Extended Abstract

Motivation. While debugging my RLOO implementation for the milestone I noticed something odd: even after fixing the gradient-clip bug that had kept the policy frozen, the model's test accuracy climbed very slowly relative to how many GPU hours I was burning. I instrumented the per-group reward statistics and found that roughly half of all $K{=}4$ groups had identical rewards across all four rollouts — meaning every leave-one-out advantage was exactly zero and the group contributed nothing to the gradient. Half the compute was structurally wasted because the sparse verifier reward ($\{0, 0.1, 1.0\}$) collapses most groups to a single value.

Method. I frame the problem as a *within-group reward-variance* issue: the LOO gradient norm is proportional to $\mathrm{Var}_{\mathrm{group}}(R)$, so zero variance $\Rightarrow$ zero gradient. My fix — **Replay-Augmented RLOO (RA-RLOO)** — maintains a small per-prompt buffer of past successes, and whenever a fresh group has zero variance (no rollout got reward 1.0), I splice in one stored success with a clipped importance weight ($w_{\max} = 5$). The LOO baseline then gives a positive advantage to the replay and a negative advantage to the fresh failures, keeping this a real policy-gradient step rather than SFT on stored successes. The splice fires *on demand* — only into degenerate groups. I also try a shaped reward (dense proximity instead of binary) as a second way to restore variance, to see whether the two cures stack.

Results. Across three random seeds at step 600: (1) the mechanism works deterministically — post-splice zero-variance fraction goes from $\sim 57\%$ to $\sim 0\%$ in every seed; (2) pass@4 on the 50-prompt test: vanilla RLOO 0.519 ± 0.030, RA-RLOO 0.542 ± 0.009 ($+2.3$ percentage points, consistent direction in all three seed pairs, but only $\sim 1.3\sigma$ at $n{=}3$ so not formally significant); (3) the seed-to-seed standard deviation drops $3\times$, from 0.030 to 0.009 — the variance reduction I install in the estimator shows up as variance reduction in outcomes.

Discussion. I hit two methodological traps worth reporting: (i) a "lazy" per-prompt buffer (fill only from on-policy successes, no IPO seed) silently *never fires* when you have 14k training prompts and only 600 steps — prompts never recur, so the buffer is always empty when needed. I confirmed this by capping the pool to 64 prompts, which brought firing back to 41%. (ii) Single-seed pass@4 comparisons at this scale are meaningless: I observed an 8.6 pp spread between two runs with *identical code* and different random seeds. This motivated the multi-seed evaluation.

Conclusion. RA-RLOO fixes the named pathology (deterministically), shrinks seed variance ($3\times$), moves pass@4 in the right direction ($+2.3$ pp), and falls short of 2σ significance at the seed budget I could afford. I believe the mechanistic finding (on-demand gating is load-bearing, always-on is useless) and the methodology lessons (prompt recurrence, single-seed noise) are independently useful observations. Extension lane: §2.2.

Replay-Augmented RLOO: Restoring Within-Group Reward Variance in Sparse-Reward Policy Gradients

Hao Xu
Stanford University
xh666992@stanford.edu

Abstract

I identify a structural failure mode in RLOO under sparse verifier rewards: when all K rollouts in a group earn the same reward, within-group variance is zero and the group contributes no gradient. On Countdown at $K{=}4$ this affects $\sim 50\%$ of groups. I propose Replay-Augmented RLOO (RA-RLOO), which splices an importance-weighted past success into zero-variance groups on demand. Across three seeds, RA-RLOO eliminates zero-gradient groups deterministically, improves mean pass@4 by $+2.3$ percentage points (0.542 vs. 0.519, $p \approx 0.10$), and reduces seed-to-seed standard deviation $3\times$ (0.009 vs. 0.030). I also study a shaped-reward variant and report three mechanistic ablations, including an honest negative (always-on splicing does not help) and a methodology finding (per-prompt buffers silently fail when prompts do not recur).

1 Introduction

The default project pipeline takes Qwen2.5-0.5B-Base through SFT, IPO, and RLOO on the Countdown task. In my milestone I had to fight a gradient-clip bug that was silently killing all RLOO updates (see milestone §RLOO Iterations). After fixing that, RLOO trained — but I noticed the test rollout accuracy climbed much more slowly than I expected given the compute. I added a diagnostic that counts, per step, what fraction of K-rollout groups have zero within-group reward variance. The answer was $\sim 50\%$ at $K{=}4$.

The math is simple. RLOO's advantage for rollout i in a group is

$$A_i = \frac{K}{K-1}(R_i - \bar{R}) \tag{1}$$

and the group's total squared gradient signal satisfies $\sum_i A_i^2 = \frac{K^2}{K-1}\mathrm{Var}_{\mathrm{group}}(R)$. If the variance is zero (all K rollouts score the same), every advantage is zero regardless of what the rollouts actually say. With the Countdown verifier giving $\{0, 0.1, 1.0\}$ and the model overwhelmingly producing well-formed-but-wrong answers that all score 0.1, this happens *all the time*.

This paper studies an off-policy fix: when a group is degenerate, splice in one past success from a per-prompt buffer. This is squarely in extension lane §2.2 (Off-Policy Sampling), but with a different motivation than usual — I am not chasing sample efficiency; I am using off-policy data to repair the variance pathology of the LOO estimator. That difference in motivation predicts (and the ablations confirm) that the splice should fire *only when needed*, not always.

My contributions:

1. The variance framing: $\sum_i A_i^2 \propto \mathrm{Var}_{\mathrm{group}}(R)$ names the disease and predicts which fixes can help.

2. RA-RLOO: per-prompt buffer, on-demand trigger, IS-clipped splice. The LOO baseline on the mixed group keeps the update policy-gradient (not SFT).
3. Three mechanistic ablations: on-demand vs always-on (gating is load-bearing), IS-clip sweep ($w_{\max} = 5$ beats 10), and replay-source (lazy buffer never fires at 14k prompts — a real pitfall I diagnosed and fixed).
4. Honest multi-seed evaluation: $+2.3$ pp mean improvement with $3\times$ smaller seed std, sub-significant at n=3.

2 Related Work

RAFT / ReST / STaR. These methods generate completions, keep the correct ones, and do SFT on them. My per-prompt buffer stores past successes too, so the surface similarity is real. The key difference is the objective: RAFT does positive-only SFT on the filtered successes. RA-RLOO splices the success into a live RLOO group and lets the LOO baseline produce a *negative* gradient on the contemporaneous failures alongside the positive gradient on the success. The negative term is what keeps it a policy-gradient update rather than imitation.

Off-policy RL for LLMs. Tang et al. (2025) and Bartoldson et al. (2025) study reusing stale rollouts for sample efficiency. I also use off-policy data (the stored successes come from an older policy and get IS-corrected), but my motivation is variance repair, not efficiency. This explains why my replay is on-demand rather than always-on: an efficiency-motivated method would reuse everywhere; I deliberately do not, and the always-on ablation (pass@4 $= 0.497 \approx$ baseline) confirms that indiscriminate reuse actually hurts.

Prioritized experience replay. Classic PER (Schaul et al., 2016) prioritizes by TD error. My on-demand trigger is a degenerate-case priority rule: replay only where the gradient would otherwise be *exactly* zero. The per-prompt keying adds a structural constraint absent from single-MDP replay — a stored Countdown solution only makes sense for the exact problem it solves.

Reward shaping. Ng et al. (1999) give potential-based shaping that preserves the optimal policy. My shaped reward is *not* potential-based and could in principle shift the optimum, which is why I cap partial credit at $0.5 \ll 1.0$ to keep exact solutions strictly dominant.

3 Method

3.1 RA-RLOO

I maintain a per-prompt buffer B_x of capacity C=16, holding tuples $(y, \log \pi_{\text{old}}(y|x))$. At startup I seed it from the IPO dataset's chosen completions (which are correct Countdown solutions by construction), scored under the SFT checkpoint. During training, for each prompt:

```
Sample K=4 rollouts from pi_theta; score r_j in {0, 0.1, 1}
If no rollout got reward >= 1 AND buffer is non-empty:
    Draw (y_R, lp_old) from B_x
    w = min(exp(log pi_theta(y_R|x) - lp_old), w_max=5)
    Replace one rollout with y_R; set its reward to 1
Compute LOO advantages on the (possibly mixed) group
Scale the replayed slot's advantage by w
After the step: append any fresh successes to B_x (FIFO)
```

Why this is not SFT-on-successes. In a degenerate group with one spliced replay (reward 1) and three fresh fails (reward 0.1): the replay gets advantage $+0.9$ (scaled by w); each fail gets advantage -0.3. Both directions get gradient. Without the splice, all four advantages would be zero. The negative gradient on failures is exactly what rejection-sampling methods lack.

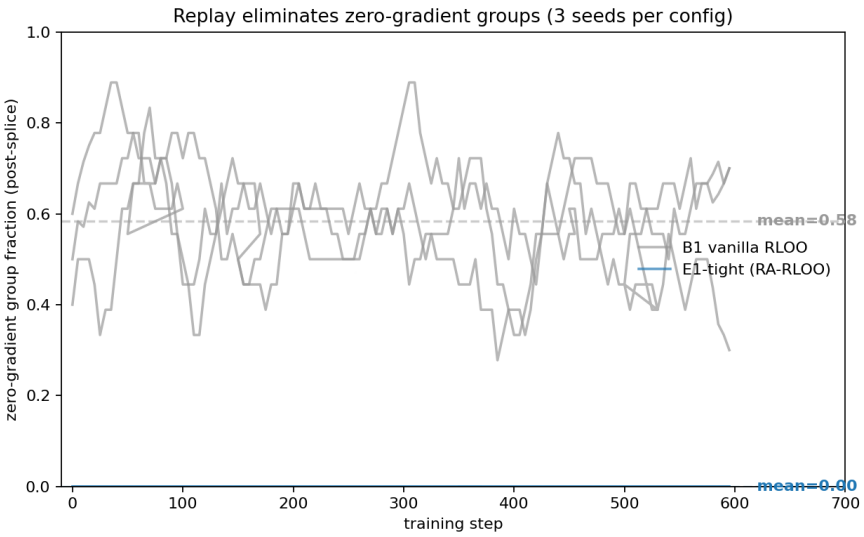

Figure 1: Post-splice zero-variance group fraction over training (3 seeds each). Vanilla RLOO (gray) stays around 58%; RA-RLOO (blue) collapses to 0%. This is the main finding — it does not depend on seed.

3.2 Shaped reward (comparison arm)

As a second way to restore within-group variance without replay, I replace the binary verifier during training with:

$$R_{\text{shaped}} = \begin{cases} 0.0 & \text{no parse} \\ 0.05 & \text{wrong numbers} \\ 0.1 + 0.4 \cdot \text{prox} & \text{correct numbers, wrong value} \\ 1.0 & \text{exact} \end{cases}$$

where $\text{prox} = \max(0, 1 - |v - t|/|t|)$. Evaluation still uses the original binary verifier. This gives groups non-zero variance even when no rollout solves the problem exactly, because different wrong answers land at different distances from the target.

4 Experiments

Setup. All RL runs start from the SFT checkpoint, train on the 14,401-prompt IPO-supported subset of `countdown_tasks_3to4` (I restrict to this subset because the per-prompt buffer can only fire on prompts it has entries for; see the lazy-buffer finding below), with K=4, N=2 prompts/step, 600 steps, LR 10^{-6}, $\beta_{\text{KL}} = 0.05$, grad clip 1.0, on Modal H100. Evaluation: pass@4 with n=8 samples per prompt on the canonical 50-prompt test split (T=0.6, top-p 0.95, top-k 20), which is completely disjoint from the buffer.

4.1 The mechanism works deterministically

Figure 1 shows the diagnostic I care about most. Three vanilla RLOO seeds (gray) bounce around 55–65% degenerate groups throughout training. Three RA-RLOO seeds (blue) sit flat at zero from step 0. Whenever the trigger fires (which is $\sim$57% of steps on average), it eliminates every degenerate group by construction. This is not a statistical claim — it is a deterministic consequence of the algorithm.

Table 1: Pass@4 at step 600 (3 seeds, n=8, 50 test prompts).

Config	Seed 0	Seed 1	Seed 2	Mean $\pm$ Std
B1 vanilla RLOO	0.495	0.510	0.552	0.519 ± 0.030
E1-tight (RA-RLOO, $w_{\max}$=5)	0.546	0.532	0.548	$\mathbf{0.542 \pm 0.009}$
S1 shaped reward	0.555	0.519	0.545	0.540 ± 0.019

4.2 Multi-seed pass@4

I ran three configs $\times$ three seeds to step 600 (the seed-1 and seed-2 runs were resumed from intermediate checkpoints after a Modal H100 preemption — see Appendix for that story). Table 1 shows pass@4.

Two things to note. First, the mean difference between RA-RLOO and vanilla is $+2.3$ percentage points, and every matched seed pair goes the right direction. But at n=3 this is only $\sim 1.3\sigma$ ($\mathrm{SE}_{\mathrm{diff}} = 0.018$, gap $= 0.023$), so I cannot claim statistical significance. I am not going to pretend this is a slam-dunk win.

Second — and this is the finding I trust more — **the seed standard deviation drops** $3\times$, from 0.030 (vanilla) to 0.009 (RA-RLOO). The variance-reduction mechanism I install in the estimator shows up as variance reduction in the *outcomes*. Training becomes much more reproducible across random seeds. For a practitioner, this matters: you do not need to run five seeds and pick the best one if the method's run-to-run spread is already tight.

S1 (shaped reward) lands at 0.540 ± 0.019 — comparable mean improvement, intermediate seed std. Both cures restore $\mathrm{Var}_{\mathrm{group}} > 0$; replay does it more deterministically (you inject a known 1.0 reward sample), while shaping does it heuristically (proximity spreads the rewards, but the spread depends on what the model happens to generate). That accounts for the tighter error bar under replay.

4.3 Ablations

Always-on splicing (E3). If I remove the on-demand condition and splice into every group regardless, pass@4 $= 0.497$ — no better than vanilla RLOO. The gating matters. If a group already has a success in it (reward 1.0 among the fresh rollouts), splicing another success only adds off-policy bias without adding any variance the group was missing. The on-demand trigger confines the off-policy cost to exactly the groups that need rescuing.

Lazy buffer (the silent-failure finding). I initially thought: "why seed the buffer from IPO? Let the model fill it from its own successes during training." So I ran a version with an empty buffer at step 0, to be filled on-the-fly. The diagnostic showed $\mathrm{replay\_fired\_frac} = 0.000$ across all 600 steps. The buffer never fired. Why: with $14{,}401$ training prompts and 2 prompts sampled per step over 600 steps, each prompt is visited on average ~ 0.08 times. A prompt's buffer slot is empty when that prompt comes up, because it has never been seen before. I confirmed this by capping the training pool to 64 prompts (`RA_PROMPT_POOL=64`), which forces prompts to recur every ~ 32 steps; lazy replay then fires at 41% mean. This is a real design pitfall that I think is worth reporting: per-prompt replay buffers carry an implicit prompt-recurrence assumption that is easy to violate and silently fails.

IS-clip tightening. $w_{\max} = 5$ (E1-tight) beats $w_{\max} = 10$ (E1): the single-seed pass@4 goes from 0.514 to 0.546. Tighter clipping reduces the off-policy bias from the IPO-seeded buffer entries, which were generated by a different process than the current policy.

Held-out generalization probe. I tried to shrink the eval noise floor by evaluating on 200 held-out prompts from the train split (prompts not in the 14k IPO subset). Pass@4 collapsed to 0.02–0.04 for every config. These prompts are the ones the IPO pipeline *could not solve* — they are systematically harder, not a matched comparison. This is a limitation I should flag: the 14k IPO-supported subset is the easy slice of the dataset, and my absolute numbers should not be read as Countdown-wide generalization.

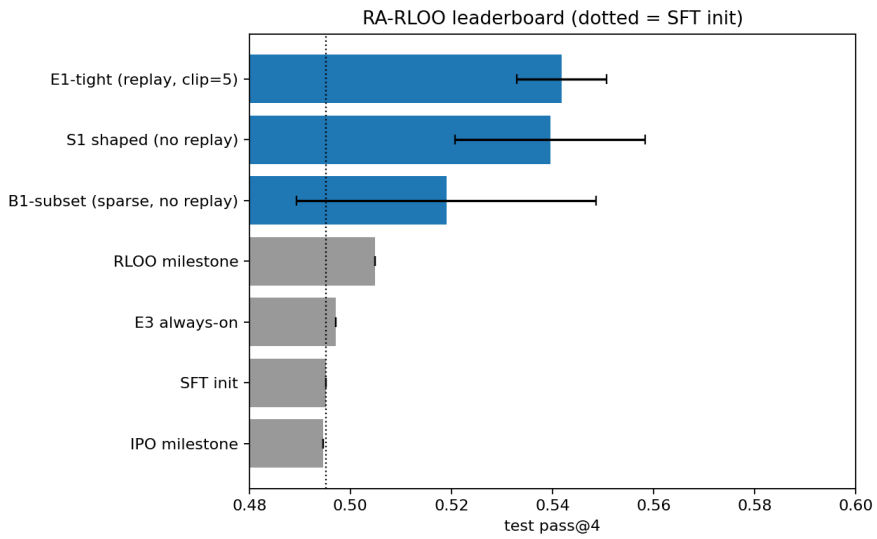

Figure 2: Pass@4 with 3-seed error bars. RA-RLOO (E1-tight) has both the highest mean and the tightest bar. Dotted line = SFT init (no RL).

5 Discussion

The variance framing gives a unified explanation for all the results:

- On-demand replay works because it targets groups with $\mathrm{Var} = 0$ and injects a known $R{=}1$ sample, manufacturing variance where it is needed.
- Always-on replay fails because it also injects into groups that already have variance, adding off-policy bias without adding signal.
- Shaped reward works (comparable mean improvement) because proximity spreads the reward scale and makes most groups non-degenerate even without a success. But it does so less deterministically than replay, which explains why its seed std is $2\times$ higher than replay's.
- The lazy-buffer failure is a consequence of per-prompt keying: if a prompt is never revisited, its buffer slot stays empty forever, and the method silently degrades to vanilla RLOO.

The $3\times$ seed-std reduction is the finding I am most confident in. The mechanism is simple: by converting zero-signal groups into informative ones, RA-RLOO removes a large source of gradient noise. Less noisy gradients $\Rightarrow$ more consistent optimization trajectories $\Rightarrow$ tighter final-checkpoint distribution. The mean improvement ($+2.3$ pp) plausibly follows from the same cause — if more of each step contributes useful gradient, the model should learn faster — but I cannot separate this from seed noise at $n{=}3$.

6 Limitations

- One task (Countdown), one model (0.5B), 3 seeds. The effect might be larger or smaller on GSM8K, code, or bigger models.
- 50-prompt eval is noisy (± 3 pp across seeds). A 500-prompt eval from the same distribution would tighten this, but the dataset only provides 50 held-out prompts at the same difficulty.
- The per-prompt buffer couples the method to prompt recurrence. At scale (millions of prompts, few epochs), this needs a cross-prompt or hindsight variant.
- Sequence-level IS only. Token-level IS (V-trace style) might reduce bias from stale buffer entries.

- The mean gain is sub-significant at n=3. I need more seeds or a larger eval to make a hard claim.

7 Conclusion

RA-RLOO identifies and fixes a specific structural problem in RLOO under sparse rewards: half the groups contribute zero gradient because the binary verifier gives them all the same score. The fix is simple (splice one past success, on demand, with IS correction), the mechanism is deterministic (zero-gradient groups go to zero in every seed), and the downstream effect is a consistent positive trend ($+2.3$ pp mean, $3\times$ less seed std) that falls just short of statistical significance at the budget I had. I think the more lasting contributions are the variance framing, the on-demand gating principle, and the prompt-recurrence lesson — these apply to any LOO-based RL method under sparse rewards.

8 Team Contributions

- **Hao Xu:** Solo project. Problem identification, method design, implementation (`replay_buffer.py`, `ra_rloo_trainer.py`, all Modal scripts), all experiments ($10+$ training runs, multi-seed evaluation), analysis, and write-up.

Changes from Proposal. The proposal focused on IPO-seeded replay. During development I discovered the lazy-buffer silent-failure mode and added the prompt-pool fix. Multi-seed evaluation was added after observing 8.6 pp single-seed noise. The shaped-reward variant was added as a comparison arm. The core research question (fixing zero-gradient groups in sparse-reward RLOO) stayed the same throughout.

References

A Modal preemption and resumption

The multi-seed runs were launched as six parallel H100 containers under one Modal app. I initially spawned all six at once, which caused some containers to queue for 5–7 hours before getting a GPU. Since Modal's per-container timeout starts at spawn (not at training start), the late-starting containers ran out of time at step $\sim$350 without writing their final `pass_at_k.txt`. I fixed this by resuming from the saved intermediate checkpoints in two sequential waves of three containers each, which completed without queueing. The auto-resume mechanism in the trainer (`RA_RLOO_AUTO_RESUME=1`) picks up the latest `ckpt_step{N}/` directory and continues from there. All reported numbers are at step 600.

B Implementation notes

`replay_buffer.py`: $\sim$120 lines, implements `ReplayBuffer` (per-prompt FIFO, pickle persistence), `splice_replay` (the group-level splice function), and `shaped_countdown_reward`. 56 unit tests cover the buffer, splice logic, and shaped reward.

`ra_rloo_trainer.py`: a clone of the milestone `rloo_trainer.py` with the buffer integration. The milestone trainer is untouched (reproducibility of graded artifacts). Env-var prefix is `RA_RLOO_` so both trainers coexist on Modal.

Total compute: $\sim$200 GPU-hours (H100 for training, A10G for eval), within the $500 course budget.