training dataset subproblems.
- **Teacher-derived easy subproblems (2-number).** A teacher model initialized from the provided SFT checkpoint solved each problem from `countdown_test_500`. Its solutions were decomposed by a rule-based parser. All attempts were kept, even if subproblems were generated incorrectly.
  - `test_easy_teacher`: subproblems from the teacher's solutions to the 500 test problems (1,164 entries).
  - `test_easy_teacher_aug`: `countdown_test_easy_teacher` augmented with n=4 teacher samples per parent problem 3,707 entries. This dataset was generated after initial experimentation with `test_easy_teacher` with the hopes of providing a more robust set of training examples, as the original dataset was quite small.
  - `train_easy_teacher`: subproblems from the teacher's solutions to train problems.

- **Random subproblems (2-number and 3-number)**. Each random problem was built by drawing operands uniformly from [1, 99] and picking a target at random from [10, 100] that is provably reachable by combining all operands with the four standard operations of arithmetic.
  - `rung_2num_random`: random 2-number problems, enumerated programmatically (no AI), 5,000 entries.
  - `rung_3num_random`: random 3-number problems, enumerated programmatically (no AI), 5,000 entries.

4.2 Experimental Conditions

We conducted multiple experiments to assess the potentially varying effects of training datasets and schedules on Countdown performance.

- **Experiment 1: Dataset Source.** Do Gemini-generated vs. SFT Teacher-generated training datasets have different effects? Does training on testing subproblems vs. training subproblems have different effects?
  - **Condition 1:** `base_hardonly`. 120 steps of `train_hard_25k` only.
  - **Condition 2:** `test_gemini`. 36 steps of `test_easy_gemini` then 84 steps of `train_hard_25k`.
  - **Condition 3:** `test_teacher`. 36 steps of `test_easy_teacher` then 84 steps of `train_hard_25k`.
  - **Condition 4:** `test_teacher_aug`. 36 steps of `test_easy_teacher_aug` then 84 steps of `train_hard_25k`.
  - **Condition 5:** `train_gemini`. 36 steps of `train_easy_gemini` then 84 steps of `train_hard_25k`.
  - **Condition 6:** `train_teacher`. 36 steps of `train_easy_teacher` then 84 steps of `train_hard_25k`.
- **Experiment 2: Shuffled Training.** Does shuffling the easy training subproblems and hard training problems have any effects as compared to providing all easy subproblems before all hard problems?
  - **Condition 1:** `test_teacher_ordered`. 36 steps of `test_easy_teacher` then 84 steps of `train_hard_25k`.
  - **Condition 2:** `train_teacher_ordered`. 36 steps of `train_easy_teacher` then 84 steps of `train_hard_25k`.
  - **Condition 3:** `test_teacher_shuffled`. Same problems and amount of problems as Condition 1, but training is randomly shuffled rather than ordered.
  - **Condition 4:** `train_teacher_shuffled`. Same problems and amount of problems as Condition 2, but training is randomly shuffled rather than ordered.
- **Experiment 3: Curriculum Relevance.** Does training on random subproblems have any effects as compared to SFT Teacher-generated subproblems?
  - **Condition 1:** `random_two`. 36 steps of `rung_2num_random`, followed by 84 steps of `train_hard_25k`.
  - **Condition 2:** `teacher_two`. 36 steps of `test_easy_teacher`, followed by 84 steps of `train_hard_25k`.
- **Experiment 3.5: Curriculum Scaffolding Ablation.** Does training on a scaffolding of 2-number then 3-number problems, as compared to solely 2-number problems, improve performance?
  - **Condition 1:** `random_two`. 36 steps of `rung_2num_random`, followed by 84 steps of `train_hard_25k`.
  - **Condition 2:** `random_three`. 36 steps of `rung_3num_random`, followed by 84 steps of `train_hard_25k`.
  - **Condition 3:** `random_scaffolded`. 18 steps of `rung_2num_random`, followed by 18 steps of `rung_3num_random`, followed by 84 steps of `train_hard_25k`.

- **Experiment 4: Fixed Alternate Scheduling.** Do other fixed scheduling techniques such as alternating blocks, demonstrate any improvements?
  - **Condition 1:** `gemini_alternating`. 30-step alternating blocks of `test_easy_gemini` and `train_hard_25k`.
  - **Condition 2:** `teacher_aug_alternating`. 30-step alternating blocks of `test_easy_teacher_aug` and `train_hard_25k`.
  - **Condition 3:** `teacher_aug_expanded`. 100 steps of `test_easy_teacher_aug`, followed by 100 steps of `train_hard_25k`, followed by 50 steps of `test_easy_teacher_aug`, followed by 50 steps of `train_hard_25k`.
  - **Condition 4:** `equal_scaffolding`. 40 steps of `rung_2num_random`, followed by 40 steps of `rung_3num_random`, followed by 40 steps of `train_hard_25k`.
  - **Condition 5:** `interleaving`. 4 cycles of 10 steps of `rung_2num_random`, followed by 10 steps of `rung_3num_random`, followed by 10 steps of `train_hard_25k`.
  - **Condition 6:** `hard_heavy`. 6 cycles of 2 steps of `rung_2num_random`, followed by 2 steps of `rung_3num_random`, followed by 16 steps of `train_hard_25k`.
- **Experiment 5: Probabilistic Alternate Scheduling.** Do other probabilistic scheduling techniques such as Gaussian or Cosine scheduling demonstrate any improvements?
  - **Condition 1:** `gemini_cosine`. Cosine decay schedule ($p_{\text{easy}}(t) = \frac{1}{2}(1 + \cos(\pi t))$) with `test_easy_gemini` as the easy dataset. p_{easy} starts at 1.0 and decays smoothly to 0.0 over 90 steps.
  - **Condition 2:** `3num_cosine`. Same as Condition 1 except with `rung_3num_random` as the easy dataset.
  - **Condition 3:** `teacher_aug_cosine`. Same as Condition 1 except with `test_easy_teacher_aug` as the easy dataset.
  - **Condition 4:** `gemini_gaussian`. Gaussian decay schedule ($p_{\text{easy}}(t) = \exp(-t^2/2\sigma^2)$, $\sigma = 0.25$) with `test_easy_gemini` as the easy dataset. Easy exposure is concentrated in the first $\sim$25% of training and decays rapidly thereafter.
  - **Condition 5:** `3num_gaussian`. Same as Condition 4 except with `rung_3num_random` as the easy dataset.
  - **Condition 6:** `teacher_aug_gaussian`. Same as Condition 4 except with `test_easy_teacher_aug` as the easy dataset.

These experiments differ from a standard easy-to-hard curriculum in two ways. First, instead of using only generic easier Countdown problems, we explicitly tested whether easy examples are more useful when they correspond to intermediate computations from target testing problems. This lets us distinguish the effect of general arithmetic practice from the effect of practicing reusable subcomputations that appear inside full solutions.

Second, we compared multiple ways of exposing the model to easy and hard problems: ordered easy-to-hard training, shuffled mixtures, fixed alternating schedules, scaffolded 2-number to 3-number transitions, and continuous probabilistic schedules. This allowed us to test not only whether curriculum examples help, but also whether the timing and smoothness of the curriculum affects reasoning performance. Together, these design choices make our approach more targeted than simply adding extra easy data: we study curriculum relevance, curriculum source, and curriculum schedule as separate factors in RLOO training for arithmetic reasoning.

5 Results

5.1 Quantitative Evaluation

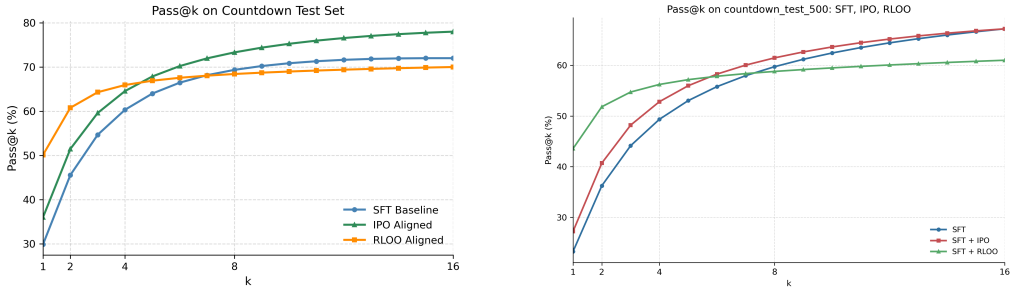
5.1.1 Baseline Metrics: SFT, IPO, RLOO

The SFT model provided the base reasoning checkpoint, achieving about 30% pass@1 on the original 50-problem benchmark. This showed that supervised fine-tuning taught the model the task format and some arithmetic reasoning, but still left substantial room for improvement.

IPO gave a modest improvement over SFT. On the 50-problem benchmark, the best IPO checkpoint reached 36% pass@1. On the larger 500-problem set, IPO improved pass@1 from 23.3% to 27.2%.

However, SFT and IPO both reached 67.2% pass@16 on the 500-problem set, suggesting that IPO mainly made correct solutions appear earlier in the sampled responses rather than increasing the total number of solvable problems.

RLOO produced the strongest pass@1 improvement. It reached about 50% pass@1 on the 50-problem benchmark and 43.6% pass@1 on the 500-problem set. This suggests that verifier-based reinforcement learning was especially effective at increasing single-sample correctness. However, on the 500-problem set, RLOO reached 61.0% pass@16, below SFT and IPO, indicating that RLOO improved the chance of getting a correct answer on the first sample, but did not improve coverage across multiple attempts.



Original 50-problem benchmark | Larger 500-problem benchmark

Figure 1: Pass@k comparison for the SFT, IPO, and RLOO baseline models on the original 50-problem Countdown benchmark and the larger 500-problem evaluation set.

5.1.2 Experiment 1: Dataset Source

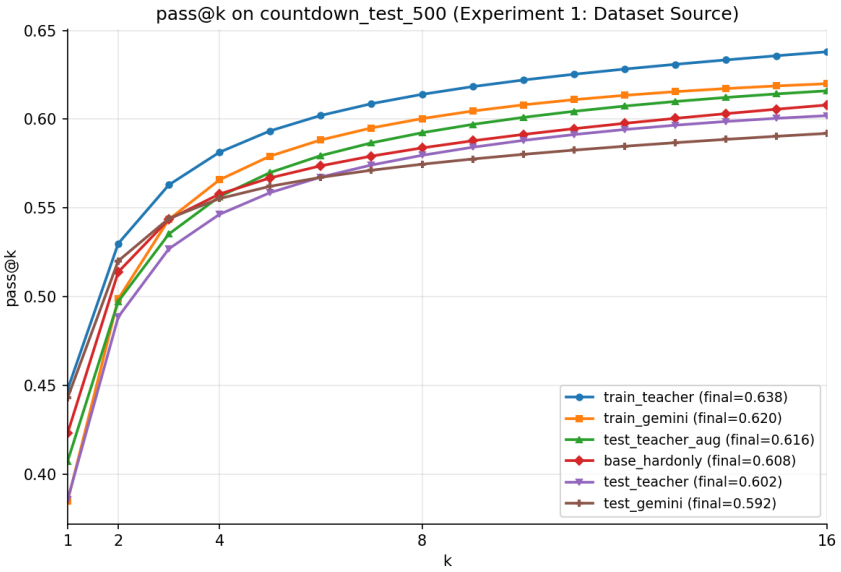


Figure 2: Pass@k on the 500-problem Countdown evaluation set for Gemini vs. SFT Teacher-generated easy subproblem datasets.

To test whether the source of easy curriculum problems affects performance, we compared four easy-subproblem curricula (varying the generator – Gemini or SFT Teacher, and the problem origin – test or training set), plus an augmented teacher variant, against the baseline RLOO trained only on hard problems.

Only one curriculum significantly improved greedy accuracy over the hard-only baseline, which reached a pass@1 of 41.2%. `train_teacher` raised pass@1 to 46.8%, a +5.6 percentage point gain (95% CI [+1.6, +9.6], $p = 0.048$). `test_gemini` produced a comparable increase to 46.4% (+5.2 percentage points, 95% CI [+1.4, +9.0]) but did not survive multiple-comparison correction ($p = 0.054$). The remaining conditions did not improve greedy accuracy: `test_teacher` (38.2%) and `train_gemini` (37.2%) were lower than baseline, and `test_teacher_aug` was statistically indistinguishable from it (41.4%, $p = 1.0$). No condition significantly improved coverage: `train_teacher` achieved the highest pass@16 at 63.8% (+3.0 percentage points, 95% CI [+0.6, +5.4]), but this gain did not survive correction ($p = 0.12$). Notably, neither the generator nor the subproblem origin produced a consistent advantage. The two strongest curricula came from opposite generator–origin combinations, indicating that easy-subproblem curricula provide at best small, condition-specific gains over training on hard problems alone.

5.1.3 Experiment 2: Shuffled Training

We next tested whether the order of teacher-generated curriculum examples affected downstream Countdown performance. For each teacher-data source, we compared an ordered curriculum against a shuffled version while keeping the training setup fixed. We focus on the 500-problem evaluation set, since the 50-problem benchmark showed similar trends but with less statistical power.

For the test-teacher curriculum, shuffling did not produce a statistically significant change relative to the ordered curriculum. The shuffled model achieved 38.0% pass@1 compared to 36.6% for the ordered model, but this difference was not significant under paired permutation testing (difference = +1.4 percentage points, 95% CI [-0.3, +3.0], p = 0.109). At pass@16, the shuffled model reached 59.8% compared to 60.6% for the ordered model, which was also not significant (p = 0.617).

For the train-teacher curriculum, shuffling improved pass@1 relative to ordering. The shuffled model achieved 35.8% pass@1 compared to 33.2% for the ordered model, a significant +2.6 percentage point improvement (95% CI [+0.7, +4.5], p = 0.006). However, this advantage disappeared at pass@16: the shuffled model reached 60.8% compared to 61.2% for the ordered model, a non-significant difference (p = 0.873).

Overall, these results suggest that curriculum ordering had only a limited effect. Shuffling helped single-sample accuracy for the train-teacher curriculum, but this improvement did not translate into solving more problems across 16 attempts. For the test-teacher curriculum, ordering and shuffling were statistically indistinguishable. This indicates that the content of the teacher-generated curriculum may matter more than the precise ordering of examples, at least under the schedules tested here.

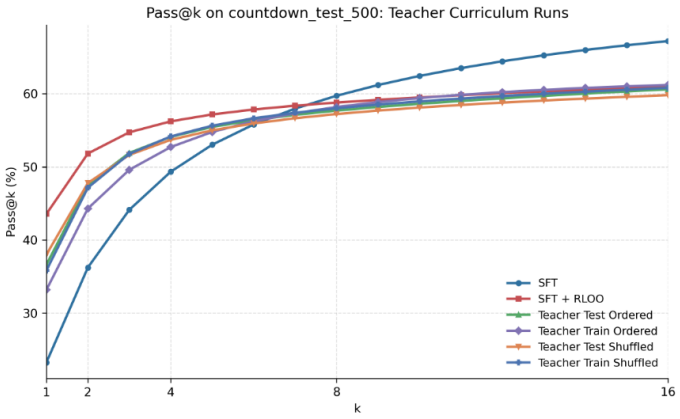


Figure 3: Pass@k on the 500-problem Countdown evaluation set for the non-curriculum RLOO baseline and the teacher-generated curriculum models. Ordered and shuffled curriculum variants are shown for both train-derived and test-derived teacher datasets.

5.1.4 Experiment 3: Curriculum Relevance

To isolate the role of curriculum relevance, we compared two SFT-teacher-generated easy-problem curricula before hard-problem training. This comparison tests whether the benefit comes simply from easier arithmetic practice, or from easy examples that rehearse intermediate computations needed for the final hard problems.

On the 500-problem evaluation set, the subproblem curriculum improved pass@1 from 34.8% to 36.6% over the unrelated-easy curriculum, a significant +1.9 percentage point gain (95% CI [+0.1, +3.7], $p = 0.045$). This suggests that easy examples are most useful when they rehearse computations that are directly relevant to the later hard problems, rather than serving only as generic arithmetic warmup. The subproblem curriculum also achieved a higher pass@16, reaching 60.6% compared to 58.2% for the unrelated-easy curriculum, though this broader coverage gain was not statistically significant (95% CI [-0.2, +5.0], $p = 0.103$).

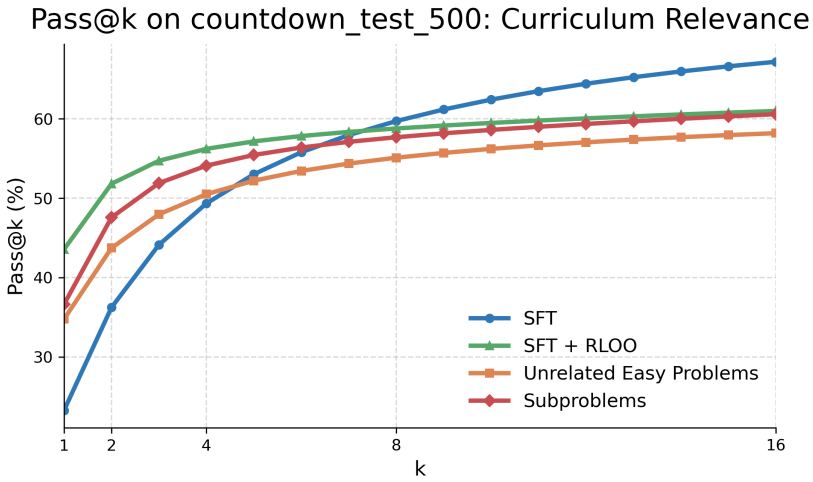


Figure 4: Pass@k on the 500-problem Countdown evaluation set for SFT, the non-curriculum SFT+RLOO baseline, and two curriculum relevance conditions. Both curriculum datasets were generated by the SFT teacher; the unrelated-easy curriculum uses generic 2-number problems, while the other uses 2-number subproblems derived from the hard problems.

5.1.5 Experiment 3.5: Curriculum Transition Ablation

We next tested whether gradually increasing the difficulty of random easy-problem curricula improves downstream Countdown performance. All three curricula used SFT-teacher-generated random arithmetic problems that were not constructed as subproblems of the target hard examples. We compared training on 2-number random problems before hard problems, training on 3-number random problems before hard problems, and a gradual schedule that trained on 2-number random problems, then 3-number random problems, then hard problems.

On the 500-problem evaluation set, both curricula that included 3-number random problems substantially improved pass@1 over the 2-number-only curriculum. The 3-number-to-hard schedule improved pass@1 from 34.8% to 39.9% ($p_{\text{Holm}} < 0.001$), while the gradual 2-to-3-to-hard schedule improved pass@1 to 40.5% ($p_{\text{Holm}} < 0.001$). This suggests that 3-number warmup problems provide a better bridge to the full Countdown task than 2-number warmup alone, likely because they require more multi-step arithmetic while remaining easier than the target hard problems.

The gradual schedule showed its clearest advantage at pass@16. It reached 62.2% pass@16, compared to 58.2% for the 2-number-to-hard curriculum ($p_{\text{Holm}} = 0.009$) and 59.4% for the 3-number-to-hard curriculum ($p_{\text{Holm}} = 0.049$). Unlike the pass@1 comparison, where the gradual schedule was not significantly better than starting directly with 3-number problems, the pass@16 result suggests that smoothing the transition from easier to harder problems may improve broader problem coverage across multiple attempts.

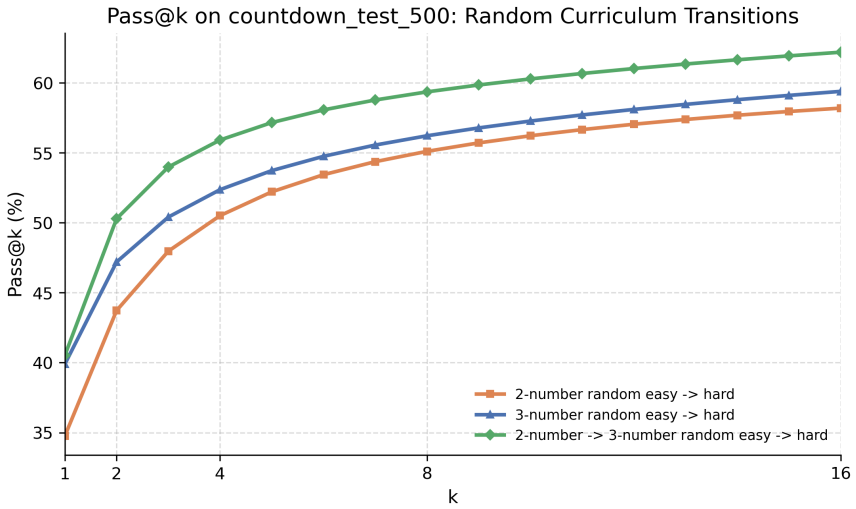


Figure 5: Pass@k on the 500-problem Countdown evaluation set for random easy-problem curricula with different difficulty transitions. The gradual 2-number to 3-number to hard schedule improves pass@16 over both direct random-curriculum schedules.

5.1.6 Experiment 4: Fixed Alternate Scheduling

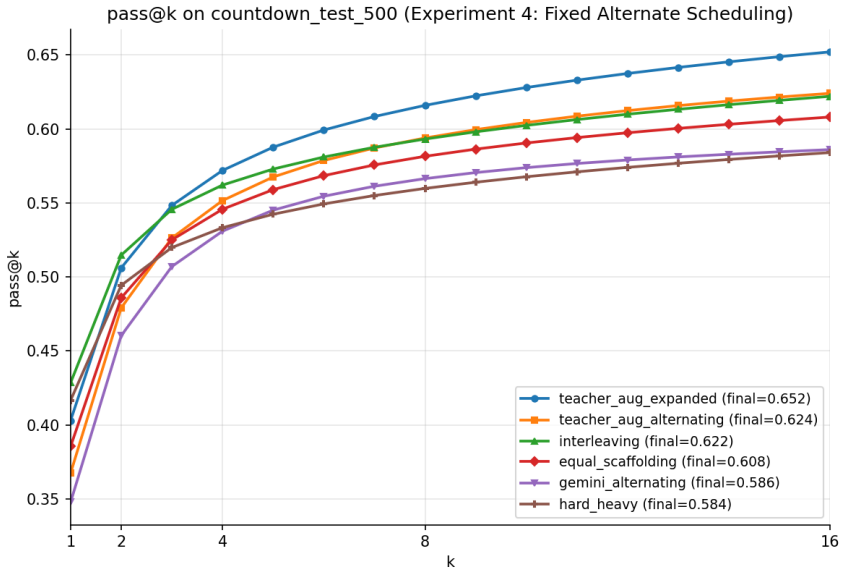


Figure 6: Pass@k on the 500-problem Countdown evaluation set for fixed alternate scheduling techniques. The expanded run with the teacher augmented dataset performs significantly better than the RLOO baseline for pass@k.

To test whether fixed scheduling strategies beyond a single easy-to-hard transition could produce additional gains, we compared six static curriculum schedules against the baseline RLOO trained only on hard problems.

None of the six schedules improved greedy accuracy over RLOO, which reached a pass@1 of 43.0%. The two alternating-block schedules were significantly worse: `gemini_alternating` fell to 33.0% pass@1, a 10.0 percentage point drop (95% CI $[-14.0, -6.2]$, $p < 0.001$), and `teacher_aug_alternating` fell to 36.4% (-6.6 percentage points, 95% CI $[-10.6, -2.6]$, $p =$

0.012). The scaffolded schedules were closer but still not better: `interleaving` was statistically indistinguishable from RLOO (42.6% pass@1, -0.4 percentage points, 95% CI $[-4.6, +3.8]$, $p = 1.0$), as were `equal_scaffolding` (41.0%) and `hard_heavy` (41.2%). The only positive effect appeared in coverage rather than greedy accuracy: `teacher_aug_expanded` raised pass@16 from 61.0% to 65.2%, a significant $+4.2$ percentage point gain (95% CI $[+1.6, +6.8]$, $p = 0.016$), while its pass@1 was not significantly different from RLOO (40.2%, -2.8 percentage points, $p = 1.0$). These results indicate that fixed scheduling does not improve the greedy policy over pure RLOO; its only measurable benefit is improved solution coverage from revisiting easy problems after hard training.

5.1.7 Experiment 5: Probabilistic Alternate Scheduling

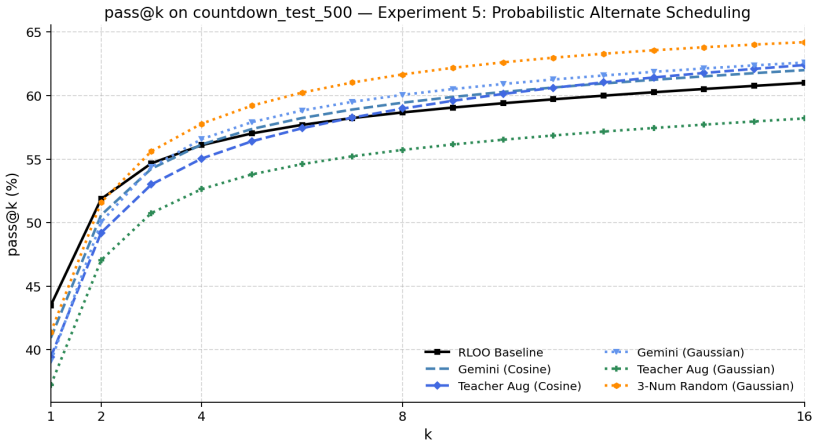


Figure 7: Pass@k on the 500-problem Countdown evaluation set for the RLOO baseline and the six probabilistic scheduling conditions. Dashed lines show cosine-scheduled conditions and dotted lines show Gaussian-scheduled conditions, across the Gemini, augmented teacher, and 3-number random easy datasets.

To test whether smoothing curriculum transitions with continuous probabilistic decay functions could improve over fixed-block schedules, we evaluated six conditions combining two decay functions (cosine and Gaussian) with three easy datasets (Gemini-derived subproblems, augmented teacher subproblems, and 3-number random problems). Rather than switching between datasets at predetermined step boundaries, each condition samples from the easy dataset with probability $p(t)$ at training step t, where $t \in [0, 1]$ is normalized progress. The cosine schedule sets $p(t) = \frac{1}{2}(1 + \cos(\pi t))$, decaying smoothly from full easy exposure to full hard exposure. The Gaussian schedule sets $p(t) = \exp\left(-t^2/2\sigma^2\right)$ with $\sigma = 0.25$, concentrating easy exposure in the first quarter of training and decaying sharply thereafter. All six conditions ran for 90 training steps.

None of the probabilistic conditions improved pass@1 over the RLOO baseline of 43.5%. Among cosine-scheduled conditions, Gemini cosine was the strongest at 40.9% pass@1 (-2.6 percentage points), followed by Teacher Aug cosine at 39.4% (-4.1 percentage points). The 3-number random cosine condition could not be evaluated due to a file error. Among Gaussian-scheduled conditions, the 3-number random Gaussian model performed best at 41.4% pass@1 (-2.1 percentage points), while Gemini Gaussian reached 39.1% (-2.4 percentage points) and Teacher Aug Gaussian was the weakest overall at 37.2% (-6.3 percentage points).

Despite underperforming at pass@1, several conditions showed improved coverage at higher k. The 3-number random Gaussian model reached approximately 64% pass@16, above the RLOO baseline of 61%, while Gemini cosine, Teacher Aug cosine, and Gemini Gaussian each reached approximately 62% pass@16. Teacher Aug Gaussian was the only condition that remained below the baseline across all k, reaching approximately 58% pass@16.

5.2 Qualitative Analysis

5.2.1 Overall trends

Across our curriculum experiments, curriculum training did not consistently outperform the non-curriculum RLOO baseline. Most curriculum-trained models matched or slightly underperformed the RLOO baseline, suggesting that our curricula did not produce a broad improvement over standard RLOO fine-tuning. However, the ablations revealed that curriculum design still mattered: computationally relevant subproblems improved pass@1 relative to unrelated easy problems, while smoother difficulty transitions improved pass@16 relative to direct jumps from easy to hard problems.

For the curriculum relevance ablation, the subproblem curriculum improved pass@1 over the unrelated easy-problem curriculum. Since both curricula were generated by the same teacher pipeline, this suggests that the gain was not simply due to additional teacher-generated arithmetic practice. Instead, the improvement indicates that easy examples are more useful when they rehearse intermediate computations needed for the final hard problems. The pass@16 improvement was in the same direction but not statistically significant, so this effect appears strongest for making correct solutions more likely early in sampling rather than clearly expanding the total set of problems solved across 16 attempts.

The curriculum transition ablation showed a different kind of curriculum effect. Training on 3-number random problems before hard problems improved pass@1 compared to training only on 2-number random problems, and the gradual 2-number to 3-number to hard schedule achieved a similar pass@1 gain. The gradual schedule was most distinctive at pass@16, where it outperformed both direct schedules. This suggests that intermediate difficulty examples may provide a useful bridge between simple arithmetic and full Countdown reasoning, with the smoother transition helping the model solve a broader set of problems across multiple attempts.

The later scheduling experiments reinforce this interpretation. Fixed alternate and probabilistic schedules did not improve greedy pass@1 over the RLOO baseline, and in some cases performed significantly worse. However, several schedules improved or nearly improved pass@16. In the fixed alternate setting, the expanded augmented-teacher curriculum significantly improved pass@16 while leaving pass@1 statistically unchanged. Similarly, in the probabilistic scheduling experiment, several conditions underperformed at pass@1 but reached higher pass@16 values than the baseline. This suggests that these schedules may have helped the model explore a broader set of candidate solutions without making the first sampled solution more reliable. Qualitatively, this is consistent with rollouts where curriculum-trained models produce more varied multi-step attempts, but still suffer from arithmetic mistakes, invalid number use, or answer-formatting failures.

Overall, these results suggest that our curriculum interventions were partially successful. They did not yield a model that consistently beat the strong RLOO baseline, likely because the curriculum datasets were relatively small and the additional training signal was not strong enough to broadly improve reasoning ability. However, the ablations show that the choice of curriculum examples and the transition between difficulty levels both affected model behavior. In the qualitative rollout analysis below, we examine representative successes and failures to see whether curriculum-trained models reuse relevant intermediate computations, make fewer arithmetic mistakes, or recover differently across multiple sampled attempts.

5.2.2 Subproblem curricula encourage reusable intermediate computations

A representative success case appears in the subproblem curriculum. For the hard problem with numbers $[3, 35, 84]$ and target 63, the subproblem-trained model finds the solution by first computing the reusable intermediate

$$84/3 = 28,$$

then combining it with the remaining number:

$$28 + 35 = 63.$$

Its rollout explicitly follows this decomposition:

```
84 / 3 = 28
28 + 35 = 63
This works! Let's verify:
```

```
84/3 = 28
28 + 35 = 63
<answer> (84/3) + 35 </answer>
```

This is exactly the kind of computation that the subproblem curriculum was designed to rehearse: an easier two-number subproblem, $[84, 3] \rightarrow 28$, becomes an intermediate step in the full hard problem. This does not prove that the model memorized that specific subproblem, but it suggests that subproblem training can bias the model toward decompositions that are useful for the final task. By contrast, random or unrelated easy problems provide arithmetic practice but do not necessarily train intermediate values that compose into the hard target.

5.2.3 Invalid arithmetic despite correct answer format

The failures, however, show why these behavioral improvements did not always translate into higher measured accuracy. Many incorrect rollouts have the right surface format but fail the actual Countdown constraints. For example, on the problem $[74, 5, 20, 88] \rightarrow 50$, one model writes:

```
74 - 5 = 70
70 - 20 = 50
<answer> (74 - 5) - 20 </answer>
```

The answer is formatted correctly, but the arithmetic is wrong: $74 - 5 = 69$, not 70, and the final expression evaluates to 49, not 50. It also omits the input number 88, violating the requirement that every number be used exactly once. This pattern occurred frequently in the rollouts: the model often identifies a plausible path, writes an answer in the required tags, and then receives only the format reward because the expression is invalid or numerically incorrect.

5.2.4 Extraction failures: solved but misformatted

Because our scorer reads only the final `<answer>` span and requires every input number to be used exactly once, a model can derive a correct solution while receiving no credit. On the 500-problem test set with 16 samples per problem, this failure mode is essentially absent in the SFT model: its low pass@1 reflects incompetence rather than formatting issues. However, the issue is induced by RLOO, which loses a noticeable fraction of completions this way, and curriculum training does not reliably help.

Inspection reveals that the primary cause is a failure to stop generating. The model emits a correct `<answer>`, then continues verifying or trying new approaches, and thus overwrites it with later incorrect answer tags. Since the scorer keeps the last answer span, these completions are marked wrong even though a valid solution appeared earlier in the response. This means that some of the apparent pass@1 gaps reflect an answer-extraction artifact rather than a pure reasoning gap.

For example, on the problem with numbers $[44, 19, 35]$ and target 98, the model emits the correct answer twice, then keeps trying different approaches and overwrites it:

```
<think>... 35 + 19 = 54; 54 + 44 = 98! ...</think>
<answer>(35 + 19) + 44</answer>        <-- correct

<think>... Let me try one more approach ...</think>
<answer>(44 - 35) + 19</answer>        <-- wrong
<answer>(44 * 35) / 19</answer>        <-- final, wrong
```

The completion contains a valid solution, but the final answer tag is incorrect, so the scorer marks it wrong.

A related case occurs for numbers $[63, 95, 96]$ and target 64. The model finds the valid solution $63 + (96 - 95)$, but later answer tags incorrectly inject the target value itself as an operand:

```
<think>... 96 - 95 = 1; 63 + 1 = 64. This works! ...</think>
<answer>(63 + (96 - 95))</answer>       <-- correct

<think>... Let me try one last sequence ...</think>
```

```
<answer>(96 - 63 + 95) - 64</answer>    <-- uses 64, invalid
<answer>(63 + 96 - 95) - 64</answer>    <-- final, invalid
```

5.2.5 Gradual difficulty transitions produce more complete attempts

The transition ablation shows a different qualitative effect. Models trained with a smoother $2 \to 3 \to$ hard curriculum often produced more complete multi-step attempts than models trained with a direct $2 \to$ hard transition. For the problem $[49, 41, 73] \to 17$, the direct 2-number curriculum initially finds the right idea in its reasoning:

```
73 - 41 = 32
49 - 32 = 17
```

but then outputs only:

```
<answer>73 - 41</answer>
```

which is incomplete and does not use all numbers. By contrast, the gradual curriculum produces a complete expression:

```
49 + 41 = 90
90 - 73 = 17
<answer> (49 + 41) - 73 </answer>
```

This suggests that intermediate 3-number training may help the model bridge from simple arithmetic to full Countdown-style composition. The direct 2-number curriculum teaches useful local operations, but the model can still fail to assemble them into a final expression that uses every number. The gradual schedule more often carries the reasoning through to a complete multi-step answer, which is consistent with its stronger pass@16 performance.

6 Discussion

Our results suggest that curriculum learning for small reasoning models is more delicate than simply adding easier examples before harder ones. Although the curriculum-trained models did not consistently outperform the RLOO baseline, the ablations showed that the structure of the curriculum mattered. Subproblems that were computationally related to the final hard problem were more useful than unrelated easy arithmetic problems, and smoother transitions from 2-number to 3-number to hard Countdown problems produced stronger pass@16 performance than direct jumps in difficulty. These findings suggest that curriculum learning can shape model behavior, but that the benefits depend strongly on the quality, relevance, and scale of the curriculum data.

The fixed alternate and probabilistic scheduling experiments further suggest that curriculum scheduling mostly affected coverage rather than greedy accuracy. Neither class of schedules improved pass@1 over RLOO, and some alternating schedules significantly hurt pass@1. The clearest positive signal appeared at higher k: the expanded augmented-teacher fixed schedule significantly improved pass@16, and several probabilistic schedules reached approximately 62–64% pass@16 despite underperforming at pass@1. This pattern suggests that revisiting easier problems after hard training may encourage exploration or preserve a wider set of solution strategies, but it does not necessarily improve the model's default greedy policy.

This distinction is important because pass@1 and pass@16 measure different kinds of improvement. A pass@1 gain would indicate that the model's most likely completion has become more reliable. A pass@16 gain, by contrast, can indicate that the model is capable of finding a correct solution somewhere in its sampled distribution, even if that solution is not consistently ranked first. Our scheduling results therefore point to a limited but useful effect: curriculum schedules may broaden the model's search behavior, but additional training or decoding interventions are needed to turn that broader coverage into reliable first-attempt accuracy.

A major limitation of our experiments was the size of the curriculum datasets. Generating subproblem curricula required repeatedly solving hard Countdown problems and decomposing those solutions into intermediate arithmetic steps. Because we had limited compute and API credits, the resulting

curriculum datasets were relatively small compared to the full hard-problem training distribution. This likely weakened the curriculum signal: even if the subproblems were useful, the model may not have seen enough of them for the effect to dominate standard RLOO training on hard problems. Larger curriculum datasets might produce stronger and more stable gains, especially for pass@1, where the subproblem curriculum showed the clearest improvement.

A second limitation was the difficulty of generating high-quality subproblems. Our pipeline asked a teacher model to first solve a hard Countdown problem and then decompose that solution into smaller subcomputations. This creates a bottleneck: if the teacher fails to solve the original hard problem, then the generated subproblems are either missing, invalid, or not actually useful for the target problem. In some cases, the teacher could generate easy arithmetic tasks, but these tasks were not guaranteed to correspond to a valid decomposition of the hard instance. This weakens the intended curriculum because the model may practice operations that look plausible but do not help solve the final problem.

We initially expected that using a stronger external teacher would reduce this problem, so we also generated subproblem curricula with Gemini. However, Gemini was not consistently reliable on the hard Countdown problems either. In the curriculum-source ablation, neither the generator nor the origin of the hard problems produced a stable advantage. The best condition, `train_teacher`, improved pass@1 from the hard-only baseline of 41.2% to 46.8%, a $+5.6$ percentage point gain. The `test_gemini` curriculum produced a similar pass@1 of 46.4%, but this result did not survive multiple-comparison correction. Other Gemini- and teacher-generated variants underperformed or matched the baseline: `train_gemini` reached only 37.2%, `test_teacher` reached 38.2%, and the augmented teacher condition was essentially unchanged at 41.4%. No condition significantly improved pass@16 after correction.

These results suggest that the limiting factor was not simply whether the easy curriculum was generated by Gemini or by the SFT teacher. Rather, the broader issue was the reliability of the curriculum-generation process itself. Curriculum generation is most useful when the teacher can reliably identify the hidden structure of a hard problem and expose that structure through valid intermediate examples. If the teacher, whether Gemini or the SFT model, fails to solve the hard instance or produces noisy decompositions, then the resulting curriculum can reinforce unhelpful or invalid reasoning patterns. Future work should therefore use a stronger solver, an exact symbolic search algorithm, or a verifier-guided generation pipeline to ensure that every subproblem is valid and genuinely useful for the corresponding hard problem.

Our qualitative analysis also revealed that measured performance was affected by answer formatting and extraction failures. RL-trained models often found a correct solution somewhere in the completion but then continued generating, overwrote the correct answer with an incorrect final answer tag, or used invalid operands in a later expression. Since the scorer only evaluates the final `<answer>` span, these completions receive no credit even when they contain correct reasoning. This suggests that some apparent reasoning failures are partly artifacts of the decoding and evaluation protocol. Adding explicit stop tokens, training the model to terminate after a valid answer, or using a verifier to select the first valid expression could recover some of this lost performance.

More broadly, our findings highlight a challenge for improving reasoning in small language models. Learning strategies such as decomposition, intermediate computation reuse, and gradual skill transfer are difficult to induce from sparse RL rewards alone. Curriculum learning offers a promising way to make these strategies more explicit, but only when the curriculum examples are accurate, relevant, and numerous enough to affect training. For small models, noisy or weak curricula may not provide enough signal to overcome arithmetic errors, formatting pathologies, and limited generalization capacity.

6.1 Broader Impact

This project studies methods for improving mathematical reasoning in small language models. If successful, such methods could make reasoning-capable systems cheaper and more accessible, reducing the need for very large models in educational, scientific, or productivity settings. However, our results also show that training small models to reason reliably remains difficult. Models may appear to follow a correct strategy while still making subtle arithmetic errors, violating constraints, or

producing invalid final answers. This is especially important in domains where correctness matters: improving reasoning ability must be paired with strong verification and careful evaluation.

Our work also suggests that automated curriculum generation should be used cautiously. If teacher-generated data are noisy, invalid, or produced by a model with similar weaknesses to the student, then the curriculum can reinforce flawed reasoning patterns rather than correct them. Future systems that use model-generated curricula should include external validation, exact solvers, or human oversight when correctness is important.

7 Conclusion

We investigated whether curriculum learning can improve reinforcement learning for Countdown arithmetic reasoning. Overall, curriculum training did not consistently outperform a strong non-curriculum RLOO baseline. However, our ablations showed that curriculum design mattered: computationally relevant subproblems were more useful than unrelated easy problems, and gradual difficulty transitions improved performance across multiple samples. The later scheduling experiments sharpen this conclusion: alternate and probabilistic curricula sometimes improved coverage at higher k, but they did not improve greedy accuracy, suggesting that curriculum learning affected exploration more than reliable single-sample reasoning. These results suggest that curricula can shape the reasoning behavior of small language models, even when they do not produce large aggregate gains.

The main takeaway is that useful curricula must teach the structure of the target task, not merely provide easier examples. Subproblem curricula helped when they rehearsed intermediate computations that could be reused in hard problems, while smoother transitions helped models produce more complete multi-step attempts. At the same time, the benefits were limited by small dataset size, noisy teacher-generated decompositions, and answer-formatting failures during evaluation.

Future work should focus on generating larger and cleaner curriculum datasets, ideally using exact symbolic solvers or verifier-guided pipelines rather than relying only on teacher model outputs. Another promising direction is to improve the evaluation and decoding protocol so that models are rewarded for the first valid solution they find and trained to stop after producing it. Finally, future experiments could test whether curriculum learning becomes more effective with stronger teachers, larger student models, or adaptive schedules that introduce harder problems only after the model has mastered simpler intermediate skills.

In summary, curriculum learning did not solve Countdown reasoning in our setting, but it revealed useful signals about how small models learn arithmetic strategies. The results point toward a more careful view of curriculum design: the easiest examples are not necessarily the most helpful ones; the most helpful examples are those that expose the reusable computations needed for harder reasoning.

8 Team Contributions

- **Amrita Malhotra** implemented SFT, generated the datasets for the Gemini and teacher-generated subproblems, generated the random subproblem datasets, and implemented a variety of scheduling strategies, focusing on the fixed strategies.
- **Giselle Rivera** implemented RLOO and a variety of scheduling strategies, focusing on the probabilistic strategies.
- **Zofia Dudek** implemented IPO, created the general algorithm for the extension, helped to generate the random subproblem datasets, implemented shuffling strategies, and conducted evaluations.

All members contributed to the final report equally. While we fully ideated the extension ourselves, we would like to disclose that we used AI tools in the extension to help with automating training runs, debugging issues with disconnection from Modal, and conducting statistical analyses.

Changes from Proposal Originally, we had planned for each team member to take one of the static, dynamic-random-subproblems, and dynamic-related-subproblems tasks. We did not end up implementing a dynamic strategy (as our training rewards plateaued and thus did not indicate that it would be useful), and we instead divided the workload for generating datasets and creating a variety of training strategies based on the ongoing results of experimentation.

References

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2023. A General Theoretical Paradigm to Understand Learning from Human Preferences. *arXiv e-prints* (Oct. 2023), arXiv:2310.12036. arXiv:2310.12036 doi:10.48550/arXiv.2310.12036

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

Wouter Kool, Herke van Hoof, and Max Welling. 2019. Buy 4 REINFORCE Samples, Get a Baseline for Free! *Deep Reinforcement Learning Workshop, NeurIPS* (2019).

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research* 21, 181 (2020), 1–50.

Shubham Parashar et al. 2025. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. *arXiv preprint arXiv:2506.06632* (2025).

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).