LET THE BATCH DECIDE, ON- OR OFF-POLICY: EFFECTIVE SAMPLE SIZE FOR RL POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) is structurally harder than supervised learning because the policy changes the data distribution it learns from. The resulting fragility is especially visible in large-model training using nominally on-policy RL, where the training and rollout systems can differ in numerical precision, sampling and other implementation details. The common approach handles this difference by constraining the update with a fixed clip range, so the algorithm becomes more sensitive to its configuration and must be retuned whenever the task, model scale or distribution mismatch changes. We argue that the fragility traces to two linked concerns that this one fixed range has to cover. The first is a trust-region concern: an update should not move the policy too far from its current value. The second is an off-policy concern: data from older or different behavior policies should influence the update only to the extent that the update remains reliable. Neither concern is a constant to set in advance, and the policy ratios of the current batch show how severe each one is. We adopt P3O, a simple yet effective batch-adaptive objective from prior work that replaces fixed clipping with the normalized effective sample size of the per-token policy ratios, which we take against the rollout engine's log-probabilities. The same statistic caps the score-function weight and sets the strength of an off-policy regularizer, so the update stays close to the on-policy update when the ratios are close to one. When stale or mismatched data concentrate the ratios, the update tightens but keeps a nonzero score-function weight on every token. Experiments on math reasoning show that P3O compares favorably with tuned baselines and, under FP8 rollouts on Qwen3-8B-Base, avoids the late collapse of GRPO. P3O adds no new objective hyper-parameters and removes the clip range.

1 INTRODUCTION

Post-training large (language) models often means optimizing a policy by sampling completions and scoring them with a reward model, a human-preference-derived objective, or a task verifier (Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022; DeepSeek-AI, 2025). Because these scores are assigned to discrete sampled text and are usually not differentiable through the policy, policy-gradient reinforcement learning (RL) is a natural tool for increasing expected score. RL is a fragile optimization process whose outcomes depend heavily on hyper-parameters and implementation details (Henderson et al., 2018; Engstrom et al., 2020; Andrychowicz et al., 2021; Fakoor et al., 2020a; Huang et al., 2022). This fragility sharpens at large-model scale, where runs are expensive, rollouts are costly to regenerate, and small choices (clip range, allowed rollout staleness, etc.) determine whether training improves, stalls, or becomes unstable. These choices are often retuned for each task, model scale, and rollout infrastructure.

Two concerns underlie this fragility. The first is a *trust-region* concern, that each gradient update should not move the policy too far from its current value (Schulman et al., 2015; 2017). The second is an *off-policy* concern, that data collected by older or different behavior policies should influence the current update only to the extent that the update remains reliable (Fakoor et al., 2020b; 2024; Espeholt et al., 2018). The two concerns are linked, since a surrogate objective estimated from behavior-policy samples is accurate only for policies close to the behavior policy.

Off-policy data are common in practice and arise from several sources, including rollouts reused across optimizer steps, rollouts from earlier policies in asynchronous pipelines (Noukhovitch et al.,

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.

2025), demonstrations or rollouts from heterogeneous policies (the batch-RL setting; Levine et al., 2020; Fakoor et al., 2021; Liu et al., 2023), and practical mismatches between the rollout and training engines such as numerical precision (Qi et al., 2025), sampling strategy, or expert routing (Ma et al., 2026). Mishandling such data is consequential, since stale or mismatched samples can destabilize training, yet discarding them forfeits sample efficiency because each rollout is then consumed by a small number of updates (Fakoor et al., 2019).

Proximal policy optimization (PPO) and its critic-free variant GRPO cover both concerns with a single fixed-clip mechanism (Schulman et al., 2017; Shao et al., 2024), where a fixed range controls the per-step policy change and is also relied on to limit the off-policy variance. This is simple, but the range is a strong algorithmic choice made before training starts, and it determines which tokens receive gradient, how much stale data can be reused, and how quickly the policy is allowed to move. Decoupled-loss methods separate the proximal policy used for clipping from the behavior policy used for importance correction and improve learning from stale data (Hilton et al., 2022). Recent objectives make the range asymmetric and add dynamic sampling (DAPO; Yu et al., 2025), clip a sequence-level ratio (GSPO; Zheng et al., 2025), or remove the lower bound and cap a stop-gradient importance weight at a fixed upper value (CISPO; MiniMax et al., 2025).

The premise of this paper is that much of this fragility comes from fixing the clip before training, and that changing the shape of the clip (asymmetric, sequence-level, one-sided) still leaves a range or cap to fix in advance. We build on the observation that the per-token policy ratios of the current batch show how severe each concern is. Using only their normalized effective sample size (ESS; Kong, 1992), P3O (Fakoor et al., 2019) sets the cap on the score-function weight and regularizes toward the behavior policy, with no new objective hyper-parameter. Unlike fixed-clip methods, whose range is set before training starts, P3O thus adapts the cap and the regularization to how far apart the behavior and current policies are in each batch. When the two match, every ratio is one and, with group-relative advantages, P3O's gradient equals that of GRPO; as they drift apart, the cap tightens and the regularizer grows. The same objective therefore applies whether the batch is on- or off-policy, while fixed-clip objectives may need a decoupled loss (Hilton et al., 2022) or a retuned clip.

Our main contribution is to adopt P3O for RL post-training of large models, where, to our knowledge, it has not been studied. The original P3O, designed for actor-critic training on games and continuous control, caps only replayed data, since its fresh rollouts are on-policy. In large-model post-training, there can be a log-probability gap between the rollout and training engines even for a fresh rollout, by design (e.g., lower-precision rollouts) or unintentionally (e.g., different kernels or expert routing). We therefore cap every token, with each ratio taken against the rollout engine's recorded log-probability, so one batch can mix several policy versions. We use group-relative advantages in place of a critic. Used unchanged in synchronous and asynchronous training (Fig. 5), with the GRPO hyper-parameters of each experiment and no clip sweep, this one objective compares favorably with tuned baselines on math reasoning across several Qwen models. The clearest difference appears when Qwen3-8B-Base is trained in BF16 with FP8 rollouts, where the quantization mismatch reaches every token even though no rollout is stale. GRPO is level with or slightly above P3O at iteration 15 but below the untrained model on all five benchmarks at iteration 30, while P3O stays above it (Fig. 4b and Table 2).

2 Background

Given a pre-trained model, our objective is to fine-tune it using reinforcement learning so that completions sampled from it receive high scores. For a prompt x, the model assigns a probability to a completion $y = (y_1, \dots, y_T)$ autoregressively,

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x,\, y_{<t}), \tag{1}$$

where each y_t is a token from a finite vocabulary $\mathcal{V}$, $y_{<t} = (y_1, \dots, y_{t-1})$ is the prefix at position t, and T is the completion length. To simplify, we write $c_{<t} = (x, y_{<t})$ for the conditioning context.

Token-level MDP. We cast this autoregressive generation as a finite-horizon MDP in which the state at position t is $s_t = (x, y_{<t})$, the action is the next token $y_t \in \mathcal{V}$, and a scalar terminal reward

$r(x, y) \in \mathbb{R}$ is assigned to the full completion.[1] The agent's objective is to maximize the expected return $J(\pi_\theta) = \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}[\, r(x, y)\,]$, where x is a prompt from the training set and y is a completion sampled from π_θ. Using the likelihood-ratio trick (Williams, 1992), the policy gradient of $J(\pi_\theta)$ is

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} r(x, y)\, \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right]. \tag{2}$$

This is known as the REINFORCE policy-gradient estimator (Williams, 1992). It is an on-policy estimator: it optimizes the same policy that was used to collect the data. Despite its simplicity, the policy-gradient estimator has high variance because the update uses sampled trajectories and their corresponding rewards as noisy estimates of how good each action was, so the same policy can produce very different rewards across rollouts, especially with long horizons and delayed rewards (Sutton & Barto, 2018). The standard remedy is to subtract a control variate (baseline) b that does not depend on the sampled action y_t, replacing r by an advantage $A = r - b$. For any such baseline the gradient remains unbiased, while for a well-chosen b its variance is typically reduced substantially. Actor-critic methods learn a state-dependent value function $V_\phi(s_t) = V_\phi(x, y_{<t})$ and use it as the baseline at each step.

Group-relative advantage. Learning a value function alongside the policy is usually expensive when training large models, because the value function can itself be a network of comparable size to the policy and inherits all the difficulties of training a large model. GRPO (Shao et al., 2024) sidesteps this by replacing the learned baseline with a within-batch reward statistic. For each prompt x_p, it samples a *group* of G completions $\{y_{p,j}\}_{j=1}^{G}$ and uses the group's reward statistics as the baseline. Writing $r_{p,j} := r(x_p, y_{p,j})$ for the reward of completion j in group p, the group-relative advantage is

$$A_{p,j} = \frac{r_{p,j} - \frac{1}{G}\sum_{k=1}^{G} r_{p,k}}{\sqrt{\frac{1}{G-1}\sum_{k=1}^{G}\left(r_{p,k} - \frac{1}{G}\sum_{l=1}^{G} r_{p,l}\right)^2 + \delta}}, \tag{3}$$

where $\delta > 0$ avoids division by zero. The group standard deviation rescales the advantage so it is comparable across prompts. Both statistics depend on $r_{p,j}$, so the gradient is biased in general.

Trust region policy optimization. A separate optimization concern, beyond the variance and baseline issues above, is that a single gradient update can move π_θ too far from its current value and destabilize training, even when the data are on-policy. Trust-region methods address this by constraining each update to stay close to the current policy. Where TRPO solves this problem with a complex second-order method (Schulman et al., 2015), PPO (Schulman et al., 2017) takes a simpler approach by using first-order methods to keep new policies close to old. Concretely, PPO samples data from a snapshot $\pi_{\theta_{\text{old}}}$ of the policy, takes several optimizer steps on these data, and clips the per-token policy ratio $\rho_t(\theta)$ before applying it,

$$\mathcal{L}_{\text{PPO}}(\theta) = -\mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[\min\big(\rho_t(\theta)\, A,\ \text{clip}(\rho_t(\theta),\, 1-\epsilon_\ell,\, 1+\epsilon_h)\, A\big)\right], \quad \rho_t(\theta) = \frac{\pi_\theta(y_t \mid x, y_{<t})}{\pi_{\theta_{\text{old}}}(y_t \mid x, y_{<t})} \tag{4}$$

with the clip range $[1-\epsilon_\ell,\, 1+\epsilon_h]$ fixed before training. Because $\pi_{\theta_{\text{old}}}$ is close to π_θ in PPO's intended setting, ρ_t stays close to one for most tokens and the clip discourages large policy changes.

From on-policy to off-policy data. The on-policy estimator in Eq. (2) is unbiased but sample-inefficient, since each rollout is consumed by a single gradient step. In practice the data we train on are rarely drawn from π_θ exactly. In such cases the data were sampled from a *behavior policy* π_{b} but the loss is evaluated under the current π_θ, so the empirical gradient computed on samples from π_{b} is a biased estimate of $\nabla_\theta J(\pi_\theta)$.

The standard fix is importance sampling (Fakoor et al., 2020b; 2024; Resnick, 2013), which restores unbiasedness by changing the measure from π_θ to π_{b}. For any function f,

$$\mathbb{E}_{z\sim\pi_\theta}[f(z)] = \int \pi_\theta(z)\, f(z)\, dz = \int \pi_{\text{b}}(z)\, \frac{\pi_\theta(z)}{\pi_{\text{b}}(z)}\, f(z)\, dz = \mathbb{E}_{z\sim\pi_{\text{b}}}\left[\frac{\pi_\theta(z)}{\pi_{\text{b}}(z)}\, f(z)\right], \tag{5}$$

[1] The formulation extends to per-token rewards $r_t(x, y_{\le t})$ by replacing $r(x, y)$ with the reward-to-go $\sum_{t'=t}^{T} r_{t'}(x, y_{\le t'})$ inside the sum in Eq. (2).

provided that the Radon-Nikodym derivative $d\pi_\theta / d\pi_{\mathrm{b}}$ is well defined. Applied to the RL objective, this rewrites the expected return as an expectation under the behavior policy,

$$J(\pi_\theta) = \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}\Big[r(x,y)\Big] = \mathbb{E}_{x,\, y\sim\pi_{\mathrm{b}}(\cdot|x)}\left[\frac{\pi_\theta(y \mid x)}{\pi_{\mathrm{b}}(y \mid x)}\, r(x,y)\right], \tag{6}$$

so $J(\pi_\theta)$ can be evaluated on data drawn from π_{b}. In our token-level setting we work with the per-token importance ratio

$$\rho_t(\theta) = \frac{\pi_\theta(y_t \mid x, y_{<t})}{\pi_{\mathrm{b}}(y_t \mid x, y_{<t})}, \tag{7}$$

which generalizes the PPO-snapshot ratio in Eq. (4) to any behavior policy π_{b}, and apply this correction directly inside the per-token gradient. Unlike Eq. (6), this correction weights each token by its own ratio only, so the resulting gradient is biased, as in PPO.

Remark (Importance sampling is unbiased but high variance). Importance sampling restores unbiasedness in Eq. (6), but the per-token ratios $\rho_t(\theta)$ are bounded below by zero and unbounded above, so a few tokens that were unlikely under π_{b} can produce ratios many orders of magnitude larger than one and dominate the gradient estimate. The standard remedy is to clip $\rho_t(\theta)$ before applying it, but as we discuss in Sec. 3, fixed clipping is itself fragile.

RL in language modeling. When fine-tuning language models with RL, the objective is often regularized by a KL penalty toward a frozen reference policy π_{θ_0}, the same policy used to initialize π_θ (typically a pre-trained or supervised-fine-tuned model, the latter trained on ground-truth tokens via teacher forcing; Cen et al., 2025),[2]

$$J_{\mathrm{LM}}(\pi_\theta) = \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}\left[r(x,y) - \eta \sum_{t=1}^{T} \mathrm{KL}\big(\pi_\theta(\cdot \mid x, y_{<t}) \,\big\|\, \pi_{\theta_0}(\cdot \mid x, y_{<t})\big)\right], \tag{8}$$

where $\eta \geq 0$ controls how far π_θ is allowed to drift from π_{θ_0} (Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022). The KL anchor is intended to keep π_θ from drifting too far from π_{θ_0} and potentially from forgetting its language-modeling capabilities. We set $\eta = 0$ in all experiments.

3 Policy-on Policy-off Policy Optimization (P3O)

The PPO clipped surrogate in Eq. (4) contains two hyper-parameters, ϵ_ℓ and ϵ_h, that specify the clip range $[1-\epsilon_\ell,\, 1+\epsilon_h]$ around the per-token ratio ρ_t. From here on, ρ_t is the ratio of Eq. (7), where the behavior policy π_{b} is the policy whose token log-probabilities the rollout engine recorded (for PPO, $\pi_{\mathrm{b}} = \pi_{\theta_{\mathrm{old}}}$). The effect of ϵ_ℓ and ϵ_h is most easily seen by considering the per-token contribution to Eq. (4) under the two signs of A. For tokens with $A > 0$, this contribution simplifies to

$$-\min(\rho_t,\, 1+\epsilon_h)\; A. \tag{9}$$

The objective is improved by increasing $\pi_\theta(y_t \mid c_{<t})$, which raises ρ_t. Once $\rho_t > 1+\epsilon_h$, the contribution hits the bound $-(1+\epsilon_h)\, A$, and the new policy gains no further credit by increasing the probability of the token. For tokens with $A < 0$, the contribution becomes

$$-\max(\rho_t,\, 1-\epsilon_\ell)\; A, \tag{10}$$

where the $\max$ replaces the $\min$ because A is negative. The objective is improved by decreasing $\pi_\theta(y_t \mid c_{<t})$ and hence ρ_t. Once $\rho_t < 1-\epsilon_\ell$, the contribution hits $-(1-\epsilon_\ell)\, A$, and the new policy gains no further credit by decreasing the probability of the token. In both cases, $(\epsilon_\ell, \epsilon_h)$ specify how far the new policy can move from π_{b} while continuing to improve the surrogate; clipping is a regularizer that removes the incentive to change the policy dramatically from one update to the next.

The effect of $(\epsilon_\ell, \epsilon_h)$ depends on the regime. When the batch is fresh and on-policy, $\rho_t \approx 1$ for most tokens and the clip is rarely active. When the batch is stale or generated by a different policy, ρ_t can

[2] More generally, π_{θ_0} can be any reference policy that shares the token vocabulary $\mathcal{V}$ with π_θ. When the vocabularies differ, the per-token KL is no longer well defined and the regularization must be applied at the sequence level.

be far from one for many tokens, and $(\epsilon_\ell, \epsilon_h)$ then determines what fraction of the batch contributes to the update. If $(\epsilon_\ell, \epsilon_h)$ is too small, many tokens are clipped before they can contribute; if it is too large, high-variance importance-weighted gradients can destabilize training. A range chosen for one regime may therefore fail in the other, and since it is fixed before training, $(\epsilon_\ell, \epsilon_h)$ in practice has to be tuned for the task, model scale, and degree of off-policy mismatch.

Prior work re-parameterizes the clip but does not remove it. Most previous works retain at least one of these two hyper-parameters and differ in how they re-parameterize them. GRPO (Shao et al., 2024) uses Eq. (4) with a symmetric range $\epsilon_\ell = \epsilon_h$. DAPO (Yu et al., 2025) relaxes the symmetry, allowing $\epsilon_\ell \neq \epsilon_h$ to handle positive- and negative-advantage updates separately.[3] GSPO (Zheng et al., 2025) moves the clipped ratio from the token level to the sequence level via a geometric mean, but introduces sequence-level analogues $\epsilon_\ell^{\text{seq}}, \epsilon_h^{\text{seq}}$ that still must be chosen in advance. CISPO (MiniMax et al., 2025) effectively keeps only ϵ_h, as a fixed cap $1 + \epsilon_h$ on a stop-gradient importance weight. Across these objectives, $(\epsilon_\ell, \epsilon_h)$ survives in some form (see Table 1).

Effective sample size. Tuning $(\epsilon_\ell, \epsilon_h)$ for large models is expensive. The normalized effective sample size (ESS) of the policy ratios in the current batch (Kong, 1992; Fakoor et al., 2019) provides a way to set the cap from the batch. Concretely, the ESS is given by

$$\mathrm{ESS}(\mathcal{B}; \theta) = \frac{\widehat{\mathbb{E}}_{\mathcal{B}}[\rho_t]^2}{\widehat{\mathbb{E}}_{\mathcal{B}}[\rho_t^2]} \in \left[\frac{1}{|\mathcal{B}|}, 1\right], \tag{11}$$

where $\mathcal{B}$ is the set of valid response tokens in the current micro-batch, $|\mathcal{B}|$ is its size, and $\widehat{\mathbb{E}}_{\mathcal{B}}$ denotes the empirical average over $\mathcal{B}$. Each data-parallel worker computes the sums of ρ_t and ρ_t^2 and its valid-token count, and these are all-reduced before the ESS is formed (Algorithm 2). The ratios are already computed for the loss, so this adds one small reduction and no forward pass. We use $e_{\mathcal{B}} = \mathbf{sg}(\mathrm{ESS}(\mathcal{B}; \theta))$ to denote its detached value (treated as a constant for backpropagation), where $\mathbf{sg}(\cdot)$ is the stop-gradient operator. The ESS is close to one when the ratios are nearly uniform and falls when a few large ratios dominate, as happens with stale or mismatched data. Observe that, at each context, ρ_t has mean one under π_{b} and second moment one plus the chi-square divergence $\chi^2(\pi_\theta \,\|\, \pi_{\mathrm{b}})$, so the ESS falls as the behavior and current policies move apart. It averages over tokens and does not multiply ratios along a response, so response length by itself does not lower it.

P3O for large-model post-training. We adopt the P3O objective of Fakoor et al. (2019). P3O replaces the fixed clip in Eq. (4) with two terms whose strength is set by the batch ESS,

$$\mathcal{L}_{\mathrm{P3O}}(\theta) = \mathbb{E}_{x,\, y\sim\pi_{\mathrm{b}}(\cdot|x)}\Bigg[\sum_{t=1}^{T}\Big(-\mathbf{sg}(\min\{\rho_t,\, e_{\mathcal{B}}\})\, \log \pi_\theta(y_t \mid c_{<t})\, A \\ + (1 - e_{\mathcal{B}})\, \mathrm{KL}\big(\pi_\theta(\cdot \mid c_{<t}) \,\big\|\, \pi_{\mathrm{b}}(\cdot \mid c_{<t})\big)\Big)\Bigg], \tag{12}$$

where the per-token KL, written in the same form as Eq. (8), vanishes when $\pi_\theta = \pi_{\mathrm{b}}$. In practice we approximate this KL at the sampled token by $\log \rho_t + 1/\rho_t - 1$, the estimator GRPO uses for its reference KL (Shao et al., 2024), since we store only that token's log-probability under π_{b}. Because y_t is drawn from π_{b} and not from π_θ, this term matches the KL in expectation only to second order in $\rho_t - 1$ (Sec. A).

P3O makes two structural changes to Eq. (4). First, it replaces the fixed clip range $(\epsilon_\ell, \epsilon_h)$ with the data-driven cap $e_{\mathcal{B}}$. Second, it adds a regularizer with coefficient $(1 - e_{\mathcal{B}})$ that grows as the behavior and current policies move apart and vanishes only when all ratios in the batch are equal. Both adaptations are driven by a single statistic of the current batch, so Eq. (12) adds no objective hyper-parameter. Replacing $e_{\mathcal{B}}$ by the fixed cap $1 + \epsilon_h$ and dropping the KL term gives CISPO (MiniMax et al., 2025), proposed six years after P3O.

[3] Note that, in addition to its asymmetric clip, DAPO changes how the data and the reward are used: it removes prompt groups in which all completions receive the same reward (dynamic sampling) and penalizes responses that approach the length limit (overlong penalty). Both changes are complementary to the idea of this paper and can be applied to P3O directly; we do not test this combination, as our focus is on the objective itself. We use DAPO's token-level loss normalization for all methods.

Table 1: **Comparison of policy objectives.** Here $\rho_t = \pi_\theta(y_t \mid x, y_{<t}) / \pi_{\text{b}}(y_t \mid x, y_{<t})$. **Red**/**green**: fixed hyper-parameters (high/low). **Purple**: auxiliary choices. **Blue**: batch-adaptive quantities. Auxiliary entropy and reference-policy KL terms are omitted.

Method	**Policy objective**
GRPO / DAPO	$-\min\left(\rho_t A,\ \text{clip}\left(\rho_t,\ 1-\epsilon_\ell,\ 1+\epsilon_h\right) A\right)$
GSPO	$-\min\left(S_\theta(y) A,\ \text{clip}\left(S_\theta(y),\ 1-\epsilon_\ell^{\text{seq}},\ 1+\epsilon_h^{\text{seq}}\right) A\right), \quad S_\theta(y) = \exp\left(\frac{1}{T}\sum_{t=1}^{T} \log \rho_t\right)$
Decoupled	$-\mathbf{sg}\left(\text{clip}\left(\frac{\pi_{\text{prox}}}{\pi_{\text{b}}},\ 0,\ c_w\right)\right) \min\left(\frac{\pi_\theta}{\pi_{\text{prox}}} A,\ \text{clip}\left(\frac{\pi_\theta}{\pi_{\text{prox}}},\ 1-\epsilon_\ell,\ 1+\epsilon_h\right) A\right)$
CISPO	$-\mathbf{sg}(\min\{\rho_t,\ 1+\epsilon_h\})\ \log \pi_\theta(y_t \mid x, y_{<t})\, A$ (lower bound effectively removed)
P3O	$-\mathbf{sg}(\min\{\rho_t,\ e_{\mathcal{B}}\})\ \log \pi_\theta(y_t \mid x, y_{<t})\, A\ +\ (1-e_{\mathcal{B}})\, \text{KL}(\pi_\theta \,\|\, \pi_{\text{b}})$

Properties of the update. The clip in Eq. (4) removes a token's gradient once its ratio passes the bound on the side its advantage favors. In contrast, the score-function weight $\min\{\rho_t, e_{\mathcal{B}}\}$ is at most one and never zero, so, as in CISPO, the cap never removes a token's gradient. When $e_{\mathcal{B}} < 1$, every token with $\rho_t > e_{\mathcal{B}}$, even one with ρ_t close to one, gets the weight $e_{\mathcal{B}}$. If $\rho_t = 1$ for every token, then $e_{\mathcal{B}} = 1$, every weight is one and the KL weight is zero, so the P3O gradient equals the GRPO gradient. Off-policy, the cap and the KL term add further bias, as in any truncated or clipped surrogate.

Relation to the original P3O. Fakoor et al. (2019) proposed P3O for actor-critic training on Atari and MuJoCo, where fresh rollouts come from the policy being trained, so it applies the cap only to replayed data. Its update adds an unweighted policy-gradient term on fresh rollouts to the capped term $\min\{\rho, c\}$ with $c = \text{ESS}$ and an exact KL penalty with weight $1 - \text{ESS}$. In large-model pipelines the rollout and training engines can assign different log-probabilities even to a fresh rollout, so we drop the unweighted term and apply the other two to every token, taking each ratio against the rollout engine's recorded log-probability. Tokens from several policy versions, as in asynchronous training, can then share a batch, and the group-relative advantage of Eq. (3) replaces the critic.

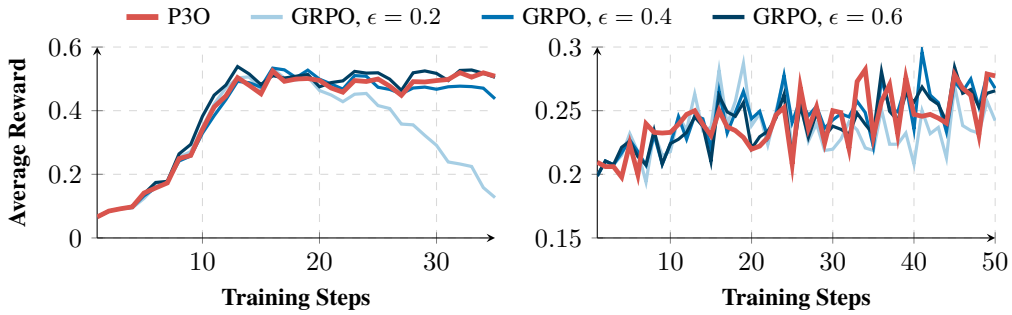
Off-policy data in large-model training. At large-model scale, a common fix for off-policy data is a decoupled-loss objective (Hilton et al., 2022; Fu et al., 2025) that clips against a proximal snapshot π_{prox} rather than against π_{b}. This stabilizes training but adds choices that are fixed before training, including the construction of π_{prox}, its lifetime, and, in practice, a cap c_w on an outer behavior weight. P3O needs none of them. The same $e_{\mathcal{B}}$ that drives the cap on fresh data falls as the behavior and current policies move apart, so we use Eq. (12) unchanged in every setting of Sec. 4, including asynchronous training (Fig. 5), without any additional hyper-parameter or special code path. Sec. D describes and tests a two-anchor extension of P3O, and Sec. E covers further related work.

4 Experiments

We evaluate P3O across three axes: (*i*) sensitivity to the clipping hyper-parameters $(\epsilon_\ell, \epsilon_h)$ that P3O eliminates entirely; (*ii*) stability under off-policy data arising from asynchronous training and from practical mismatches in numerical precision and sampling temperature; and (*iii*) downstream benchmark performance on held-out mathematical reasoning tasks. In all settings we compare against GRPO, which shares the same base objective (Sec. 2; Algorithms 1 and 2) but relies on a fixed clip range. P3O uses the GRPO hyper-parameters of each experiment family (Table 6), and Sec. B.1 lists the departures in an additional study on Qwen3-4B-Base.

4.1 Experiment setup

Experiments use the open-source models Qwen3-4B-Thinking-2507 and Qwen3-8B-Base (Qwen Team, 2025), the latter for the FP8 experiments, and Qwen2.5-1.5B-Instruct (Qwen Team, 2024). We train on a 32,251-problem training split of DeepScaleR-Preview, a dataset of 40,315 mathematics problem-answer pairs compiled from AIME (1984–2023), AMC (prior to 2023), Omni-MATH, and



(a) Qwen3-4B-Thinking-2507

(b) Qwen2.5-1.5B-Instruct

Figure 1: **Training reward of GRPO with each clip range $\epsilon \in \{0.2, 0.4, 0.6\}$ versus P3O run once with no clip hyper-parameter.** Over the last five steps, P3O's mean reward is 0.510 against 0.196, 0.467 and 0.514 for $\epsilon = 0.2$, 0.4 and 0.6 on Qwen3-4B-Thinking-2507, and 0.263 against 0.241, 0.268 and 0.263 on Qwen2.5-1.5B-Instruct.

Still (Luo et al., 2025). Binary rewards are assigned by matching the model's output against the DeepScaleR-Preview reference; to isolate algorithmic differences between P3O and GRPO, we use no reward shaping or auxiliary bonuses in either method (the DAPO baseline in Sec. B.2 keeps its overlong penalty). GRPO uses $\epsilon = 0.4$ in the Qwen3 off-policy experiments. Each configuration is run once. The clip, temperature and FP8 runs use 8 NVIDIA H100 GPUs in a distributed stack that separates training and rollout engines and synchronizes policy weights under a shared scheduler. Training, evaluation, hardware, and benchmark details are in Tables 5 to 8 and Sec. B and F.

4.2 Effects of hyper-parameters

Clip range. We test the sensitivity to the clip range by sweeping the symmetric clip $\epsilon_\ell = \epsilon_h = \epsilon \in \{0.2, 0.4, 0.6\}$ for GRPO and contrasting with a single P3O run. These runs take one optimizer step per iteration, so no rollout is stale and ρ_t departs from one only through the log-probability gap between the rollout and training engines. As shown in Fig. 1, GRPO's reward trajectory varies substantially with ϵ on Qwen3-4B-Thinking-2507 and little on Qwen2.5-1.5B-Instruct, while P3O remains stable on both. Other clip-based objectives such as DAPO (Yu et al., 2025) and GSPO (Zheng et al., 2025) also fix a clip range before training; we report DAPO in two settings with one range each (Sec. B.2) and did not run GSPO or CISPO.

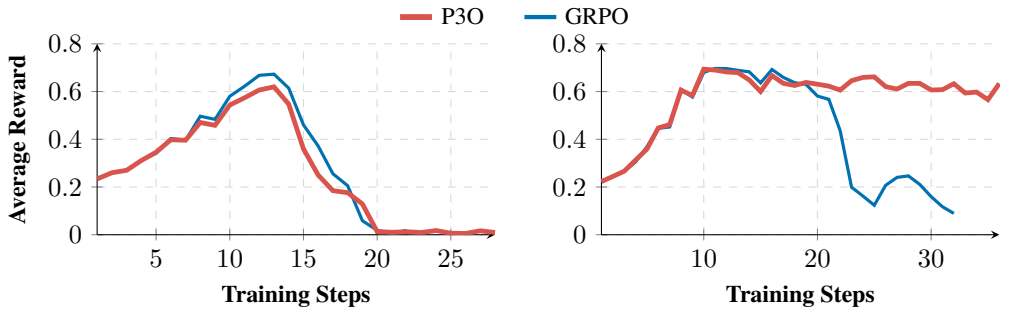
4.3 Off-policy data

In asynchronous training of Qwen3-4B-Thinking-2507 on GSM8K (Cobbe et al., 2021; Fig. 5), GRPO with the decoupled loss is ahead in the middle of training, and P3O is ahead at 22 of the last 23 steps. The two experiments below isolate two other sources of off-policy data and compare P3O with GRPO under each.

Temperature of rollouts. We sample rollouts of Qwen3-4B-Thinking-2507 at temperature $T = 0.6$ and $T = 1.2$. The rollout engine records log-probabilities before temperature scaling and the training engine computes them without temperature, so neither ρ_t (Eq. (7)) nor the ESS (Eq. (11)) reflects the change in the sampling distribution, and we report this experiment as an empirical comparison.

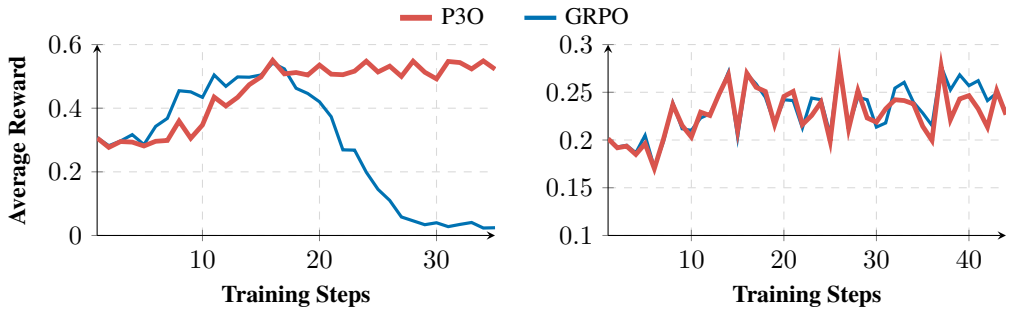
As shown in Fig. 2, at $T = 1.2$ the two methods stay within 0.04 of each other up to step 19 and peak at nearly the same reward (0.694 for P3O and 0.697 for GRPO). GRPO then falls from 0.567 at step 21 to 0.199 at step 23 and ends at 0.089, while P3O stays between 0.56 and 0.70 from step 10 to step 36. At $T = 0.6$ both methods peak at step 13 and then collapse, and P3O is below GRPO at 15 of the 22 steps both runs share, so P3O does not prevent this failure. On Qwen2.5-1.5B-Instruct (Fig. 6), both methods also collapse at $T = 0.6$, and neither degrades at $T = 1.2$.

BF16 train + FP8 rollout. The practical off-policy mismatch described in Sec. 1 is directly instantiated by mixed-precision pipelines: rollouts generated by an FP8-quantized policy carry



(a) Qwen3-4B-Thinking-2507 (Temperature 0.6) (b) Qwen3-4B-Thinking-2507 (Temperature 1.2)

Figure 2: **Training reward on off-policy data introduced through the varied sampling temperature of rollouts.** Sampling rollouts at a temperature other than 1.0 shifts the distribution of sampled tokens. At $T = 1.2$ only GRPO collapses; at $T = 0.6$ both methods do. Qwen2.5-1.5B-Instruct results are in Fig. 6.



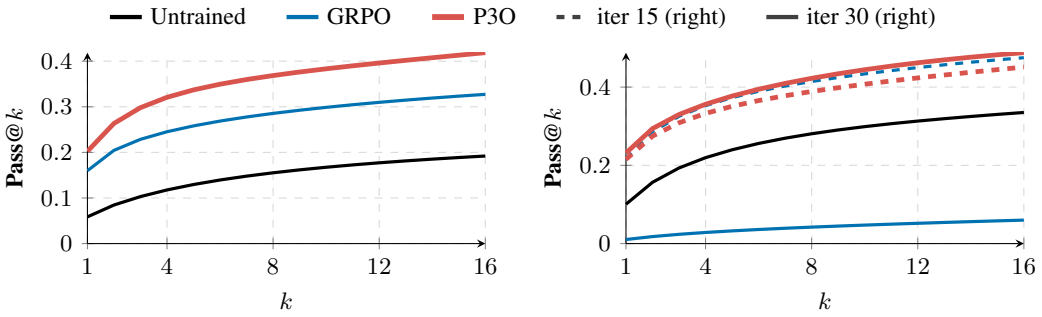
(a) Qwen3-8B-Base (b) Qwen2.5-1.5B-Instruct

Figure 3: **Training reward on off-policy data introduced through the BF16 train + FP8 rollout scheme.** Because accuracy can collapse with long FP8 rollouts (Xi et al., 2026), we use 16,384-token rollouts for Qwen3-8B-Base. On Qwen3-8B-Base, GRPO collapses after its peak at step 16 while P3O does not. On Qwen2.5-1.5B-Instruct, the curves overlap and neither degrades.

different token-level log-probabilities than the BF16 training model, shifting the per-token ratio ρ_t (Eq. (7)) away from one. Rollouts generated in lower precision can be faster to generate (vLLM Team, 2024). A common training strategy is to use BF16 precision for training and FP8 while creating rollouts, but this strategy can lead to collapse in reward performance, particularly for long rollouts (Xi et al., 2026). In practice, dynamic quantization methods are used where the latest trained policy is moved to the rollout engine with BF16 precision, then dynamically quantized to FP8 for rollout generation (vLLM Team, 2024; Xi et al., 2026). We test whether P3O (Eq. (12)) handles this mismatch without any change to the training configuration. GRPO takes its ratio against the same recorded log-probabilities and uses no decoupled loss or other correction for the mismatch. Both runs are configured for 500 iterations with 50 warmup iterations, so the learning rate is still rising at the evaluated iterations 15 and 30 (Table 5). As shown in Fig. 3a, GRPO on Qwen3-8B-Base is ahead during the first 15 steps, and the two methods peak at step 16 at nearly the same reward (0.542 for GRPO and 0.550 for P3O). GRPO then falls below its starting reward from step 22 and below 0.1 from step 27, while P3O stays above 0.49 up to step 35. On Qwen2.5-1.5B-Instruct (Fig. 3b) the curves overlap and neither method degrades; GRPO is slightly higher over the last five steps (0.248 against 0.234). The gain from P3O is thus the absence of the late collapse on Qwen3-8B-Base, not a higher peak.

4.4 Benchmark results

To test whether the differences in training reward carry over to held-out task performance, we evaluate the Qwen3-4B-Thinking-2507 checkpoints of the clip study and the Qwen3-8B-Base checkpoints



(a) Clip variants (4K-token eval)

(b) FP8 variants (16K-token eval)

Figure 4: **Pass@k averaged over all five held-out benchmarks (AIME24/25/26, AMO-Bench, AMC).** *Left*: clip variants of Qwen3-4B-Thinking-2507 (iter 35) at 4K-token evaluation; GRPO is averaged over $\epsilon \in \{0.2, 0.4, 0.6\}$, and Untrained is Qwen3-4B-Thinking-2507 before training. P3O is above this average at every k. *Right*: BF16 train + FP8 rollout variants of Qwen3-8B-Base at 16K-token evaluation, and Untrained is Qwen3-8B-Base before training. At iter 30 the average pass@k of GRPO is at most 0.060 and lies below the untrained model at every k, while that of P3O is above it.

of the FP8 study on five mathematical reasoning benchmarks (Table 2 in the appendix). These are AIME24 (Zhang & Math-AI, 2024), AIME25 (OpenCompass, 2025), AIME26 (Zhang & Math-AI, 2026), AMO-Bench (An et al., 2025), and AMC (Mathematical Association of America, 2024). AIME25 consists of the 15 problems of AIME 2025 I, and AMC of 45 problems from the 2024 AMC 12A and 12B. The DeepScaleR-Preview training set contains no samples from these benchmarks, so the table measures generalization.

Fig. 4 shows that the training-time stability differences seen in the reward curves also matter at evaluation time. In the clip study (Fig. 4a), P3O is above the average of the three GRPO runs at every k, although GRPO with $\epsilon = 0.6$ alone is above P3O on every benchmark at a 16,384-token budget (Table 4). At this budget, every clip-study checkpoint, trained with 1,024-token rollouts, scores below the untrained model on every benchmark (Sec. B.3). In a clip study on Qwen3-4B-Base with a DAPO baseline (Table 3), P3O lies between the best and the worst GRPO clip on every benchmark but is the best on none. In the FP8 rollout setting (Fig. 4b), P3O retains benchmark performance from iteration 15 to iteration 30, while GRPO degrades sharply.

5 Discussion

RL is structurally and algorithmically fragile, and large-model post-training sharpens that fragility because rollouts are expensive, hyper-parameters are consequential, and tuning costs compound with model scale. Recent approaches respond by adding or re-parameterizing fixed choices (asymmetric clip ranges, behavior-weight caps, etc.) tuned per task and model. We take the opposite path. The amount of trust placed in an update should be adaptive rather than pre-committed before training begins. Using the normalized effective sample size of the current policy ratios, the P3O objective replaces fixed clipping with a batch-adaptive score-function cap and matching regularizer, which behaves like the on-policy update when the ratios are close to one and tightens on stale or mismatched data. This does not make arbitrary off-policy data reliable, since meaningful token-level ratios and adequate support in the behavior data are still required (Sec. A), but it turns off-policy mismatch into a measured batch property and removes the need for a clip range or behavior-weight cap in the objective.

Removing the fixed clip also opens space for downstream work. Because P3O makes off-policy data directly usable inside one objective, future methods can combine off-policy reuse (for sample efficiency) with on-policy collection (for exploration) without retuning a clip range or behavior-weight cap for each setting. We view this paper as a starting point for that direction and for batch-adaptive methods that improve large-model post-training by reducing the surface of pre-committed knobs.

AI USE STATEMENT

We did not use generative AI tools to develop the method or theory, design or run experiments, or generate, clean, or reformat data. We used them mainly for writing and presentation assistance, including grammar correction, improving sentence flow, clarity, and readability, polishing the text, checking references and related work, writing code to improve the formatting and visual presentation of plots, and verifying reported experimental results against the associated logs. All AI-assisted content was reviewed and verified by the authors.

REFERENCES

Shengnan An, Xunliang Cai, Xuezhi Cao, Xiaoyu Li, Yehao Lin, Junlin Liu, Xinxuan Lv, Dan Ma, Xuanlin Wang, Ziwen Wang, and Shuang Zhou. AMO-Bench: Large language models still struggle in high school math competitions, 2025. URL https://arxiv.org/abs/2510.26768. 9

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? A large-scale study. In *International Conference on Learning Representations*, 2021. 1

Zhepeng Cen, Yao Liu, Siliang Zeng, Pratik Chaudhari, Huzefa Rangwala, George Karypis, and Rasool Fakoor. Bridging the training-inference gap in LLMs by leveraging self-generated tokens. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. 4

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 7, 18

DeepSeek-AI. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. 1

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020. 1

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1407–1416, 2018. 1

Rasool Fakoor, Pratik Chaudhari, and Alexander J. Smola. P3O: Policy-on policy-off policy optimization. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019*, volume 115 of *Proceedings of Machine Learning Research*, pp. 1017–1027, 2019. 2, 5, 6, 14

Rasool Fakoor, Pratik Chaudhari, and Alexander J Smola. DDPG++: Striving for simplicity in continuous-control off-policy reinforcement learning. *arXiv preprint arXiv:2006.15199*, 2020a. 1

Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. Meta-Q-Learning. In *International Conference on Learning Representations*, 2020b. 1, 3

Rasool Fakoor, Jonas W Mueller, Kavosh Asadi, Pratik Chaudhari, and Alexander J Smola. Continuous doubly constrained batch reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 11260–11273, 2021. 2

Rasool Fakoor, Jonas Mueller, Zachary C. Lipton, Pratik Chaudhari, and Alexander J. Smola. Time-varying propensity score to bridge the gap between the past and present. In *International Conference on Learning Representations*, 2024. 1, 3

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 36256–36282, 2025. 6, 18

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018. 1

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17086–17098, 2022. 2, 6, 18

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. 1

Augustine Kong. A note on importance sampling using standardized weights. Technical Report 348, Department of Statistics, University of Chicago, 1992. 2, 5

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020. 2

Yuhang Li, Reena Elangovan, Xin Dong, Priyadarshini Panda, and Brucek Khailany. QuRL: Efficient reinforcement learning with quantized rollout. In *International Conference on Learning Representations*, 2026. 19

Yao Liu, Pratik Chaudhari, and Rasool Fakoor. Budgeting counterfactual for offline RL. In *Advances in Neural Information Processing Systems*, volume 36, pp. 5729–5751, 2023. 2

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, 2025. URL https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset. Notion Blog. 7

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Fuli Luo, and Zhifang Sui. Stabilizing MoE reinforcement learning by aligning training and inference routers. In *International Conference on Machine Learning*, 2026. 2

Mathematical Association of America. 2024 AMC 12A and AMC 12B. https://huggingface.co/datasets/rawsh/2024_AMC12, 2024. American Mathematics Competitions. 9

MiniMax, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, et al. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025. 2, 5

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 *International Conference on Learning Representations*, 2025. 1

OpenCompass. AIME 2025 I: OpenCompass dataset, AIME2025-I configuration, 2025. URL https://huggingface.co/datasets/opencompass/AIME2025. Complete 15-problem configuration. 9

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022. 1, 4

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. PipelineRL: Faster on-policy reinforcement learning for long sequence generation. *Transactions on Machine Learning Research*, 2026. 20

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via FP16. *arXiv preprint arXiv:2510.26788*, 2025. 2

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/. 6

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388. 6

Sidney I. Resnick. *A Probability Path*. Modern Birkhäuser Classics. Birkhäuser, 2013. 3

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015. 1, 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 1, 2, 3

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 preprint arXiv:2402.03300*, 2024. 2, 3, 5

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pp. 3008–3021, 2020. 1, 4

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. 3

vLLM Team. FP8 quantization — vLLM documentation. https://docs.vllm.ai/en/v0.5.0.post1/quantization/fp8.html, 2024. Accessed: 2026-05-06. 8

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. 3

Haocheng Xi, Charlie Ruan, Peiyuan Liao, Yujun Lin, Han Cai, Yilong Zhao, Shuo Yang, Kurt Keutzer, Song Han, and Ligeng Zhu. Jet-RL: Enabling on-policy FP8 reinforcement learning with unified training and rollout precision flow. *arXiv preprint arXiv:2601.14243*, 2026. 8

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan 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, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. 2, 5, 7, 16

Yifan Zhang and Team Math-AI. American invitational mathematics examination (AIME) 2024, 2024. 9

Yifan Zhang and Team Math-AI. American invitational mathematics examination (AIME) 2026, 2026. 9

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025. 2, 5, 7

Zhengyang Zhuge, Hao Yu, Xin Wang, Zheng Li, Yizhong Cao, Dayiheng Liu, and Jianwei Zhang. QUADS: Stabilizing NVFP4 reinforcement learning for MoE via QUantization-error alignment across dual sides. *arXiv preprint arXiv:2607.15810*, 2026. 19

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019. 1, 4

A ALGORITHM PSEUDOCODE

We present pseudocode for GRPO and P3O as implemented in our training framework. Both algorithms share the same rollout phase and use group-relative advantages (Eq. (3)); they differ only in how the policy update is computed. GRPO clips the per-token policy ratio with a fixed clip range $(\epsilon_\ell, \epsilon_h)$, while P3O replaces this fixed clip with a batch-adaptive ESS cap and an adaptive KL regularizer. Throughout both algorithms, $\langle\cdot\rangle_M$ denotes the sum over the valid (non-padded) tokens indicated by mask M divided by the number of valid tokens in the whole rollout batch, $\mathbf{sg}(\cdot)$ is the stop-gradient operator, and $c_{<t} = (x, y_{<t})$ is the conditioning context at position t. Thus, within a rollout batch, the loss is the average over completions of the summed per-token terms, multiplied by the number of completions over the number of valid tokens, so this normalization changes the scale of the gradient of each batch but not its direction. GRPO uses the same normalization. We set $\beta_{\text{ent}} = 0$ in all experiments (H_θ is the entropy of $\pi_\theta(\cdot \mid c_{<t})$), and both algorithms omit the reference-KL term of Eq. (8), since $\eta = 0$.

Algorithm 1 GRPO

Require: Policy π_θ; group size G; clip $(\epsilon_\ell, \epsilon_h)$; entropy coefficient β_{ent}

Rollout: for each prompt x_p, sample G completions with the rollout engine, record their token log-probabilities $\log \pi_{\text{b}}$, and compute

$$A_{p,j} = \frac{r_{p,j} - \bar{r}_p}{\sqrt{\frac{1}{G-1}\sum_k (r_{p,k} - \bar{r}_p)^2 + \delta}}$$

where $\bar{r}_p = \frac{1}{G}\sum_k r_{p,k}$, and $A_t = A_{p,j}$ for every token y_t of $y_{p,j}$.
Update: for each micro-batch $\mathcal{B}$ with mask M:
1: $\rho_t \leftarrow \exp\big(\log \pi_\theta(y_t \,|\, c_{<t}) - \log \pi_{\text{b}}(y_t \,|\, c_{<t})\big)$
2: $\mathcal{L} \leftarrow -\big\langle \min\big(\rho_t A_t,\ \text{clip}(\rho_t, 1-\epsilon_\ell, 1+\epsilon_h)\, A_t\big)\big\rangle_M - \beta_{\text{ent}} \langle H_\theta \rangle_M$
3: Backward pass; accumulate gradients
After all micro-batches: one optimizer step

Algorithm 2 P3O

Require: Policy π_θ; group size G; entropy coefficient β_{ent} *(no clip range)*

Rollout: identical to Algorithm 1
Update: for each micro-batch $\mathcal{B}$ with mask M:
1: $\rho_t \leftarrow \exp\big(\log \pi_\theta(y_t \,|\, c_{<t}) - \log \pi_{\text{b}}(y_t \,|\, c_{<t})\big)$
2: $e_{\mathcal{B}} \leftarrow \mathbf{sg}\Big(\big(\sum_M \rho_t\big)^2 \big/ \big(|M| \cdot \sum_M \rho_t^2\big)\Big)$ (sums and $|M|$ all-reduced across data-parallel workers)
3: $\mathcal{L} \leftarrow -\langle \mathbf{sg}(\min\{\rho_t, e_{\mathcal{B}}\}) \log \pi_\theta A_t \rangle_M + (1 - e_{\mathcal{B}}) \langle \log \rho_t + 1/\rho_t - 1 \rangle_M - \beta_{\text{ent}} \langle H_\theta \rangle_M$
4: Backward pass; accumulate gradients
After all micro-batches: one optimizer step

The KL term. Algorithm 2 replaces the per-token KL of Eq. (12) by $h(\rho_t) = \log \rho_t + 1/\rho_t - 1$ at the sampled token, which is nonnegative and zero only at $\rho_t = 1$. Since y_t is drawn from π_{b}, its mean at each context is given by

$$\mathbb{E}_{\pi_{\text{b}}}[h(\rho_t)] = \chi^2(\pi_{\text{b}} \,\|\, \pi_\theta) - \text{KL}(\pi_{\text{b}} \,\|\, \pi_\theta),$$

and not $\text{KL}(\pi_\theta \,\|\, \pi_{\text{b}}) = \mathbb{E}_{\pi_{\text{b}}}[\rho_t h(\rho_t)]$. GRPO uses h for its reference KL with tokens drawn from $\pi_{\theta_{\text{old}}}$, which is close to π_θ in its intended setting, so it uses no such weight. For example, with $\pi_\theta = (0.9, 0.1)$ and $\pi_{\text{b}} = (0.5, 0.5)$, the KL is about 0.368, whereas the mean of h is about 1.267. With $\delta_t = \rho_t - 1$, both $h(\rho_t)$ and $\rho_t h(\rho_t)$ equal $\delta_t^2/2 + O(\delta_t^3)$, so both means equal $\chi^2(\pi_\theta \,\|\, \pi_{\text{b}})/2$ to second order, and the loss term matches the KL when the two policies are close. Fakoor et al. (2019) write $\text{KL}(\pi_{\text{b}} \,\|\, \pi_\theta)$ in their gradient but state that they use $\text{KL}(\pi_\theta \,\|\, \pi_{\text{b}})$, which they compute from

Training Method	AIME24	AIME25	AIME26	AMO-Bench	AMC
Clip Variants (Qwen3-4B-Thinking-2507, $\epsilon \in \{0.2, 0.4, 0.6\}$, 4K tokens)					
Untrained model (4K tokens)	0.029	0.033	0.006	0.008	0.217
GRPO (clip avg)	$0.122^{\pm 0.083}$	$0.160^{\pm 0.123}$	$0.126^{\pm 0.087}$	$\mathbf{0.012}^{\pm 0.007}$	$0.381^{\pm 0.195}$
P3O	**0.165**	**0.183**	**0.160**	0.010	**0.493**
FP8 Variants (Qwen3-8B-Base, BF16 train + FP8 rollout, 16K tokens)					
Untrained model (16K tokens)	0.077	0.121	0.056	0.021	0.228
FP8 Rollout GRPO Iter 15	0.154	0.250	0.160	0.021	0.499
FP8 Rollout GRPO Iter 30	0.002	0.000	0.002	0.019	0.029
FP8 Rollout DAPO Iter 15	0.071	0.142	0.065	0.025	0.349
FP8 Rollout DAPO Iter 30	0.006	0.004	0.004	0.015	0.075
FP8 Rollout P3O Iter 15	0.158	0.238	**0.179**	0.024	0.478
FP8 Rollout P3O Iter 30	**0.173**	**0.254**	0.175	**0.026**	**0.529**

Table 2: **Benchmark results (pass@1) across trained checkpoints.** Each group starts with its untrained model. Clip variants: Qwen3-4B-Thinking-2507 checkpoints from the clip study at iteration 35, evaluated with a 4,096-token budget; GRPO shows the mean $\pm$ population standard deviation across $\epsilon \in \{0.2, 0.4, 0.6\}$ (per-clip values at a 16,384-token budget are in Table 4). FP8 variants: Qwen3-8B-Base trained in BF16 with FP8 rollouts and evaluated with a 16,384-token budget; GRPO uses $\epsilon = 0.4$; DAPO uses $(\epsilon_\ell, \epsilon_h) = (0.4, 0.48)$ and a different learning-rate schedule (Sec. B.2). Each row other than the GRPO average is a single run, and entries are rounded half up. **Bold**: best trained checkpoint within each group per column.

stored action distributions. We keep this direction, but since we store only the sampled token's log-probability, we use $h(\rho_t)$ in its place. The derivative of $(1 - e_{\mathcal{B}})h(\rho_t)$ with respect to $\log \pi_\theta(y_t \mid c_{<t})$ is $(1 - e_{\mathcal{B}})(1 - 1/\rho_t)$, whose magnitude grows without bound as $\rho_t \to 0$ when $e_{\mathcal{B}} < 1$. Thus the cap bounds the score-function weight but not the whole per-token gradient.

Computing the ESS. Each micro-batch of Algorithm 2 gets its own $e_{\mathcal{B}}$, computed over its tokens and those of the micro-batches that the other data-parallel workers process at the same time (Table 6). Since each $e_{\mathcal{B}}$ depends on which tokens share a micro-batch, the same rollouts split differently can give a different cap and KL weight, and thus a different update. The micro-batches are formed by a seeded random shuffle of the rollout batch. Note that the empirical ESS measures how concentrated the observed ratios are; it is not a distance between π_θ and π_{b}. It does not change when every ratio is multiplied by the same constant: if $\rho_t = 0.5$ for every token, then $e_{\mathcal{B}} = 1$, the KL weight is zero and every score-function weight is 0.5. When tokens are sampled from π_{b} and π_{b} is positive wherever π_θ is, ρ_t has mean one at each context, so such a scaling is unlikely in a large batch, and the relation to $\chi^2(\pi_\theta \,\|\, \pi_{\mathrm{b}})$ after Eq. (11) holds in this population sense. However, a finite batch can miss rare tokens with large ratios, and its ESS can then be much higher than the population value. We therefore use the ESS to set the cap and the KL weight, not as a guarantee that the two policies are close.

B Benchmark results

Table 2 reports pass@1 for each checkpoint on all five held-out benchmarks. Pass@1 is the fraction of correct completions among 16 samples per problem, averaged over the problems of a benchmark. For $k \leq 16$, pass@k (Fig. 4) is the probability that at least one of k samples drawn without replacement from these 16 is correct, averaged in the same way. The untrained Qwen3-8B-Base and the DAPO checkpoints were evaluated in a later round than the FP8 GRPO and P3O checkpoints, the untrained model with a later revision of the evaluation code (the DAPO evaluations do not record their revision). All use the same sampling settings (Table 7).

Training Method	Iter.	AIME24	AIME25	AIME26	AMO-Bench	AMC
Untrained model	–	0.031	0.058	0.031	0.013	0.119
GRPO ($\epsilon = 0.2$)	50	0.071	0.088	0.029	0.010	0.271
GRPO ($\epsilon = 0.4$)	40	**0.108**	0.129	0.113	0.008	**0.381**
GRPO ($\epsilon = 0.6$)	50	0.104	0.125	0.100	0.008	0.365
P3O	50	0.088	0.125	0.085	0.009	0.354
DAPO ($\epsilon_\ell = 0.2$, $\epsilon_h = 0.28$)	30	0.106	**0.175**	**0.115**	**0.013**	0.347

Table 3: **Clip study on Qwen3-4B-Base: pass@1 at a 16,384-token budget.** Iter. is the evaluated checkpoint. GRPO with $\epsilon = 0.6$ and P3O differ only in the objective, while GRPO with $\epsilon = 0.2$ and 0.4 use a different GPU split and fewer completions per iteration (Sec. B.1). The DAPO run differs from the other runs in GPU split, batch size, schedule, prompt format and rollout engine, so it is not a matched comparison (Sec. B.2). Each row is a single run, and entries are rounded half up. **Bold**: best trained run per column.

B.1 Clip study on Qwen3-4B-Base

Table 3 reports a clip study on Qwen3-4B-Base, evaluated on the same benchmarks with a 16,384-token budget. The runs use 16,384-token rollouts at temperature 1.0, four completions per prompt, 512 requested prompts per iteration, 50 configured iterations, seed 42 and the optimizer settings of Table 5. Each rollout batch receives one optimizer step. The runs differ in their GPU split, which changes the number of completions per iteration. GRPO with $\epsilon = 0.2$ and $\epsilon = 0.4$ uses six training and two rollout GPUs and generates $512 \times 4 = 2{,}048$ completions per iteration. GRPO with $\epsilon = 0.6$ and P3O use two training and six rollout GPUs; each rollout engine takes 64 prompts, so the 512 requested prompts are rounded up to two full batches of 384 prompts, which gives 3,072 completions per iteration. In the logged configurations, the $\epsilon = 0.2$ and $\epsilon = 0.4$ runs differ only in the clip, and the $\epsilon = 0.6$ and P3O runs differ only in the objective. The evaluated checkpoints are iterations 50, 40, 50 and 50 for $\epsilon = 0.2$, 0.4, 0.6 and P3O, at which point the runs have generated 102,400, 81,920, 153,600 and 153,600 completions. For each run we evaluate the saved checkpoint that we judged strongest. The training prompts contain no system prompt, while the benchmark prompts contain one. The first row of Table 3 evaluates the untrained Qwen3-4B-Base with the same settings. P3O lies between the best and the worst GRPO clip on every benchmark but is the best on none.

B.2 DAPO baselines

Tables 2 and 3 include DAPO (Yu et al., 2025) as an additional baseline. The Qwen3-4B-Base run uses the clip pair $(\epsilon_\ell, \epsilon_h) = (0.2, 0.28)$ and is evaluated at iteration 30. The Qwen3-8B-Base run with FP8 rollouts uses $(0.4, 0.48)$ and is evaluated at iterations 15 and 30. Both runs use dynamic sampling, which drops prompt groups whose completions all receive the same reward and, unlike the original DAPO, does not sample new groups to replace them. They also use an overlong penalty with a 3,277-token buffer and a factor of 0.5; the benchmark evaluation applies no penalty. Both use five training and three rollout GPUs, 16,384-token rollouts, four completions per prompt and 256 requested prompts per iteration. With 16 prompts per rollout engine, the request is rounded up to 288 prompts, so each iteration generates 1,152 completions, as in the FP8 GRPO and P3O runs (Table 6). At the last logged iteration, dynamic sampling kept 432 of these completions in the Base run and 240 in the FP8 run.

The FP8 run uses a different learning-rate schedule from the FP8 GRPO and P3O runs. It configures 45 iterations with 4 warmup iterations, whereas the GRPO and P3O runs configure 500 iterations with 50 warmup iterations; their logs end at iterations 45 and 35, respectively, and Fig. 3a shows the first 35. At the evaluated iterations 15 and 30, the GRPO and P3O learning rates are therefore still rising (6.9×10^{-6} at iteration 15 and 8.7×10^{-6} at iteration 30 in both runs, against a peak of 10^{-5}), while the DAPO run is past its warmup. The Base run differs from the GRPO and P3O runs of Sec. B.1 in its GPU split, its batch (1,152 instead of 2,048 or 3,072 completions per iteration) and its schedule (40 configured iterations, and about 34,560 completions at the evaluated iteration 30). It

Training Method	AIME24	AIME25	AIME26	AMO-Bench	AMC
Untrained model	0.371	0.471	0.396	0.019	0.618
GRPO ($\epsilon = 0.2$)	0.033	0.008	0.017	0.001	0.103
GRPO ($\epsilon = 0.4$)	0.138	0.171	0.154	0.014	0.461
GRPO ($\epsilon = 0.6$)	**0.202**	**0.304**†	**0.235**	**0.018**	**0.565**
P3O	0.165	0.183	0.163	0.009	0.503

Table 4: **Clip study on Qwen3-4B-Thinking-2507: pass@1 at a 16,384-token budget.** The checkpoints are those of Table 2 (iteration 35), trained with 1,024-token rollouts. †A second evaluation of the same checkpoint path, run later with a newer revision of the evaluation code, gives 0.242 (58 instead of 73 correct completions out of 240; Sec. B.3). Each row is a single run, and entries are rounded half up. **Bold**: best trained run per column.

also differs in its training prompts, which contain the system prompt used at evaluation, and in its rollout engine, which does not run in batch-invariant mode (a vLLM setting in which the output for a request does not depend on how requests are batched). Its row in Table 3 is therefore not a matched comparison.

B.3 Qwen3-4B-Thinking-2507 checkpoints at a 16,384-token budget

Table 4 evaluates the Qwen3-4B-Thinking-2507 checkpoints of Table 2 with a 16,384-token budget. At this budget every trained checkpoint scores below the untrained model on every benchmark. GRPO with $\epsilon = 0.6$ is the best trained run on all five benchmarks, and P3O is second on four of them. The response lengths differ sharply between the runs. For GRPO with $\epsilon = 0.4$ on all five benchmarks and for P3O on the three AIME sets, the mean response length equals the budget at both 4,096 and 16,384 tokens, so every one of these completions runs to the budget; the verifier still scores these truncated completions. P3O stays just below the budget on AMO-Bench and AMC (4,011 and 4,047 tokens at 4K; 16,028 and 16,080 at 16K), and GRPO with $\epsilon = 0.2$ averages 15,932 to 16,294 tokens at 16K. GRPO with $\epsilon = 0.6$ stops early, with means of 1,378 to 2,215 tokens at both budgets, and the untrained model averages 11,005 to 16,162 tokens at 16K.

The evaluations in Table 4 come from two rounds that used different revisions of the evaluation code. The untrained model, GRPO with $\epsilon = 0.4$, the AIME25 entry of GRPO with $\epsilon = 0.6$ and the three AIME entries of P3O come from the first round, and the remaining entries from the second; the 4K-token entries of Table 2 all come from the first round. At this budget the $\epsilon = 0.6$ checkpoint path gives 73 correct AIME25 completions out of 240 in the first round and 58 in the second, and we do not know the cause; Table 4 reports the first-round value. On the other four benchmarks of this checkpoint, the first-round 4K-token and second-round 16K-token evaluations differ by at most 9 correct completions, and its responses are short at both budgets, so we see no systematic difference between the two rounds there.

C Additional off-policy experiments

C.1 Asynchronous training

In asynchronous training, the one-step and two-step pipeline variants show the same ordering: GRPO is ahead in the middle of training and P3O at the end. The two variants use the same maximum lag of 10 policy versions and differ only in the number of optimizer steps per iteration. We show the one-step setting in Fig. 5; its caption also gives the two-step results.

C.2 Sampling temperature on Qwen2.5-1.5B-Instruct

Fig. 6 shows the Qwen2.5-1.5B-Instruct runs at $T = 0.6$ and $T = 1.2$: at $T = 0.6$ both methods collapse, and at $T = 1.2$ neither degrades.

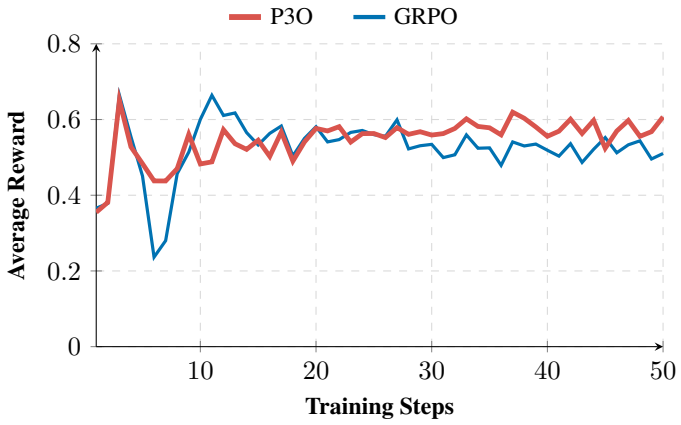
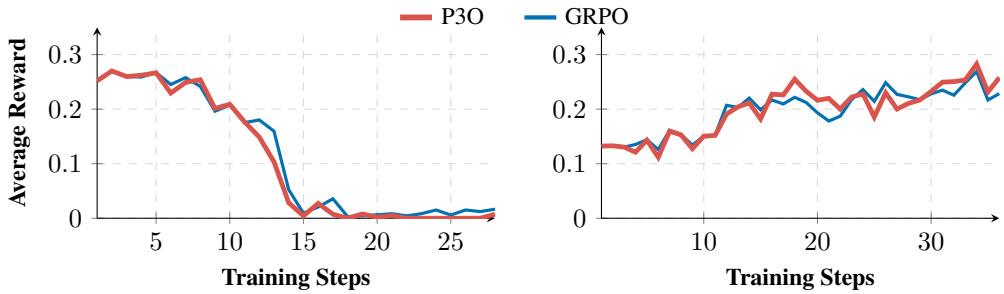


Figure 5: **Asynchronous-training comparison between P3O and GRPO under one optimizer step per iteration.** Qwen3-4B-Thinking-2507 is trained on GSM8K (Cobbe et al., 2021) with 1,024-token rollouts, and the replay buffer keeps rollouts that are at most 10 policy versions old. Rollouts are generated by a stale policy while the training engine continues updating. GRPO uses the decoupled loss (Hilton et al., 2022; Fu et al., 2025) with $\epsilon = 0.4$ and $c_w = 5$, while P3O uses Eq. (12) unchanged. GRPO takes its proximal snapshot before the single optimizer step of each iteration, so $\pi_{\text{prox}} = \pi_\theta$ at that step, the ϵ clip does not bind, and the GRPO gradient equals that of the CISPO row of Table 1 with the cap $1 + \epsilon_h$ set to $c_w = 5$. In this one-step pipeline setting, both methods peak at step 3 (0.651 for P3O and 0.665 for GRPO). GRPO then drops to 0.237 at step 6 and P3O to 0.438 at step 7, and GRPO is ahead at 10 of the 11 steps from 10 to 20. P3O is ahead at 22 of the last 23 steps and ends at 0.607 against 0.510. With two optimizer steps per iteration (not shown), P3O is ahead from iteration 37 on and ends at 0.697 against 0.618.



(a) Qwen2.5-1.5B-Instruct (Temperature 0.6)

(b) Qwen2.5-1.5B-Instruct (Temperature 1.2)

Figure 6: **Temperature results for Qwen2.5-1.5B-Instruct, corresponding to Fig. 2.** At $T = 0.6$ both methods collapse (mean reward over the last five steps of 0.001 for P3O and 0.013 for GRPO), and at $T = 1.2$ neither degrades.

D Two-anchor extension of P3O

D.1 Full formulation

One possible issue with P3O is that the single ESS $e_{\mathcal{B}}$ in Eq. (12) conflates two distinct drifts in the batch: the difference between π_θ and the behavior policy π_{b}, and the within-iteration difference between π_θ and its pre-update snapshot π_{prox}. When both drifts are large but in different directions, a single anchor cannot distinguish them. A natural extension replaces the single-anchor regularizer of Eq. (12) with a KL toward an ESS-weighted mixture of the behavior policy π_{b} and a proximal snapshot π_{prox} of π_θ taken at the start of each iteration. The full loss is

$$\mathcal{L}_{\text{ext}}(\theta) = -\mathbb{E}_{\pi_{\text{b}}}\big[\mathbf{sg}(\min\{r_b,\, e_{\text{mix}}\})\, \log \pi_\theta(y_t \mid c_{<t})\, A\big] + (1 - e_{\text{mix}})\, \mathrm{KL}\big(\pi_\theta \,\big\|\, \pi_{\text{mix}}\big), \qquad (13)$$

with the following per-batch quantities, where r_b is the ratio ρ_t of Eq. (7) and e_b equals $e_{\mathcal{B}}$ (Eq. (11)):

$$
\begin{aligned}
r_b &= \frac{\pi_\theta(y_t \mid c_{<t})}{\pi_{\mathrm{b}}(y_t \mid c_{<t})}, \quad r_{\mathrm{prox}} = \frac{\pi_\theta(y_t \mid c_{<t})}{\pi_{\mathrm{prox}}(y_t \mid c_{<t})}, \\
e_b &= \frac{\left(\sum_{\mathcal{B}} r_b\right)^2}{|\mathcal{B}| \sum_{\mathcal{B}} r_b^2}, \quad e_{\mathrm{prox}} = \frac{\left(\sum_{\mathcal{B}} r_{\mathrm{prox}}\right)^2}{|\mathcal{B}| \sum_{\mathcal{B}} r_{\mathrm{prox}}^2}, \\
e_{\mathrm{mix}} &= \min(e_b,\, e_{\mathrm{prox}}), \\
\pi_{\mathrm{mix}}(\cdot \mid c_{<t}) &= \frac{(1-e_b)\,\pi_{\mathrm{b}}(\cdot \mid c_{<t}) + (1-e_{\mathrm{prox}})\,\pi_{\mathrm{prox}}(\cdot \mid c_{<t})}{(1-e_b) + (1-e_{\mathrm{prox}})}.
\end{aligned} \tag{14}
$$

As in P3O, e_b and e_{prox} (and thus e_{mix} and the two mixture weights) are detached, and π_{prox} receives no gradient. When $1 - e_{\mathrm{mix}} \le 10^{-6}$, which includes the case $e_b = e_{\mathrm{prox}} = 1$ where π_{mix} is undefined, we set the regularizer to zero. Note that both ESS values are computed on the tokens of $\mathcal{B}$, which are sampled from π_{b}. Thus e_{prox} is not the importance-sampling ESS of samples from π_{prox}; it measures the relative spread of the ratios r_{prox} over $\mathcal{B}$ and equals one when they are all equal (e.g., when $\pi_\theta = \pi_{\mathrm{prox}}$). Each anchor is weighted by its $(1 - \mathrm{ESS})$, so a fully on-policy axis (whose ESS approaches one) drops out of the mixture and the regularizer pulls only toward the mismatched anchor. The construction reduces to P3O when $e_{\mathrm{prox}} \to 1$: the proximal weight then vanishes, $\pi_{\mathrm{mix}} \to \pi_{\mathrm{b}}$ and $e_{\mathrm{mix}} \to e_b$, recovering Eq. (12). If instead $e_b \to 1$, the behavior weight vanishes and $\pi_{\mathrm{mix}} \to \pi_{\mathrm{prox}}$, so the regularizer anchors at the proximal snapshot and not at π_{b}. The construction adds no hyper-parameter beyond what P3O already uses. The second anchor has an effect only when an iteration takes more than one optimizer step: with one step, as in the clip and FP8 runs (Table 6), $\pi_{\mathrm{prox}} = \pi_\theta$ at that step, so $e_{\mathrm{prox}} = 1$ and the extension reduces to P3O.

D.2 Training stability

Fig. 7 reports training curves for the two-anchor extension of P3O described in Sec. D.1, alongside P3O and GRPO baselines. All three algorithms are trained on Qwen3-4B-Thinking-2507 using the DeepScaleR-Preview dataset with temperature 1.2 and a rollout length of 4,096 tokens. The number of optimizer steps per iteration of the two-anchor runs was not recorded.

The two-anchor extension comes close to the peak reward of P3O and GRPO (0.673 against 0.694 and 0.697) during the first 16 steps but then collapses abruptly, falling to near-zero reward (0.012) before the run terminates at step 24. P3O remains stable throughout the full training run, and GRPO collapses later, falling from 0.567 at step 21 to 0.199 at step 23 (Fig. 2). We have not identified the cause of the collapse. Addressing this instability (e.g., through tighter proximal resets, adaptive KL weighting, or learning-rate schedules) is left as future work.

Fig. 8 presents a complementary experiment at default rollout temperature with 1,024-token rollouts (its P3O and GRPO curves are the clip runs of Table 6), where the same three methods are compared and GRPO again uses ϵ=0.4.

E Additional related work

Quantized rollouts. QuRL (Li et al., 2026) generates rollouts with a quantized (INT8 or FP8) actor. To prevent long-term collapse, it sets the clip range adaptively from the ratio between the full-precision and the quantized actor, and it rescales the weights (invariant scaling) so that the small weight changes between RL steps are not lost to quantization. Its finding that a fixed clip range is not enough under quantized rollouts is consistent with our FP8 result on Qwen3-8B-Base (Fig. 3a). The two methods differ in how they respond: QuRL makes the clip adaptive for this source of mismatch, whereas P3O removes the clip and sets its cap and regularizer from the batch ESS, with the same objective in every setting we test. Invariant scaling acts on the quantized weights and not on the objective, so it could be combined with P3O; we have not tested this combination. QUADS (Zhuge et al., 2026) studies NVFP4 rollouts of mixture-of-experts models trained in BF16 with GRPO and aligns the quantization error on both sides: the trainer fake-quantizes the weights, and the rollout engine compensates the activation channels with the largest error. QUADS leaves the GRPO objective unchanged, so it is complementary to P3O as well.

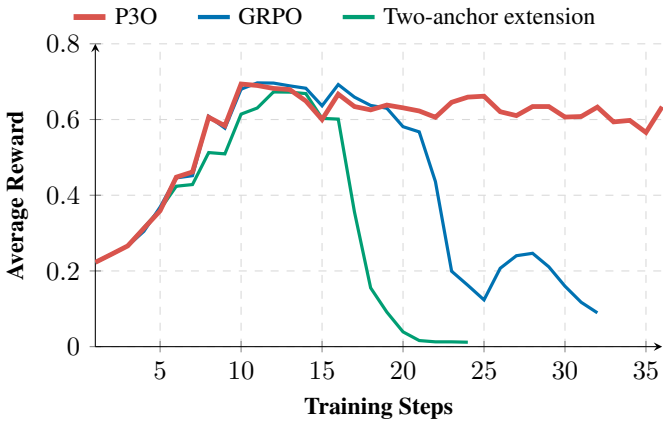


Figure 7: **Training curves of the two-anchor extension, P3O, and GRPO on Qwen3-4B-Thinking-2507.** All runs use the DeepScaleR-Preview dataset with temperature 1.2 and a rollout length of 4,096 tokens. The two-anchor extension peaks at a reward comparable to P3O and GRPO but undergoes a sharp collapse after step 16, dropping to near-zero reward by step 24 before the run terminates. P3O stays between 0.56 and 0.70 from step 10 to step 36, while GRPO collapses later, falling from 0.567 at step 21 to 0.199 at step 23.

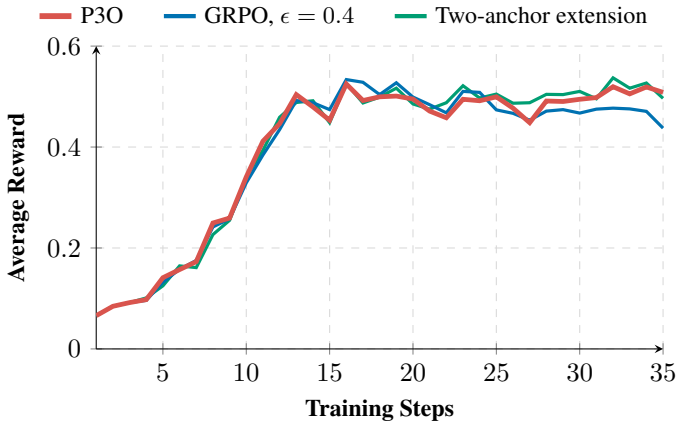


Figure 8: **Training curves of the two-anchor extension, P3O, and GRPO (ϵ=0.4) on Qwen3-4B-Thinking-2507 at default temperature.** All runs use the DeepScaleR-Preview dataset with a rollout length of 1,024 tokens. Unlike the temperature-1.2 regime (Fig. 7), the two-anchor extension remains stable throughout training and ends at 0.497, against 0.508 for P3O and 0.438 for GRPO with ϵ=0.4; over the last five steps it averages 0.515 against 0.510 for P3O.

In-flight weight updates. PipelineRL (Piché et al., 2026) runs generation and training concurrently and updates the weights of the rollout engines after each optimizer step without restarting the sequences in progress, so the tokens of one sequence can come from several policy versions. Since P3O takes each ratio per token against the log-probability that the rollout engine recorded (Sec. 3), such sequences need no single behavior policy. We have not tested P3O with in-flight weight updates.

F Additional Experimental Details

This section summarizes the settings for the main runs.

Table 6 lists the settings of the three Qwen3 experiment families: the clip and temperature runs train Qwen3-4B-Thinking-2507 with 1,024- and 4,096-token rollouts on six training and two rollout GPUs, and the FP8 runs train Qwen3-8B-Base with 16,384-token rollouts on five training and three rollout GPUs. The benchmark prompts contain a system prompt; the FP8 runs train with it and the clip

Hyper-parameter	Value
Optimizer	AdamW
Learning rate	1e-5
Betas (β_1, β_2)	(0.9, 0.95)
Weight decay	0.01
Gradient clipping	1.0
LR scheduler	WarmupCosineLR (10% warmup ratio)

Table 5: **Shared optimizer settings used across all experiments.** The warmup covers 10% of the configured iterations of each run, rounded down. The FP8 GRPO and P3O runs configure 500 iterations, so their checkpoints at iterations 15 and 30 are within the 50-iteration warmup (Sec. B.2).

Hyper-parameter	Clip Runs	Temperature Runs	FP8 Runs
Model	Qwen3-4B-Thinking-2507	Qwen3-4B-Thinking-2507	Qwen3-8B-Base
Training / rollout GPU split	6 train + 2 rollout	6 train + 2 rollout	5 train + 3 rollout
Optimizer steps / iteration	1	–	1
Train micro-batch / GPU	8	4	1
Gradient accumulation	1	2	1
Rollout batch size / GPU	64	64	16
Requested prompts / iteration	512	512	256
Completions / iteration	2,048	2,048	1,152
Max tokens	1,024	4,096	16,384
Sampling temperature	1.0	$T \in \{0.6, 1.2\}$	1.0
GRPO clip setting	$\epsilon \in \{0.2, 0.4, 0.6\}$	$\epsilon = 0.4$	$\epsilon = 0.4$

Table 6: **Per-experiment training configuration.** Completions per iteration are the requested prompts times four completions per prompt, after the rollout engines round the prompt count up to full batches (256 requested prompts become 288 with three engines of 16 prompts each). In the clip and FP8 runs, each iteration takes one optimizer step on the gradients accumulated over all its micro-batches, so the gradient-accumulation row (a DeepSpeed setting) does not change the update. The micro-batch row, together with the number of training GPUs, sets the tokens over which P3O computes each ESS (Eq. (11)). The configuration files of the temperature runs were not archived; their column gives the settings from our run notes, which do not record the number of optimizer steps per iteration. The Qwen2.5-1.5B-Instruct GRPO clip runs use the settings of the clip column other than the model; the configurations of the other Qwen2.5-1.5B-Instruct runs were not archived.

Hyper-parameter	Value
Shared benchmark-evaluation settings	
Temperature	1.0
Top-p	0.95
Samples / problem	16
Rollout Batch Size / GPU	16
Rollout GPUs	8
Data Workers	8
Benchmarks (problems)	AIME24 (30), AIME25 (15), AIME26 (30), AMO-Bench (50), AMC (45)
4K-token budget (Qwen3-4B-Thinking-2507 clip study)	
Max Response Tokens	4,096
16K-token budget (FP8, Qwen3-4B-Base and Qwen3-4B-Thinking-2507 checkpoints)	
Max Response Tokens	16,384
Model & system configuration	
Precision	bfloat16
Tensor Parallel Size	1

Table 7: **Benchmark-evaluation settings, grouped by evaluation budget.** We evaluate with a 4K-token or a 16K-token budget; all other sampling settings and the GPU allocation are the same for every benchmark and checkpoint.

runs without it. Table 2 evaluates the clip-study checkpoints at 4,096 tokens and all others at 16,384 tokens; Sec. B.3 also evaluates the clip-study checkpoints at 16,384 tokens.

Setting	Value
Accelerator Configuration	
GPU Model	NVIDIA H100 (clip, temperature and FP8 runs) NVIDIA A100 80GB (Fig. 5)
Total GPUs per run	8
Evaluation partition	8 rollout GPUs
Tensor Parallel Size	1
Runtime Configuration	
Training Precision	bfloat16
Rollout Precision	bfloat16; FP8 only in the FP8 runs
Rollout Engine Mode	vLLM batch-invariant where logged (off in FP8 and DAPO runs)
Distributed Training	DeepSpeed ZeRO-3
Trainer Attention Backend	Flash Attention 2
Workload Summary	
Training Dataset	DeepScaleR-Preview; GSM8K for Fig. 5
Training Models	Qwen3-4B-Thinking-2507, Qwen3-4B-Base, Qwen3-8B-Base, Qwen2.5-1.5B-Instruct
Evaluation Rollout Batch Size / GPU	16
Software Environment	
Python version	3.13.1 or 3.12 (recommended per code revision)
NVIDIA driver	CUDA 12.x-compatible; driver version $\geq$ 525.85 recommended
CUDA toolkit	12.2 (tested); toolkit versions $\geq$ 12.2 supported
PyTorch build	CUDA 12.6 wheels
Training and Rollout Stack	
PyTorch / TorchVision	CUDA-enabled install
DeepSpeed	0.18.9
vLLM	0.19.0
Transformers	4.57.6
Ray	2.54.1
FlashAttention	2.8.3 (built from source)

Table 8: **Compute layout and software environment for the reported runs.** The eight GPUs are partitioned differently across runs (Table 6; Sec. B.1 and B.2). The software versions are those that the training code pins or recommends at the revisions used for the runs; the runtime versions were not logged.

Reproducibility. Each configuration is run once with a fixed seed for Python, NumPy, PyTorch, and rollout sampling. We further reduce nondeterminism using vLLM batch-invariant mode[4], deterministic cuBLAS and cuDNN settings, disabled cuDNN benchmarking, and a fixed `PYTHONHASHSEED`. Some nondeterminism remains, e.g., from distributed floating-point reductions.

[4] https://docs.vllm.ai/en/latest/features/batch_invariance/