Extended Abstract

Sample-Efficient RLOO via Off-Policy Rollout Reuse

Problem. REINFORCE Leave-One-Out (RLOO) is a simple, strong policy-gradient method for fine-tuning language models with a verifier reward. However, it is strictly on-policy and discards every rollout after a single gradient step. On a 0.5B model and an arithmetic-reasoning task, generating those rollouts dominates the cost of training, while the gradient step itself is cheap. We ask how far a single batch of rollouts can be reused, and what controls the off-policy bias that reuse creates.

Approach. We reuse each sampled batch for K inner updates. The behavior policy μ is fixed when the batch is drawn; the policy π drifts as we take inner steps. We compute μ with the same model that takes the gradient, so at the first inner step ($k = 0$) the importance ratio is exactly one and the reuse axis starts from an exact on-policy origin. We compare three corrections of the policy-gradient term: (i) none (plain REINFORCE), (ii) clipped importance sampling (IS), and (iii) a clipped PPO surrogate. We track the realized drift each step with two cheap signals: (i) the effective-sample-size fraction (ESS) and (ii) the KL from π to μ. Our contribution is a drift-budgeted adaptive controller that keeps reusing a batch until the drift signal crosses a cutoff then resamples. It spends reuse where it is cheap (near on-policy) and stops before it becomes expensive, with no per-task tuning of K.

Results. Importance-sampling reuse improves sample efficiency: at a fixed update budget, $K = 4$ matches or beats the on-policy baseline at $4\times$ fewer rollouts and a $2.3\times$ wall-clock speedup (held-out pass@1 0.42 ± 0.02 over three seeds versus 0.39). IS learns best, plain REINFORCE collapses to zero for any reuse, and PPO is competitive (0.35 ± 0.01). A single off-policy law relates final accuracy to realized drift across every run, and a measured mechanism explains the gain. Reuse raises gradient-direction consistency until off-policy bias dominates, producing an inverted-U shape that peaks at $K = 4$. The adaptive controller reaches the fixed-K efficiency frontier with no K to choose and beats a blind decay schedule. We also report the limitations: the on-policy baseline overtrains past ~ 60 updates (entropy collapse, pass@1 $0.52 \rightarrow 0.08$), and the reuse benefit concentrates on the easier problems.

Significance. A cheap drift signal measured inside the trainer turns rollout reuse into a tuning-free knob that makes on-policy RL fine-tuning more sample and time efficient at equal or better quality. Every number traces to a logged artifact, and the headline claims carry error bars over three seeds on a held-out set.

Sample-Efficient RLOO via Off-Policy Rollout Reuse: A Drift-Budgeted Adaptive Controller

Zhaoyang Li
Stanford University
zhaoyangli@stanford.edu

Abstract

RLOO is on-policy and discards each rollout after one update, so rollout sampling dominates the cost of fine-tuning a language model with a verifier reward. We study how far a batch of rollouts can be reused and which off-policy correction controls the resulting bias. On the Countdown task with an SFT-initialized Qwen2.5-0.5B, we run a controlled study of the reuse factor K crossed with three corrections (none / IS / PPO), measuring off-policyness with behavior log probabilities. IS improves sample efficiency. At equal updates, $K = 4$ matches or beats the on-policy baseline at $4\times$ fewer rollouts and a $2.3\times$ wall-clock speedup (held-out pass@1 0.42 vs. 0.39). The corrections also separate cleanly. IS learns best, naive reuse collapses to zero, and PPO is competitive (0.35). We identify a measured mechanism (reuse raises gradient-direction consistency until drift dominates), unify every run under a simple law, and turn that law into a drift-budgeted adaptive reuse controller that reaches the fixed-K frontier with no tuning and beats a blind decay schedule. There are a few limitations: naive reuse diverges, the on-policy baseline over-trains, and the benefit concentrates on easier problems. Every number traces to a logged artifact.

1 Introduction

RLOO [1] is a simple and strong policy-gradient method for fine-tuning language models with a verifier reward. It samples a group of G responses per prompt, scores each with the reward, uses a leave-one-out baseline, and updates the policy once. It is strictly on-policy, as the next update needs a fresh batch of rollouts. For verifier-reward reasoning tasks, generating those rollouts (autoregressive decoding with fast inference) is the wall-clock bottleneck, while the gradient step on a small model is cheap. The method therefore spends most of its time producing data it uses exactly once.

This raises a direct question: *how far can we reuse a single batch of rollouts, and which correction keeps learning stable as the data become off-policy?* Reuse trades a controlled amount of off-policy bias for fewer sampling rounds. The trade is favorable only if the bias is both measured and contained. We study this on Countdown, a small arithmetic-reasoning task with a rule-based reward, using a fixed model and a fixed evaluation protocol so that every comparison is internally consistent.

Our contributions are:

- A controlled study of the reuse factor K crossed with three off-policy corrections (none / IS / PPO), with per-step drift diagnostics computed inside the trainer so the on-policy point is exact (Sec. 5).
- An empirical off-policy law: final accuracy is governed by the realized drift, not by the reuse count or correction that produced it (Sec. 7), together with a measured gradient-variance mechanism that explains why moderate reuse helps (Sec. 6).

- A drift-budgeted adaptive reuse controller that reaches the fixed-K sample-efficiency frontier with no per-task tuning of K and beats a blind decay schedule (Sec. 8).

2 Related Work

Policy gradients for language models. RLHF with PPO [7, 5] is the standard pipeline; RLOO [1] shows that a REINFORCE-style estimator with a leave-one-out baseline is competitive and simpler, and GRPO [8] uses a group-relative baseline at scale, as in DeepSeek-R1 [3]. These methods are on-policy by construction. **Preference optimization.** DPO [6] and IPO [2] optimize from a fixed preference dataset; IPO regresses the implicit reward margin to a fixed target, which we use to warm-start the policy. **Off-policy correction and efficiency.** A clipped surrogate (PPO) and importance weighting are the classical tools for reusing data within a trust region. Asynchronous RLHF [4] decouples generation from learning, trading a controlled amount of off-policyness for throughput. Our work differs by treating off-policyness itself as the controlled variable: we measure it cheaply each step and *budget* it with a simple controller, rather than fixing a reuse count or a clip range in advance.

3 Method

RLOO recap. For a prompt x we sample a group of G responses $\{y_i\}$ from the current policy π_θ and score each with the verifier reward r_i. The leave-one-out baseline gives the advantage

$$A_i = r_i - \frac{1}{G-1}\sum_{j\neq i} r_j = \frac{G}{G-1}\Big(r_i - \bar{r}\Big), \tag{1}$$

and the on-policy update maximizes $\sum_i A_i \log \pi_\theta(y_i \mid x)$, with an entropy bonus and a small KL penalty to the reference policy.

Rollout reuse. We reuse each sampled batch for K inner updates. The responses are sampled once from a behavior policy $\mu = \pi_{\theta_0}$ and stored with their behavior log-probabilities $\log \mu(y_i \mid x)$. As we take inner steps, π_θ drifts from μ, so inner steps $k \geq 1$ are off-policy. We compute $\log \mu$ with the same model that takes the gradient, so at inner step $k = 0$ the importance ratio is exactly one, $\mathrm{ESS}{=}1$, and the drift is zero. The reuse axis therefore starts from an exact on-policy origin, which is what makes the correction comparison and the controller interpretable. Larger K means fewer sampling rounds for the same number of updates, hence fewer rollouts.

Three corrections. Let $\rho_i = \pi_\theta(y_i)/\mu(y_i)$ be the per-sequence importance ratio from response token log probabilities. The policy-gradient term is one of:

$$\text{none:} \quad -\mathbb{E}[A_i \log \pi_\theta(y_i)] \tag{2}$$

$$\text{IS:} \quad -\mathbb{E}[\mathrm{clip}(\mathrm{sg}[\rho_i], \leq w_{\max}) A_i \log \pi_\theta(y_i)] \tag{3}$$

$$\text{PPO:} \quad -\mathbb{E}\big[\min(\rho_i A_i, \mathrm{clip}(\rho_i, 1{\pm}\epsilon) A_i)\big] \tag{4}$$

Methods none and IS use the score-function form, where the gradient flows only through the explicit $\log \pi_\theta$ factor and the weight is a detached coefficient ($\mathrm{sg}[\rho_i]$, stop-gradient); none is a negative control and IS ($w_{\max}{=}10$) down-weights only the worst-overweighted samples. PPO ($\epsilon{=}0.2$) builds a differentiable ratio and lets autograd differentiate the clipped surrogate, with hard-clamping to keep the updates inside the trust region.

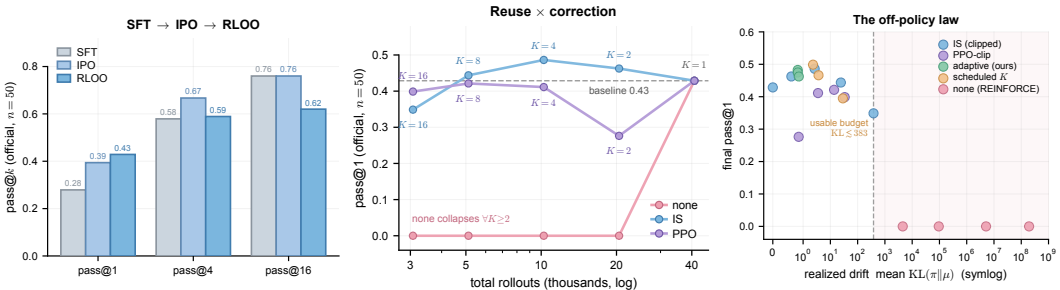
Drift diagnostics. Each step we log two cheap signals from the same forward pass: the effective-sample-size fraction

$$\mathrm{ESS}/N = \frac{(\sum_i w_i)^2}{N \sum_i w_i^2} \tag{5}$$

with $w_i = \rho_i$, and a KL-to-behavior proxy

$$\widehat{\mathrm{KL}}(\pi \| \mu) = \frac{1}{2}\mathbb{E}\left[(\log \rho_i)^2\right] \tag{6}$$

ESS falls from 1 and KL grows from 0 as a batch is reused.



(a) pipeline pass@k (b) sample efficiency (c) the off-policy law

Figure 1: **Reuse buys sample efficiency.** (a) pass@k improves across the supervised, preference, and RLOO stages. (b) pass@1 versus the rollouts consumed for three corrections; markers are reuse factors K, the dashed line is the on-policy baseline. Importance-sampling reuse peaks at $K = 4$ above the baseline at $4\times$ fewer rollouts; naive reuse collapses. (c) final pass@1 versus realized drift across all runs, a single law (Sec. 7).

Adaptive reuse controller. Our method replaces the fixed K with a stopping rule on the drift signal. Within one sampling round we repeat: take one RLOO update (IS correction) on the batch; read the drift signal σ; stop and resample when σ crosses the budget b (ESS below b, or KL above b) or a cap $K_{\max}$ is reached. The controller uses more reuse early (near on-policy, high ESS) and less reuse late (after θ has moved), automatically and with no per-task tuning of K.

4 Experimental Setup

Task and reward. Countdown: given a multiset of numbers and a target, produce an arithmetic expression that uses each number once. The reward is 1.0 for a correct expression, 0.1 for a parseable but wrong answer, and 0 for no answer. **Model.** Qwen2.5-0.5B, warm-started from a supervised checkpoint. **Training.** Batch 128 prompts, group size 8 (1024 rollouts per sampling round), learning rate 10^{-5} constant, entropy and KL coefficients 10^{-3}, $w_{\max}$=10, ϵ=0.2, on-policy sampling temperature 1.0. The dataloader is seeded so every run sees the same prompt order, removing a fewer-unique-prompts confound as K grows. **Evaluation.** We report pass@k with the unbiased estimator on two splits: a fixed 50-prompt test set ($\pm$0.07 pass@1 noise), and a contamination-free 256-prompt held-out set reserved from the training tail and never trained on ($\pm$0.03), which we use for the seeded headline claims. **Compute.** Fully local on a four-H100 node; each run uses one GPU, alternating sampling and the update. Update budgets are fixed within each comparison.

5 Reuse and correction

Figure 1b is the headline. Because the on-policy point is exact, the importance-sampling curve *rises* with reuse, peaking at $K = 4$ with pass@1 0.486 at only 10k rollouts (above the on-policy baseline of 0.429 at $4\times$ fewer rollouts), then falls for $K\geq 8$ (0.444 at $K = 8$, 0.349 at $K = 16$): an inverted-U in K. Moderate reuse therefore helps, and Sec. 6 traces this to reduced gradient variance. Naive reuse without a correction collapses to pass@1 0 for *every* $K \geq 2$, a clean negative result: uncorrected reuse drives the policy into extreme drift and divergence. PPO is competitive, reaching 0.41–0.42 (peaking at $K = 8$); the clean ordering is IS $\gtrsim$ PPO $\gg$ none, with IS edging PPO on the held-out set. Pipeline accuracy improves monotonically across the supervised, preference, and RLOO stages (Fig. 1a).

6 Why reuse helps or hurts

The off-policy state inside a round. Figure 2 reads the drift inside each reuse round. ESS starts at 1.0 (exact on-policy) and falls as the policy drifts; KL starts at 0 and grows. The corrections separate by how hard they brake. At $K = 8$, the mean KL to behavior is 13.7 (PPO), 24.2 (IS), and 5.1×10^6 (none), with mean ESS 0.32, 0.28, and 0.14 respectively, so naive reuse drifts to divergence while IS

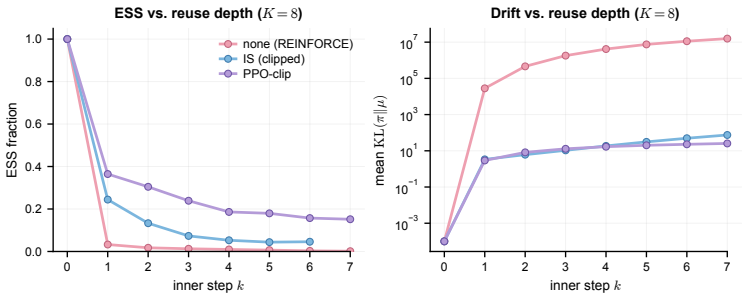


Figure 2: **Drift inside a reuse round** ($K = 8$)**.** ESS fraction (left) and KL-to-behavior (right) versus inner step k, one line per correction. Both start at the exact on-policy origin (ESS=1, KL= 0); naive reuse drifts to divergence while IS and PPO stay bounded.

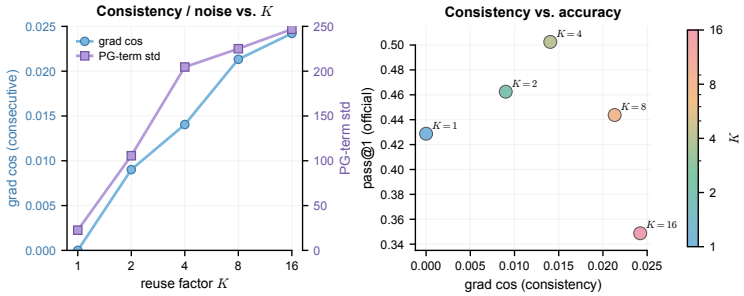


Figure 3: **The variance mechanism.** (left) Gradient-direction consistency rises with K while the within-batch policy-gradient noise grows. (right) Accuracy nonetheless peaks at $K = 4$ and falls for larger K despite still-rising consistency, the bias-variance turnover.

and PPO stay bounded. Stability alone, however, cannot explain why IS beats the on-policy baseline at equal updates.

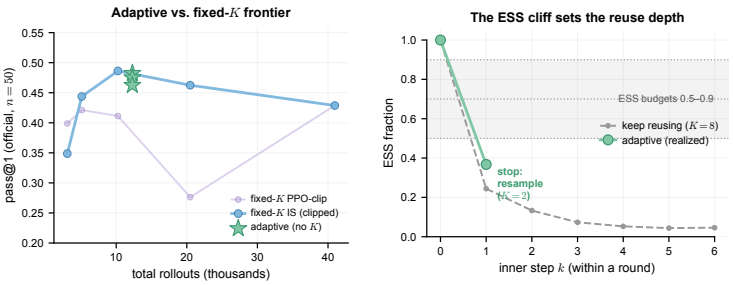
The variance mechanism. Figure 3 measures the mechanism directly. The cosine similarity between consecutive update gradients rises with K (0.000 at $K = 1$, where each step is a fresh batch, to 0.024 at $K = 16$), so reuse gives a more consistent, lower-variance optimization direction. The trade is favorable only while drift is small. We find that pass@1 peaks at $K = 4$ and then falls even as consistency keeps rising ($K = 16$ has the highest consistency but the lowest pass@1), which is exactly the bias-variance trade-off behind the inverted-U. Reuse buys a real variance reduction at the cost of a little off-policy bias, and the trade is good up to $K \approx 4$.

7 The off-policy law

Figure 1c shows final pass@1 versus the training-mean KL-to-behavior (log scale; drift runs from ≈ 0 at the exact on-policy point to 1.9×10^8). Accuracy stays high (0.40–0.49 for IS, PPO, and the adaptive controller) while mean drift is below ~ 30, and collapses to 0 once drift exceeds $\sim 10^3$, the regime where the uncorrected runs are driven into divergence. IS, PPO, and adaptive all cluster in the high-accuracy, low-drift corner, and only naive reuse occupies the far-right collapse zone. The usable drift budget—the largest realized drift that still yields a working policy—is KL $\lesssim$ a few$\times 10^2$ (IS at $K = 16$ still reaches 0.349 at KL$\approx$383, whereas naive reuse at $K = 2$ already collapses at KL$\approx$4500). The best runs all live at KL $\lesssim 25$. This budget is what the controller targets.

8 A drift-budgeted adaptive controller

Figure 4 overlays the adaptive runs on the fixed-K frontier. Because the on-policy ESS is exactly 1.0, the three budgets we try (0.5, 0.7, 0.9) all land on the IS frontier (pass@1 0.482, 0.475, 0.463 at 12k rollouts) without choosing K. The best adaptive run matches the best fixed-K point (IS $K = 4$, 0.486) at a comparable rollout cost. The reuse the budget buys is a constant 2: ESS falls off a cliff,



(a) on the fixed-K frontier (b) the ESS cliff sets the reuse depth

Figure 4: **The controller reaches the frontier with no tuning.** (a) the adaptive runs (stars) land on the fixed-K importance-sampling Pareto frontier. (b) ESS collapses from 1.0 to ≈ 0.37 in one off-policy step, so every budget in $[0.5, 0.9]$ stops after the first inner step and the realized reuse is a constant 2.

from 1.0 to ≈ 0.37 in a single off-policy step (Fig. 4b), so any budget in $[0.5, 0.9]$ is crossed after the first inner step and the three budgets are nearly indistinguishable. The controller therefore spends reuse where ESS is high and resamples once it falls, automatically.

A blind reuse schedule is the natural non-adaptive alternative. In a head-to-head at a matched rollout budget, the adaptive controller (held-out pass@1 0.413) matches fixed $K = 4$ (0.415) and beats a blind $4{\to}1$ decay schedule (0.361), so measuring drift per round does real work over a hand-set decay.

9 Benchmark, efficiency, and limits

Seeded benchmark. Three seeds on each headline configuration, on the contamination-free held-out set (Fig. 5a), confirm the ordering: IS $K = 4$ pass@1 0.422 ± 0.023, adaptive 0.411 ± 0.006, PPO $K = 4$ 0.353 ± 0.010, above the on-policy baseline 0.388. The IS–PPO gap (0.069) is several standard deviations (significant), and the adaptive controller matches IS $K = 4$ within error bars while choosing no K.

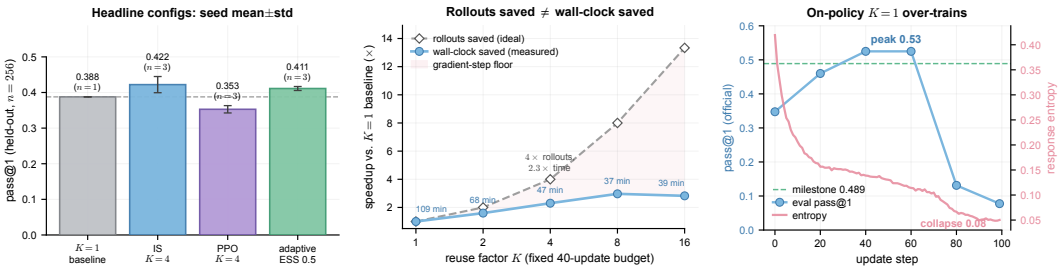
Wall-clock efficiency. The rollout saving is real but does not transfer one-to-one to wall-clock (Fig. 5b). At a fixed update budget, $K = 4$ finishes in 47 min and the adaptive controller in 41 min, versus 109 min for the on-policy baseline, a $2.3\times$ speedup. Because only the sampling rounds are skipped while the gradient steps are a fixed floor, the rollout saving keeps growing with K (to $13\times$) but the wall-clock saving plateaus near 2.3–$3\times$.

Failure modes. We report three honest limits. (i) Naive reuse diverges for any $K \geq 2$. (ii) The on-policy baseline over-trains, such that a full-length run peaks at pass@1 0.525 around 40–60 updates, then collapses to 0.08 by 100 updates as its response entropy falls from 0.42 to 0.05 while its training reward stays high (Fig. 5c), so we cap the budget where every run sits near its peak, before this entropy cliff. (iii) The reuse benefit concentrates where the policy already has signal. Split by problem size, reuse helps the easier three-number problems (IS $K = 1 \to K = 4$: $0.568 \to 0.646$) but is flat on the harder four-number problems ($0.254 \to 0.244$).

Robustness. Table 1 lists every run with pass@k, the realized rollout and wall-clock budgets, and the mean drift diagnostics. The on-policy $K = 1$ point sits at ESS$=1.0$ and drift 0, and the separation between corrections is monotone in realized drift, consistent with the off-policy law.

10 Discussion

Reuse helps when the realized drift stays small and a correction damps the worst updates, while it hurts when drift compounds across inner steps, which shows up as entropy collapse and format reward-hacking under naive reuse. The three corrections expose a clear bias-variance-stability trade-off. Naive reuse has no brake, so reuse drives it to divergence. IS keeps the score-function gradient and only down-weights the worst-overweighted samples, so it learns aggressively and is best on pass@1. PPO keeps a differentiable clipped ratio and is a close second. Off-policyness should be



(a) seeded benchmark (b) wall-clock efficiency (c) the baseline over-trains

Figure 5: **Benchmark, efficiency, and a limit.** (a) Held-out pass@1 (mean±std, three seeds; baseline a single run): IS $K = 4$ and the adaptive controller sit above the baseline; the IS–PPO gap is significant. (b) rollouts saved ($\sim K$) outrun wall-clock saved (2.3–3×) because the gradient steps are a fixed floor. (c) the full-length on-policy baseline peaks then over-trains as entropy collapses.

Table 1: All runs; pass@k on the fixed 50-prompt test split (the held-out numbers are reported in the text). roll.(k) = thousands of rollouts; wall(m) = wall-clock minutes; $\overline{KL_\mu}$ = mean KL-to-behavior; $\overline{ESS}$ = mean ESS fraction. Bold marks a sample-efficiency win (matches or beats the on-policy baseline at fewer rollouts); italic rows are the adaptive controller.

run	K	pass@1	pass@4	pass@16	roll.(k)	wall(m)	$\overline{KL_\mu}$	$\overline{ESS}$
IS, K=1	1	0.429	0.589	0.620	41.0	109	0.0000	1.000
IS, K=1 (full budget)	1	0.077	0.212	0.400	102.4	261	0.0000	1.000
IS, K=2	2	**0.463**	0.618	0.700	20.5	68	0.6123	0.694
IS, K=4	4	**0.486**	0.697	0.760	10.2	47	2.6530	0.421
IS, K=4 (seed 2)	4	**0.547**	0.706	0.740	10.2	48	2.6589	0.430
IS, K=4 (seed 3)	4	**0.474**	0.675	0.760	10.2	47	2.5205	0.449
IS, K=8	8	**0.444**	0.683	0.800	5.1	37	24.2350	0.281
IS, K=16	16	0.349	0.618	0.780	3.1	39	382.5112	0.302
none, K=2	2	0.000	0.000	0.000	20.5	73	4507.7614	0.583
none, K=4	4	0.000	0.000	0.000	10.2	51	94467.7396	0.285
none, K=8	8	0.000	0.000	0.000	5.1	40	5114039.1989	0.135
none, K=16	16	0.000	0.000	0.000	3.1	40	193588074.8977	0.072
PPO, K=2	2	0.276	0.483	0.600	20.5	68	0.8545	0.670
PPO, K=4	4	**0.411**	0.607	0.660	10.2	46	3.5243	0.491
PPO, K=4 (seed 2)	4	**0.409**	0.590	0.660	10.2	48	4.0588	0.454
PPO, K=4 (seed 3)	4	0.376	0.565	0.640	10.2	47	4.4698	0.475
PPO, K=8	8	**0.421**	0.628	0.700	5.1	36	13.6647	0.323
PPO, K=16	16	**0.399**	0.644	0.720	3.1	39	33.6023	0.174
scheduled 4→1	–	**0.466**	0.690	0.760	12.3	39	3.6745	0.664
scheduled 8→1	–	**0.395**	0.664	0.780	1.0	6	28.7848	0.287
scheduled const 4	–	**0.499**	0.694	0.800	12.3	57	2.3111	0.435
adaptive, ESS 0.5	–	**0.482**	0.663	0.720	12.3	41	0.8231	0.684
adaptive, ESS 0.5 (seed 2)	–	**0.509**	0.700	0.780	12.3	42	0.8948	0.675
adaptive, ESS 0.5 (seed 3)	–	**0.471**	0.669	0.780	12.3	40	0.8844	0.692
adaptive, ESS 0.7	–	**0.475**	0.683	0.740	12.3	41	0.8230	0.691
adaptive, ESS 0.9	–	**0.463**	0.663	0.720	12.3	40	0.8572	0.672

measured in the same model that takes the gradient. With an exact on-policy origin, the drift axis starts at zero, the correction comparison is unambiguous, and an ESS budget is set on an intuitive scale. There are also a few limitations. For example, we use a single small task and a 0.5B model; the public test set contains only 50 prompts, with the reserved held-out set tightening the estimates to ± 0.03; our update budgets are capped below the over-training cliff; and we use a disk-based sampler/learner hand-off instead of a true asynchronous pipeline.

11 Conclusion

We asked how far a single batch of RLOO rollouts can be reused and which off-policy correction contains the resulting bias. Measuring drift in the same model that takes the gradient anchors the reuse axis at an exact on-policy origin, which is what makes the correction comparison and the controller interpretable. The main results are as follows.

- Importance-sampling reuse improves sample efficiency, with $K = 4$ matching or beating the on-policy baseline at $4\times$ fewer rollouts and a $2.3\times$ wall-clock speedup;
- The corrections separate cleanly as IS $\gtrsim$ PPO $\gg$ none, with naive reuse diverging;
- A single off-policy law ties final accuracy to the realized drift rather than to the reuse count or correction that produced it;
- A gradient-variance mechanism explains the inverted-U in K that peaks at $K = 4$.
- Turning the law into a one-line drift-budgeted stopping rule yields a tuning-free controller that reaches the fixed-K efficiency frontier with no K to choose and beats a blind decay schedule.
- A cheap drift signal, always available inside the trainer, therefore makes on-policy RL fine-tuning more sample and time efficient at equal or better quality.

Team Contributions

Zhaoyang Li: Everything.

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, 2024.

[2] 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, 2023.

[3] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[4] Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models, 2024.

[5] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

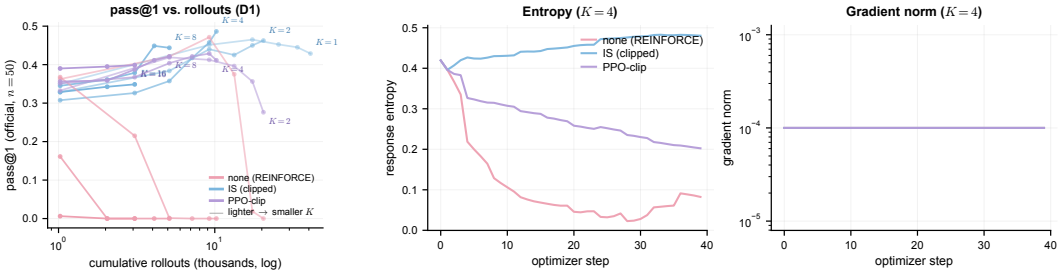
[6] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[8] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

A Additional Figures

Figure 6 shows two supporting diagnostics. (a) On a comparable budget axis (cumulative rollouts rather than the outer update step, which is not comparable across K), the pass@1 trajectories make the inverted-U in K visible over training. (b) Response entropy and gradient norm over training for each correction at $K = 4$: naive reuse loses entropy fastest, preceding its collapse.



(a) pass@1 vs. cumulative rollouts

(b) entropy and grad-norm ($K = 4$)

Figure 6: **Supplementary diagnostics.** (a) is one panel; (b) is two panels at twice the width, so all three panels render at a comparable size.