Extended Abstract

Adversarial Co-evolution of a Difficulty Judge and an LLM Generator with an RLOO Policy for Cold-Start Countdown Reasoning

Viveak Ravichandiran vravicha@stanford.edu

Motivation. RL-based recommenders fail in the *cold-start* regime: new users and items have no interaction history, so the reward signal that policy-gradient methods depend on is essentially absent, and alignment tends to overfit to well-represented ("warm") cases. I study this failure mode in a controlled, verifiable setting by re-casting the CS224R Countdown arithmetic task as a cold-start proxy: frequent 3-number prompts act as *warm* cases and rare 4-number prompts as *cold* cases. The central question is whether a co-evolving curriculum can shrink the warm–cold accuracy gap without sacrificing warm performance.

Method. On top of the default SFT $\rightarrow$ IPO $\rightarrow$ RLOO pipeline (Qwen2.5-0.5B), I build three cold-start extensions that share a cold-augmented training distribution (75% 4-number prompts). **(S4) Static Augmentation** mixes a fixed pool of 50k synthetically generated 4-number problems into RLOO. **(S5) Co-Evolution** adds a *DifficultyJudge* that, every 25 steps, measures per-prompt solve rates over a rolling window and replaces the synthetic cold pool with the specific cold prompts the policy currently fails, keeping the curriculum at the model's frontier. **(S6) LLM-Gen** replaces the rule-based generator with a Qwen2.5-Instruct model that *proposes* new 4-number problems; every proposal is filtered by the exact rule-based verifier so only solvable problems enter training (LLM generator $\rightarrow$ symbolic verifier $\rightarrow$ solver). All systems are evaluated with the verifier (a perfect, free reward checker), which I also exploit at test time.

Implementation. I implement SFT, IPO, and RLOO from scratch with vLLM sampling and Ray-based GPU orchestration, plus the DifficultyJudge, the synthetic and LLM generators, and the verifier-filtered curriculum. All systems are evaluated identically (50 problems, 16 samples, τ=0.6) using the unbiased Pass@k estimator, reported separately on warm and cold subsets.

Results. RL alignment *widens* the warm–cold gap (SFT 0.151 $\rightarrow$ RLOO 0.453 at Pass@1), the direct analogue of popularity bias. My core ablation supports adaptivity: **Co-Evolve (S5) outperforms Static Augmentation (S4) in this experiment** (Pass@1 0.564 vs. 0.432; cold 0.363 vs. 0.226). However, neither augmentation beats a well-trained plain RLOO@250 (0.621, cold 0.404): adding a cold curriculum on top of an already-strong policy trades overall accuracy for gap reduction. Tuning the curriculum dose (lower cold ratio, stronger KL anchor) recovers most of this cost, reaching 0.591, within bootstrap noise of RLOO@250. **LLM-Gen (S6) *destabilizes* RLOO**: Pass@1 drops to 0.343 while Pass@16 is preserved (0.760); the model retains its solutions but loses single-sample sharpness. A training-duration control shows this is not generic over-training (plain RLOO improves $0.515 \rightarrow 0.621$ from step 100 to 250). Finally, because Countdown is verifiable, a best-of-k decoder reaches 0.82 accuracy at k=16 with no extra training.

Discussion. The verifier-filtered LLM generator appears to hurt learning despite guaranteeing per-problem correctness, suggesting the bottleneck at 0.5B scale is curriculum *stability* under RLOO, not problem validity. A smaller gap is not automatically good: LLM-Gen attains the smallest gap (0.243) by degrading warm accuracy, illustrating a degenerate way to "close" it.

Conclusion. Adaptive, frontier-targeted curricula perform better than static oversampling in this experiment, but cold augmentation does not surpass strong on-policy RL, and an LLM problem generator further destabilizes training at small scale. The verifiable-reward setting that makes Countdown a clean cold-start proxy also yields large, free test-time-compute gains.

Adversarial Co-evolution of a Difficulty Judge and an LLM Generator with an RLOO Policy for Cold-Start Countdown Reasoning

Viveak Ravichandiran
Stanford University
vravicha@stanford.edu

Abstract

I study the cold-start problem of RL-based personalization through the lens of the CS224R Countdown arithmetic task, treating rare 4-number prompts as cold cases and frequent 3-number prompts as warm cases. On top of the default SFT/IPO/RLOO pipeline for Qwen2.5-0.5B, I introduce three cold-start extensions: static synthetic augmentation, an adaptive co-evolutionary curriculum driven by a DifficultyJudge, and a verifier-filtered LLM problem generator. I find that RL alignment widens the warm–cold gap (a popularity-bias analogue), that the adaptive curriculum outperforms static augmentation in this experiment, but that no cold-augmentation strategy beats a well-trained plain RLOO policy (though careful curriculum dosing closes the gap to within statistical noise), and that an LLM generator destabilizes RLOO at this scale (Pass@1 collapses while Pass@16 is preserved). I further show that exploiting the verifier at test time yields large, training-free accuracy gains. My contribution is a controlled, verifiable study of when co-evolutionary curricula help and when they hurt.

1 Introduction

Reinforcement learning from verifiable or preference rewards is now standard for post-training language models, but it inherits a structural weakness from recommender systems: the *cold-start* problem. New users or items have little or no interaction history, so the reward signal that policy-gradient methods rely on is sparse or absent, and training overfits to well-represented cases [9, 11]. The same pathology appears in reasoning RL: prompts whose solution patterns are rare in the training distribution receive little effective gradient and lag behind common ones.

I study this failure mode in a fully controlled, verifiable setting. The CS224R Countdown task asks a model to combine a set of numbers with $+, -, \times, \div$ to reach a target; correctness is checked by a rule-based verifier. I re-cast Countdown as a cold-start proxy: **warm** prompts are frequent 3-number instances and **cold** prompts are rare 4-number instances with larger search spaces. This isolates the cold-start phenomenon while removing the noise of a learned reward model: the verifier is a ground-truth oracle, so any warm–cold gap reflects optimization, not reward error.

I ask three questions. **(RQ1)** Does standard alignment (SFT, IPO, RLOO) widen or narrow the warm–cold gap? **(RQ2)** Can a co-evolutionary curriculum, in which a judge continually re-targets training toward the cold prompts the policy currently fails, reduce the gap better than static synthetic augmentation or plain RLOO? **(RQ3)** Does replacing the rule-based problem generator with an *LLM* generator (the verifiable analogue of an LLM user simulator [19, 9]) further help? I answer RQ1 affirmatively (alignment widens the gap), give a nuanced answer to RQ2 (adaptive outperforms

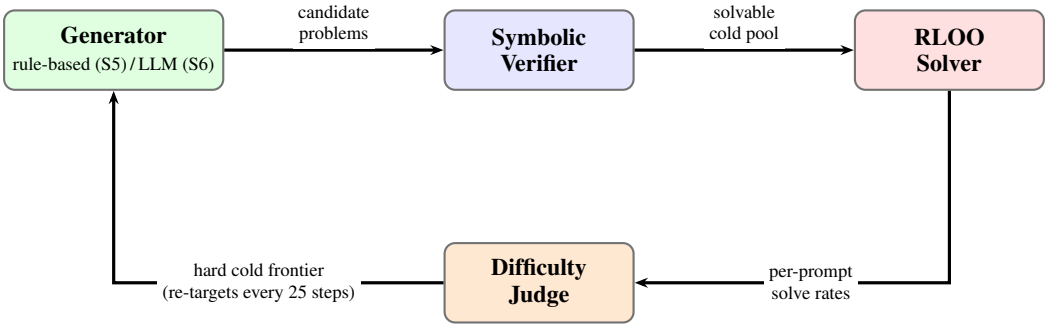

Figure 1: Co-evolutionary loop. A generator proposes 4-number (cold) problems; a symbolic verifier keeps only solvable ones; the RLOO solver trains on the cold-augmented pool; the DifficultyJudge measures solve rates and re-targets the generator at the prompts the solver currently fails. S4 removes the Judge (static pool); S5 uses a rule-based generator; S6 uses an LLM generator.

static, but neither beats strong plain RLOO), and a cautionary answer to RQ3 (the LLM generator destabilizes RLOO at 0.5B scale). I additionally show that the verifier enables large, free test-time-compute gains.

My central and most counter-intuitive finding is a *validity-is-not-sufficiency* effect: even though every LLM-generated problem is verifier-certified solvable, training on these correct-by-construction problems destabilizes RLOO, isolating curriculum *stability*, rather than data correctness, as the binding constraint at small scale. To my knowledge this distinction has not been isolated in prior work on LLM data generation, user simulators, or self-play, which generally assume that valid synthetic data helps; surfacing it in a fully verifiable setting, where validity can be guaranteed rather than estimated, is the main contribution of this study.

2 Related Work

Cold-start recommendation with LLMs. ColdLLM [9] uses an LLM simulator to synthesize surrogate feedback for new items, and Li et al. [11] show reasoning-based LLM fine-tuning improves cold-start ranking. Both confirm that LLMs can produce plausible surrogate signal where real interactions are absent, but their simulators are *fixed* after training and do not adapt as the policy improves. My co-evolution loop instead re-targets the generator at the policy's current failures every few steps.

Self-play and co-evolution. SPRec [7] uses a model's own prior outputs as negatives to debias recommendation, and AFL [2] co-trains a recommender and a user agent with supervised signals. Self-play has driven strong reasoning gains in theorem proving [5] and multi-agent finetuning [16]. Unlike SPRec, my adversarial signal comes from a *distinct* judge/generator rather than the policy's own samples; unlike AFL, my loop is driven by an RLOO policy-gradient objective with a verifiable reward.

Curriculum learning. Easy-to-hard curricula [13] and solve-rate-adaptive curricula [3] improve reasoning RL by keeping tasks at the edge of learnability, echoing the "stepping-stone" generation of DeepSeek-R1 [4]. My DifficultyJudge is an instance of this principle specialized to the cold/warm split and implemented with a verifiable reward.

RL fine-tuning and test-time compute. I build on RLOO [1] and IPO [8] (cf. DPO [14]), the Countdown/SoS formulation [6], and the TinyZero reward scheme [12]. My test-time-compute analysis follows Snell et al. [15] and the generative-verifier view [18]; uniquely, Countdown admits a *perfect* verifier, so best-of-k is an exact accuracy.

3 Method

3.1 Cold-start proxy and baselines

I partition Countdown into **warm** (3-number) and **cold** (4-number) prompts; cold prompts have a strictly larger combinatorial search space and are rarer in the base distribution, mirroring tail users/items. The default pipeline provides three baselines, all initialized from the official SFT checkpoint of Qwen2.5-0.5B: **SFT** (next-token loss on completions), **IPO** [8] on preference pairs, and **RLOO** [1], a REINFORCE-leave-one-out policy gradient with the rule-based verifier reward (1.0 correct, 0.1 format-only, 0.0 wrong) [12], an entropy bonus, and a KL penalty to the SFT reference. I sample $K{=}8$ rollouts per prompt with vLLM [10] and apply per-sample importance weighting to correct for sampler/trainer log-prob mismatch. Formally, with policy π_θ and frozen SFT reference π_{ref}, the three default-pipeline objectives are

$$\mathcal{L}_{\text{SFT}}(\theta) = -\,\mathbb{E}_{(x,y)\sim\mathcal{D}} \sum_{t=1}^{|y|} \log \pi_\theta(y_t \mid x, y_{<t}), \tag{1}$$

$$\mathcal{L}_{\text{IPO}}(\theta) = \mathbb{E}_{(x,y_w,y_l)}\Big(h_\theta - \tfrac{1}{2\beta}\Big)^2, \quad h_\theta = \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}, \tag{2}$$

$$\mathcal{L}_{\text{RLOO}}(\theta) = -\sum_{j=1}^{K} \text{sg}(w_j A_j)\, \log \pi_\theta(y_j \mid x) \;-\; \alpha_H\, \mathcal{H}[\pi_\theta(\cdot \mid x)] \;+\; \alpha_{\text{KL}}\, \text{KL}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big), \tag{3}$$

where the leave-one-out advantage and the importance weight (behavior sampler μ vs. target π_θ) are

$$A_j = r_j - \frac{1}{K-1}\sum_{i\neq j} r_i, \qquad w_j = \exp\big(\log \pi_\theta(y_j \mid x) - \log \mu(y_j \mid x)\big), \tag{4}$$

$r_j \in \{0, 0.1, 1.0\}$ is the verifier reward, sg denotes stop-gradient, β the IPO regularizer, and w_j is clipped to a maximum for stability.

3.2 S4: Static Augmentation

I oversample cold prompts to 75% of each batch and add a fixed pool of 50,000 synthetic 4-number problems. Synthetic problems are guaranteed valid by construction: I enumerate reachable targets for random number tuples and keep only solvable (number, target) pairs. This tests whether simply *seeing* more cold problems closes the gap.

3.3 S5: Co-Evolutionary Curriculum (DifficultyJudge)

Static augmentation is non-adaptive: as the solver improves, a fixed pool becomes stale. S5 adds a **DifficultyJudge** that tracks, for each training prompt, the fraction of its K rollouts that are fully correct, averaged over a rolling window of 5 steps. Every 25 steps it flags cold prompts whose windowed solve rate is below a threshold (0.3) and *replaces the synthetic cold pool* with these hard, real cold prompts (hardest first), keeping the curriculum pinned to the solver's current frontier (Fig. 1). The solver is initialized from RLOO@250 and trained for a further 150 steps. This realizes an adaptive, RL-based analogue of an adaptive user simulator [2, 3] under a verifiable reward. Formally, let $r_i^{(\tau)}(p) \in \{0, 0.1, 1.0\}$ be the reward of rollout i for prompt p at step τ. The DifficultyJudge maintains a windowed solve rate $\bar{s}_t(p)$ and, at update step t, the hard-cold frontier $\mathcal{H}_t$:

$$\bar{s}_t(p) = \frac{1}{W}\sum_{\tau=t-W+1}^{t} \frac{1}{K}\sum_{i=1}^{K} \mathbb{1}\big[\, r_i^{(\tau)}(p) = 1.0 \,\big], \qquad \mathcal{H}_t = \big\{\, p : \text{cold}(p) \,\wedge\, \bar{s}_t(p) < \tau_{\text{hard}} \,\big\}, \tag{5}$$

with window $W{=}5$ and hardness threshold $\tau_{\text{hard}}{=}0.3$. Every $C{=}25$ steps the synthetic cold pool $\mathcal{P}_{\text{cold}}$ is replaced by $\mathcal{H}_t$ (sorted hardest-first), and each training batch $\mathcal{B}$ is drawn with a fixed cold fraction ρ:

$$\mathcal{B} \sim \rho\, \mathcal{P}_{\text{cold}} \;\oplus\; (1-\rho)\, \mathcal{P}_{\text{warm}}, \qquad \rho = 0.75. \tag{6}$$

```
Algorithm:  Verifier-filtered co-evolutionary curriculum (S5/S6)
init solver π from RLOO@250; cold_pool ← generate_and_verify(N)
for step t = 1...T:
   batch ← 25% warm ⊕ 75% cold (from cold_pool)
   rollouts, rewards ← sample_and_score(π, batch)          // vLLM + verifier
   judge.update(prompt → solve_rate over window w=5)
   π ← RLOO_update(π, rollouts, rewards)
   if t mod 25 = 0:
     hard ← judge.hard_cold(threshold = 0.3)
     cand ← generator.propose(hard)            // rule-based (S5) or LLM (S6)
     cold_pool ← {c ∈ cand :  verifier.solvable(c)}  // correctness guarantee
```

Figure 2: The curriculum loop. S4 omits lines 5 and 7–10 (static pool).

3.4 S6: LLM Problem Generator (LLM-Gen)

S5's generator is heuristic. S6 replaces it with a Qwen2.5-Instruct model that is *prompted* with a few examples of the current hard cold frontier and asked to propose new 4-number problems at matching difficulty, the verifiable analogue of an LLM user simulator [19, 9]. Because LLMs readily emit unsolvable problems, *every* proposal is passed through the same symbolic verifier used for the reward; only solvable problems enter the pool, and I bucket them by the number of reaching expressions as a difficulty proxy. The pipeline is therefore **LLM generator** $\rightarrow$ **symbolic verifier** $\rightarrow$ **solver**: the LLM supplies novelty and the verifier guarantees correctness. The generated pool is a drop-in replacement for S4/S5's cold pool, so the training loop is otherwise unchanged. Concretely, conditioning the generator on the current frontier $\mathcal{H}_t$,

$$\mathcal{P}_{\text{cold}}^{\text{LLM}} = \{\, p \sim \text{LLM}(\mathcal{H}_t) \;:\; \text{solvable}(p) \,\}, \qquad d(p) = \big|\{\, e \in \mathcal{E}(p.\text{nums}) : \text{eval}(e) = p.\text{target} \,\}\big|, \tag{7}$$

where $\mathcal{E}(\cdot)$ enumerates left-to-right arithmetic expressions over the four numbers, $\text{solvable}(p) \iff d(p) > 0$ is the verifier filter, and the solution count $d(p)$ is the difficulty proxy used for bucketing.

3.5 Verifier-guided test-time compute

Because the Countdown verifier is exact, a deployable best-of-k decoder can return the first sample that provably hits the target; its accuracy is exactly Pass@k. I therefore report Pass@k both as a diversity diagnostic and as a real test-time-compute curve [15].

4 Experimental Setup

Model and data. All systems use Qwen2.5-0.5B [17] initialized from the official SFT checkpoint; RLOO/IPO use the course Countdown datasets. **RLOO hyperparameters:** AdamW, lr 10^{-5} (constant), weight decay 10^{-4}, batch 128 prompts, group size K=8, entropy coefficient 0.001, KL coefficient 0.001, gradient clipping 1.0, training temperature 1.0 (top-p=1, top-k disabled), up to 250 steps. **Curriculum:** cold ratio 0.75, judge window 5, judge threshold 0.3, pool update every 25 steps; S6 uses a Qwen2.5-Instruct generator with verifier filtering. **Evaluation.** Following the course protocol, I sample 16 responses per prompt at τ=0.6, top-p=0.95, top-k=20 on 50 held-out problems (24 warm, 26 cold) and report the unbiased estimator Pass@$k = \mathbb{E}_x[1 - \binom{n-c}{k} / \binom{n}{k}]$ with n=16, separately on warm and cold subsets. I define Gap@1 = Warm@1 $-$ Cold@1. The verifier follows the TinyZero scheme [12].

5 Results

5.1 Quantitative evaluation

Table 1 and Figure 3 summarize all systems.

(RQ1) Alignment widens the warm–cold gap. The gap grows monotonically with alignment strength at Pass@1: SFT 0.151 $\rightarrow$ IPO 0.305 $\rightarrow$ RLOO@250 0.453 (Fig. 4). RLOO maximizes the

Table 1: Pass@k and warm/cold breakdown (50 problems, 16 samples, τ=0.6). Gap@1 = Warm@1 − Cold@1. **Bold** = best in column among RL systems. Pass@1 bootstrap 95% CI half-widths (5000 resamples): SFT ±.06, IPO ±.08, RLOO ±.10, S4 ±.09, S5 ±.10, S5′ ±.10, S6 ±.07; see Fig. 4 for per-bar intervals.

Method	Pass@1	Pass@4	Pass@8	Pass@16	Warm@1	Cold@1	Gap@1
SFT	0.286	0.622	0.737	0.800	0.365	0.214	**0.151**
IPO	0.370	0.661	0.727	0.760	0.529	0.224	0.305
RLOO@250	**0.621**	**0.799**	**0.827**	**0.840**	**0.857**	**0.404**	0.453
RLOO + Static Aug (S4)	0.432	0.687	0.740	0.760	0.656	0.226	0.430
RLOO + Co-Evolve (S5)	0.564	0.763	0.798	0.820	0.781	0.363	0.418
+ dose-tuned (S5′)	0.591	0.787	0.814	0.820	0.810	0.389	0.420
RLOO + LLM-Gen (S6)	0.343	0.641	0.714	0.760	0.469	0.226	0.243

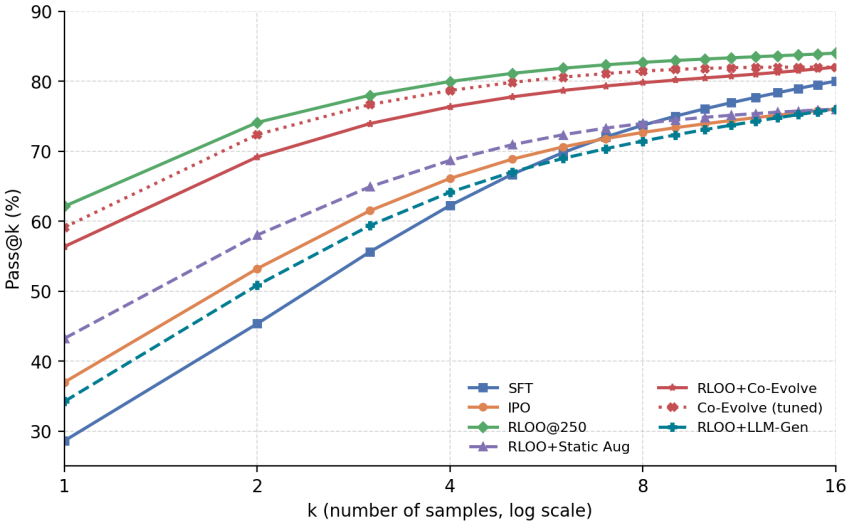

Figure 3: Pass@k for all six systems. Plain RLOO@250 attains the highest Pass@k; both cold-augmentation variants (Static Aug, Co-Evolve) and LLM-Gen trade single-sample accuracy for coverage and converge near k=16.

verifier reward, but reward mass concentrates on warm prompts, a direct analogue of popularity bias in recommenders, here arising purely from optimization since the reward is exact.

(RQ2) Adaptive outperforms static, but neither beats strong RLOO. The core S4-vs-S5 ablation is a key comparison: **Co-Evolve outperforms Static Augmentation in this experiment** (Pass@1 0.564 vs. 0.432; cold 0.363 vs. 0.226; warm 0.781 vs. 0.656). Adaptively re-targeting the curriculum at the frontier avoids the "flooding" failure of static oversampling, which depresses warm learning. However, *neither* augmentation surpasses a well-trained plain RLOO@250 (Pass@1 0.621, cold 0.404): because Co-Evolve is initialized from RLOO@250 and then trained on a 75%-cold distribution, the cold curriculum trades ∼6 points of overall accuracy for gap reduction. The hypothesis that co-evolution beats *static augmentation* is supported by these results; the stronger claim that it beats *standard RLOO* is not.

Curriculum dose-response. The 75%-cold ratio, not the curriculum itself, is the culprit. Re-running Co-Evolve from RLOO@250 with a gentler dose (40% cold, KL coefficient 0.02, lr 5×10^{-6}, 100 steps) recovers warm accuracy (0.781→0.810) and lifts cold (0.363→0.389), raising overall Pass@1 to 0.591 (Table 1, row S5′) and closing the gap to plain RLOO@250 from 0.057 to 0.030. The curriculum cost is therefore real but largely *tunable*: aggressive cold oversampling pays for cold coverage with warm forgetting, while a KL-anchored low dose nearly eliminates that tax. As I show next, the residual 0.030 gap is within bootstrap noise.

Table 2: Training-duration control for plain RLOO. More steps help the baseline, making generic over-training less likely as the explanation for S6's lower accuracy.

Method	Pass@1	Warm@1	Cold@1
RLOO@100	0.515	0.727	0.320
RLOO@250	0.621	0.857	0.404

Table 3: Multi-seed Pass@1 (mean $\pm$ std over three independent vLLM decoding seeds) for the four core RL systems. Decoding variance is small; this isolates sampling randomness and is complementary to the wider bootstrap intervals (Table 1), which capture test-set variance.

Method	Pass@1	Warm@1	Cold@1
RLOO@250	0.617 ± 0.003	0.839 ± 0.003	0.412 ± 0.003
RLOO + Co-Evolve (S5)	0.560 ± 0.001	0.763 ± 0.003	0.373 ± 0.001
+ dose-tuned (S5′)	0.599 ± 0.001	0.812 ± 0.005	0.402 ± 0.005
RLOO + LLM-Gen (S6)	0.341 ± 0.003	0.425 ± 0.011	0.264 ± 0.007

(RQ3) The LLM generator destabilizes RLOO. Despite the verifier guaranteeing that every generated problem is solvable, LLM-Gen is the weakest RL system (Pass@1 0.343), regressing below even Static Aug. The failure is diagnostic rather than catastrophic: **Pass@16 is preserved** (0.760, equal to Static Aug) while only Pass@1 collapses, and LLM-Gen has by far the steepest best-of-k slope (Fig. 5). The model still *contains* correct solutions; it has lost single-sample sharpness, the canonical signature of RLOO mode-flattening. Table 2 makes the simplest confound less likely: plain RLOO *improves* from step 100 (0.515) to step 250 (0.621), so 250 steps is not inherently too many. The instability is specific to training on the LLM-generated curriculum at 0.5B scale.

Test-time compute is large and free. Every system's Pass@16 vastly exceeds its Pass@1, and all curves converge near k=16 (Fig. 3): the models *know* solutions but need several attempts on cold prompts. Since the verifier is exact, a best-of-k decoder realizes these gains at deployment with no extra training; Co-Evolve reaches 0.82 at k=16 (Fig. 5).

Statistical significance. With only 50 test problems, per-prompt bootstrap 95% confidence intervals on Pass@1 (5000 resamples) are wide: RLOO@250 $0.621\,[0.52, 0.72]$, dose-tuned Co-Evolve $0.591\,[0.50, 0.69]$, Co-Evolve $0.564\,[0.46, 0.67]$, Static Aug $0.432\,[0.34, 0.52]$, LLM-Gen $0.343\,[0.27, 0.42]$. The strongest systems' intervals overlap substantially, so RLOO@250's edge over the dose-tuned curriculum is *within noise*. The differences I treat as robust are the large ones (Static Aug vs. Co-Evolve, and any RL system vs. LLM-Gen), together with the directional trend (the warm–cold gap widening with alignment) that holds for *every* system. I therefore frame conclusions in terms of these robust effects rather than small Pass@1 deltas, and verify decoding robustness with a multi-seed evaluation next.

Multi-seed robustness. To go beyond a single evaluation I re-ran the four core RL systems at three independent vLLM decoding seeds (Table 3). Decoding-seed standard deviation is small (≤ 0.011 on Pass@1), the means agree with the representative-seed numbers of Table 1 to within 0.02, and the system ranking is unchanged. The dose-tuned curriculum's cold accuracy ($0.402 \pm .005$) nearly matches plain RLOO@250 ($0.412 \pm .003$), and its overall Pass@1 ($0.599 \pm .001$) trails it by under two points. Crucially, this small decoding variance does *not* shrink the bootstrap intervals above: the dominant source of uncertainty is the 50-problem test set, not sampling randomness, so the strongest systems remain statistically overlapping. Training-seed variance (a fresh RL run per system) is the clearest remaining robustness check and is left to future work.

5.2 Qualitative analysis

Co-evolution dynamics (Fig. 4, right). The DifficultyJudge behaves as intended: the cold solve rate dips to ≈ 0.28 as the policy explores, then climbs to ≈ 0.35 after the first pool update at step 25, while warm solve rate holds near 0.82 throughout; the KL anchor to RLOO@250 prevents forgetting even as 75% of the batch is cold.

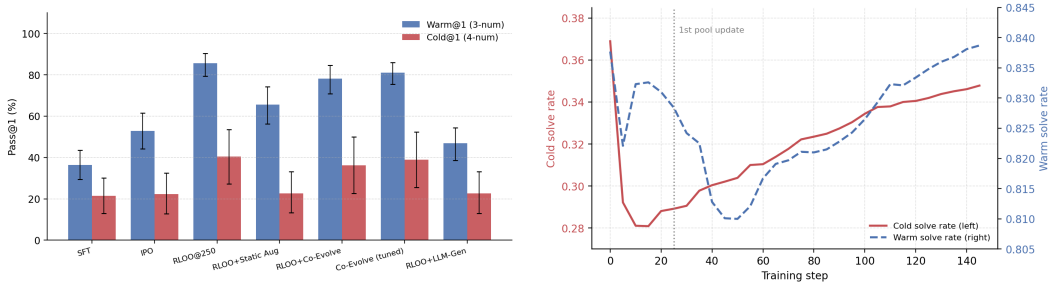

Figure 4: Left: Warm vs. cold Pass@1 with bootstrap 95% CI error bars (5000 resamples over the 50-problem test set). The gap widens through alignment; Co-Evolve restores *both* warm and cold relative to Static Aug; LLM-Gen "closes" the gap only by degrading warm. Overlapping intervals among the strongest systems indicate the top-end Pass@1 differences are within sampling noise. Right: Co-Evolve training dynamics: after the first pool update (step 25), the DifficultyJudge lifts the cold solve rate ($0.28 \rightarrow 0.35$) while the warm solve rate holds near 0.82 (no forgetting).

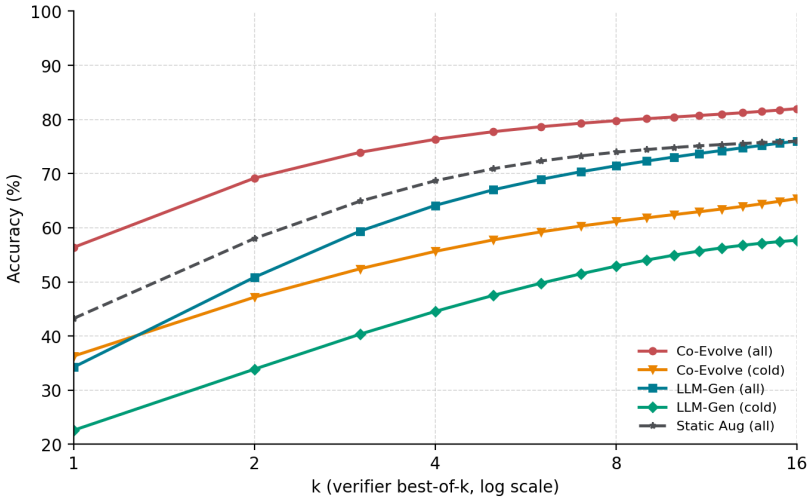

Figure 5: Verifier-guided test-time compute (best-of-k = Pass@k). LLM-Gen (S6) starts lowest but has the steepest slope, recovering to match Static Aug by k=16, evidence that S6 lost single-sample sharpness, not knowledge.

A degenerate way to close the gap. LLM-Gen attains the *smallest* Gap@1 (0.243) of any system (Table 1), but this is an artifact: its cold accuracy (0.226) is identical to Static Aug, and the gap shrinks only because warm accuracy *fell* to 0.469. This is a concrete illustration that gap reduction is not a sufficient objective: a model can equalize warm and cold by getting worse at both.

Cold failure taxonomy. Table 4 decomposes every cold rollout into correct / format-only-but-wrong-arithmetic / no-answer-or-invalid. Across all RL systems the dominant cold failure is a *well-formed equation with wrong arithmetic* (55–76%), not a parsing or formatting error: the strong systems emit a valid `<answer>` essentially always (no-answer $\leq 1\%$) yet the underlying search is wrong. This localizes the cold bottleneck to arithmetic *search*, not output format, and explains why test-time sampling helps so much (Fig. 5): a different random search frequently succeeds. Static Aug and LLM-Gen produce the most confident-but-wrong cold equations (73%, 76%), mirroring their depressed single-sample accuracy.

Universally hard cold instances. Six 4-number problems are solved by *no* system at any of the 16 samples, e.g. `[85,45,69,74]` $\rightarrow$`94`, `[26,71,58,38]` $\rightarrow$`40`, and `[51,61,91,48]` $\rightarrow$`44`. These share large operands and non-round targets that require multiplicative or division steps deep in the search tree, exactly the cold structure already flagged as hardest at the SFT stage, indicating a capability ceiling at 0.5B scale that curriculum alone does not breach.

Table 4: Composition of cold (4-number) rollouts: fraction correct (1.0), format-only with wrong arithmetic (0.1), and no-answer/invalid (0.0). The cold bottleneck is wrong arithmetic, not formatting.

Method	Correct	Format-only (wrong arith.)	No-answer / invalid
SFT	0.21	0.69	0.09
IPO	0.22	0.55	0.22
RLOO@250	0.40	0.59	0.01
Static Aug (S4)	0.23	0.73	0.05
Co-Evolve (S5)	0.36	0.62	0.01
dose-tuned (S5′)	0.39	0.61	0.00
LLM-Gen (S6)	0.23	0.76	0.01

Sample rollouts. The models exhibit explicit search-and-verify behavior. On a warm prompt, `[44, 19, 35]` $\rightarrow$ `98`, Co-Evolve produces `<think> 44 + 19 = 63; 63 + 35 = 98 </think> <answer> (44 + 19) + 35 </answer>` (reward 1.0). On a cold prompt the policy backtracks: for `[63, 95, 96]` $\rightarrow$ `64` it tries `96 - 63 = 33; 33 * 95` (too big), then `63 + 96 = 159; 159 - 95 = 64` (reward 1.0). LLM-Gen's typical cold failure is a well-formatted but arithmetically wrong first sample whose later samples include the correct equation, consistent with its preserved Pass@16.

6 Discussion

Validity is not sufficiency. A notable result is that a generator whose every output is verifier-certified solvable still appears to degrade learning. This points to the bottleneck: at 0.5B scale the limiting factor appears to be the *stability* of RLOO under a shifting, hard, LLM-shaped distribution, rather than the correctness of the problems. **Limitations.** The test set is small (50 problems) and each system was trained with a *single* training seed; I therefore report bootstrap 95% confidence intervals over the evaluation set (Table 1, Fig. 4) and three-seed decoding means for the core RL systems (Table 3), which together quantify problem- and decoding-level variance but not training-seed variance. Because these intervals are wide ($\pm.06$ to $\pm.10$ on Pass@1) and overlap among the strongest systems, I base every claim on effects that exceed them (the adaptive-vs-static gap, the LLM-Gen collapse, and the monotone widening of the warm–cold gap with alignment) rather than on small Pass@1 deltas; multi-seed training is the clearest next step. The cleanest S6 control, evaluating S6 at the same 150 steps as S5 rather than 250, was not completed under the deadline, so duration and generator are partially entangled for S6 (Table 2 controls only the plain-RLOO baseline). The LLM generator (1.5B-Instruct) is also a different model family than the 0.5B solver, which may shift the cold distribution off-policy. **Broader impact.** The cold-start framing transfers to recommendation and personalization under data scarcity; my finding that adaptive curricula help but LLM-simulated data can destabilize small-model RL is a useful caution for LLM-simulator pipelines [9, 19].

7 Conclusion

On a fully verifiable cold-start proxy, RL alignment widens the warm–cold gap; an adaptive co-evolutionary curriculum (DifficultyJudge) outperforms static synthetic augmentation in these experiments but does not surpass a strong plain RLOO policy; and replacing the rule-based generator with a verifier-filtered LLM generator destabilizes RLOO at 0.5B scale, collapsing single-sample accuracy while preserving coverage. The primary take-home is a *validity-is-not-sufficiency* principle: in a setting where every generated problem is provably correct, that correctness is still not enough for a useful curriculum, because verifier-certified-correct LLM data can destabilize on-policy RL. The binding constraint at small scale is therefore training stability rather than data validity, a caution for the fast-growing class of LLM-simulator and synthetic-data pipelines that assume valid data is helpful data. Secondary take-homes are that *how* cold data is selected matters more than *how much* of it there is, that gap reduction must be paired with absolute accuracy to be meaningful, and that a perfect verifier yields large, free test-time-compute gains. Future work: match S6 to S5's training budget, vary generator scale and on-policyness, and add a verifier reranker or generative verifier [18] to convert the preserved Pass@16 into deployable single-shot accuracy.

8 Team Contributions

- **Viveak Ravichandiran:** 100% of the work: problem formulation; implementation of SFT, IPO, and RLOO (vLLM sampling, Ray orchestration, importance weighting); the DifficultyJudge, synthetic and LLM generators, verifier-filtered curriculum, and evaluation pipeline; all experiments, figures, analysis, and writing.

Use of AI tools (extension only): AI assistance was used solely on the extension (the DifficultyJudge, the synthetic and LLM generators, and the plotting and evaluation utilities), while the core SFT, IPO, and RLOO implementation and all experimental results are the author's own.

Changes from Proposal. The proposal framed the extension as an LLM "user simulator" co-evolving with the policy. I realized this in two stages: a rule-based adaptive curriculum (Co-Evolve, the strongest extension) and a verifier-filtered LLM generator (LLM-Gen, the proposal's literal LLM simulator). The LLM variant produced a mixed/negative result, which I report and analyze rather than discard, consistent with the project's emphasis on methodology over peak score.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

[2] Shihao Cai, Jizhi Zhang, Keqin Bao, Chongming Gao, Qifan Wang, Fuli Feng, and Xiangnan He. Agentic feedback loop modeling improves recommendation and user simulation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2025.

[3] Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*, 2025.

[4] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[5] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

[6] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

[7] Chongming Gao, Ruijun Chen, Shuai Yuan, Kexin Huang, Yuanqing Yu, and Xiangnan He. Sprec: Self-play to debias llm-based recommendation. In *Proceedings of the ACM Web Conference (WWW)*, 2025.

[8] Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences. *arXiv preprint arXiv:2310.12036*, 2023.

[9] Feiran Huang, Yuanchen Bei, Zhenghang Yang, Junyi Jiang, Hao Chen, Qijie Shen, Senzhang Wang, Fakhri Karray, and Philip S. Yu. Large language model simulator for cold-start recommendation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining (WSDM)*, 2025.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

[11] Shijun Li, Yu Wang, Jin Wang, Ying Li, Joydeep Ghosh, and Anne Cocos. Llm reasoning for cold-start item recommendation. In *Proceedings of the ACM Web Conference (WWW)*, 2026.

[12] Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. `https://github.com/Jiayi-Pan/TinyZero`, 2025.

[13] Shubham Parashar et al. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. *arXiv preprint arXiv:2506.06632*, 2025.

[14] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[15] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[16] Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. *arXiv preprint arXiv:2501.05707*, 2025.

[17] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[18] Lunjun Zhang, Hritik Bansal, Mehran Kazemi, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[19] Zijian Zhang, Shuchang Liu, Ziru Liu, Rui Zhong, Qingpeng Cai, Xiangyu Zhao, Chunxu Zhang, Qidong Liu, and Peng Jiang. Llm-powered user simulator for recommender system. *arXiv preprint arXiv:2412.16984*, 2024.

A Reproducibility

Figures are produced by `Final_report/make_report_figures.py` from the canonical eval files in `downloaded_eval_results/`. The LLM generator, verifier filter, and difficulty bucketing are in `extension/llm_problem_generator.py`; the DifficultyJudge in `extension/difficulty_judge.py`; the curriculum dataset in `rloo_trainer/rloo_dataset.py`.

B Additional Experiments: Self-Consistency

As a verifier-free alternative to the best-of-k decoder of Figure 5, I evaluate *self-consistency*: the majority-voted final answer across the 16 samples (MV@16). Table 5 compares single-sample Pass@1, MV@16, and the verifier best-of-16 (Pass@16). Majority vote recovers a large share of the Pass@1-to-Pass@16 gap *without* querying the verifier at inference, and every system benefits, but it stays below the verifier best-of-16 because it cannot exploit ground-truth checkability (e.g. Co-Evolve: $0.564 \rightarrow 0.780$ with MV vs. 0.820 with the verifier). The across-system ordering is preserved, consistent with the main results.

Table 5: Verifier-free self-consistency (MV@16) vs. single-sample Pass@1 and verifier best-of-16 (Pass@16), over the same 50 problems and 16 samples. Majority vote helps every system but trails the verifier.

Method	Pass@1	MV@16	Pass@16
SFT	0.286	0.720	0.800
IPO	0.370	0.720	0.760
RLOO@250	0.621	0.800	0.840
Static Aug (S4)	0.432	0.720	0.760
Co-Evolve (S5)	0.564	0.780	0.820
LLM-Gen (S6)	0.343	0.700	0.760

C Implementation Details

RL orchestration. Sampling and updates alternate via Ray actors: a vLLM sampling worker and a PyTorch (Hugging Face) update worker are created and torn down each step, with GPU memory explicitly freed (`tear_down` before `kill`) between transitions to avoid out-of-memory failures when reloading weights. Checkpoints are written every 25 steps with resume support (checkpoint path plus a start-step offset).

Importance weighting. Because rollouts are sampled with vLLM but scored under the Hugging Face policy, I compute a per-sequence log-space ratio w_j (Eq. 4) between update-time and sampling-time log-probabilities and clip it to a maximum before multiplying the RLOO advantage, which stabilizes the off-policy correction.

Verifier and difficulty proxy. Solvability and the difficulty count $d(p)$ (Eq. 7) enumerate left-to-right expressions $((a \circ b) \circ c) \circ d$ over the four numbers with integer-only operations (no division by zero, no non-integer division). This matches the reward's arithmetic and is *stricter* than full parenthesization, so accepted problems are guaranteed solvable (no false positives) at the cost of occasionally rejecting parenthesization-only solvable ones.

LLM generator. The S6 generator (Qwen2.5-1.5B-Instruct) is prompted with up to eight current frontier examples; outputs are parsed by regex into (numbers, target) tuples, range-filtered to $[1, 99]$, de-duplicated, passed through the verifier, and bucketed by solution count (hard ≤ 2, medium ≤ 8, easy otherwise). Generation runs offline and writes a drop-in cold pool.

Compute. Training and evaluation run on Modal (H100/A100). Evaluation uses τ=0.6, top-p=0.95, top-k=20, and 16 samples per prompt; multi-seed evaluation (Table 3) varies the vLLM sampling seed.