How Off-Policy Can GRPO Be? Efficient μ-GRPO with Negative-Advantage Veto

Anonymous authors
Paper under double-blind review

Abstract

Group Relative Policy Optimization (GRPO) has been a key driver of recent progress in reinforcement learning with verifiable rewards (RLVR) for large language models, but it is typically trained in a low-staleness, near-on-policy regime that incurs substantial system overhead. We ask a simple question: How off-policy can GRPO be? We show that GRPO-style algorithms can tolerate substantially larger rollout staleness than previously assumed, and propose μ-GRPO, an RL training framework that organizes training into a small number (*e.g.*, four) of large sequential generation–optimization stages. This design induces high rollout staleness while greatly reducing rollout–optimization switching overhead. To stabilize learning under stale data, μ-GRPO combines relaxed clipping, which preserves useful stale-rollout gradients, with negative-advantage veto, which removes destabilizing post-trigger suffix updates in negative-advantage responses. Across five language models and multiple math reasoning benchmarks, μ-GRPO matches or exceeds the performance of standard GRPO while achieving around $2\times$ speedup in wall-clock training time, establishing a substantially improved performance–efficiency trade-off for LLM reinforcement learning.

1 Introduction

Reinforcement Learning with Verifiable Rewards (RLVR) has rapidly reshaped the landscape of large language model (LLM) reasoning, driving landmark systems such as OpenAI o1 (OpenAI, 2024) and DeepSeek R1 (Guo et al., 2025). A dominant narrative attributes the success of these RL approaches, most notably PPO-style methods (Schulman et al., 2017) such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and its variants (Yu et al., 2025), to their on-policy design, where rollouts are freshly generated and closely aligned with the current policy. This freshness is widely viewed as important for stable credit assignment and iterative self-improvement.

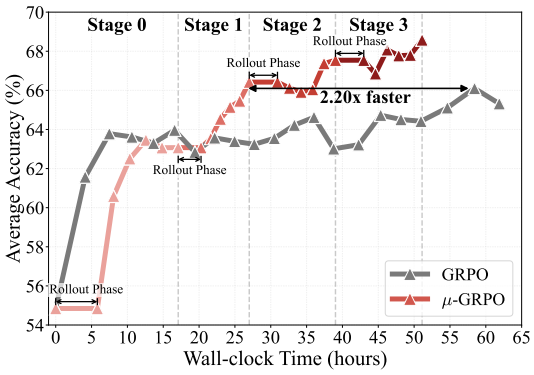

Figure 1: **Comparing GRPO and μ-GRPO.** Average accuracy across five math benchmarks over wall-clock time. μ-GRPO uses four large rollout–optimization stages, inducing high rollout staleness while reducing rollout–training switching overhead. It reaches GRPO's performance with a $2.2\times$ wall-clock speedup on DeepSeek-7B (Guo et al., 2025). Both methods are trained for 4096 updates.

On closer inspection, however, standard GRPO implementations are almost never strictly on-policy. The off-policyness arises from multiple sources. One primary source is data staleness: most RL training frameworks (Sheng et al., 2025; Zhu et al., 2025; Noukhovitch et al., 2025; Zhong et al., 2025; He et al., 2025) use rollouts generated by policies from several updates earlier, resulting in a *near* on-policy regime that trades off rollout freshness for efficiency (Hugging Face, 2025). Throughout this paper, we refer to this commonly used low-staleness, near-on-policy implementation as *standard GRPO*. Another source of off-policyness stems from training–inference

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

mismatch. In practice, rollouts are typically generated using high-throughput inference backends like vLLM (Kwon et al., 2023), while policy optimization is performed using sharded training engines like FSDP (Zhao et al., 2023). This introduces numerical discrepancies between rollout generation and optimization. Empirically, however, LLM RL has proven tolerant to such off-policyness when appropriate stabilization techniques are applied (Liu et al., 2025; Qi et al., 2025).

Taken together, these observations raise a natural question: *How far can we increase rollout staleness in GRPO without compromising learning stability or reasoning performance?*

This question is also central to system efficiency. Standard GRPO implementations repeatedly alternate between rollout generation and policy optimization, requiring many weight synchronizations between inference and training engines. To make such switching practical, systems often keep training state resident during rollout generation, reducing memory available for high-throughput inference (Sheng et al., 2025; Zhu et al., 2025). Increasing rollout staleness could amortize rollout generation over many more policy updates and thereby reduce these overheads. While prior work has explored limited staleness, these approaches do not push staleness far enough to fully decouple rollout generation from optimization (Zheng et al., 2026; Xi et al., 2026).

In this paper, we show that GRPO-style algorithms can tolerate substantially larger rollout staleness than commonly assumed, but only if the failure mode of high-staleness optimization is handled explicitly. Our diagnosis reveals that stale rollouts retain useful learning signal: relaxed clipping recovers strong early learning under large staleness, but later introduces a localized instability rather than generic off-policy collapse. We trace it to negative-advantage trajectories that cross an off-support boundary: once a prefix becomes poorly supported by the current policy, token-level GRPO can still apply substantial updates to later suffix tokens. This prefix-support mismatch explains why token-level clipping alone fails under large staleness.

Motivated by this diagnosis, we propose **μ-GRPO**, a staged GRPO framework for high-staleness training. Instead of frequent small generation–optimization cycles, μ-GRPO uses a few large stages that execute rollout generation and policy optimization sequentially: each stage first freezes the current policy to generate a large static rollout dataset, and then consumes this dataset over many policy updates using stored behavior log-probabilities as anchors, resulting in high rollout staleness. To stabilize optimization under extreme staleness, μ-GRPO combines relaxed clipping with negative-advantage veto (NAV), which filters destabilizing post-trigger suffix updates in negative-advantage responses while preserving useful stale-rollout gradients.

Empirically, μ-GRPO converts GRPO's staleness tolerance into substantial wall-clock speedups. As illustrated in Figure 1, μ-GRPO completes training in a few stages (e.g., four), reaching the best accuracy of standard GRPO $2.2\times$ faster in wall-clock time and ultimately exceeding it on DeepSeek-7B[1] (Guo et al., 2025). Across five language models and a diverse set of math reasoning benchmarks, μ-GRPO matches or improves on the average accuracy of standard GRPO on four out of five models. At the same time, it reduces rollout generation time by $1.83\times$ and overall wall-clock training time by $1.53\times$ on average. These results show that high-staleness GRPO is not only viable, but can improve the performance–efficiency trade-off of scalable LLM reinforcement learning.

2 Preliminaries: GRPO and Rollout Staleness

We briefly review Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and define rollout staleness. Let β denote the behavior policy that generates rollouts and π_θ the policy being optimized.

GRPO objective. For each prompt $x \sim \mathcal{D}$, the behavior policy β samples a group of G responses $\{y_i\}_{i=1}^{G}$, where $y_i = (a_{i,1}, \ldots, a_{i,T_i})$. Each response receives a scalar reward r_i. GRPO estimates advantages by normalizing rewards within the group:

$$A_i = \frac{r_i - \mathrm{mean}(\{r_1, \ldots, r_G\})}{\mathrm{std}(\{r_1, \ldots, r_G\})}. \tag{1}$$

The same response-level advantage A_i is assigned to all tokens in response y_i.

[1] DeepSeek-7B denotes the Qwen2.5-Math-7B model distilled from DeepSeek-R1. link to Hugging Face models

For token $a_{i,t}$, with prefix state $s_{i,t} = (x, a_{i,<t})$, define the token-level importance ratio $\rho_{i,t}(\theta) = \pi_\theta(a_{i,t} \mid s_{i,t})/\beta(a_{i,t} \mid s_{i,t})$. GRPO optimizes

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\,\{y_i\}\sim\beta}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=1}^{T_i}\min\left(\rho_{i,t}(\theta)A_i,\ \mathrm{clip}\left(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)A_i\right)\right]. \quad (2)$$

The implemented training loss is $-\mathcal{J}_{\mathrm{GRPO}}$. A KL term $-\lambda_{\mathrm{KL}} D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})$ is optionally applied.

Rollout staleness μ. GRPO training alternates between rollout generation and policy optimization. At the beginning of each cycle, the current policy is frozen as the behavior policy β, which generates a fixed rollout batch used for several policy updates before the next refresh.

Let B_{train} denote the number of prompt groups generated in one rollout phase, where each prompt group contains G sampled responses. Let B_{mini} denote the number of prompt groups consumed by one policy update. Therefore, a single rollout generation phase supports $\mu = B_{\mathrm{train}}/B_{\mathrm{mini}}$ model updates, which we refer to as the *rollout staleness*.

During these μ updates, θ changes while the behavior policy β and the stored rollout log-probabilities remain fixed. Thus later updates within the same cycle optimize against increasingly stale data. A strictly on-policy algorithm corresponds to $\mu = 1$, where every update uses freshly generated rollouts.

GRPO is therefore a *near on-policy* algorithm, with μ typically set to a small value (e.g., $\mu = 4$ (Zheng et al., 2025b; Wang et al., 2025; Yu et al., 2025; MiniMax et al., 2025; Zheng et al., 2025a)) to control policy drift under PPO-style clipped updates. While it is generally understood that a large μ degrades performance, **we show in this work that substantially larger values of μ are feasible**, achieving comparable or improved performance while accelerating training.

Throughout the paper, one *model update* refers to one optimizer update on one minibatch, and *batch size* refers to B_{mini}, the number of prompt groups for one optimizer update, unless stated otherwise.

3 Diagnosing GRPO's Staleness Tolerance

Large-μ training promises faster training through extensive data reuse, but requires understanding when stale rollouts remain useful and when they become destabilizing. This section shows why naive high-staleness GRPO breaks (Section 3.1), identifies low-ratio negative-advantage tokens as collapse triggers (Section 3.2), and localizes the destabilizing updates to the post-trigger suffix (Section 3.3).

3.1 The Clipping Dilemma under High Staleness

Under low staleness, the behavior policy β and current policy π_θ remain close, so the standard $[0.8, 1.2]$ interval clips only outlying updates. Under high staleness, however, rollouts sampled from the same β are reused for many updates, causing the current policy π_θ to drift away from the rollout distribution. Figure 2 illustrates this clipping dilemma at $\mu = 1024$. With the standard clipping bound $[0.8, 1.2]$, an increasing fraction of rollout tokens fall outside the interval and are clipped, suppressing their gradient contribution (Figure 2, left). As a result, the run plateaus at a lower accuracy (Figure 2, right). In contrast, relaxing the clipping range to $[0, 5]$ keeps the clipping ratio low and recovers strong early learning, indicating that stale rollouts still contain useful gradient signal. However, this signal is not continuously safe to use: after the initial improvement phase, the relaxed-clipping run collapses mid-training. These results show that high-staleness GRPO is limited not by the absence of useful signal in stale rollouts, but by the difficulty of exposing that signal while suppressing destabilizing updates as π_θ drifts from β.

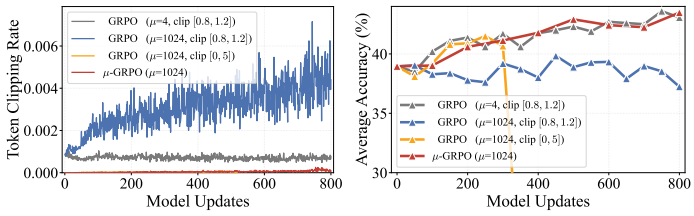

Figure 2: **Clipping creates a high-staleness dilemma.** At $\mu = 1024$, the $[0.8, 1.2]$ bound clips many more tokens and plateaus at lower accuracy (blue), while relaxed clipping $[0, 5]$ recovers early gains but later collapses (yellow); standard GRPO and μ-GRPO are shown for reference. Results are on Qwen2.5-Math-7B.

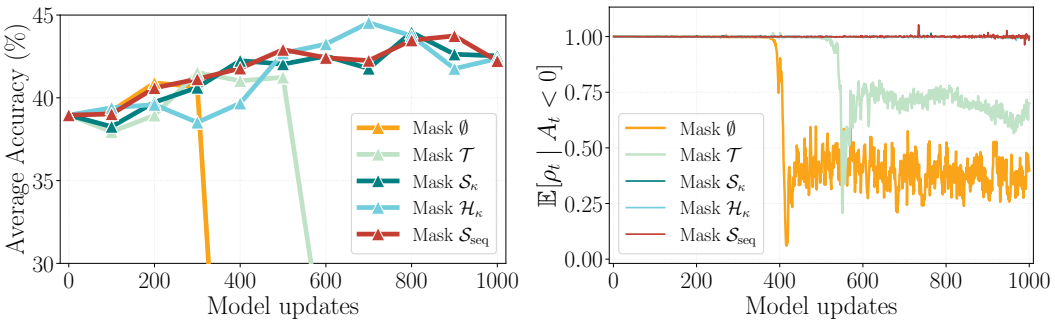

Figure 3: **Localizing the harmful updates.** Under relaxed clipping $[0, 5]$, masking only trigger tokens $\mathcal{T}$ still collapses, while masking the non-trigger suffix $\mathcal{H}_\kappa$ or broader scopes keeps both accuracy and negative-advantage importance ratios stable. Results are on Qwen2.5-Math-7B.

3.2 Negative-Advantage Drift as a Collapse Signal

Hereafter we drop the response index i, writing $y = (a_1, \ldots, a_T)$ and $A_t = A_i$ for the response-level advantage of token t. To locate which part of the relaxed update becomes dangerous, we focus on negative-advantage tokens. For GRPO, a negative advantage means that the response scored below its group average, so the update decreases the probability of the tokens that produced it. Under high staleness, many updates use rollouts from the same frozen behavior policy β. With relaxed clipping, the policy can repeatedly suppress negative-advantage actions sampled from this fixed distribution.

This repeated suppression creates a simple diagnostic. The importance ratio $\rho_t = \pi_\theta(a_t \mid s_t)/\beta(a_t \mid s_t)$ compares a token's likelihood under the current and behavior policies. A small ratio on negative-advantage tokens means that actions once plausible under β have become much less likely under π_θ. Figure 3 (masking scopes defined in Section 3.3) shows that this is exactly what happens in collapsed runs: Mask $\emptyset$ and Mask $\mathcal{T}$ show a sharp drop in $\mathbb{E}[\rho_t \mid A_t < 0]$, whereas $\mathcal{S}_\kappa$, $\mathcal{H}_\kappa$, and $\mathcal{S}_{\text{seq}}$ keep it near one. Collapse is thus accompanied by over-suppression of low-reward behavior-sampled responses.

With the relaxed clipping range $[0, 5]$, the gradient of the per-token loss ℓ_t for a negative-advantage token is $\nabla_\theta \ell_t(\theta) = |A_t|\, \rho_t(\theta) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$. Thus, relaxed clipping allows negative-advantage updates to keep suppressing behavior-sampled actions over a much wider ratio range.

This motivates using low-ratio negative-advantage tokens as collapse triggers. Prior work on training–inference mismatch similarly captures low-ratio events to mask out unstable updates (Liu et al., 2025). Here, we treat such events as boundaries where useful stale learning begins to turn into destructive policy drift, and then ask which subsequent updates must be masked to prevent collapse.

3.3 Post-Trigger Suffix Tokens as the Source of Collapse

Are low-ratio negative-advantage tokens themselves the harmful updates, or do they mark a boundary after which the rest of the response becomes unsafe to learn from? A token is a *trigger token* if it has negative advantage and its importance ratio falls below a small threshold: $A_t < 0$ and $\rho_t(\theta) < \tau_c$, where τ_c is a small threshold (we use $\tau_c = 10^{-4}$). For a response y, the *trigger boundary* is the first trigger-token position $\kappa(y) = \min\{t : A_t < 0,\ \rho_t(\theta) < \tau_c\}$, with $\kappa(y) = \infty$ if no such token exists.

With this defined, we compare five masking scopes, all applied only to negative-advantage responses:

- No-Mask Baseline $\emptyset$: Keeps the full, unaltered response.
- Trigger-token Mask $\mathcal{T}(y)$: Removes all trigger tokens.
- Suffix Mask $\mathcal{S}_\kappa(y)$: Removes all tokens from the trigger boundary (exclusive) to the end.
- Non-trigger Suffix Mask $\mathcal{H}_\kappa(y) = \mathcal{S}_\kappa(y) \setminus \mathcal{T}(y)$: Masks the suffix tokens excluding trigger tokens.
- Sequence Mask $\mathcal{S}_{\text{seq}}(y)$: Masks the whole response once any trigger appears.

Figure 3 shows the results. Under relaxed clipping $[0, 5]$, both no-mask baseline and trigger-token mask $\mathcal{T}(y)$ collapse. Thus, low-ratio tokens are primarily *triggers*, rather than *the dominant harmful updates*. By contrast, non-trigger suffix mask $\mathcal{H}_\kappa(y)$ stabilizes training, and the broader masks $\mathcal{S}_\kappa(y)$ and $\mathcal{S}_{\text{seq}}(y)$ are also stable because they include the same post-boundary suffix.

Position, not masking volume. We further verify that stability is determined by *where* updates are removed rather than *how many*. In the same setting as Figure 3, for each triggered response we retain an equal number of non-trigger tokens either immediately before or immediately after $\kappa(y)$ and veto the rest, so both settings keep exactly the same number of negative-advantage updates. Measured by average accuracy over the five math benchmarks, pre-trigger retention remains stable (41.2% at update 400), whereas post-trigger retention collapses to 17.6%, with $\mathbb{E}[\rho_t \mid A_t < 0]$ deteriorating only in this setting (Appendix C.3). The takeaway is that *collapse is driven by post-trigger suffix tokens with non-tiny local ratios, not by the trigger tokens alone*.

Why Post-Trigger Tokens Become Harmful. A trigger marks the point after which tokens are conditioned on prefixes that π_θ is unlikely to generate. Define the prefix occupancy ratio as

$$R_{t-1}(\theta) = \prod_{j=1}^{t-1} \rho_j(\theta) = \frac{\pi_\theta(a_{<t} \mid x)}{\beta(a_{<t} \mid x)}. \tag{3}$$

Once a trigger appears at position κ, its tiny ratio enters every later R_{t-1}, so teacher-forcing training continues to update suffix tokens behind a prefix with low current-policy support.

This is the correction that token-level GRPO omits. Its importance weight and clipping both depend only on the local ratio $\rho_t(\theta)$, without accounting for the probability of reaching the prefix on which that token is conditioned. Post-boundary suffix tokens with non-tiny ratios therefore stay inside the clipping range and keep receiving large updates, even though the prefixes needed to reach them are already unlikely; this is why token-level clipping alone cannot prevent collapse under high staleness.

Trigger tokens in $\mathcal{T}(y)$ already have tiny local ratios and hence small gradient coefficients, whereas tokens in $\mathcal{H}_\kappa(y)$ inherit low prefix support while retaining non-tiny ratios; this is why masking $\mathcal{H}_\kappa(y)$, but not $\mathcal{T}(y)$, prevents collapse. Appendix E formalizes this view.

4 METHOD: μ-GRPO

μ-GRPO. Building on this diagnosis, we propose a staged training framework that separates rollout generation and policy optimization into large-μ phases. Instead of tightly interleaving generation and optimization as in standard GRPO, μ-GRPO generates rollouts in bulk and then performs many policy updates on the resulting static dataset. Each stage consists of two phases:

Phase 1: High-Throughput Rollout Generation. At the beginning of each stage k, we freeze the current policy as the rollout policy β_k (with $\beta_0 \leftarrow \pi_{\text{base}}$) and use it only for rollout generation. Given the stage prompt set $\mathcal{X}_k = \{x_i\}_{i=1}^{B_{\text{train}}}$, the rollout policy β_k samples a response group $\mathcal{Y}_i = \{y_{i,g}\}_{g=1}^{G}$ for each prompt x_i. For each generated response $y = (a_1, \ldots, a_T)$ to prompt x, we compute its group-normalized advantage A as in Eq. 1 and store tokenwise behavior log-probabilities $b_{1:T}$ with $b_t = \log \beta_k(a_t \mid x, a_{<t})$. The resulting static rollout dataset is

$$\mathcal{D}_k = \{(x_i, y, A, b_{1:T}) : x_i \in \mathcal{X}_k,\ y \in \mathcal{Y}_i\}.$$

The frozen rollout engine emits behavior log-probabilities during decoding, avoiding an additional behavior-policy forward pass. Once rollout generation is complete, the rollout engine can be released from memory, and $\mathcal{D}_k$ is passed to the optimization phase.

Phase 2: Policy Optimization with Large μ. The policy is then optimized on the static rollout dataset $\mathcal{D}_k$ for $\mu = B_{\text{train}}/B_{\text{mini}}$ updates before the next rollout stage. By making B_{train} large, μ-GRPO operates with μ on the order of a thousand, reusing the same behavior policy over many updates. For a token a_t in response y to prompt x, with stored behavior log-probability b_t, we compute the importance ratio as

$$\rho_t(\theta) = \exp\left(\log \pi_\theta(a_t \mid x, a_{<t}) - b_t\right).$$

Motivated by the analysis in Section 3, μ-GRPO applies *negative-advantage veto* (NAV) before the GRPO update. For each negative-advantage response y, the veto rule checks whether any token satisfies $\rho_t(\theta) < \tau_c$. We consider three veto scopes that contain the diagnosed non-trigger suffix $\mathcal{H}_\kappa$: $\mathcal{H}_\kappa$, $\mathcal{S}_\kappa$, and $\mathcal{S}_{\text{seq}}$. We use $\mathcal{S}_{\text{seq}}$ by default for simplicity and robustness, and ablate the scope choice

in Appendix C.2. Under this default, once a trigger is detected, the entire response is masked. Thus, in the objective below, $m_t(\theta)$ is zero for all tokens in a triggered negative-advantage response and equals one otherwise. The mask is recomputed at each update and is not differentiated through.

With this mask, we optimize a relaxed clipped GRPO objective with clipping range $[0, 5]$:

$$J_{\mu\text{-GRPO}}(\theta) = \mathbb{E}_{\mathcal{B}\sim\mathcal{D}_k}\left[\frac{1}{|\mathcal{B}|}\sum_{(x,y,A,b_{1:T})\in\mathcal{B}}\frac{1}{T}\sum_{t=1}^{T} m_t(\theta)\min\left(\rho_t(\theta)A, \text{clip}(\rho_t(\theta), 0, 5)A\right)\right].$$

The two components are complementary: relaxed clipping exposes the stale-rollout signal but collapses without NAV (Figure 2), while under the standard range $[0.8, 1.2]$, NAV is nearly inactive (0.004% of tokens vetoed) and leaves accuracy unchanged (Appendix C.4).

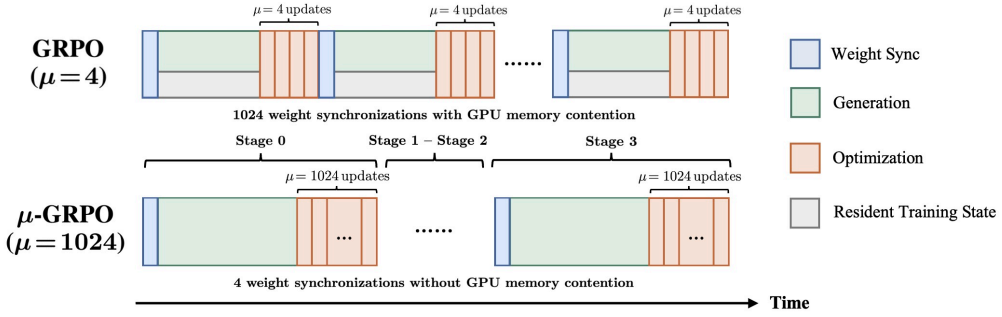

Figure 4: **System-level execution of standard GRPO and μ-GRPO.** Over 4096 model updates, standard GRPO with $\mu = 4$ requires 1024 rollout refreshes and weight synchronizations, while μ-GRPO with $\mu = 1024$ requires only four. Large-stage rollout generation reduces synchronization overhead and avoids GPU memory contention during generation.

Multi-Stage μ-GRPO. Both standard GRPO and μ-GRPO can be viewed as repeated generation–optimization cycles (Figure 4), differing in the scale of each cycle: standard GRPO refreshes the behavior policy after small rollout batches, keeping μ low, whereas μ-GRPO generates a large static rollout dataset and performs many updates on it. After stage k consumes $\mathcal{D}_k$, the updated policy is frozen as the rollout policy for stage $k+1$, and a fresh dataset $\mathcal{D}_{k+1}$ is generated. This design separates two timescales: within each stage, the rollout policy stays fixed for many updates to amortize generation; across stages, it is refreshed only a few times to improve rollout quality. At matched logging granularity, μ-GRPO shows per-update reward fluctuation comparable to GRPO (Figure 5). Within a stage, its training reward stays roughly flat because it is measured on the fixed dataset $\mathcal{D}_k$; it rises sharply at the first refresh and closely tracks GRPO thereafter. Meanwhile, evaluation accuracy of the current policy continues to improve within stages (Figure 1), so this sparse refresh schedule translates the large-μ efficiency gain into stable evaluation performance.

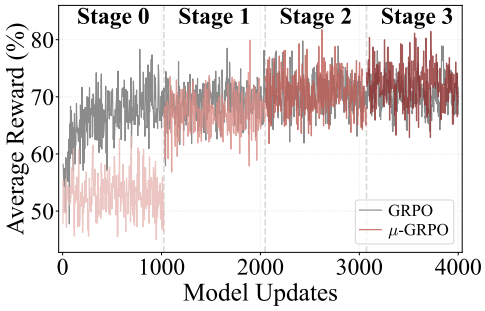

Figure 5: **Training-set reward** on DeepSeek-7B. Each point averages four consecutive updates for both methods, matching standard GRPO's once-per-rollout-phase logging.

System-Level Efficiency. Figure 4 compares the execution schedules of standard GRPO and μ-GRPO over 4096 model updates. Standard GRPO with $\mu = 4$ requires 1024 rollout refreshes and weight synchronizations between a high-throughput inference engine, such as vLLM (Kwon et al., 2023) or SGLang (Zheng et al., 2024), and a sharded training engine, such as FSDP (Zhao et al., 2023) or Megatron-LM (Shoeybi et al., 2019); each refresh can incur sharded-parameter materialization, transfer to the rollout engine, and inference-layout conversion. Frequent alternation also creates GPU memory contention, since systems often keep model parameters and optimizer states resident during rollout to switch quickly, reducing memory available for high-throughput inference. μ-GRPO reduces both costs: at $\mu = 1024$, it performs only four rollout stages, each with one weight synchronization. In each stage, the frozen rollout policy generates a large static dataset as a standalone phase, after which

the rollout engine is released and the trainer consumes the data using stored behavior log-probabilities. Because training state need not remain co-resident during rollout, more device memory is available for larger rollout batches and higher decoding parallelism. This reduces synchronization overhead and memory contention, enabling higher rollout throughput.

5 EXPERIMENTS

Models & Datasets. We evaluate μ-GRPO on five language models: Llama-3.2-3B-Instruct (Llama Team, 2024), Qwen2.5-Math-7B (Yang et al., 2024), Qwen3-Base-1.7B and 8B (Yang et al., 2025), and DeepSeek-7B (Guo et al., 2025). We use a context length of 4k for Qwen2.5-Math-7B, which is the maximum supported by this model, and 8k for all other models. For training, we adopt the DeepScaleR (Luo et al., 2025) mathematics dataset.

Training & Evaluation. We implement all methods in verl (Sheng et al., 2025), with vLLM (Kwon et al., 2023) for rollout generation and FSDP (Zhao et al., 2023) for actor optimization. Each configuration is trained for 4096 model update steps with batch size 32, giving the same rollout-sample budget. We evaluate GRPO under both low- and high-staleness settings ($\mu = 4$ and $\mu = 1024$), and set $\mu = 1024$ for μ-GRPO, M2PO (Zheng et al., 2026), and BAPO (Xi et al., 2026). μ-GRPO thus runs four rollout stages of 1024 updates each (Figure 1); high-staleness GRPO runs in the same staged pipeline, without relaxed clipping or negative-advantage veto. We also evaluate GPPO (Su et al., 2025), CISPO (MiniMax et al., 2025), and GSPO (Zheng et al., 2025a), which collapse or fall to the level of high-staleness GRPO at $\mu = 1024$ (Appendix D.3). For GRPO, we set $\epsilon = 0.2$ and omit the KL loss following prior work (Yu et al., 2025; Zheng et al., 2026). For M2PO and BAPO, we use the recommended configurations from their papers and released code (Appendix B).

We use a fully aligned wall-clock measurement protocol on identical hardware configurations with $4\times$H200 GPUs, including both rollout generation and policy optimization. For evaluation, we use five widely adopted math reasoning benchmarks: AMC23/24 (Art of Problem Solving, n.d.b), AIME24/25 (Art of Problem Solving, n.d.a), and MATH500 (Hendrycks et al., 2021; Lightman et al., 2024). Models are evaluated every 200 model updates, and we report the checkpoint with the best average performance across all benchmarks. More experimental details are provided in Appendix B.

5.1 MAIN RESULTS

We analyze the results in Table 1 along two dimensions: final reasoning performance and wall-clock efficiency, measured by rollout-generation time T_{rollout} and end-to-end training time T_{total}.

μ-GRPO vs. High-Staleness GRPO: Recovering performance under the same large-μ regime. Naively increasing GRPO to $\mu = 1024$ reduces rollout-generation and wall-clock time, but degrades accuracy. Under the same high-staleness setting, μ-GRPO recovers this performance loss while retaining the efficiency benefits of large-stage rollout execution. It improves average accuracy from 41.8% to 48.0% on Qwen2.5-Math-7B and from 60.2% to 68.6% on DeepSeek-7B. The gains are visible on harder benchmarks such as AIME24/25. This consistent gap shows that the stabilization mechanisms in μ-GRPO are critical for turning high-staleness data reuse into effective learning.

μ-GRPO vs. Standard GRPO and Off-Policy Baselines: Better performance–efficiency trade-off. Against standard GRPO ($\mu = 4$), μ-GRPO matches or improves average accuracy on four out of five models, with the largest gain on DeepSeek-7B (68.6% vs. 66.1%), while reducing T_{rollout} by $1.83\times$ and T_{total} by $1.53\times$ on average. Compared to M2PO at $\mu = 1024$, μ-GRPO achieves competitive accuracy (0.6 points higher on average) while reducing T_{rollout} by $1.87\times$ and T_{total} by $1.49\times$. BAPO remains stable at $\mu = 1024$ but trails μ-GRPO on all five models (by 2.1 points on average) while taking $1.59\times$ longer in T_{total}. M2PO and BAPO confirm that high-staleness rollouts can remain learnable, while μ-GRPO more directly converts this tolerance into rollout and wall-clock efficiency gains. Overall, μ-GRPO achieves a stronger performance–efficiency trade-off. On three additional benchmarks, Gaokao (Zhang et al., 2023), Minerva Math (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024), evaluated on the same checkpoints, μ-GRPO attains the best eight-benchmark average on four of the five models (Appendix D.2). Policy entropy on freshly sampled generations also shows no greater collapse than under standard GRPO (Appendix D.4).

Table 1: Performance (%) and efficiency comparison across five math reasoning benchmarks and five models. **Bold** marks the best Avg.; underline marks the fastest T_{total} among methods no worse than standard GRPO in Avg. Overall, μ-GRPO achieves the strongest performance–efficiency trade-off.

METHODS	μ	AMC23	AMC24	AIME24	AIME25	MATH500	AVG. ↑	T_{rollout} (h) ↓	T_{total} (h) ↓
LLAMA-3.2-3B-INSTRUCT									
GRPO	4	25.0	20.6	12.3	0.2	55.1	22.6	15.1	22.7
GRPO	1024	21.9	12.2	9.8	0.6	46.8	18.3	13.2	19.3
M2PO	1024	30.6	15.0	10.6	0.2	50.7	21.4	15.3	21.3
BAPO	1024	29.1	17.2	12.6	0.2	48.2	21.5	15.9	23.5
μ-GRPO	1024	29.4	17.7	14.0	0.5	52.8	**22.9**	12.7	<u>18.6</u>
QWEN2.5-MATH-7B									
GRPO	4	68.1	42.8	29.0	15.8	81.0	47.3	8.8	23.7
GRPO	1024	61.3	36.2	23.3	9.8	78.2	41.8	3.8	17.9
M2PO	1024	66.9	40.6	29.6	14.6	82.5	46.8	6.7	20.2
BAPO	1024	66.8	37.4	24.7	13.2	79.1	44.2	7.2	24.3
μ-GRPO	1024	63.1	47.2	31.5	16.5	81.7	**48.0**	4.7	<u>18.1</u>
QWEN3-BASE-1.7B									
GRPO	4	42.5	27.2	7.5	3.8	66.6	29.5	11.2	20.5
GRPO	1024	32.5	19.3	7.7	4.5	63.5	25.5	6.5	13.3
M2PO	1024	35.0	24.4	10.0	4.4	66.6	28.1	12.9	24.2
BAPO	1024	33.2	21.3	8.9	4.0	65.0	26.5	12.8	23.9
μ-GRPO	1024	40.8	24.9	7.5	6.5	68.7	**29.7**	5.2	<u>12.4</u>
QWEN3-BASE-8B									
GRPO	4	68.8	53.3	23.5	19.4	86.3	**50.3**	21.3	<u>58.2</u>
GRPO	1024	59.6	43.2	17.6	11.2	74.6	41.2	7.4	23.6
M2PO	1024	65.6	53.9	24.2	19.6	87.1	50.1	23.2	49.7
BAPO	1024	67.8	48.9	23.9	18.3	78.6	47.5	23.9	54.1
μ-GRPO	1024	69.4	51.1	21.5	17.4	81.8	48.2	7.9	25.9
DEEPSEEK-7B									
GRPO	4	91.4	66.7	46.8	33.2	92.4	66.1	26.0	61.9
GRPO	1024	82.6	63.5	39.5	26.8	88.7	60.2	22.3	54.5
M2PO	1024	90.6	69.4	50.8	35.0	93.0	67.8	27.9	68.3
BAPO	1024	87.7	70.2	49.5	36.9	91.1	67.1	23.6	67.8
μ-GRPO	1024	93.1	68.9	50.2	37.9	92.8	**68.6**	21.4	<u>51.3</u>

Evaluation Results on Coding Tasks. As a test beyond math reasoning, we train Qwen2.5-Coder-7B (Hui et al., 2024) on `code_contests` (Li et al., 2022) and evaluate on LiveCodeBench (Jain et al., 2025), using the main experimental setup with a minibatch size of 16. Starting from 8.2% accuracy, GRPO and μ-GRPO reach comparable accuracies of 39.3% and 38.6%. However, μ-GRPO reduces wall-clock training time from 98.9h to 45.7h, yielding a 2.16× speedup. This suggests that large-μ training's efficiency benefit transfers to coding without sacrificing performance.

Asynchronous Execution. Unlike the synchronous staged setting in Figure 4, we evaluate GRPO and μ-GRPO in a fully asynchronous setup with rollout workers and trainers running in parallel on separate devices. We use $4\times$H200 GPUs, split evenly between rollout workers and trainers. Rollout samples are streamed through a queue, and workers periodically receive synchronized policy parameters. With periodic synchronization, samples may lag behind the trainer policy; we cap the maximum lag for async μ-GRPO to match the $\mu = 1024$ staleness level of the synchronous setting. Async μ-GRPO uses the same infrastructure as async GRPO, but replaces the trainer-side update with relaxed clipping and negative-advantage veto. This tests whether μ-GRPO's staleness tolerance transfers from serial staged training to parallel asynchronous execution. As shown in Table 2, async μ-GRPO maintains comparable accuracy while reducing training time from 22.3h to 17.4h; the trainer idle ratio, defined as the fraction of total trainer step time spent waiting for rollout samples, drops from 24.2% to 0.6%.

Table 2: Asynchronous execution on Qwen2.5-Math-7B.

Method	Max Lag	Avg.↑	T_{total} (h)↓	Idle Ratio (%) ↓
GRPO	4	47.9	22.3	24.2
μ-GRPO	1024	47.7	**17.4**	**0.6**

5.2 Ablation Study

Table 3: **Fixed generation budget.** Varying batch size on a fixed rollout dataset.

Batch	μ	AMC23	AMC24	AIME24	AIME25	MATH500	Avg.
32	1259	61.9	40.6	24.8	13.8	80.0	**44.2**
64	629	63.1	39.4	23.3	13.3	79.7	43.8
128	314	61.3	36.1	21.5	13.5	80.7	42.6
256	157	62.5	32.8	23.1	11.9	79.3	41.9

Table 4: **Fixed optimization budget.** Varying μ with 4096 updates.

μ	AMC23	AMC24	AIME24	AIME25	MATH500	Avg.	#Stage
512	65.6	43.9	30.8	13.3	82.9	47.3	8
1024	63.1	47.2	31.5	16.5	81.7	**48.0**	4
2048	65.7	40.6	28.1	16.9	79.1	46.1	2
4096	66.9	37.2	26.0	16.0	82.3	45.7	1

Effect of Staleness with a Fixed Generation Budget. All ablations use Qwen2.5-Math-7B. To isolate the value of reusing stale rollout data, we first generate a fixed rollout dataset by sampling 8 responses for each prompt in the DeepScaleR training set, and then optimize only on it. Under this setting, the batch size determines both the number of updates and the effective staleness μ: smaller batches yield more updates but higher staleness, while larger batches reduce staleness but shorten training. Table 3 varies the batch size in $\{32, 64, 128, 256\}$. Batch size 32 ($\mu = 1259$) achieves the best average accuracy of 44.2%, while batch size 256 drops to 41.9%. This suggests that rollout data remain useful for many updates, and that additional updates can outweigh the cost of higher staleness.

Effect of Staleness with a Fixed Optimization Budget. Table 4 fixes the total number of model updates and varies the effective staleness μ by changing the rollout refresh frequency, or equivalently the number of generation–optimization stages. Performance peaks at $\mu = 1024$ (four stages) and declines as the number of stages decreases, to 46.1% with two stages and 45.7% with a single stage ($\mu = 4096$). This suggests that stale rollouts remain useful over many updates, but occasional refreshes are still important for maintaining rollout quality. We therefore adopt $\mu = 1024$ by default.

6 Related Work

We summarize key related work here and provide a more detailed discussion in Appendix F. Practical RLVR systems are partially off-policy due to rollout reuse (Shao et al., 2024; Hugging Face, 2025), rollout–training decoupling in modern RL systems (Sheng et al., 2025; Zhu et al., 2025; Fu et al., 2025; Noukhovitch et al., 2025; Zhong et al., 2025; He et al., 2025), and training–inference mismatch (Qi et al., 2025). Prior work stabilizes off-policy RLVR by modifying the clipping or trust region, including ratio clipping (Yu et al., 2025), approximate trust regions (Fu et al., 2025), sequence-level clipping (Zheng et al., 2025a), asymmetric clipping (Le Roux et al., 2025), gradient-preserving clipping (Su et al., 2025; MiniMax et al., 2025), and adaptive clipping bounds (Xi et al., 2026), or by masking tokens whose importance weights have a large batch-level second moment (Zheng et al., 2026). In contrast, we push GRPO to extreme rollout staleness, diagnose its failure mode as localized negative-advantage suffix drift, and stabilize training with a simple trigger–scope filter that vetoes post-trigger updates in negative-advantage responses. Our work complements synchronous / hybrid orchestration systems (Sheng et al., 2025; Zhu et al., 2025) and asynchronous RLVR systems (Fu et al., 2025; Zhong et al., 2025; Han et al., 2025; Noukhovitch et al., 2025): we study how to increase GRPO's algorithmic tolerance to rollout staleness, reducing synchronization pressure in both settings. μ-GRPO realizes this tolerance in a synchronous large-stage framework, where rollout generation and optimization are executed sequentially across stages rather than overlapped asynchronously.

7 Conclusion

In this work, we revisit the assumption that GRPO must remain in a low-staleness, near-on-policy regime. We find that stale rollouts retain useful learning signals far beyond this regime, and the instability of high-staleness training is localized to post-trigger suffix updates in negative-advantage responses. Motivated by this diagnosis, we propose μ-GRPO, a staged framework that decouples rollout generation from policy optimization and stabilizes training under extreme staleness via relaxed clipping and negative-advantage veto. Across multiple models and math reasoning benchmarks, μ-GRPO converts this staleness tolerance into a substantially improved performance–efficiency trade-off, delivering notable wall-clock speedups while largely preserving or improving final performance. Extending high-staleness training to broader domains and larger models remains future work.

AI USE STATEMENT

We used generative AI tools, including LLM-based chat assistants and AI coding assistants, in two limited ways: (1) for writing assistance, such as improving the grammar, readability, and presentation of the manuscript; and (2) as coding assistants (e.g., Cursor and Claude Code) for routine tasks such as debugging syntax errors and refactoring simple functions. Generative AI tools were not used for research ideation, algorithm design, experimental analysis, or drawing conclusions; all conceptual and scientific contributions are entirely the work of the authors. We have reviewed all AI-assisted content and take full responsibility for the final content of this work.

ETHICS STATEMENT

This work studies a training algorithm for reinforcement learning of language models. All experiments use publicly available pretrained models and publicly available datasets (DeepScaleR (Luo et al., 2025) for math reasoning and `code_contests` (Li et al., 2022) for coding), and evaluation uses public benchmarks. The work involves no human subjects and no collection of personal data. By reducing the cost of RL post-training, μ-GRPO may make reasoning-model development more accessible, but may also accelerate the development of more capable language models, whose downstream impacts depend on how the resulting systems are deployed; we encourage applications of high-staleness RL training to follow standard safety evaluation and responsible deployment practices.

REPRODUCIBILITY STATEMENT

Section 4 specifies the μ-GRPO objective, including the relaxed clipping range and the negative-advantage veto rule. Appendix B lists the models, datasets, training hyperparameters, efficiency-measurement protocol, and evaluation protocol for all main, coding, and asynchronous experiments; Appendix D.3 gives the configurations used for the additional off-policy baselines. All methods are implemented in the open-source verl framework (Sheng et al., 2025) with vLLM (Kwon et al., 2023). Hyperparameter sensitivity is reported in Appendix C.5, and complete statements and proofs of the theoretical results are given in Appendix E.

REFERENCES

Art of Problem Solving. AIME problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`, n.d.a.

Art of Problem Solving. AMC problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AMC_Problems_and_Solutions`, n.d.b.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *NeurIPS*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025.

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. AsyncFlow: An asynchronous streaming RL framework for efficient LLM post-training. *arXiv:2507.01663*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating LLM reinforcement learning with RhymeRL. *arXiv:2508.18588*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021.

Hugging Face. Open R1: A fully open reproduction of deepseek-r1. `https://github.com/huggingface/open-r1`, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv:2409.12186*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *ICLR*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

Nicolas Le Roux, Marc Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alexandre Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Tóth, and Sam Work. Tapered off-policy REINFORCE - stable and efficient reinforcement learning for large language models. In *NeurIPS*, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *NeurIPS*, 2022.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 2022.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Zhuo Jiang. When speed kills stability: Demystifying RL collapse from the training-inference mismatch. `https://richardli.xyz/rl-collapse`, 2025.

Llama Team. The Llama 3 herd of models. *arXiv:2407.21783*, 2024.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B model by scaling RL. `https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2`, 2025.

MiniMax, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv:2506.13585*, 2025.

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. In *ICLR*, 2025.

OpenAI. OpenAI o1 system card. *arXiv:2412.16720*, 2024.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via FP16. *arXiv:2510.26788*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

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. *arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *EuroSys*, 2025.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv:1909.08053*, 2019.

Zhenpeng Su, Leiyu Pan, Xue Bai, Dening Liu, Guanting Dong, Jiaming Huang, Minxuan Lv, Wenping Hu, Fuzheng Zhang, Kun Gai, and Guorui Zhou. Klear-Reasoner: Advancing reasoning capability via gradient-preserving clipping policy optimization. *arXiv:2508.07629*, 2025.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. In *NeurIPS*, 2025.

Zhiheng Xi, Xin Guo, Yang Nan, Enyu Zhou, Junrui Shen, Wenxiang Chen, Jiaqi Liu, Jixuan Huang, Xun Deng, Zhihao Zhang, Honglin Guo, Zhikai Lei, Miao Zheng, Guoteng Wang, Peng Sun, Rui Zheng, Hang Yan, Tao Gui, Qi Zhang, and Xuanjing Huang. BAPO: Stabilizing off-policy reinforcement learning for LLMs via balanced policy optimization with adaptive clipping. In *ICLR*, 2026.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *NeurIPS*, 2025.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, Xipeng Qiu, Tianxiang Sun, Peng Li, Shiqiao Meng, Yanjun Zheng, Jun Zhan, Zhangyue Yin, Xiannian Hu, Guofeng Quan, and Qixiang Wang. Evaluating the performance of large language models on GAOKAO benchmark. *arXiv:2305.12474*, 2023.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *PVLDB*, 2023.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv:2507.18071*, 2025a.

Haizhong Zheng, Yang Zhou, Brian R. Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for LLM reasoning via selective rollouts. In *NeurIPS*, 2025b.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before collapse: How far can off-policy RL reach with stale data on LLMs? In *ICLR*, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *NeurIPS*, 2024.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, et al. StreamRL: Scalable, heterogeneous, and elastic RL for LLMs with disaggregated stream generation. *arXiv:2504.15930*, 2025.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. `https://github.com/THUDM/slime`, 2025.

A Overview

This appendix provides details and analyses that support the main text. Appendix B describes the experimental setup, including model and dataset details, training hyperparameters, efficiency measurement, evaluation protocols, and the coding and asynchronous experiment settings. Appendix C provides additional analyses of negative-advantage veto and relaxed clipping, including veto-rate statistics, negative-advantage veto scope ablations, a volume-matched control isolating trigger position, a NAV $\times$ clipping factorial, stabilization hyperparameter sensitivity, and low-staleness μ-GRPO. Appendix D reports variance across random seeds, evaluation on three additional benchmarks, a comparison with other off-policy RLVR methods under the same high-staleness setting, and policy entropy during training. Appendix E formalizes the prefix-support mismatch mechanism behind retained harmful suffixes. Appendix F discusses additional related work.

B Detailed Experimental Settings

B.1 Main Experiment Settings

Models & Dataset. All experiments are conducted on five backbone language models: Llama-3.2-3B-Instruct (Llama Team, 2024), Qwen2.5-Math-7B (Yang et al., 2024), Qwen3-Base-1.7B and 8B (Yang et al., 2025), and DeepSeek-7B (Guo et al., 2025). DeepSeek-7B denotes the Qwen2.5-Math-7B model distilled from DeepSeek-R1. Following model specifications, we use a context length of 4k tokens for Qwen2.5-Math-7B, which is the maximum supported by this model, and 8k tokens for all other models. All models are trained on the DeepScaleR mathematics dataset (Luo et al., 2025). For each training instance, we use a unified instruction-style prompt template. The model is instructed to reason step by step and return the final answer enclosed in `\boxed{}` tags, which enables reliable answer extraction during training and evaluation.

Prompt Template

Please solve the following math problem: {{Question Description}}. The assistant first thinks about the reasoning process step by step and then provides the user with the answer. Return the final answer in `\boxed{}` tags, for example `\boxed{1}`. Let's solve this step by step.

Training. All methods are implemented in verl (Sheng et al., 2025), with vLLM (Kwon et al., 2023) for rollout generation and FSDP (Zhao et al., 2023) for actor optimization. Policy optimization is performed on the actor network only. We use AdamW with a constant learning rate of 1×10^{-6}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay 0.01. Following prior work (Yu et al., 2025; Zheng et al., 2026), we omit the explicit KL-divergence regularization term. Rewards are binary: 1.0 if the extracted answer is correct, and 0.0 otherwise. All main experiments use random seed 42; Appendix D.1 reports results over additional seeds. For rollout generation, we use temperature 1.0 and sample 8 responses per prompt. Each configuration is trained for 4096 model updates with batch size 32, giving the same rollout-sample budget across methods. We evaluate GRPO under both low- and high-staleness settings, $\mu=4$ and $\mu=1024$, and set $\mu=1024$ for μ-GRPO, M2PO, and BAPO; μ-GRPO thus uses four rollout stages, each followed by 1024 model updates. The high-staleness GRPO baseline uses the same two-phase large-stage rollout pipeline as μ-GRPO, but optimizes the standard GRPO objective with default clipping and no negative-advantage veto. This isolates the effect of the proposed stabilization mechanism from staged rollout execution. For GRPO, we set $\epsilon=0.2$. For μ-GRPO, we use the sequence-level negative-advantage veto scope $\mathcal{S}_{\text{seq}}$, a conservative superset of the harmful suffix $\mathcal{H}_\kappa$ identified in Section 3, and set $\tau_c=10^{-4}$; Appendix C.2 ablates alternative scopes. For M2PO, we use the recommended second-moment threshold of 0.04. For

BAPO, we use the recommended configuration: $\rho_0 = 0.5$, an adaptive lower-bound range of $[0.6, 0.9]$, an adaptive upper-bound range of $[1.2, 3.0]$, and adjustment steps of 0.05 and 0.1.

Efficiency Measurement. All efficiency experiments use identical $4\times$H200 GPU configurations without parameter offloading. We report two timing metrics. T_{rollout} denotes the cumulative wall-clock time of all rollout-generation phases across training. T_{total} denotes the end-to-end wall-clock training time, including rollout generation, policy optimization, synchronization, data movement, and scheduling overhead. This aligned timing protocol is used for all methods in Table 1.

Evaluation. We evaluate trained models on AMC23/24 (Art of Problem Solving, n.d.b), AIME24/25 (Art of Problem Solving, n.d.a), and MATH500 (Hendrycks et al., 2021; Lightman et al., 2024). Models are evaluated every 200 model updates, and we report the checkpoint with the best average performance across all benchmarks. Diagnostic runs (Figures 2 and 3 and Appendix C.3) are evaluated every 100 model updates, and the additional off-policy baselines in Appendix D.3 every 50 updates. Evaluation uses pass@1 accuracy. For AIME24/25, pass@1 is averaged over 16 sampled generations; for AMC23/24 and MATH500, pass@1 is averaged over 4 sampled generations. For Gaokao, Minerva Math, and OlympiadBench (Appendix D.2), pass@1 is averaged over 4 sampled generations. All evaluations use the same context length as training.

B.2 Additional Experiment Settings

Coding Experiment Details. For the coding experiment, we train Qwen2.5-Coder-7B (Hui et al., 2024) on the `code_contests` dataset (Li et al., 2022) and evaluate on LiveCodeBench (Jain et al., 2025). We use an 8k context length and report pass@4 accuracy. The training setup follows the main experiments except that the per-update batch size is 16. Both GRPO and μ-GRPO are trained for 4096 updates; μ-GRPO uses four rollout stages with $\mu = 1024$. Wall-clock time is measured with the aligned protocol on $4\times$H200 GPUs.

Asynchronous Experiment Details. For the asynchronous experiment, we use the fully asynchronous policy trainer in verl, where rollout workers and trainers are placed on separate devices and run concurrently. We use $4\times$H200 GPUs, with two allocated to rollout workers and two to trainers. Rollout samples are streamed through a message queue, and policy parameters are periodically synchronized from the trainer to the rollout workers. We synchronize parameters every 4 trainer updates for both async runs. Async GRPO uses no additional rollout staleness allowance, while async μ-GRPO allows up to 255 synchronization intervals of stale rollouts. This gives a maximum rollout lag of $(1 + 255) \times 4 = 1024$ updates, matching the $\mu = 1024$ synchronous setting. Trainer idle ratio is measured as the fraction of total trainer step time spent waiting for rollout samples.

C Additional Analysis of Negative-Advantage Veto and Relaxed Clipping

This section provides additional analyses and ablations for the two stabilization components of μ-GRPO: negative-advantage veto (NAV) and relaxed clipping. We first analyze how often NAV vetoes token-level updates during training, then compare different NAV masking scopes, isolate the effect of trigger position from masking volume, separate NAV from relaxed clipping, study the sensitivity to the NAV threshold and clipping upper bound, and finally test whether the μ-GRPO update rule remains compatible with the standard low-staleness regime.

C.1 Fraction of Tokens Vetoed by NAV

To better understand how much training signal is filtered by NAV, we report the fraction of tokens vetoed during μ-GRPO training. Recall that under the sequence-level NAV rule, for a negative-advantage response, once a token's importance ratio falls below the threshold τ_c, NAV masks all tokens in that response, including those preceding the trigger (scope $\mathcal{S}_{\text{seq}}$). We define the NAV-vetoed token fraction as the number of tokens masked by NAV divided by the total number of tokens in the optimization batch. Since NAV masks are recomputed online during optimization, this fraction can

vary across updates within a stage. These statistics correspond to the main μ-GRPO runs reported in Table 1, all of which use $\mathcal{S}_{\text{seq}}$.

Table 5: Mean NAV-vetoed token fraction in each stage, computed as vetoed tokens divided by all tokens in the optimization batch. Avg. is the unweighted average across stages.

Model	Stage 0	Stage 1	Stage 2	Stage 3	Veto Avg.
Llama-3.2-3B-Instruct	6.08%	3.28%	0.78%	2.42%	3.14%
Qwen2.5-Math-7B	15.24%	0.27%	0.27%	0.32%	4.03%
Qwen3-Base-1.7B	7.07%	0.22%	0.23%	0.42%	1.99%
Qwen3-Base-8B	3.78%	0.81%	2.83%	4.72%	3.04%
DeepSeek-7B	0.61%	0.01%	0.01%	0.05%	0.17%

Table 5 reports the mean NAV-vetoed token fraction in each stage for all five models, along with the unweighted average across stages. Overall, the average vetoed fraction remains small, ranging from 0.17% on DeepSeek-7B to 4.03% on Qwen2.5-Math-7B. This shows that NAV removes only a limited subset of updates rather than discarding a large portion of the rollout data. The result supports the view that NAV primarily suppresses harmful off-support negative-advantage updates while preserving most of the training signal.

For most models, the vetoed fraction is especially small in later stages. For Qwen2.5-Math-7B, the mean vetoed token fraction is 15.24% in Stage 0 but drops to below 0.4% in all subsequent stages. Similarly, Qwen3-Base-1.7B decreases from 7.07% in Stage 0 to below 0.5% afterward, while DeepSeek-7B remains below 0.7% across all stages. These trends suggest that NAV is most active when the model is early in training or when the rollout distribution is more prone to off-support negative-advantage updates, and becomes increasingly sparse once the staged training process stabilizes. Qwen3-Base-8B is an exception: its vetoed fraction remains above 0.8% in every stage and peaks at 4.72% in Stage 3.

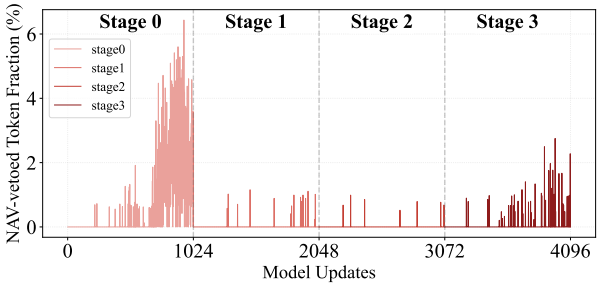

Figure 6: NAV-vetoed token fraction during μ-GRPO training on DeepSeek-7B.

To complement the averages in Table 5, Figure 6 visualizes the NAV-vetoed token fraction throughout training for DeepSeek-7B across the four stages. The curve shows that the vetoed token fraction remains close to zero for most updates, with only occasional peaks within individual stages. Overall, these results show that NAV stabilizes high-staleness training through sparse, response-level intervention.

Response-level view. Because the default scope $\mathcal{S}_{\text{seq}}$ masks an entire triggered response, the fraction of vetoed responses directly measures how many rollouts NAV discards. Table 6 reports, for each stage, the fraction of response-update instances that have negative advantage and contain at least one trigger token. Each minibatch contains 256 responses (32 prompt groups $\times$ 8 responses), so, for example, 12.30% corresponds to about 31 responses per update. For four of the five models, triggering is most frequent in Stage 0 and becomes much sparser after the first rollout refresh. NAV therefore discards only a small subset of responses rather than broadly removing training data.

C.2 NAV Scope Ablation

In the main experiments, we use the sequence-level NAV scope $\mathcal{S}_{\text{seq}}$ as the default choice. For a negative-advantage response, once a token with importance ratio below τ_c is detected, it vetoes all tokens in that response. The diagnostic experiments in Section 3 identify the non-trigger suffix $\mathcal{H}_\kappa$ as

Table 6: Fraction of response-update instances vetoed by NAV (negative-advantage responses containing at least one trigger token) in each stage. Avg. is the unweighted average across stages.

Model	Stage 0	Stage 1	Stage 2	Stage 3	Avg.
Llama-3.2-3B-Instruct	7.23%	3.11%	0.80%	2.38%	3.38%
Qwen2.5-Math-7B	12.30%	0.38%	0.34%	0.29%	3.33%
Qwen3-Base-1.7B	8.50%	0.23%	0.17%	0.38%	2.32%
Qwen3-Base-8B	2.36%	0.71%	2.88%	2.92%	2.22%
DeepSeek-7B	0.49%	0.01%	0.01%	0.02%	0.13%

the critical subset that must be removed to prevent collapse. This suggests that masking scopes that include $\mathcal{H}_\kappa$, such as $\mathcal{S}_\kappa$ and $\mathcal{S}_{\text{seq}}$, should also stabilize high-staleness optimization.

To examine the effect of the veto scope, we compare three NAV scopes under the main experimental setting: 4096 model updates with four rollout stages, corresponding to $\mu = 1024$ updates per stage. The scopes are the non-trigger suffix $\mathcal{H}_\kappa$, the suffix $\mathcal{S}_\kappa$, and the sequence-level scope $\mathcal{S}_{\text{seq}}$. We report the fraction of vetoed tokens in each stage, computed as the number of vetoed tokens divided by the total number of tokens in the optimization batch. We also report the average veto rate across stages and the average accuracy across the five math benchmarks.

Table 7: NAV scope ablation on Qwen2.5-Math-7B. Veto rates are reported per stage.

NAV Scope	Stage 0	Stage 1	Stage 2	Stage 3	Veto Avg.	Avg. Acc.
$\mathcal{H}_\kappa$	11.70%	0.18%	0.21%	0.17%	3.07%	47.9
$\mathcal{S}_\kappa$	13.40%	0.22%	0.23%	0.22%	3.52%	47.6
$\mathcal{S}_{\text{seq}}$	15.24%	0.27%	0.27%	0.32%	4.03%	**48.0**

All three scopes remain stable and achieve similar average accuracy. The minimal scope $\mathcal{H}_\kappa$ vetoes the fewest tokens, whereas $\mathcal{S}_{\text{seq}}$ vetoes the most because it removes the entire negative-advantage response once a trigger is detected. The comparison suggests that the additional prefix tokens retained by $\mathcal{H}_\kappa$ and $\mathcal{S}_\kappa$ provide limited benefit once a trigger has appeared. Since the response has negative advantage and has already crossed an off-support boundary under the current policy, the remaining useful contribution from the pre-trigger prefix is small compared with the risk of continuing to update on the same trajectory. Consistent with this view, the more conservative sequence-level scope achieves the best average accuracy despite vetoing more tokens. We therefore use $\mathcal{S}_{\text{seq}}$ in the main experiments because it is simple, robust, and slightly outperforms the other scopes in this ablation.

C.3 Isolating Trigger Position from Masking Volume

The scope comparison in Figure 3 and Table 7 shows that masks containing the post-trigger suffix $\mathcal{H}_\kappa$ stabilize training. However, these scopes also remove many negative-advantage tokens, and when the first trigger appears early in a response, $\mathcal{H}_\kappa$, $\mathcal{S}_\kappa$, and $\mathcal{S}_{\text{seq}}$ all cover most of it. Their stability could therefore reflect generic downweighting of a highly off-policy negative-advantage sample rather than a suffix-specific effect. Since masking volume and position covary across these scopes, Table 7 alone cannot separate the two explanations. This subsection first reports where triggers occur and then presents a control that fixes masking volume by construction.

Where does the first trigger occur? We measure the normalized first-trigger position $\kappa(y)/T$ for all triggered negative-advantage response-update instances of Qwen2.5-Math-7B in Stage 0, where $\kappa(y)$ is the first token with $A_t < 0$ and $\rho_t < \tau_c = 10^{-4}$, and T is the response length. Table 8 shows that triggers are often early, particularly at the beginning of Stage 0, but shift later as training progresses and are not confined to the response prefix.

Volume-matched pre- versus post-trigger retention. We run Qwen2.5-Math-7B in Stage 0 with relaxed clipping $[0, 5]$. For each triggered negative-advantage response y, let $m(y)$ be the smaller of the number of tokens preceding $\kappa(y)$ and the number of non-trigger tokens following it. We compare two settings:

Table 8: Normalized first-trigger position $\kappa(y)/T$ of triggered negative-advantage responses during Stage 0 on Qwen2.5-Math-7B.

Stage-0 updates	Mean κ/T	Median κ/T	Fraction with $\kappa/T > 0.5$
0–256	0.19	0.17	5%
256–512	0.26	0.22	11%
512–768	0.33	0.24	18%
768–1024	0.36	0.41	23%

- **Pre-trigger retention:** retain the $m(y)$ non-trigger tokens immediately preceding $\kappa(y)$ and veto all other tokens of y.
- **Post-trigger retention:** retain the first $m(y)$ non-trigger tokens following $\kappa(y)$ and veto all other tokens of y.

All other responses are unchanged. The two settings therefore retain exactly the same number of negative-advantage token updates per triggered response and differ only in whether the retained tokens lie before or after the first low-ratio trigger.

Table 9: Volume-matched retention control on Qwen2.5-Math-7B (Stage 0, clipping $[0, 5]$). Average accuracy (%) over the five math benchmarks.

Setting	@0	@100	@200	@300	@400
Pre-trigger retention	38.7	39.5	40.7	41.0	41.2
Post-trigger retention	38.7	39.2	39.9	40.1	17.6

As shown in Table 9, the two settings track closely through update 300, after which post-trigger retention collapses to 17.6% while pre-trigger retention remains stable at 41.2%. The negative-advantage importance-ratio diagnostic $\mathbb{E}[\rho_t \mid A_t < 0]$ deteriorates only under post-trigger retention, matching the failure signature of the no-mask baseline. Because retained update volume is identical by construction, generic downweighting cannot account for the divergence; the position of the retained updates relative to the low-support boundary does. This provides direct evidence for the post-trigger failure mode described in Section 3.3 and formalized in Appendix E.

C.4 Isolating NAV from Relaxed Clipping

μ-GRPO combines two components, relaxed clipping and NAV. The main text shows that relaxed clipping without NAV collapses (Figure 2), and Table 1 reports GRPO with standard clipping [0.8,1.2] at $\mu = 1024$ without NAV. To complete the factorial, we additionally run NAV under the standard clipping range $[0.8, 1.2]$ on Qwen2.5-Math-7B at $\mu = 1024$, changing only the clipping range and whether NAV is enabled.

Table 10: NAV $\times$ clipping factorial on Qwen2.5-Math-7B at $\mu = 1024$. Average accuracy over the five math benchmarks.

Clipping range	NAV	Stability	Avg. (%)	NAV veto rate
$[0.8, 1.2]$	No	Stable, plateaus	41.8	–
$[0.8, 1.2]$	Yes	Stable, plateaus	41.7	0.004%
$[0, 5]$	No	Collapses	–	–
$[0, 5]$	Yes	Stable	**48.0**	4.03%

Table 10 shows that under standard clipping, NAV is effectively inactive, vetoing only 0.004% of tokens, and changes neither stability nor accuracy. This is expected: once a negative-advantage token's ratio falls below 0.8, the lower clipping bound zeroes its gradient, so ratios rarely approach the trigger threshold τ_c. Standard clipping is stable but plateaus at 41.8%, while relaxed clipping restores useful stale-rollout gradients but collapses without NAV. Only the combination is both stable and high-performing, reaching 48.0%. Relaxed clipping thus recovers the high-staleness learning signal, while NAV suppresses the destructive negative-advantage updates that this relaxation exposes.

C.5 Stabilization Hyperparameter Sensitivity

We study the sensitivity of μ-GRPO to two stabilization-related hyperparameters: the NAV threshold τ_c and the relaxed clipping upper bound. All experiments follow the main experimental setting: Qwen2.5-Math-7B is trained for 4096 model updates with four rollout stages, corresponding to $\mu = 1024$ updates per stage. We report the average accuracy across the five math benchmarks.

NAV Threshold Sensitivity. The NAV threshold τ_c determines when a negative-advantage response is treated as crossing the off-support boundary and triggers sequence-level vetoing. Since this threshold is intended to detect extreme low-ratio events, it should be sufficiently small; an overly large threshold may trigger NAV prematurely and remove useful updates.

Table 11: NAV threshold sensitivity.

NAV threshold τ_c	Avg. Acc.
10^{-1}	45.2
10^{-2}	46.4
10^{-4}	**48.0**
10^{-6}	47.6

We sweep τ_c over $\{10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}\}$. As shown in Table 11, $\tau_c = 10^{-4}$ achieves the best average accuracy of 48.0%, while 10^{-6} remains close at 47.6%. In contrast, larger thresholds degrade performance, with 10^{-1} dropping to 45.2%. These results suggest that NAV is robust once the threshold is small enough, but setting it too large can over-veto negative-advantage responses before they truly indicate an off-support boundary.

Clipping Upper Bound Sensitivity. We also ablate the upper bound of the relaxed clipping interval. In high-staleness training, the standard GRPO clipping range can overly suppress useful gradients from stale but still informative rollouts. μ-GRPO therefore uses a relaxed clipping range, while relying on NAV to remove destabilizing negative-advantage suffix updates. With the staged pipeline fixed, we vary the clipping upper bound over $\{3, 5, 10, +\infty\}$, where $+\infty$ denotes no upper clipping. Table 12 shows that an upper bound of 5 achieves the best average accuracy, reaching 48.0%. A tighter bound of 3 slightly underperforms at 47.3%, suggesting that it still restricts some useful stale-rollout gradients. Increasing the bound to 10 gives comparable performance at 47.8%. Removing the upper bound entirely leads to a clear drop to 43.6%, showing that although relaxed clipping is beneficial, some upper bound control remains necessary for stable optimization. Thus, a moderately relaxed upper bound best balances useful stale-rollout gradients and stability.

Table 12: Clipping upper bound sensitivity.

Clipping upper bound	Avg. Acc.
3	47.3
5	48.0
10	47.8
$+\infty$	43.6

C.6 Low-Staleness μ-GRPO

We further test whether the μ-GRPO update rule is compatible with the standard low-staleness regime. Specifically, we run μ-GRPO with $\mu = 4$, matching the rollout refresh frequency of standard GRPO while enabling the relaxed clipping range $[0, 5]$ and NAV. This isolates the effect of the update rule from large-staleness rollout reuse. We report the average accuracy across the five math benchmarks and the average token clipping fraction during training.

As shown in Table 13, low-staleness μ-GRPO slightly improves average accuracy over standard GRPO, from 47.3% to 47.7%. It also reduces the average clipping fraction from 4×10^{-4} to 8×10^{-7}, indicating that the relaxed clipping rule avoids unnecessary clipping when rollout staleness is small. In this low-staleness setting, the NAV veto fraction remains zero throughout training, so the mild gain is primarily attributable to relaxed clipping rather than NAV intervention. These results show

Table 13: Low-staleness μ-GRPO on Qwen2.5-Math-7B.

METHODS	μ	AMC23	AMC24	AIME24	AIME25	MATH500	AVG.	CLIP FRAC
GRPO	4	68.1	42.8	29.0	15.8	81.0	47.3	4×10^{-4}
μ-GRPO	4	69.2	42.1	30.1	14.9	82.3	**47.7**	8×10^{-7}

that the μ-GRPO update is compatible with fresh-data learning, and suggest that the standard GRPO clipping range can be overly conservative even when rollouts are relatively fresh.

This result should not be interpreted as a comparison between on-policy and off-policy training. Rather, it shows that the optimization rule itself matters: standard low-staleness GRPO is not necessarily the strongest GRPO-style update. In the main high-staleness setting, μ-GRPO combines this more permissive update rule with large-stage rollout reuse and NAV-based stabilization, enabling stable and efficient learning under much larger policy drift.

D EXTENDED EVALUATION

This section reports variance across random seeds, extends the evaluation of μ-GRPO to three additional held-out benchmarks, compares it with other off-policy RLVR objectives under the same high-staleness setting, and tracks policy entropy during training.

D.1 VARIANCE ACROSS RANDOM SEEDS

To assess run-to-run variability, we repeat standard GRPO ($\mu = 4$) and μ-GRPO ($\mu = 1024$) on Qwen2.5-Math-7B with three random seeds, keeping all other settings identical to Table 1. We use seeds 42, 43, and 44; all runs in Table 1 use seed 42. For each seed, Avg. is taken at the checkpoint with the best average accuracy, as in the main experiments.

Table 14: Results over three random seeds on Qwen2.5-Math-7B for standard GRPO ($\mu = 4$) and μ-GRPO ($\mu = 1024$). The last row of each method reports the mean over seeds for individual benchmarks and mean $\pm$ std for Avg. and T_{total}.

METHODS	SEED	AMC23	AMC24	AIME24	AIME25	MATH500	AVG. ↑	T_{total}(h) ↓
GRPO	42	68.1	42.8	29.0	15.8	81.0	47.3	23.7
	43	67.0	42.7	28.7	15.4	81.1	47.0	23.9
	44	67.4	42.5	27.7	15.1	80.2	46.6	22.9
	Mean	67.5	42.7	28.5	15.4	80.8	47.0±0.4	23.5±0.5
μ-GRPO	42	63.1	47.2	31.5	16.5	81.7	48.0	18.1
	43	63.9	45.9	32.5	16.2	81.6	48.0	17.3
	44	62.2	47.6	32.9	15.4	80.9	47.8	17.4
	Mean	63.1	46.9	32.3	16.0	81.4	47.9±0.1	17.6±0.4

Table 14 shows that the seed-42 results in Table 1 are representative. Across seeds, μ-GRPO attains 47.9 ± 0.1 average accuracy, compared with 47.0 ± 0.4 for standard GRPO; the 0.9-point gap exceeds the seed-to-seed standard deviation of either method, and μ-GRPO varies less across seeds. Its efficiency advantage is consistent, with T_{total} of 17.6 ± 0.4 h versus 23.5 ± 0.5 h for standard GRPO.

D.2 ADDITIONAL BENCHMARKS

Following the evaluation protocol of M2PO (Zheng et al., 2026), we additionally evaluate all methods in Table 1 on Gaokao (Zhang et al., 2023), Minerva Math (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). We use the same checkpoints as in Table 1, with no retraining or checkpoint reselection. Avg.$_5$ is copied from Table 1, and Avg.$_8$ averages all eight benchmarks.

As shown in Table 15, the extended evaluation preserves the conclusions of Table 1: μ-GRPO attains the best Avg.$_8$ on four of the five models while remaining faster than both GRPO ($\mu = 4$) and M2PO

Table 15: Performance (%) on three additional benchmarks using the checkpoints of Table 1. Avg.$_5$ is copied from Table 1; Avg.$_8$ averages all eight benchmarks. **Bold** marks the best Avg.$_8$ per model.

METHODS	μ	AVG.$_5$ ↑	GAOKAO	MINERVA	OLYMPIAD	AVG.$_8$ ↑	T_{total} (h) ↓
LLAMA-3.2-3B-INSTRUCT							
GRPO	4	22.6	41.3	24.2	22.9	25.2	22.7
GRPO	1024	18.3	39.1	16.1	16.8	20.4	19.3
M2PO	1024	21.4	45.6	20.4	18.8	24.0	21.3
BAPO	1024	21.5	43.3	21.5	21.0	24.1	23.5
μ-GRPO	1024	22.9	45.7	23.1	21.5	**25.6**	18.6
QWEN2.5-MATH-7B							
GRPO	4	47.3	62.2	33.1	43.5	46.9	23.7
GRPO	1024	41.8	57.0	30.1	38.0	41.7	17.9
M2PO	1024	46.8	63.8	32.2	41.6	46.5	20.2
BAPO	1024	44.2	60.1	30.0	39.9	43.9	24.3
μ-GRPO	1024	48.0	61.1	33.4	41.7	**47.0**	18.1
QWEN3-BASE-1.7B							
GRPO	4	29.5	54.8	27.6	28.8	32.4	20.5
GRPO	1024	25.5	48.9	23.8	24.6	28.1	13.3
M2PO	1024	28.1	56.0	30.8	31.0	32.3	24.2
BAPO	1024	26.5	54.2	33.9	30.1	31.3	23.9
μ-GRPO	1024	29.7	56.4	29.5	29.3	**33.0**	12.4
QWEN3-BASE-8B							
GRPO	4	50.3	70.8	39.3	51.1	51.6	58.2
GRPO	1024	41.2	65.1	33.1	41.8	43.3	23.6
M2PO	1024	50.1	73.5	39.5	51.7	**51.9**	49.7
BAPO	1024	47.5	71.1	36.8	47.8	49.2	54.1
μ-GRPO	1024	48.2	72.9	38.0	53.6	50.7	25.9
DEEPSEEK-7B							
GRPO	4	66.1	80.1	43.2	60.6	64.3	61.9
GRPO	1024	60.2	74.4	39.8	54.7	58.8	54.5
M2PO	1024	67.8	78.0	45.3	58.1	65.0	68.3
BAPO	1024	67.1	79.3	43.1	55.8	64.2	67.8
μ-GRPO	1024	68.6	80.5	44.1	60.2	**66.0**	51.3

on all five. On Qwen3-Base-8B, M2PO and GRPO ($\mu = 4$) lead by 1.2 and 0.9 points, but require 49.7 h and 58.2 h compared with 25.9 h for μ-GRPO, consistent with the performance–efficiency trade-off reported in the main text.

D.3 COMPARISON WITH OTHER OFF-POLICY RLVR METHODS

Several recent objectives aim to stabilize RLVR under off-policy data (Section 6). To test whether they transfer to the extreme staleness regime of μ-GRPO, we port GPPO (Su et al., 2025), CISPO (MiniMax et al., 2025), and GSPO (Zheng et al., 2025a) into our pipeline and train Qwen2.5-Math-7B on DeepScaleR with the same staged infrastructure, minibatch size, $\mu = 1024$, and 4096-update budget, changing only each method's objective and hyperparameters. We report the eight benchmarks of Appendix D.2. We follow the recommended or standard configurations from the original papers and released repositories:

- **GPPO:** $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$, i.e., clipping range $[0.8, 1.28]$.
- **CISPO:** clipping threshold 0.2; since the original paper does not specify a threshold, we use the standard PPO value.
- **GSPO:** sequence-level clipping with lower and upper bounds of 3×10^{-4} and 4×10^{-4}, as recommended in the released repository.

Table 16: Performance (%) and efficiency of off-policy RLVR methods on Qwen2.5-Math-7B at $\mu = 1024$ (standard GRPO at $\mu = 4$ shown for reference). GRPO, M2PO, BAPO, and μ-GRPO rows are from Table 1. **Bold** marks the best Avg.

METHODS	μ	AMC 23/24	AIME 24/25	MATH500	GAOKAO	MINERVA	OLYMPIAD	AVG.$_5$ ↑	AVG.$_8$ ↑	T_{total}(h) ↓
GRPO	4	68.1/42.8	29.0/15.8	81.0	62.2	33.1	43.5	47.3	46.9	23.7
GRPO	1024	61.3/36.2	23.3/9.8	78.2	57.0	30.1	38.0	41.8	41.7	17.9
M2PO	1024	66.9/40.6	29.6/14.6	82.5	63.8	32.2	41.6	46.8	46.5	20.2
BAPO	1024	66.8/37.4	24.7/13.2	79.1	60.1	30.0	39.9	44.2	43.9	24.3
GPPO	1024	collapsed: Avg.$_8$ 14.1 at update 50; run terminated								
CISPO	1024	collapsed: Avg.$_8$ 13.0 at update 100; run terminated								
GSPO	1024	61.3/35.6	23.1/12.1	77.5	57.7	29.4	38.8	41.9	41.9	18.4
μ-GRPO	1024	63.1/47.2	31.5/16.5	81.7	61.1	33.4	41.7	**48.0**	**47.0**	18.1

Table 16 shows that under matched extreme staleness, GPPO and CISPO collapse rapidly, reaching 14.1 Avg.$_8$ at update 50 and 13.0 at update 100, respectively, both below the base model's Avg.$_8$ of 33.7; we therefore terminated these runs early. These results use the recommended configurations and apply specifically to the $\mu = 1024$ regime; we do not claim failure in the moderate-staleness regimes for which these methods were developed. Among completed runs, GSPO remains stable but reaches only 41.9 Avg.$_5$, comparable to tightly clipped GRPO at $\mu = 1024$ (41.8), while clipping 22–38% of sequences across stages. For reference, M2PO reaches 46.8 Avg.$_5$, whereas μ-GRPO achieves the best accuracy, 48.0 Avg.$_5$ and 47.0 Avg.$_8$, in 18.1 h, exceeding GRPO at $\mu = 4$ while reducing training time from 23.7 h to 18.1 h.

D.4 POLICY ENTROPY

Optimizing for 1024 updates on each frozen rollout dataset could reduce output diversity. We therefore track policy entropy for μ-GRPO ($\mu = 1024$) and standard GRPO ($\mu = 4$). Because the within-stage training stream contains prefixes sampled from a stale behavior policy, entropy computed on those prefixes would not reflect current-policy generations. Instead, at each stage boundary we freshly sample from the current policy using the same prompts and sampling protocol as the five benchmarks in Table 1, compute the entropy of the next-token distribution at every generated prefix, and average over all generated tokens and examples (in nats).

Table 17: Policy entropy (nats) of freshly sampled current-policy generations at each stage boundary.

METHODS	μ	MODEL UPDATES 0	1024	2048	3072	4096
LLAMA-3.2-3B-INSTRUCT						
GRPO	4	0.117	0.092	0.087	0.073	0.061
μ-GRPO	1024	0.117	0.097	0.084	0.067	0.057
QWEN2.5-MATH-7B						
GRPO	4	0.160	0.031	0.023	0.018	0.015
μ-GRPO	1024	0.160	0.049	0.039	0.033	0.029
QWEN3-BASE-1.7B						
GRPO	4	0.104	0.056	0.042	0.039	0.038
μ-GRPO	1024	0.104	0.083	0.063	0.055	0.053
QWEN3-BASE-8B						
GRPO	4	0.137	0.053	0.044	0.042	0.034
μ-GRPO	1024	0.137	0.058	0.040	0.044	0.033
DEEPSEEK-7B						
GRPO	4	0.346	0.156	0.117	0.060	0.054
μ-GRPO	1024	0.346	0.226	0.182	0.142	0.135

Entropy decreases under both methods as training sharpens the policy (Table 17). However, μ-GRPO does not exhibit greater entropy collapse than standard GRPO: its final entropy is higher on Qwen2.5-Math-7B, Qwen3-Base-1.7B, and DeepSeek-7B, and comparable on Qwen3-Base-8B (0.033 vs. 0.034) and Llama-3.2-3B-Instruct (0.057 vs. 0.061). Together with the within-stage evaluation trends in Figure 1, this provides no evidence that rollout reuse causes systematic diversity collapse.

E Prefix-Support Mismatch in Retained Harmful Suffixes

This section formalizes the prefix-support mismatch mechanism identified in Section 3. We show why retaining $\mathcal{H}_\kappa$-tokens can be unstable even when their local importance ratios are not tiny: the behavior policy may assign non-negligible mass to these suffix tokens, while the current policy assigns very small mass to the prefixes needed to reach them.

We follow the notation in Section 2: β denotes the behavior policy, π_θ denotes the current policy, and $\rho_t(\theta)$ denotes the token-level importance ratio. For notational simplicity, we drop the response index and write a response as $y=(a_1,\dots,a_T)$. Assume a finite decoding horizon T; variable-length responses can be represented by padding with EOS tokens. Let λ be any fixed distribution over token positions $t \in \{1,\dots,T\}$ with $\lambda_t \geq 0$ and $\sum_t \lambda_t = 1$.

For a token occurrence $z=(x,y,t)$, define the behavior token-occurrence measure

$$d\nu_\beta(x,y,t) = d\mathcal{D}(x)\,\lambda_t\,\beta(y \mid x),$$

and the current-prefix occurrence measure

$$d\nu_\theta^{\mathrm{pre}}(x,y,t) = d\mathcal{D}(x)\,\lambda_t\,\pi_\theta(a_{<t} \mid x)\,\beta(a_{t:T} \mid x, a_{<t}).$$

Thus $\nu_\theta^{\mathrm{pre}}$ changes only the prefix occupancy from β to π_θ, while keeping the replay suffix conditional under β. Whenever $\beta(a_{<t} \mid x) > 0$,

$$\frac{d\nu_\theta^{\mathrm{pre}}}{d\nu_\beta}(x,y,t) = R_{t-1}(\theta) := \frac{\pi_\theta(a_{<t} \mid x)}{\beta(a_{<t} \mid x)} = \prod_{j=1}^{t-1} \rho_j(\theta),$$

where

$$\rho_t(\theta) := \frac{\pi_\theta(a_t \mid x, a_{<t})}{\beta(a_t \mid x, a_{<t})}.$$

Recall from Section 3 that the trigger boundary is the first trigger-token position

$$\kappa(y) := \min\{t :\ A(y) < 0,\ \rho_t(\theta) < \tau_c\},$$

where $A(y)$ is the response-level advantage (so $A_t = A(y)$ for all t), and $\kappa(y)=\infty$ if the set is empty. The non-trigger suffix is

$$\mathcal{H}_\kappa(y) := \{t > \kappa(y) :\ A(y) < 0,\ \rho_t(\theta) \geq \tau_c\}.$$

Let $\mathcal{U}_\mathcal{H}(\theta) \subseteq \{(x,y,t) : t \in \mathcal{H}_\kappa(y)\}$ denote any retained subset of these $\mathcal{H}_\kappa$-tokens. If the update rule retains all of $\mathcal{H}_\kappa(y)$, then $\mathcal{U}_\mathcal{H}(\theta) = \{(x,y,t) : t \in \mathcal{H}_\kappa(y)\}$.

We use the Pearson chi-square convention

$$\chi^2(P\|Q) := \int \left(\frac{dP}{dQ} - 1\right)^2 dQ,$$

with $\chi^2(P\|Q) = +\infty$ when P is not absolutely continuous with respect to Q.

Theorem 1 (Retained $\mathcal{H}_\kappa$-tokens can induce arbitrarily large prefix chi-square divergence)**.** *Let*

$$m_\mathcal{H} := \nu_\beta(\mathcal{U}_\mathcal{H}), \qquad q_\mathcal{H} := \nu_\theta^{\mathrm{pre}}(\mathcal{U}_\mathcal{H}) = \mathbb{E}_{\nu_\beta}\left[R_{t-1}(\theta)\mathbf{1}\{(x,y,t) \in \mathcal{U}_\mathcal{H}\}\right].$$

Assume $m_\mathcal{H} > 0$ *and*

$$q_\mathcal{H} \leq r\, m_\mathcal{H} \qquad \textit{for some } r \in (0,1).$$

Then, if $q_\mathcal{H} > 0$*,*

$$\chi^2(\nu_\beta\|\nu_\theta^{\mathrm{pre}}) \geq \chi^2(\mathrm{Bern}(m_\mathcal{H})\|\mathrm{Bern}(q_\mathcal{H})) = \frac{(m_\mathcal{H} - q_\mathcal{H})^2}{q_\mathcal{H}(1-q_\mathcal{H})}.$$

Consequently,

$$\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}}) \geq \frac{m_{\mathcal{H}}(1-r)^2}{r}.$$

In particular, for fixed $m_{\mathcal{H}} > 0$, the lower bound diverges as $r \downarrow 0$. If $q_{\mathcal{H}} = 0 < m_{\mathcal{H}}$, then

$$\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}}) = +\infty.$$

Proof. Let $P = \nu_\beta$, $Q = \nu_\theta^{\mathrm{pre}}$, and $U = \mathcal{U}_{\mathcal{H}}$. If $Q(U) = 0 < P(U)$, then P is not absolutely continuous with respect to Q, so $\chi^2(P\|Q) = +\infty$.

Now assume $q_{\mathcal{H}} = Q(U) > 0$. Let $L = dP/dQ$. By Cauchy–Schwarz,

$$\int_U L^2 \, dQ \geq \frac{\left(\int_U L \, dQ\right)^2}{Q(U)} = \frac{P(U)^2}{Q(U)} = \frac{m_{\mathcal{H}}^2}{q_{\mathcal{H}}}.$$

Applying the same argument to U^c,

$$\int_{U^c} L^2 \, dQ \geq \frac{P(U^c)^2}{Q(U^c)} = \frac{(1-m_{\mathcal{H}})^2}{1-q_{\mathcal{H}}}.$$

Therefore,

$$\begin{aligned}\chi^2(P\|Q) &= \int L^2 \, dQ - 1 \\ &\geq \frac{m_{\mathcal{H}}^2}{q_{\mathcal{H}}} + \frac{(1-m_{\mathcal{H}})^2}{1-q_{\mathcal{H}}} - 1 \\ &= \frac{(m_{\mathcal{H}} - q_{\mathcal{H}})^2}{q_{\mathcal{H}}(1-q_{\mathcal{H}})}.\end{aligned}$$

Using $q_{\mathcal{H}} \leq r m_{\mathcal{H}}$, we have

$$m_{\mathcal{H}} - q_{\mathcal{H}} \geq m_{\mathcal{H}}(1-r), \qquad q_{\mathcal{H}}(1-q_{\mathcal{H}}) \leq q_{\mathcal{H}} \leq r m_{\mathcal{H}}.$$

Thus

$$\chi^2(P\|Q) \geq \frac{m_{\mathcal{H}}^2(1-r)^2}{r m_{\mathcal{H}}} = \frac{m_{\mathcal{H}}(1-r)^2}{r}.$$

This proves the claim. □

Corollary 1 (The blow-up is compatible with non-tiny local ratios)**.** *Fix any threshold $\tau_c \in (0, 1]$ and any target $M > 0$. There exist behavior and current policies β and π_θ, and a retained harmful suffix set $\mathcal{U}_{\mathcal{H}}$, such that every retained suffix token in $\mathcal{U}_{\mathcal{H}}$ satisfies*

$$\rho_t(\theta) \geq \tau_c,$$

but

$$\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}}) > M.$$

Proof. Consider one prompt and two token positions. Let the first-token vocabulary contain two tokens c and b. Choose $m \in (0, 1)$, and choose

$$r < \min\{\tau_c,\, 1/2,\, \lambda_2 m/(4M)\},$$

where we assume the position weight λ_2 assigned to the second token is positive.

Let

$$\beta(c \mid x) = m, \qquad \pi_\theta(c \mid x) = rm,$$

with the remaining probability assigned to b. After prefix c, let both policies deterministically emit the second token h:

$$\beta(h \mid x, c) = 1, \qquad \pi_\theta(h \mid x, c) = 1.$$

Assign negative advantage to the trajectory $y = (c, h)$, i.e., $A(y) < 0$.

For $y = (c, h)$, the first token has

$$\rho_1(\theta) = \frac{\pi_\theta(c \mid x)}{\beta(c \mid x)} = r < \tau_c,$$

so $\kappa(y) = 1$. The second token has

$$\rho_2(\theta) = \frac{\pi_\theta(h \mid x, c)}{\beta(h \mid x, c)} = 1 \geq \tau_c,$$

so $2 \in \mathcal{H}_\kappa(y)$. Its prefix occupancy is

$$R_1(\theta) = \rho_1(\theta) = r.$$

Let the update rule retain this second-token occurrence, and define

$$\mathcal{U}_\mathcal{H} = \{(x, (c, h), 2)\}.$$

Then

$$m_\mathcal{H} = \nu_\beta(\mathcal{U}_\mathcal{H}) = \lambda_2 m, \qquad q_\mathcal{H} = \nu_\theta^{\mathrm{pre}}(\mathcal{U}_\mathcal{H}) = \lambda_2 r m.$$

By Theorem 1,

$$\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}}) \geq \frac{\lambda_2 m (1-r)^2}{r}.$$

Since $r < 1/2$, $(1-r)^2 > 1/4$, and therefore

$$\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}}) > \frac{\lambda_2 m}{4r} > M.$$

Thus the retained token can have a perfectly benign local ratio $\rho_2(\theta) = 1$, while the induced prefix mismatch is arbitrarily large. □

Remark 1 (Direction of the Pearson divergence)**.** *The divergence in Theorem 1 is the reverse Pearson divergence $\chi^2(\nu_\beta \| \nu_\theta^{\mathrm{pre}})$, which is sensitive to behavior-supported prefixes that have very small current occupancy. The opposite direction satisfies*

$$\chi^2(\nu_\theta^{\mathrm{pre}} \| \nu_\beta) = \mathbb{E}_{\nu_\beta}\left[(R_{t-1}(\theta) - 1)^2\right],$$

so $R_{t-1} \ll 1$ on a set of fixed behavior mass does not by itself force an unbounded contribution in that direction. This is why the missing-support statement should be phrased as a behavior-to-current, or reverse-Pearson, prefix divergence.

F More Related Work

Off-policy RLVR. Despite being framed as on-policy methods, practical RLVR implementations are inherently off-policy: rollout data are reused across multiple gradient updates (Shao et al., 2024; Hugging Face, 2025); modern RL systems decouple rollout generation from training for throughput (Fu et al., 2025; Zhu et al., 2025; Noukhovitch et al., 2025; Zhong et al., 2025; He et al., 2025); and training–inference mismatch from numerical precision differences can introduce additional distribution shift (Qi et al., 2025). To stabilize training under off-policy conditions, prior work has proposed various strategies to control policy updates, including ratio clipping (Yu et al., 2025), approximate trust regions (Fu et al., 2025), sequence-level clipping (Zheng et al., 2025a), asymmetric clipping (Le Roux et al., 2025), and gradient-preserving clipping (Su et al., 2025; MiniMax et al., 2025). While these methods improve RLVR under moderate staleness, most do not explicitly identify why naive large-staleness GRPO fails. Two recent methods target stale or off-policy data more directly. BAPO (Xi et al., 2026) observes that negative-advantage samples dominate the policy gradient in off-policy training and adaptively adjusts the clipping bounds to rebalance positive and negative contributions. M2PO (Zheng et al., 2026) identifies excessive clipping under staleness and the prosperity-before-collapse phenomenon, and constrains the second moment of importance weights at the batch level to selectively mask high-variance tokens. Unlike BAPO, which rebalances positive and negative contributions through adaptive clipping bounds, and M2PO, which filters tokens by batch-level second-moment statistics, we localize the harmful updates to the suffix after a low-ratio trigger in negative-advantage responses and remove them with a simple trigger–scope filter. Moreover, our work introduces an off-policy training paradigm that fully decouples generation and optimization to fundamentally address the efficiency bottleneck.

RLVR Efficiency. RLVR systems improve efficiency through both synchronous/hybrid orchestration and asynchronous execution. In synchronous pipelines, rollout generation and policy optimization often alternate between inference and training engines. To make frequent switching practical, systems may keep part of the training state resident on GPU during rollout, reducing reload overhead but limiting memory available for high-throughput inference. Hybrid orchestration frameworks such as HybridFlow/verl improve scheduling across these engines (Sheng et al., 2025; Zhu et al., 2025), but frequent rollout refreshes and weight synchronization can remain a bottleneck when the RL algorithm only tolerates low staleness. Asynchronous and streaming systems instead overlap rollout workers and trainers through queues or streamed data transfer (Fu et al., 2025; Zhong et al., 2025; Han et al., 2025; Noukhovitch et al., 2025). These systems improve utilization, but still benefit from staleness-robust objectives because larger tolerated rollout lag reduces synchronization pressure. μ-GRPO is complementary to both directions: by stabilizing GRPO at $\mu = 1024$, it reduces rollout refreshes and weight synchronizations, enabling rollout generation to run as a standalone phase that can fully use GPU memory for inference, while also permitting larger lag in asynchronous pipelines.