Extended Abstract

Motivation. Our project focuses on the extension component of the default Countdown RL pipeline: whether the training task distribution can be improved through self-play. Standard RLOO optimizes a verifier reward on a fixed set of prompts, but this fixed distribution can become mismatched to the solver's ability. Tasks that are too easy provide little useful gradient, while tasks that are too hard mostly produce zero-reward rollouts. We therefore design a self-play curriculum that continuously generates new Countdown tasks near the solver's current learning frontier.

Method. Our extension introduces a proposer-solver loop. The proposer generates candidate Countdown tasks by mutating seed tasks and constructing reachable targets, while the solver is trained with RLOO on a mixture of original and generated tasks. Generated candidates are filtered for validity, exact solvability, non-triviality, and uniqueness. They are then scored using

$$R_{\text{gen}}(x) = R_{\text{diff}}(x) + \lambda R_{\text{div}}(x),$$

where R_{diff} encourages tasks near a target solve-probability band and R_{div} rewards novelty relative to an archive of previously accepted tasks. This turns task generation into a curriculum-selection problem rather than simple data augmentation.

Implementation. We implement both non-trained and trainable proposer variants. The main non-trained proposer is a heuristic UCB mutation proposer that adaptively chooses among easy, medium, and hard symbolic mutation regimes. We also implement predictor-guided selection using a learned p_{solve} model, which estimates the probability that the current solver can solve a candidate task. For the trainable version, we introduce a factorized symbolic proposer with actions

$$a = (a_{\text{num}}, a_{\text{expr}}),$$

where a_{num} controls number mutation and a_{expr} controls expression structure. This proposer is trained with REINFORCE using the continuous generation reward R_{gen}. To address predictor staleness, we add a candidate reservoir and periodic p_{solve} predictor refresh: recent candidates are labeled using current solver rollouts, appended to an online dataset, and used to retrain and reload the predictor.

Results. The strongest self-play result comes from the UCB mutation proposer. Compared with standard RLOO, UCB self-play maintains similar pass@1 but improves pass@16 from 74% to 78%, indicating better solution coverage under sampling. This supports the idea that generated curriculum tasks can expose the solver to a broader set of arithmetic structures than the fixed training distribution. In contrast, predictor-based variants with strict difficulty-band admission achieve 70–72% pass@16, below standard RLOO. Although the refresh variant reaches the highest pass@1, it does not improve pass@16 coverage.

Discussion. Our diagnostics show that the main challenge is not simply generating more tasks, but deciding which generated tasks should enter training. The UCB variant is permissive: all validity-passing candidates remain eligible for top-k selection. Predictor-based variants instead hard-reject tasks outside the predicted band $0.2 \leq \hat{p}_{\text{solve}}(x) \leq 0.6$, which can shrink the effective candidate pool and discard useful frontier tasks when the predictor is imperfect. Periodic predictor refresh and improved diversity features help make the system more mechanically correct, but they do not fully overcome the restrictiveness of hard admission within our 100-step training budget.

Conclusion. Our extension shows that self-play can improve sparse-reward RL for reasoning by dynamically adapting the training distribution. The best-performing UCB self-play variant improves pass@16, suggesting that adaptive task generation can increase solution coverage beyond standard RLOO. At the same time, our predictor-based variants reveal an important design lesson: difficulty estimation should guide curriculum selection without becoming an overly strict rejection gate. Future work should replace hard difficulty-band admission with softer difficulty penalties, run longer training, and improve learned task representations for both difficulty and diversity.

Joint Optimization of Task Difficulty and Diversity for Fine-Tuning LLMs under Sparse Rewards

Jenny Jin
Department of Computer Science
Stanford University
yqjin@stanford.edu

Jiaming Shen
Department of Computer Science
Stanford University
shenjm@stanford.edu

Jiaxin Fang
Department of Computer Science
Stanford University
jiaxinf@stanford.edu

Abstract

We study how to improve sparse-reward reinforcement learning for LLM reasoning by adapting the training task distribution rather than training only on a fixed dataset. Our extension augments RLOO on the Countdown arithmetic task with a self-play curriculum, where a proposer generates new valid and solvable tasks near the solver's current learning frontier. The proposed system scores generated tasks using a joint difficulty-diversity objective, stores selected tasks in a replay buffer, and trains the solver on a mixture of original and generated tasks. We implement several proposer variants, including a heuristic UCB mutation proposer, predictor-guided task selection with a learned p_{solve} model, and an initial trainable factorized proposer updated with REINFORCE. We also add a candidate reservoir and periodic p_{solve} predictor refresh to recalibrate task difficulty as the solver improves. Empirically, the UCB self-play curriculum improves pass@16 from 74% for standard RLOO to 78%, suggesting that adaptive task generation can expand solution coverage. However, stricter predictor-based variants underperform at pass@16, revealing that hard difficulty-band admission can over-constrain the curriculum even when the predictor is refreshed. Our results suggest that self-play is promising for sparse-reward reasoning, but effective curriculum design requires balancing difficulty control with sufficient candidate diversity and admission flexibility.

1 Introduction

Large language models can produce fluent reasoning traces, but they still struggle with tasks that require systematic search rather than pattern matching. We study this issue in the Countdown arithmetic reasoning task: given a set of numbers $\{n_1, \ldots, n_k\}$ and a target value T, the model must construct an arithmetic expression that uses each number at most once and evaluates exactly to T. This task is simple to verify but difficult to solve because it requires combinatorial search over possible expression trees. Countdown therefore provides a controlled environment for studying reinforcement learning methods for improving reasoning under sparse, rule-based rewards.

The default alignment pipeline trains a solver with supervised fine-tuning, preference optimization, and online RLOO. While RLOO is well suited to Countdown because the reward can be computed by an exact verifier, it still relies on a fixed prompt distribution. This creates a curriculum mismatch. Early in training, many tasks may be too hard and yield mostly zero reward. Later in training, many tasks may become too easy and provide little useful gradient. In both cases, a fixed dataset fails to continuously expose the solver to tasks near its current learning frontier.

Stanford CS224R 2025 Final Report

Our final project focuses on this limitation. We propose a self-play curriculum for RLOO in which a task proposer continually generates new Countdown problems adapted to the current solver. Instead of treating synthetic data generation as an offline augmentation step, we use it as an online feedback loop: the solver improves on generated tasks, and the proposer uses the solver's current behavior to select future tasks. The goal is to generate tasks that are valid, solvable, novel, and currently challenging.

The central design question is how to choose useful generated tasks. Naively generating harder Countdown problems is not sufficient: invalid or unsolvable tasks waste rollouts, overly difficult tasks produce only zero rewards, and repeated similar tasks can cause curriculum collapse. We therefore select generated tasks using a joint difficulty-diversity objective. Difficulty is measured by an estimate of the current solver's probability of solving the task, $\hat{p}_{\text{solve}}(x)$, and diversity is measured relative to an archive of previously accepted tasks. This allows the curriculum to track the moving frontier of the solver while maintaining coverage over different arithmetic structures.

Our main contribution is a self-play extension to RLOO that dynamically modifies the training distribution. The proposed pipeline consists of candidate generation, validity and novelty filtering, difficulty estimation, diversity-aware task selection, replay-buffer insertion, and RLOO updates on a mixture of original and generated tasks. We compare this extension with the default SFT, IPO, and RLOO baselines, but the primary focus of the report is whether adaptive self-play improves the quality of online RL training for reasoning.

2 Related Work

RL fine-tuning for language models. Post-training LLMs often combines supervised fine-tuning with preference- or reward-based optimization. RLHF-style methods optimize models with learned or task-specific rewards (Ouyang et al., 2022), while preference optimization methods such as IPO reshape the policy distribution from pairwise preferences (Azar et al., 2024). In Countdown, we can avoid human preference labels because correctness is exactly verifiable. Our solver update is based on REINFORCE Leave-One-Out (RLOO), which reduces variance by comparing multiple samples from the same prompt and using the other samples as a baseline (Kool et al., 2019). However, standard RLOO still trains on a fixed prompt distribution. Our extension instead changes the training distribution itself through self-play.

Self-play for LLM reasoning. Self-play has recently been used for LLM self-improvement by letting models generate new tasks or interactions instead of relying only on static datasets. Multi-Agent Evolve proposes a co-evolutionary framework where agents iteratively construct tasks and refine solutions (Chen et al., 2025). Similarly, STP applies self-play to theorem proving by alternating between conjecture generation and proof search (Dong and Ma, 2025). These works motivate our proposer-solver setup, where a proposer generates new Countdown tasks and a solver learns from them. Unlike these approaches, our curriculum does not rely only on implicit agent dynamics; we explicitly select tasks according to the current solver's estimated success probability.

Curriculum difficulty and diversity. Curriculum learning aims to present examples that are useful for the learner's current ability level. For sparse-reward reasoning, tasks that are too easy provide little gradient, while tasks that are too hard mostly produce zero reward. Our self-play curriculum targets this middle region by estimating $\hat{p}_{\text{solve}}(x)$ and selecting tasks within a desired difficulty band. We also add an archive-based diversity term to avoid curriculum collapse, where the proposer repeatedly generates similar tasks. This is related to Multiagent Finetuning, which emphasizes diversity as a way to preserve broader reasoning behaviors during self-improvement (Subramaniam et al., 2025). Our method applies this idea at the task level rather than the response level.

Agentic data generation. Agentic data-generation methods construct data tailored to downstream learning needs. For example, SceneSmith uses an agentic pipeline to generate simulation-ready indoor scenes for embodied tasks (Pfaff et al., 2026). Our setting is different because the generated data are verifiable reasoning problems, and their usefulness depends on the current solver. We therefore curate generated tasks using validity, difficulty, and diversity criteria. Compared with standard RLOO, our method adds a dynamic curriculum; compared with unconstrained self-play, it explicitly controls which generated tasks enter training.

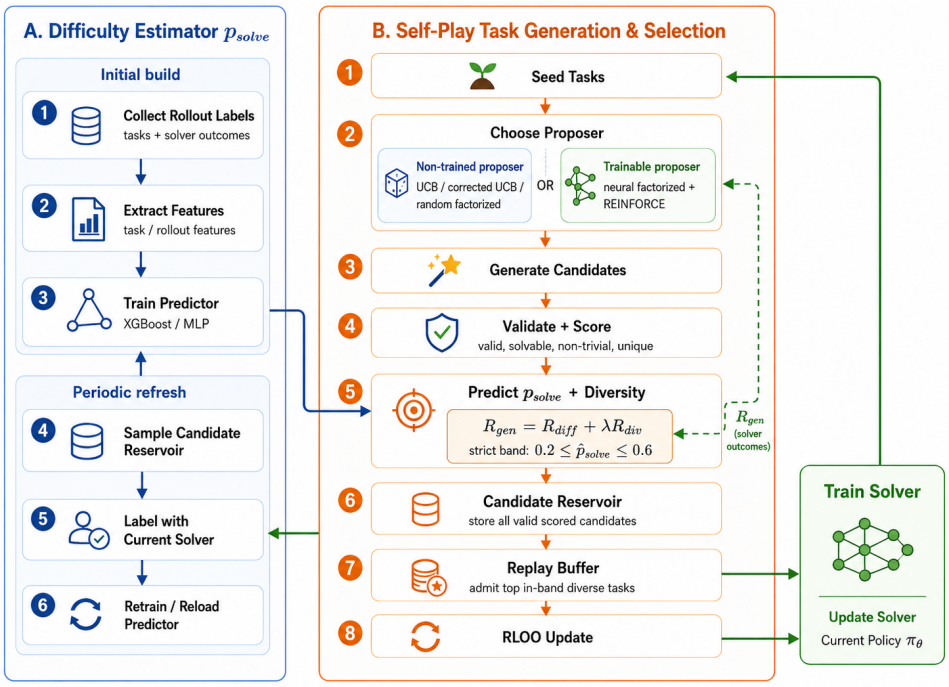

Figure 1: Overview of our self-play RLOO method. Candidate Countdown tasks are generated from seed tasks, filtered for validity and novelty, scored by estimated solver difficulty and archive-based diversity, and inserted into a replay buffer. The solver is then updated with RLOO on a mixture of original and generated tasks.

3 Method

3.1 Overview

Figure 1 summarizes our self-play RLOO pipeline. Our extension augments the default RLOO solver with a dynamic curriculum generator. The solver policy π_θ learns to solve Countdown tasks, while a proposer periodically generates new tasks near the solver's current learning frontier. Rather than generating arbitrarily hard problems, the proposer aims to produce tasks that are valid, solvable, novel, and predicted to be neither too easy nor too hard for the current solver.

At each self-play refresh, the proposer samples or mutates seed tasks, filters invalid candidates, estimates each candidate's solve probability $\hat{p}_{\text{solve}}$, and selects tasks using a joint difficulty-diversity score. The selected tasks are added to a generated replay buffer and mixed with original training examples in later RLOO updates. This turns RLOO from training on a fixed dataset into training on an adaptive curriculum.

3.2 RLOO Solver Update

For each Countdown task x, the solver samples a group of G responses:

$$y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x).$$

Each response is scored by the verifier:

$$r(y_i, x) \in \{0, 0.1, 1.0\},$$

where full reward is assigned only when the answer uses the allowed numbers correctly and evaluates to the target. RLOO computes the advantage of each response using the other samples from the same prompt as a leave-one-out baseline:

$$A_i = r(y_i, x) - \frac{1}{G-1} \sum_{j \neq i} r(y_j, x).$$

The solver is then updated with the policy-gradient objective

$$\mathcal{L}_{\text{RLOO}} = -\mathbb{E}_{x,y_i}\left[A_i \log \pi_\theta(y_i \mid x)\right] + \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}).$$

Our extension keeps this solver update unchanged. The key difference is that the prompt distribution is periodically updated by self-play.

3.3 Self-Play Task Generation and Selection

Each Countdown task is represented as

$$x = (\mathbf{n}, T),$$

where $\mathbf{n}$ is the list of input numbers and T is the target. Given seed tasks from the original data or from the generated archive, the proposer creates candidate tasks by mutating the numbers and constructing reachable targets from sampled arithmetic expression structures.

Before entering the curriculum, each candidate must pass exact symbolic checks: it must be valid, exactly solvable, non-trivial, and non-duplicated. This prevents self-play from adding invalid or unsolvable tasks that would mostly produce zero-reward rollouts.

For every valid candidate, we estimate the probability that the current solver can solve it:

$$\hat{p}_{\text{solve}}(x).$$

There are two natural ways to obtain this estimate. The rollout-based approach directly evaluates the current solver. For each generated task x, we sample $K = 8$ responses, score them with the exact Countown verifier, and compute the fraction that receive full reward, which is

$$\hat{p}_{\text{solve}}(x) = \frac{1}{K}\sum_{i=1}^{K} \mathbf{1}\{r(y_i, x) = 1.0\},$$

.

For the predictor-based variants reported in our main experiments, we instead use a learned feature-based estimator

$$\hat{p}_{\text{solve}}(x) = f_\psi(\phi(x)),$$

where $\phi(x)$ contains structural task features such as number statistics, target statistics, operator requirements, solution count, and solution-tree complexity.

We use a strict empirical difficulty band:

$$\ell \le \hat{p}_{\text{solve}}(x) \le h.$$

In our main experiments, we use $[\ell, h] = [0.2, 0.6]$. Tasks below this band are predicted to be too hard, while tasks above this band are predicted to be too easy. The difficulty reward is:

$$R_{\text{diff}}(x) = \begin{cases} 0, & \ell \le \hat{p}_{\text{solve}}(x) \le h, \\ -\left(\ell - \hat{p}_{\text{solve}}(x)\right), & \hat{p}_{\text{solve}}(x) < \ell, \\ -\left(\hat{p}_{\text{solve}}(x) - h\right), & \hat{p}_{\text{solve}}(x) > h. \end{cases}$$

To avoid curriculum collapse, we also maintain an archive $\mathcal{A}$ of accepted generated tasks and compute an archive-based diversity reward:

$$R_{\text{div}}(x) = \min_{a \in \mathcal{A}} d(\phi_{\text{tree}}(x), \phi_{\text{tree}}(a)),$$

where ϕ_{tree} summarizes solution-tree structure and operator usage. The final generation score is

$$R_{\text{gen}}(x) = R_{\text{diff}}(x) + \lambda R_{\text{div}}(x).$$

A task is admitted only if it lies in the predicted difficulty band and satisfies a minimum diversity threshold. Among admitted candidates, we select the top tasks by R_{gen} and insert them into the generated replay buffer $\mathcal{B}_{\text{gen}}$. RLOO batches are then sampled from a mixture of original training tasks and generated tasks.

3.4 Trainable Proposer and Periodic Difficulty Refresh

In addition to the original UCB mutation proposer, we implement a trainable factorized symbolic proposer. Instead of directly generating a full task with an LLM, the proposer chooses symbolic mutation actions from a factorized action space:

$$a = (a_{\text{num}}, a_{\text{expr}}),$$

where a_{num} controls how the input numbers are changed and a_{expr} controls what type of arithmetic expression is used to construct the target. The number mutation strategies include local small changes, local large changes, replacement of one number, and full resampling. The expression strategies include addition/subtraction-only targets, targets requiring multiplication, targets requiring division, and multi-step nontrivial expressions.

The neural proposer parameterizes a policy

$$q_\eta(a \mid s),$$

where the state s contains no-leak features of the seed task and recent self-play statistics such as acceptance rate, archive size, and selected-task difficulty. It is trained with a REINFORCE-style objective using the continuous generation reward:

$$\mathcal{L}_{\text{prop}} = -\mathbb{E}_{a \sim q_\eta} \left[(R_{\text{gen}} - b) \log q_\eta(a \mid s) \right] - \alpha H(q_\eta),$$

where b is a moving-average baseline and α controls entropy regularization. This allows the proposer to learn which symbolic mutation strategies tend to produce useful frontier tasks.

A second issue is that the learned p_{solve} predictor can become stale as the solver improves. To address this, we implement periodic predictor refresh. During self-play, we store all valid predictor-scored candidates in a candidate reservoir, including accepted, too-easy, and too-hard tasks. Every fixed number of self-play refreshes, we sample candidates from this reservoir using a stratified scheme over predicted difficulty regions:

$$\hat{p}_{\text{solve}} < \ell, \qquad \ell \le \hat{p}_{\text{solve}} \le h, \qquad \hat{p}_{\text{solve}} > h.$$

These sampled candidates are labeled with rollouts from the current solver to obtain empirical solve rates. The new labels are appended to an online dataset, and the XGBoost p_{solve} predictor is retrained and reloaded. This refresh mechanism helps the curriculum track the solver's moving difficulty frontier instead of relying on a fixed predictor trained before RLOO.

3.5 Algorithm

Algorithm 1 summarizes the full training loop. The solver is updated with RLOO, while the proposer and the difficulty predictor are periodically updated to improve curriculum quality.

4 Experimental Setup

We evaluate our methods on the Countdown arithmetic reasoning task. Each test example consists of a set of input numbers and a target value, and the model must generate an arithmetic expression that uses the given numbers correctly and evaluates exactly to the target. Following the default project setup, we use a rule-based verifier to compute correctness. A response is counted as correct only if it contains a valid expression in the required answer format and the expression evaluates to the target using the allowed numbers.

Our main evaluation metric is pass@k. For each test prompt, we sample up to 16 completions from the model and report whether at least one of the first k samples is correct. We evaluate on a held-out Countdown test set of 50 problems, with 16 samples per problem. This metric is useful for separating two types of improvement: pass@1 measures whether the model's highest-probability behavior becomes more reliable, while larger pass@k values measure whether the model has broader solution coverage under sampling.

We compare the following methods:

- **SFT**: supervised fine-tuning on Countdown demonstrations. This serves as the warm-started reference policy.

Algorithm 1 Self-Play RLOO with Trainable Proposer and Predictor Refresh

Initialize solver π_θ from the SFT checkpoint.
Initialize generated buffer $\mathcal{B}_{\text{gen}}$, archive $\mathcal{A}$, proposer q_η, and predictor f_ψ.
for training step $t = 1, \ldots, T$ **do**
  if $t \bmod M = 0$ **then**
    Sample seed tasks from original data and archive $\mathcal{A}$.
    Generate candidate tasks using the proposer.
    Filter candidates for validity, solvability, non-triviality, and uniqueness.
    Predict $\hat{p}_{\text{solve}}(x)$ for every valid candidate.
    Compute $R_{\text{diff}}(x)$, $R_{\text{div}}(x)$, and $R_{\text{gen}}(x)$.
    Add valid scored candidates to the candidate reservoir.
    Admit candidates satisfying the difficulty band and diversity gate.
    Add top admitted candidates to $\mathcal{B}_{\text{gen}}$ and $\mathcal{A}$.
    Update the trainable proposer using continuous reward R_{gen}.
    if predictor refresh interval is reached **then**
      Sample reservoir candidates across too-hard, in-band, and too-easy regions.
      Label sampled candidates with current solver rollouts.
      Retrain and reload the p_{solve} predictor.
    end if
  end if
  Sample a mixed batch from original data and $\mathcal{B}_{\text{gen}}$.
  Sample G responses per task from π_θ.
  Score responses with the Countdown verifier.
  Compute leave-one-out advantages.
  Update π_θ with the RLOO objective.
end for

Hyperparameter	Value
Learning rate	1×10^{-5}
Batch size	128
Group size G	8
Training steps	100
KL coefficient	0.001
Entropy coefficient	0.001
Self-play refresh interval	10 steps
Generated-task ratio	25%

Table 1: RLOO self-play training hyperparameters.

- **IPO**: preference optimization using the default pairwise preference setup.
- **RLOO**: online policy-gradient optimization with a verifier-based reward and leave-one-out advantage estimation. No self-play; trains on the fixed task distribution.
- **RLOO + Self-Play (UCB)**: augments RLOO with a UCB bandit proposer that mutates seed tasks and selects candidates by ranking all validity-passing candidates by R_{gen}, without a hard difficulty band or diversity gate. All valid candidates are eligible for selection.
- **RLOO + Self-Play (P-Solver)**: replaces the UCB proposer with a learned XGBoost $\hat{p}_{\text{solve}}$ predictor and enforces a strict difficulty band $[0.2, 0.6]$ and minimum novelty threshold. Candidates outside the band are hard-rejected before ranking. The predictor is trained once on RLOO-era rollouts and never refreshed.
- **RLOO + Self-Play (Refresh)**: extends P-Solver with periodic predictor refresh (every 5 self-play rounds, the XGBoost predictor is retrained on fresh online rollouts sampled from the candidate reservoir) and a trainable factorized neural proposer updated via REINFORCE.
- **RLOO + Self-Play (Refresh+Diversity)**: extends Refresh by replacing the 7-dimensional solution-tree-only novelty feature used in the archive diversity gate with a 13-dimensional feature that additionally includes normalized input numbers and target, eliminating spurious duplicate rejections caused by feature-space collisions.

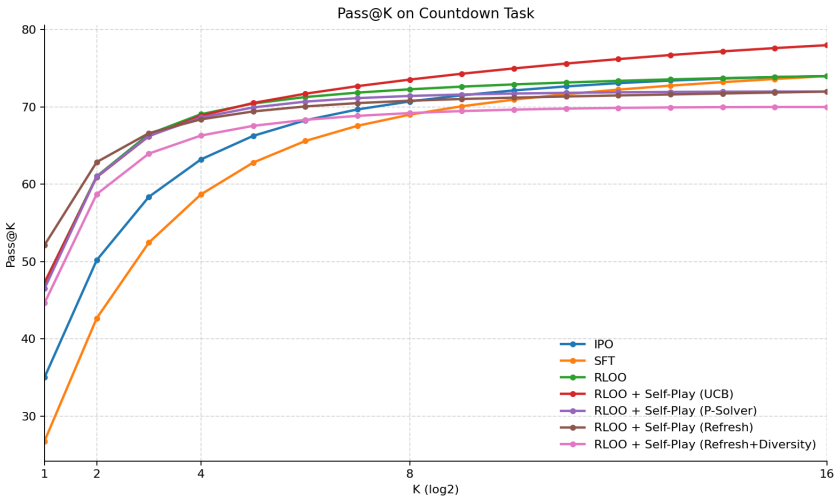

Figure 2: Pass@k on the Countdown test set with 50 problems and 16 samples per problem. RLOO improves pass@1 over SFT and IPO. Among self-play variants, UCB achieves the best pass@16 (78%). Predictor-based variants (P-Solver, Refresh, Refresh+Diversity) score 70–72% at pass@16, below the RLOO baseline, with the diversity-gate fix raising candidate admission rate but not final performance within 100 training steps.

For the self-play extension, the solver is initialized from the same SFT checkpoint as the RLOO baseline. Every 10 RLOO steps, the proposer refreshes the generated task buffer by mutating seed Countdown tasks, filtering candidates for validity and novelty, estimating candidate difficulty, and selecting tasks according to the joint difficulty-diversity objective. Selected generated tasks are added to a replay buffer and mixed with original training tasks for subsequent RLOO updates. The main RLOO self-play hyperparameters are shown in Table 4. We use group size $G = 8$, 100 training steps, KL coefficient 0.001, entropy coefficient 0.001, and a generated-task ratio of 25%.

After validity filtering (range bounds, exact solvability, non-triviality, and within-round deduplication), approximately 65–70% of raw proposals reach the scoring stage; the remaining candidates are then scored for difficulty and diversity before admission. We analyze the filtering breakdown in detail in Section 5.1.

5 Results

Figure 2 shows pass@k on the Countdown test set. Overall, online RL improves over the SFT baseline. Among our self-play variants, RLOO + Self-Play (UCB) achieves the strongest pass@16 coverage (78%), while the predictor-based variants (P-Solver, Refresh, Refresh+Diversity) score 70–72%, at or below the RLOO baseline of 74%. This suggests that UCB does not merely make the model more confident in the same high-probability solutions; instead, it expands the set of problems the model can solve when allowed to sample multiple attempts.

The results show three main trends. First, IPO improves pass@1 over SFT by shifting probability mass toward preferred solutions, but it does not substantially improve pass@16. This indicates that preference optimization mainly sharpens the model's existing solution distribution rather than expanding the range of problems it can solve. Second, standard RLOO substantially improves pass@1, reaching 47%, which confirms that online verifier-based RL is effective for Countdown. However, RLOO plateaus at 74% pass@16, suggesting that training on the fixed task distribution still leaves some parts of the test distribution uncovered. Third, RLOO + Self-Play (UCB) reaches the same pass@1 as standard RLOO (46%) but improves pass@16 to 78%, consistent with our hypothesis that generated frontier tasks improve coverage. However, the predictor-based variants (P-Solver, Refresh, Refresh+Diversity) do not improve over RLOO at pass@16, scoring 70–72%, suggesting that difficulty estimation quality is a critical bottleneck.

Method	pass@1	pass@16
SFT	27%	74%
IPO	35%	72%
RLOO	46%	74%
RLOO + Self-Play (UCB)	46%	**78%**
RLOO + Self-Play (P-Solver)	46%	72%
RLOO + Self-Play (Refresh)	**52%**	72%
RLOO + Self-Play (Refresh+Diversity)	46%	70%

Table 2: Performance comparison on the Countdown test set.

Method	Admitted	Too Hard	Too Easy	Low Diversity
RLOO + Self-Play (UCB)	∼100%	0%	0%	0%
RLOO + Self-Play (P-Solver/Refresh)	44.7%	7.1%	20.4%	27.8%
RLOO + Self-Play (Refresh+Diversity)	68.2%	3.7%	26.4%	1.7%

captionCandidate filtering breakdown per self-play variant. Percentages are of scored candidates. UCB imposes no hard rejection gate. All valid candidates are eligible for top-k selection. Predictor-based variants hard-reject candidates outside the predicted difficulty band $[0.2, 0.6]$ before ranking.

5.1 Quantitative Evaluation

Table 2 summarizes the main quantitative results. Because the final project focuses on our extension, we emphasize the comparison between standard RLOO and RLOO Self-Play. RLOO + Self-Play (UCB) matches RLOO at pass@1 (46%) and improves pass@16 by 4 percentage points (74% $\rightarrow$ 78%). Notably, Self-Play (Refresh) achieves the highest pass@1 at 52%, but does not translate this into improved pass@16 coverage (72%). This difference is meaningful because pass@16 measures whether the model has learned a broader search distribution, not just whether the first sampled answer is correct.

The improvement from RLOO to RLOO + Self-Play (UCB) is concentrated at larger k. This pattern supports the interpretation that the self-play curriculum increases diversity in the model's solution attempts. If self-play only improved local preference for already-discovered solutions, we would expect the largest gain at pass@1. Instead, the gain appears at pass@16, which suggests that training on generated valid, novel, and currently challenging tasks helps the model discover additional reasoning strategies.

The predictor-based variants (P-Solver, Refresh, Refresh+Diversity) all underperform standard RLOO. We interpret this as evidence that difficulty estimation is a sensitive component of the self-play pipeline. A learned p_{solve} predictor can score candidate tasks cheaply, but it can become miscalibrated as the solver improves during training, causing the curriculum to select tasks that are no longer near the true difficulty frontier.

Figure 4 shows two concrete failure modes that explain this gap. The left panel tracks R_{div}, the novelty of each selected task relative to the archive of previously accepted tasks (Section 3.3). In Self-Play (Refresh), R_{div} collapses to near-zero by step 30 and remains there: the 7-dimensional solution-tree feature used to measure distance is too coarse, so many genuinely distinct tasks map to identical feature vectors and are spuriously rejected as duplicates. The result is that the curriculum repeatedly selects tasks with the same arithmetic structure, providing little variety in the gradient signal and limiting the model's ability to discover new solution strategies. Refresh+Diversity fixes this by expanding the novelty feature to 13 dimensions (adding normalized numbers and target), which restores meaningful diversity throughout training and raises admission from 44.7% to 68.2%. Table 5.1 and Figure 3 break down rejection reasons across variants.

However, this structural fix alone does not recover pass@16: even with more diverse tasks, the strict difficulty band still constrains the curriculum to a narrow region, and 100 training steps may be insufficient for the benefit of a larger, more varied curriculum to compound. The right panel confirms the predictor refresh is mechanistically working: the spike in minimum predicted $\hat{p}_{\text{solve}}$ at steps 60–80 shows the refreshed predictor recalibrating to reflect the improved policy, shifting the difficulty estimates upward as the solver gets stronger. Yet even correct difficulty calibration does not

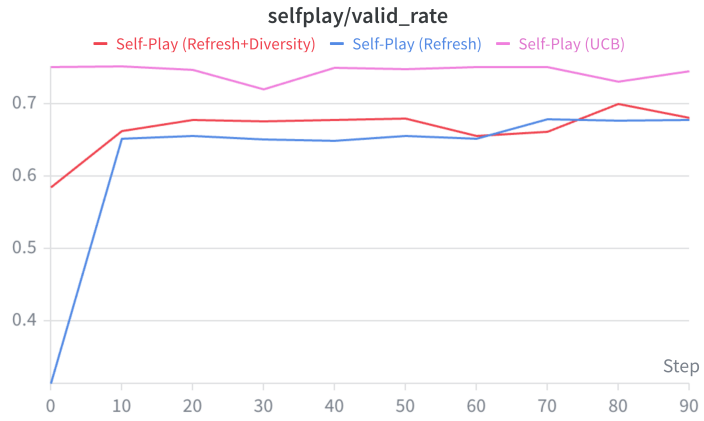

Figure 3: Candidate filtering breakdown across self-play variants. Approximately 65–70% of raw proposals pass validity and deduplication. UCB admits all remaining scored candidates (no hard band or diversity gate); predictor-based variants admit 44.7% before the novelty fix. Refresh+Diversity raises admission to 68.2% by eliminating spurious low-diversity rejections.

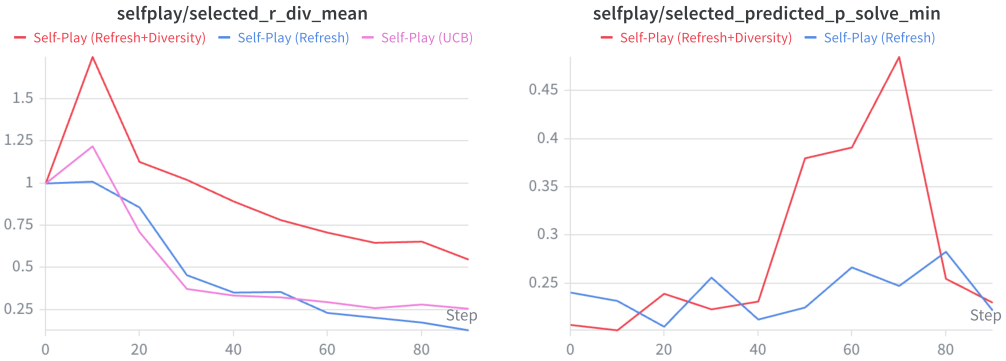

Figure 4: Self-play diagnostics across training steps. **Left**: mean diversity score (R_{div}) of selected tasks. Self-Play (Refresh) collapses to near-zero diversity by step 30 due to feature-space collisions in the 7-dim novelty metric; Refresh+Diversity maintains meaningful diversity throughout. **Right**: minimum predicted $\hat{p}_{\text{solve}}$ among selected tasks for predictor-based variants. The spike in Refresh+Diversity corresponds to the periodic predictor refresh recalibrating to the improved policy.

close the gap, suggesting the predictor-based admission mode itself, rather than its calibration quality, may be the bottleneck relative to the simpler UCB approach.

5.2 Qualitative Analysis

The quantitative results suggest that self-play mainly improves coverage rather than greedy accuracy. Standard RLOO already improves pass@1 by directly optimizing the verifier reward on sampled completions. However, because it trains on a fixed prompt distribution, its exploration is limited by the original dataset. Once the model improves on many fixed prompts, additional updates may repeatedly reinforce similar solution patterns.

The self-play curriculum changes this behavior by continually adding new tasks that are selected to be valid, novel, and near the solver's current capability frontier. These generated tasks expose the model to arithmetic structures that may be underrepresented in the original data, such as tasks requiring different operator combinations or more complex expression trees. As a result, the solver learns a broader distribution over possible reasoning paths, which is reflected in the pass@16 improvement seen for RLOO + Self-Play (UCB).

The comparison with predictor-based variants (P-Solver, Refresh, Refresh+Diversity) highlights a limitation of the current implementation. Although a learned difficulty predictor is attractive because

it avoids expensive rollout-based scoring, it must track a moving target: the solver's ability changes throughout RLOO training. If the predictor is trained only once or refreshed too infrequently, it may overestimate or underestimate candidate difficulty. This can cause the curriculum to admit tasks that are too easy, too hard, or not informative for the current solver. Therefore, our results suggest that the self-play framework is promising, but curriculum quality depends strongly on accurate online difficulty calibration.

Overall, the results support our main hypothesis: adaptive self-play is useful not simply because it creates more data, but because it creates a better training distribution for sparse-reward RL. By selecting generated tasks according to difficulty and diversity without over-constraining the admission gate, RLOO + Self-Play (UCB) improves the solver's coverage of the Countdown task space beyond what standard RLOO achieves on a fixed dataset.

6 Discussion

Why UCB outperforms predictor-based admission. The most striking result is that a simpler UCB bandit proposer outperforms more elaborate predictor-based variants across all pass@16 numbers. We believe the key factor is the admission strategy, not predictor quality. UCB scores all validity-passing candidates by R_{gen} and takes the top k with no hard rejection: every candidate that survives validity filtering is eligible to be selected. Predictor-based variants, by contrast, enforce a strict difficulty band $[0.2, 0.6]$: any candidate whose predicted solve probability falls outside this range is discarded before ranking, regardless of its diversity or rank (Table 5.1). This hard gate reduces the effective candidate pool by more than half. A miscalibrated predictor then has compounding effects: candidates near the true frontier may be incorrectly rejected as too easy or too hard, and the surviving pool is systematically biased toward whatever the predictor happens to score as in-band. UCB avoids this failure mode entirely by never discarding candidates based on a potentially stale difficulty estimate.

Mechanistic fixes work, but the bottleneck is structural. Our diagnostic analysis identified two failure modes in the predictor-based variants. First, the 7-dimensional novelty feature caused 43.7% of candidates to be spuriously rejected as duplicates. Expanding to 13 dimensions (Refresh+Diversity) eliminated this: low-diversity rejections fell from 27.8% to 1.7%, and admission rose from 44.7% to 68.2%. Second, the predictor refresh confirmed mechanistic correctness: the spike in minimum $\hat{p}_{\text{solve}}$ at steps 60–80 shows the refreshed predictor correctly tracking the improving solver. Yet neither fix recovered pass@16. This suggests the root bottleneck is the strict band admission mode itself: even a well-calibrated predictor that rejects too many candidates leaves the model training on a narrower slice of the task space than UCB's permissive top-k.

Limitations and future directions. Several factors limit our conclusions. Our evaluation uses only 50 test problems with 16 samples each, making the pass@k curves noisy; very small differences may not be statistically robust. All self-play variants run for only 100 training steps, which may be too short for predictor-based curriculum benefits to compound. Longer runs with periodic predictor refresh could allow the curriculum to track a substantially improved solver and potentially recover the gap. The p_{solve} predictor relies on handcrafted structural features, which may miss factors that determine true solver difficulty; replacing it with a rollout-based oracle or a learned embedding would improve calibration at the cost of more compute. The most direct improvement to the admission design would be replacing the hard difficulty band with a soft penalty, which would combine the permissiveness of UCB with the calibration signal of the predictor and avoid the compounding effects of hard rejection. Finally, learned novelty features from a task encoder trained to separate distinct arithmetic structures would provide a more reliable diversity signal than the current 13-dimensional handcrafted feature.

7 Conclusion

We proposed a self-play curriculum for RLOO training on Countdown arithmetic reasoning, where a task proposer continuously generates problems adapted to the current solver's difficulty frontier. Our experiments show that online self-play can meaningfully expand a model's solution coverage beyond what standard RLOO achieves on a fixed dataset. However, the benefit is sensitive to curriculum design: a permissive selection strategy that avoids hard rejection gates proves more robust in the short-training regime than a tightly gated predictor-based approach, even when the predictor is periodically recalibrated. Diagnostic analysis reveals that hard admission gates, when combined with

a potentially stale difficulty estimator, can starve the curriculum of useful candidates in ways that are difficult to recover from within a limited training budget. These findings point toward softer, more adaptive admission designs as a promising direction for making predictor-guided self-play more reliable in practice.

8 Team Contributions

- **Jenny Jin:** Implemented and trained rollout-based p_{solve} estimation for self-play difficulty calibration using direct solver rollouts.
- **Jiaming Shen:** Implemented the mutation proposer, the p_{solve} predictor for difficulty estimation, and the initial trainable factorized proposer.
- **Jiaxin Fang:** Implemented the UCB bandit proposer, the archive-based diversity gate, periodic predictor refresh, and the trainable factorized neural proposer.

Changes from Proposal The core idea from our proposal remains the same: we use self-play to generate Countdown tasks that are both challenging and diverse, and then train the solver with RLOO on this adaptive curriculum. Compared with the proposal, our final implementation makes the proposer more concrete and controllable. Instead of a free-form task generator, we use symbolic proposers that mutate numbers and construct reachable targets, including both non-trained variants and a trainable factorized proposer trained with REINFORCE. We also changed task selection from simple ranking to strict predicted-band admission, where generated tasks must satisfy $0.2 \leq \hat{p}_{\text{solve}}(x) \leq 0.6$ and pass a diversity threshold before entering the replay buffer. Finally, we added a candidate reservoir and periodic p_{solve} predictor refresh, so the difficulty estimator can be recalibrated as the solver improves during training.

References

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4447–4455.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. 2025. Multi-Agent Evolve: LLM Self-Improve through Co-evolution. *arXiv preprint arXiv:2510.23595* (2025).

Kefan Dong and Tengyu Ma. 2025. STP: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving. *arXiv preprint arXiv:2502.00212* (2025).

Wouter Kool, Herke van Hoof, and Max Welling. 2019. Buy 4 reinforce samples, get a baseline for free! (2019).

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. 2026. SceneSmith: Agentic Generation of Simulation-Ready Indoor Scenes. *arXiv preprint arXiv:2602.09153* (2026).

Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. 2025. Multiagent Finetuning: Self Improvement with Diverse Reasoning Chains. *arXiv preprint arXiv:2501.05707* (2025).