Trust the Batch, Online or Offline: Adaptive Policy Optimization for Post-Training

Anonymous Author(s)
Affiliation
Address
email

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, where the training and rollout systems differ in numerical precision, sampling, and other implementation details. Existing methods manage this fragility by adding more hyper-parameters to the training objective. Each may help in its tuned regime but makes the resulting algorithm more sensitive to its configuration, requiring retuning whenever the task, model scale, or distribution mismatch changes. This fragility traces to two concerns that current objectives entangle through hyper-parameters set before training begins. The first is a *trust-region* concern, in that each update should not move the policy too far from its current value. The second is an *off-policy* concern, in that data collected by older or different behavior policies should influence the current update only to the extent that the update remains reliable. Mishandling off-policy data is consequential, yet such data can still carry useful signal that must be weighted adaptively as training proceeds. Neither concern is a constant to set before training, and their severity is reflected in the policy-ratio distribution of the current batch. We present a simple yet effective batch-adaptive objective that replaces fixed clipping with the normalized effective sample size of the policy ratios. The same statistic caps the score-function weight and sets the strength of an off-policy regularizer. When the ratios are nearly uniform, the update stays close to the usual on-policy score-function update. When stale or mismatched data cause ratio concentration, the update tightens automatically while retaining a nonzero learning signal on high-ratio tokens. Experiments across a wide range of settings show that our method matches or exceeds tuned baselines, introducing no new objective hyper-parameters and removing several existing ones.

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 [37, 25, 19, 4]. 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, implementation details, and random seeds [12, 5, 2, 8, 14], and 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, regardless of where the data come

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

from [22, 23]. 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 [9, 10, 6]. The trust region controls the size of an update, while off-policy correction weights data by how reliably it can be used.

Off-policy data are common in practice and arise from several sources, including optimizer steps reused across iterations, demonstrations or rollouts from older or heterogeneous policies (the batch-RL setting [11]), and practical mismatches between the rollout and training engines such as numerical precision, sampling strategy, or architectural differences [20, 17]. Mishandling such data is consequential, since stale or mismatched samples destabilize training, yet discarding it forfeits sample efficiency because each rollout is then consumed by a small number of updates [7]. Current large-model RL methods commit to one regime, either generating fresh on-policy rollouts after every update or applying fixed off-policy corrections tuned for a particular staleness, which forces a choice between sample efficiency and stability before training begins.

These methods conflate the two concerns inside a single fixed-clip mechanism [23, 24], where a fixed range bounds the per-step policy change and also caps the off-policy variance. This is simple, but the range is a strong algorithmic choice made before training starts, determining 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, improving learning from stale data [13], and recent variants add asymmetric clipping, dynamic sampling, and related modifications [32]. These methods recognize the issue but still leave the amount of trust as an external choice, and the burden of tuning remains.

The premise of this paper is that fixed clipping itself, not the value of the clip range, is what causes the fragility, and iterating on the clip (symmetric, asymmetric, sequence-level) leaves the underlying problem in place. Our contribution is to step back from the fixed-clip framework rather than propose another variant, and measure how on-policy the current batch is and use that measurement to drive the update. The behavior-policy mismatch, defined as how different the current policy is from the policy that produced each completion, can be summarized by an effective sample size (ESS) [15, 7] statistic over the per-token importance ratios. ESS is close to one when the ratios are nearly uniform, and falls when a few tokens dominate. We use only ESS to drive both the score-function cap and the regularizer in the surrogate objective, removing the fixed clip range entirely.

Concretely, we adopt the P3O objective of [7] for large-model post-training, the first such application, and show that it removes the fixed clip without introducing any new hyper-parameter. The same objective handles on-policy and off-policy data automatically, with the cap loose on fresh data and tight on stale data, and unifies what prior methods split into specialized losses. Across a range of regimes, this objective compares favorably with tuned baselines despite carrying fewer hyper-parameters, demonstrating that adaptivity, not careful clip tuning, is what drives stability at scale.

Removing the clip rather than re-parameterizing it also opens space for downstream work. Because P3O makes off-policy data a feature rather than a burden, future methods can combine off-policy reuse (for sample efficiency) with on-policy collection (for exploration) inside one objective, without retuning a clip range, behavior-weight cap, or staleness budget for each setting. We view this paper as a foundation for that direction.

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 $y = (y_1, \ldots, y_T)$ 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 $\mathcal{V}$, $y_{<t} = (y_1, \ldots, y_{t-1})$ is the prefix at position t, and T is the completion length. To simplify notation, 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 [30], 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 [30]. 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 [26]. 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 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, from training instability to challenges at scale. GRPO [24] 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 $\{y_{p,j}\}_{j=1}^{G}$ from the current policy 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

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

where $\epsilon > 0$ avoids division by zero, and the variance in the denominator is computed in the population form to match the original GRPO definition. The group mean acts as a per-prompt baseline that replaces the value function, and the group standard deviation rescales the advantage so it is comparable across prompts whose reward magnitudes differ.

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 [22], Proximal Policy Optimization (PPO) [23] takes a simpler approach by using first-order methods to keep new policies close to old. Concretely, PPO samples data from a fresh snapshot $\pi_{\theta_{\text{old}}}$ taken at the start of the current optimizer epoch, 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 serves a clean trust-region role: bounding the per-step policy change.

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 the gradient step that produced it. In practice the data we train on is rarely drawn from π_θ exactly. This may be because we take several optimizer steps on each batch of completions and reuse them across training iterations, because we want to use data collected by an earlier or entirely different policy (for instance demonstrations or rollouts from a prior run), or because of a more general discrepancy between the policy we sample from and the policy we optimize. In all of these cases the data was 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 $\nabla_\theta J(\pi_\theta)$.

[1] The formulation extends to per-token rewards $r_t(x, y_{\le t})$ by replacing $r(x, y)$ with $r_t(x, y_{\le t})$ inside the sum in Eq. (2).

The standard fix is importance sampling [9, 10, 21], which restores unbiasedness by changing the measure from π_θ to π_{b}. For any function f,

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

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.

Remark (Importance sampling is unbiased but high variance). Importance sampling restores unbiasedness, 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. In practice we do not maximize $J(\pi_\theta)$ alone. When fine-tuning language models with RL, the objective is 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 [3]),[2]

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

Here $\eta > 0$ controls how far π_θ is allowed to drift from π_{θ_0} [37, 25, 19]. The KL anchor is an effective regularizer, intended to keep π_θ from drifting too far from π_{θ_0} and potentially from forgetting its language-modeling capabilities. The per-token KL is combined with the policy-gradient term under the same batch-token average (Sec. 3).

3 Approach

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 $\rho_t = \pi(y_t \mid c_{<t})/\pi_{\mathrm{old}}(y_t \mid c_{<t})$[3]. Their effect 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

$$-\min\left(\frac{\pi(y_t \mid c_{<t})}{\pi_{\mathrm{old}}(y_t \mid c_{<t})},\, 1+\epsilon_h\right) A. \tag{9}$$

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

$$-\max\left(\frac{\pi(y_t \mid c_{<t})}{\pi_{\mathrm{old}}(y_t \mid c_{<t})},\, 1-\epsilon_\ell\right) A, \tag{10}$$

where the $\max$ replaces the $\min$ because A is negative. The objective is improved by decreasing $\pi(y_t \mid c_{<t})$ and hence ρ_t. Once $\rho_t < 1-\epsilon_\ell$, the contribution hits $-(1-\epsilon_\ell)\, A$, and the new policy

[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.

[3] We drop the subscript θ and use the shorthand $c_{<t}$ from Sec. 2.

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_{old} while continuing to improve the surrogate; clipping acts as 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 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 destabilize training. There is no single value appropriate across both regimes, and $(\epsilon_\ell, \epsilon_h)$ 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 these two hyper-parameters and differ only in how they re-parameterize them. GRPO [24] uses Eq. (4) with a symmetric range $\epsilon_\ell = \epsilon_h$. DAPO [32] relaxes the symmetry, allowing $\epsilon_\ell \neq \epsilon_h$ to handle positive- and negative-advantage updates separately. GSPO [36] 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. Across these objectives, $(\epsilon_\ell, \epsilon_h)$ survives in some form (see Table 1).

Effective Sample Size to the Rescue. The value of $(\epsilon_\ell, \epsilon_h)$ is therefore consequential, and tuning it for large models is expensive. The effective sample size (ESS) of the policy ratios in the current batch [7] provides a way to determine this cap automatically, removing the need to fix $(\epsilon_\ell, \epsilon_h)$ in advance. Concretely, the ESS is given by

$$\mathrm{ESS}(\mathcal{B};\theta) = \frac{\left(\widehat{\mathbb{E}}_{\mathcal{B}}\left[\frac{\pi(y_t|c_{<t})}{\pi_{\text{old}}(y_t|c_{<t})}\right]\right)^2}{\widehat{\mathbb{E}}_{\mathcal{B}}\left[\left(\frac{\pi(y_t|c_{<t})}{\pi_{\text{old}}(y_t|c_{<t})}\right)^2\right]} = \frac{\widehat{\mathbb{E}}_{\mathcal{B}}[\rho_t]^2}{\widehat{\mathbb{E}}_{\mathcal{B}}[\rho_t^2]} \in \left[\frac{1}{|\mathcal{B}|},\, 1\right], \tag{11}$$

where $\mathcal{B}$ is the set of valid response tokens in the current training batch, $|\mathcal{B}|$ is its size, and $\widehat{\mathbb{E}}_{\mathcal{B}}$ denotes the empirical average over $\mathcal{B}$. 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 batch is on-policy and falls when a few large ratios dominate, as happens with stale or mismatched data. An objective driven by $e_{\mathcal{B}}$ therefore behaves like an on-policy update on fresh data and tightens automatically when the data drifts.

Policy-on Policy-off Policy Optimization (P3O) for large-model. We adopt the P3O objective of [7], applied here to large-model post-training for the first time. P3O replaces the fixed clip in Eq. (4) with two terms whose strength is set by the batch ESS,

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

where ρ_t is the per-token policy ratio defined in the section opening, and the per-token KL, written in the same form as Eq. (8), vanishes when the batch is on-policy and grows as it drifts.

P3O makes two structural changes to Eq. (4). First, it eliminates the fixed clip range $(\epsilon_\ell, \epsilon_h)$ entirely, replacing it with the data-driven cap $e_{\mathcal{B}}$ that adapts to the current batch. Second, it adds a regularizer with coefficient $(1 - e_{\mathcal{B}})$ that is large precisely when the batch is most off-policy and vanishes on fresh data. Crucially, while the clip in Eq. (4) discards the gradient on every token whose ratio falls outside $(1 - \epsilon_\ell,\, 1 + \epsilon_h)$, the cap $\min\{\rho_t,\, e_{\mathcal{B}}\}$ in Eq. (12) only scales the score-function gradient by a positive factor: every token in the batch contributes to the update and no data is wasted. Both adaptations are driven by a single statistic of the current batch, so Eq. (12) introduces no clipping range, no trust-region coefficient, and no staleness budget. While the form of Eq. (12) matches the original P3O, previous works for large-model RL have pursued a different path, proposing narrow variants of the GRPO clip with new hyper-parameters that succeed only in limited regimes. Adopting P3O for large-model post-training instead addresses the fragility at its source and yields a simpler and more effective solution.

Table 1: Comparison of policy objectives, with $\rho_t = \pi_\theta(y_t \mid x, y_{<t}) / \pi_{\mathrm{b}}(y_t \mid x, y_{<t})$. **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	$-\min\left(\rho_t A,\ \mathrm{clip}\left(\rho_t,\ 1-\boldsymbol{\epsilon_\ell},\ 1+\boldsymbol{\epsilon_h}\right) A\right)$
GSPO	$-\min\left(S_\theta(y)A,\ \mathrm{clip}\left(S_\theta(y),\ 1-\boldsymbol{\epsilon}^{\mathbf{seq}},\ 1+\boldsymbol{\epsilon}^{\mathbf{seq}}\right) A\right), \quad S_\theta(y) = \exp\left(\frac{1}{T}\sum_{t=1}^{T} \log \rho_t\right)$
Decoupled	$-\mathbf{sg}\left(\mathrm{clip}\left(\frac{\boldsymbol{\pi_{\mathrm{prox}}}}{\pi_{\mathrm{b}}},\ 0,\ \boldsymbol{c_w}\right)\right) \min\left(\frac{\pi_\theta}{\boldsymbol{\pi_{\mathrm{prox}}}} A,\ \mathrm{clip}\left(\frac{\pi_\theta}{\boldsymbol{\pi_{\mathrm{prox}}}},\ 1-\boldsymbol{\epsilon_\ell},\ 1+\boldsymbol{\epsilon_h}\right) A\right)$
P3O	$-\mathbf{sg}(\min\{\rho_t,\ \boldsymbol{e_{\mathcal{B}}}\})\ \log \pi_\theta(y_t \mid x, y_{<t})\, A\ +\ (1-\boldsymbol{e_{\mathcal{B}}})\, \mathrm{KL}(\pi_\theta \,\|\, \pi_{\mathrm{b}})$

Remark (ESS adapts the clip without hyper-parameters). P3O sets both the cap on the score-function weight and the regularizer coefficient from a single batch statistic, the ESS, and updates them at every gradient step from the data the optimizer is currently seeing. This one adaptive mechanism replaces the clip range $(\epsilon_\ell, \epsilon_h)$ and the auxiliary trust-region or staleness parameters that fixed-clip methods rely on, and introduces no new hyper-parameter in their place. As the off-policy degree of the batch changes during training, the cap and the regularizer track it without retuning.

Off-policy data in large-model training. At large-model scale, the data the optimizer sees is rarely strictly on-policy. Training and rollout engines differ in numerical precision, sampling, and other implementation details (Sec. 2), and reusing rollouts across optimizer epochs widens the gap further. The standard fix is a decoupled-loss objective [13] that clips against a proximal snapshot π_{prox} rather than against $\pi_{\theta_{\mathrm{old}}}$. This stabilizes training but adds new hyper-parameters, including the construction of π_{prox}, its lifetime, and a cap c_w on an outer behavior weight, all of which must be tuned for each off-policy regime. P3O sidesteps this. The same $e_{\mathcal{B}}$ that drives the cap on fresh data falls when the batch becomes off-policy, so Eq. (12) handles both regimes in a single objective without any additional hyper-parameter or special code path. Our experiments (Sec. 4) confirm that this single objective performs well across off-policy regimes that previously required specialized losses. This makes off-policy data a feature rather than a burden, since P3O can reuse rollouts from older or different policies and demonstrations to improve sample efficiency and reduce training time without a specialized loss (see Table 1 for the parameter contrast across methods).

3.1 A potential issue with P3O

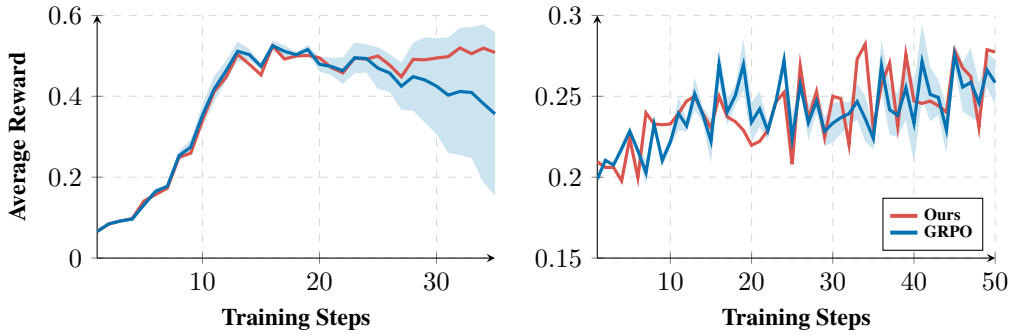
One possible issue with P3O is that the single ESS $e_{\mathcal{B}}$ in Eq. (12) conflates two distinct drifts in the batch: the difference between π_θ and the data-generating policy π_b, and the within-epoch difference between π_θ and its pre-update snapshot π_{prox}. When both drifts are large but in different directions, a single anchor cannot distinguish them. A natural extension introduces a second anchor at π_{prox} and pulls π_θ toward an ESS-weighted mixture π_{mix} of the two anchors:

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

where $r_b = \pi_\theta(y_t \mid c_{<t}) / \pi_b(y_t \mid c_{<t})$ is the per-token behavior ratio, e_{mix} is a joint ESS computed from both anchors, and π_{mix} is a per-token mixture of π_b and π_{prox} weighted by their respective $(1-e_b)$ and $(1-e_{\mathrm{prox}})$ so that the more on-policy anchor drops out (full formulation in Sec. B). This variant introduces no new hyper-parameter and reduces to P3O when either anchor is uninformative. Empirically, however, the single-anchor P3O matches or slightly outperforms this variant across the off-policy regimes considered in Sec. 4, suggesting that the behavior-axis ESS already captures most of the relevant drift signal.

4 Experiments

We evaluate P3O across three axes: (*i*) sensitivity to the clipping hyper-parameters $(\epsilon_\ell, \epsilon_h)$ that P3O eliminates entirely; (*ii*) robustness to off-policy data arising 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



(a) Qwen3-4B-Thinking-2507 (b) Qwen2.5-1.5B

Figure 1: **Sensitivity of GRPO's reward to the clip range $\epsilon \in \{0.2, 0.4, 0.6\}$ (shaded region: ± 1 std over clip values) versus P3O run once with no clip hyperparameter.** P3O's ESS-driven cap (Eq. (11)) removes this tuning burden while matching or exceeding the best GRPO variant across both model families.

same base objective (Sec. 2) but relies on a fixed clip range; both methods otherwise use identical hyper-parameters within each experiment family (Table 3).

4.1 Experiment Setup

Experiments use the open-source models Qwen3-4B-Thinking-2507 [28] and Qwen2.5-1.5B [27], trained on the DeepScaleR-Preview dataset of 40,000 mathematics problem-answer pairs compiled from AIME (1984-2023), AMC (prior to 2023), Omni-MATH, and Still [16]. 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. All runs use 8 NVIDIA H100 GPUs in a distributed stack that separates optimizer workers from rollout engines and synchronizes policy weights under a shared scheduler. Training, evaluation, hardware, and benchmark details are in Tables 3 to 6 and Sec. C and E.

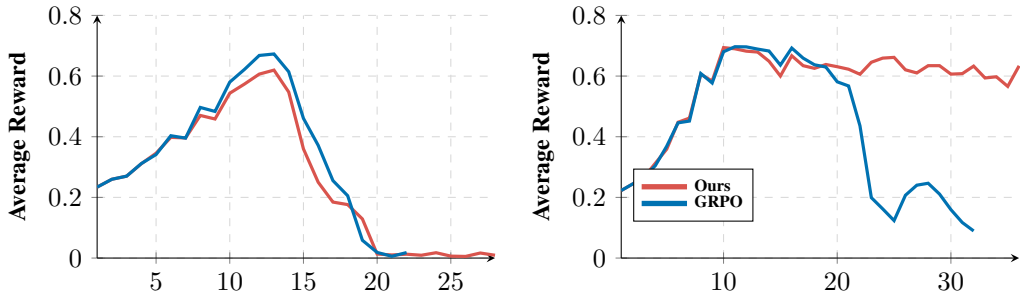
4.2 Effects of Hyperparameters

Clipping Factor. As argued in Sec. 3, the fixed clip range $(\epsilon_\ell, \epsilon_h)$ is a pre-committed choice that cannot adapt to the batch, and a value suited to on-policy data may over- or under-clip when rollouts become stale or mismatched (Eqs. (9) and (10)). We verify this sensitivity 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. As shown in Fig. 1, GRPO's reward trajectory varies substantially with ϵ (shaded region), while P3O remains stable by adjusting the score-function cap and regularizer from the batch ESS (Eq. (11)). Other clip-based baselines such as DAPO [32] and GSPO [36] retain a fixed clip range and exhibit equivalent hyper-parameter sensitivity under this ablation, so the key variable we isolate is the presence or absence of a fixed clip.

4.3 Off-Policy Data

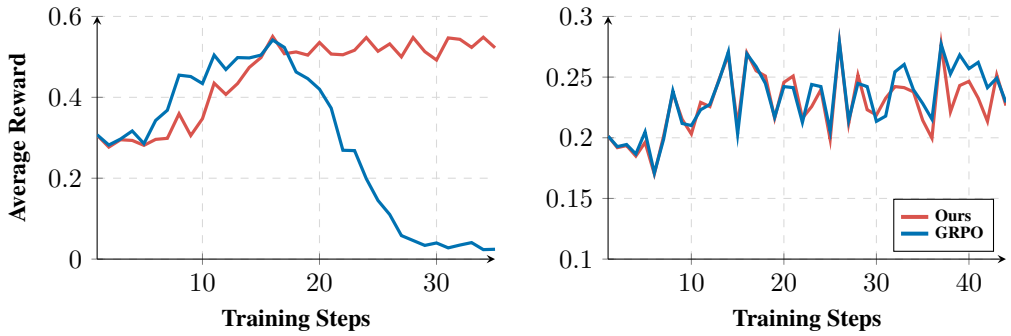
Off-policy mismatch arises in large-model RL post-training whenever rollouts are not drawn from the current policy, including from optimizer steps reused across iterations, mixed-precision inference engines, and non-standard sampling strategies (Sec. 1). The two experiments below isolate two of these sources and test whether P3O's batch ESS (Eq. (11)) handles each organically.

Temperature of Rollouts. Sampling rollouts at temperature $T \neq 1.0$ uniformly rescales token log-probabilities, shifting the per-token ratio ρ_t (Eq. (7)) by a constant factor across the entire batch. For GRPO, this offset falls either inside or outside the fixed clip interval independent of how on-policy the batch otherwise is, a bias that cannot be corrected without retuning ϵ. The ESS (Eq. (11)) directly measures this shift as ratio concentration and tightens the score-function cap and regularizer accordingly.



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

Figure 2: **P3O is robust to off-policy data introduced through the varied sampling temperature of rollouts.** Sampling rollouts at a temperature other than 1.0 introduces a distribution shift in the token-level log probabilities, creating off-policy data for Qwen3-4B-Thinking-2507. Corresponding Qwen2.5-1.5B results are deferred to Fig. 7.



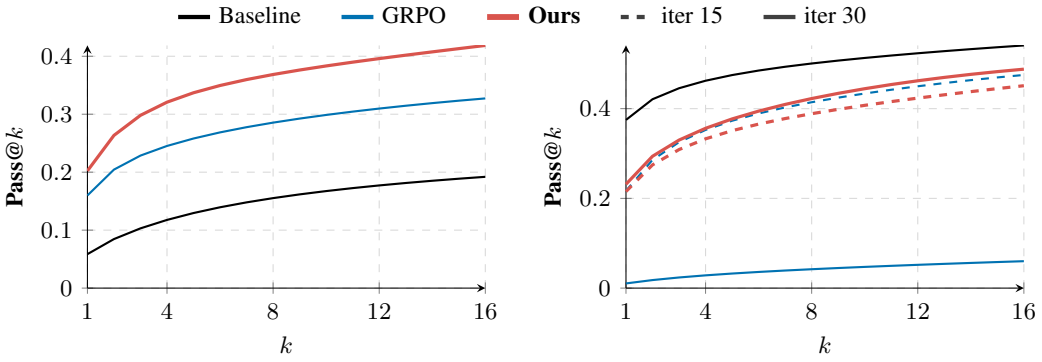
(a) Qwen3-4B-Thinking-2507 (b) Qwen2.5-1.5B

Figure 3: **P3O is robust to off-policy data introduced through the BF16 Train + FP8 Rollout training scheme.** As accuracy collapse is observed in longer rollout lengths [31], a rollout length of 16,384 tokens was used in this experiment. The demonstrated robustness of P3O to off-policy data allows for the use of faster rollout generation strategies. In contrast, GRPO's performance degrades significantly under the same conditions, highlighting its sensitivity to off-policy data.

As shown in Fig. 2, GRPO's performance degrades at both $T = 0.6$ and $T = 1.2$ relative to the on-policy baseline on Qwen3-4B, while P3O matches or exceeds standard-temperature performance without retuning. The same qualitative trend appears for Qwen2.5-1.5B in Fig. 7, confirming that the batch ESS adapts to the induced ratio shift across both model families.

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 different token-level log-probabilities than the BF16 training model, shifting the per-token ratio ρ_t (Eq. (7)) away from one. We test whether P3O's adaptive regularizer (Eq. (12)) handles this mismatch without any change to the training configuration.

Off-policy data is created when rollouts are generated by a model quantized to a different numerical precision than the training model. Rollouts generated in lower precision can be much faster to generate, leading to a faster training run overall. 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 large `max_tokens`, because the rollout policy and training policy no longer induce the same token-level probabilities [31]. 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 [29]. As shown in Fig. 3, P3O remains stable when FP8 quantization pushes importance ratios away from one, maintaining training quality without any change to the training configuration, whereas GRPO collapses later in training under the same mismatch.



(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-ratio variants at 4K-token evaluation; GRPO is averaged over $\epsilon \in \{0.2, 0.4, 0.6\}$. Ours matches or exceeds the averaged GRPO sweep without requiring a clip-ratio choice. *Right*: BF16-train + FP8-rollout variants at 16K-token evaluation. GRPO collapses by iter 30 (near-zero pass@k) while Ours retains strong performance.

4.4 Benchmark Results

To confirm that the reward-curve advantages of P3O translate to held-out task performance, we evaluate checkpoints of Qwen3-4B-Thinking-2507 trained with each method on five mathematical reasoning benchmarks (Table 7). These include AIME24 [33], AIME25 [34], AIME26 [35], AMO-Bench [1], and AMC [18], all of which are standard held-out benchmarks for mathematical reasoning. Notably, 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-sensitivity study (Fig. 4a), P3O is competitive with or better than the averaged GRPO sweep at every k while avoiding the clip-selection burden entirely. In the FP8 rollout setting (Fig. 4b), P3O retains benchmark performance much later into training, while GRPO degrades sharply—by iter 30 its pass@k is near zero across all benchmarks. Full per-benchmark results are reported in Table 7 in the appendix.

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, staleness budgets, etc.) tuned per task and model, and each addition makes the algorithm more sensitive to its configuration rather than less. 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, P3O replaces fixed clipping with a batch-adaptive score-function cap and matching regularizer, behaving like an on-policy update on fresh data and tightening automatically 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, but it turns off-policy mismatch into a measured batch property and removes the need for clip ranges, behavior-weight caps, and staleness budgets. The same mechanism also makes off-policy data directly usable inside one objective, whereas prior methods reuse it only through specialized losses or per-regime retuning. Across the regimes we test, including clip sweeps, temperature shifts, and BF16/FP8 mixed precision, P3O matches or exceeds tuned GRPO baselines with no objective hyper-parameters to set. We view this paper as a starting point for batch-adaptive methods that improve large-model post-training by reducing, rather than expanding, the surface of pre-committed knobs.

References

[1] 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. 9

[2] 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 (ICLR)*, 2021. 1

[3] 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. 4

[4] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Nature*, 645:633–638, 2025. 1

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

[6] 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*, pages 1407–1416, 2018. 2

[7] 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*, page 371, 2019. 2, 5

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

[9] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. Meta-q-learning. In *ICLR*, 2020. 2, 4

[10] 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 *ICLR*, 2024. 2, 4

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

[12] 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

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

[14] 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. 1

[15] Augustine Kong. A note on importance sampling using standardized weights. *Technical Report 348*, 1992. 2

[16] 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. Notion Blog. 7

[17] Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025. 2

[18] Mathematical Association of America. 2023 american mathematics competitions (amc 10 and amc 12). https://huggingface.co/datasets/math-ai/amc23, 2023. Dataset curated by the Math-AI community. 9

[19] 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. *arXiv preprint arXiv:2203.02155*, 2022. 1, 4

[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

[21] Sidney I Resnick. *A probability path*. Springer Science & Business Media, 2013. 4

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015. 2, 3

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

[24] 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

[25] 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 *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. 1, 4

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

[27] Qwen Team. Qwen2.5: A party of foundation models, September 2024. 7

[28] Qwen Team. Qwen3 technical report, 2025. 7, 16

[29] vLLM Team. Fp8 quantization — vllm documentation. https://docs.vllm.ai/en/v0.5.0.post1/quantization/fp8.html, 2024. Accessed: 2026-05-06. 8

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

[31] 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, 2026. 8

[32] 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

[33] Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024. 9

[34] Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025. 9

[35] Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2026, 2026. 9

[36] 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. 5, 7

[37] 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

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 from π_θ, record $\log \pi_{\text{b}} = \log \pi_\theta$, and compute

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

Update: for each micro-batch $\mathcal{B}$ with mask M:

$\rho_t \leftarrow \exp\big(\log \pi_\theta(y_t \,|\, c_{<t}) - \log \pi_{\text{b}}(y_t \,|\, c_{<t})\big)$

$\mathcal{L} \leftarrow -\big\langle \min\big(\rho_t A_t,\ \text{clip}(\rho_t, 1{-}\epsilon_\ell, 1{+}\epsilon_h)\, A_t\big)\big\rangle_M - \beta_{\text{ent}} \langle H_\theta \rangle_M$

Backward pass; 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:

$\rho_t \leftarrow \exp\big(\log \pi_\theta(y_t \,|\, c_{<t}) - \log \pi_{\text{b}}(y_t \,|\, c_{<t})\big)$

$e_{\mathcal{B}} \leftarrow \big(\sum_M \rho_t\big)^2 \big/ \big(|M| \cdot \sum_M \rho_t^2\big)$ (ESS, all-reduced across workers)

$\mathcal{L} \leftarrow -\langle \mathbf{sg}(\min\{\rho_t, e_{\mathcal{B}}\}) \log \pi_\theta A_t \rangle_M + (1{-}e_{\mathcal{B}}) \langle \text{KL}(\pi_\theta \| \pi_{\text{b}}) \rangle_M - \beta_{\text{ent}} \langle H_\theta \rangle_M$

Backward pass; optimizer step

A Algorithm Pseudocode

We present pseudocode for GRPO and P3O as implemented in our FeynRL 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 uses a fixed clip range $(\epsilon_\ell, \epsilon_h)$ to bound the per-token policy ratio, while P3O replaces this fixed clip with a batch-adaptive ESS cap and an adaptive KL regularizer, introducing no new hyper-parameters. Throughout both algorithms, $\langle \cdot \rangle_M$ denotes the mean over the valid (non-padded) tokens indicated by mask M, $\mathbf{sg}(\cdot)$ is the stop-gradient operator, and $c_{<t} = (x, y_{<t})$ is the conditioning context at position t.

B Two-anchor extension of P3O: full formulation

The extension discussed in Sec. 3.1 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 optimizer epoch. The full loss is

$$\mathcal{L}_{\text{ext}}(\theta) = -\mathbb{E}_{\pi_b}\big[\mathbf{sg}(\min\{r_b,\, e_{\text{mix}}\})\, \log \pi_\theta(y_t \mid c_{<t})\, A\big] + (1 - e_{\text{mix}})\, \text{KL}\big(\pi_\theta \,\big\|\, \pi_{\text{mix}}\big), \tag{14}$$

with the following per-batch quantities:

$$\begin{aligned}
r_b &= \frac{\pi_\theta(y_t \mid c_{<t})}{\pi_b(y_t \mid c_{<t})}, \quad r_{\text{prox}} = \frac{\pi_\theta(y_t \mid c_{<t})}{\pi_{\text{prox}}(y_t \mid c_{<t})}, \\
e_b &= \frac{\big(\sum_{\mathcal{B}} r_b\big)^2}{|\mathcal{B}| \sum_{\mathcal{B}} r_b^2}, \quad e_{\text{prox}} = \frac{\big(\sum_{\mathcal{B}} r_{\text{prox}}\big)^2}{|\mathcal{B}| \sum_{\mathcal{B}} r_{\text{prox}}^2}, \\
e_{\text{mix}} &= \min(e_b,\, e_{\text{prox}}), \\
\pi_{\text{mix}}(\cdot \mid c_{<t}) &= \frac{(1 - e_b)\, \pi_b(\cdot \mid c_{<t}) + (1 - e_{\text{prox}})\, \pi_{\text{prox}}(\cdot \mid c_{<t})}{(1 - e_b) + (1 - e_{\text{prox}})}.
\end{aligned} \tag{15}$$

Each anchor is weighted by its $(1-\text{ESS})$, 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 the single-anchor P3O regularizer when either anchor is uninformative: if $e_b \to 1$ the

behavior weight vanishes and $\pi_{\text{mix}} \to \pi_{\text{prox}}$; if $e_{\text{prox}} \to 1$ the proximal weight vanishes and $\pi_{\text{mix}} \to \pi_b$, recovering Eq. (12). The construction adds no hyper-parameter beyond what P3O already uses.

C Additional experimental details

This appendix summarizes the settings for the main runs and a small set of supplementary off-policy ablations.

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 2: Shared optimizer settings used across all experiments.

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
Global train batch size	48	48	5
Train micro-batch / GPU	8	4	1
Gradient accumulation	1	2	1
Rollout batch size / GPU	64	64	16
Rollout samples / epoch	512	512	256
Max tokens	1024	4096	16384
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 3: Per-experiment training configuration. Clip and temperature runs use the 4B model with a 6/2 GPU split; the FP8 ablation uses the 8B model with a 5/3 split and a 16K-token rollout budget.

Hyper-parameter	**Value**
Shared benchmark-evaluation settings	
Temperature	1.0
Top-p	0.95
Group Size (n_{samples})	16
Rollout Batch Size / GPU	16
Rollout GPUs	8
Data Workers	8
4K benchmark eval family (clip + temperature)	
Benchmarks	AIME24, AIME25, AIME26, AMO-Bench, AMC
Max Response Tokens	4096
16K benchmark eval family (baseline + FP8)	
Benchmarks	AIME24, AIME25, AIME26, AMO-Bench, AMC
Max Response Tokens	16384
Model & system configuration	
Precision	bfloat16
Tensor Parallel Size	1

Table 4: Benchmark-evaluation settings grouped by rollout-length family. Separate 4K-token and 16K-token evaluation regimes are used in the paper, but the sampling policy and GPU allocation are otherwise held fixed across benchmarks and checkpoints.

The main distinction across training families is the rollout budget and GPU partition. Table 3 summarizes all three experiment families: clip and temperature runs use the 4B model with a 6/2

Hardware Attribute	Value
Accelerator Configuration	
GPU Model	NVIDIA H100 (as reported in Sec. 4)
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 mixed-precision ablation
Distributed Training	DeepSpeed ZeRO-3
Attention Backend	Flash Attention 2
Workload Summary	
Training Dataset	DeepScaleR-Preview
Training Models	Qwen3-4B-Thinking-2507 and Qwen3-8B-Base
Evaluation Rollout Batch Size / GPU	16

Table 5: Compute layout for the reported runs, together with the accelerator SKU reported in the main text. The eight GPUs are partitioned differently across ablation families depending on whether the workload is optimizer-heavy or rollout-heavy.

Environment Attribute	Value
Base environment	
Python version	3.13.1 (recommended/tested)
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
Core 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 6: Software environment used for the reported experiments. The table records only the main CUDA and library versions needed to contextualize the reported pipeline.

train-rollout split, while the FP8 ablation uses the 8B model with a 5/3 split and a 16K-token rollout budget. Benchmark evaluation is likewise split into separate 4K-token and 16K-token regimes, which is why Table 7 mixes both context budgets.

Under asynchronous optimization, both the one-step and two-step pipeline variants preserve the same qualitative ordering: P3O remains more stable than GRPO as rollout staleness increases. We show the one-step setting in Fig. 5 because it is visually cleaner; the omitted two-step variant follows the same trend.

D Two-anchor extension training stability

Fig. 8 reports training curves for the two-anchor extension of P3O described in Sec. B—alongside P3O and GRPO baselines. All three algorithms are trained on Qwen3-4B-Thinking using the DeepSeek dataset with temperature 1.2 and a rollout length of 4,096 tokens.

The two-anchor extension matches the peak reward of P3O and GRPO ($\approx$0.67) during the first 16 steps but then collapses abruptly, falling to near-zero reward before the run terminates at step 24. P3O remains stable throughout the full training run, and GRPO degrades more gradually in the later steps. The collapse suggests that the dual-anchor regularizer amplifies gradient variance once the proximal snapshot drifts far from the behavior policy, destabilizing optimization under the current

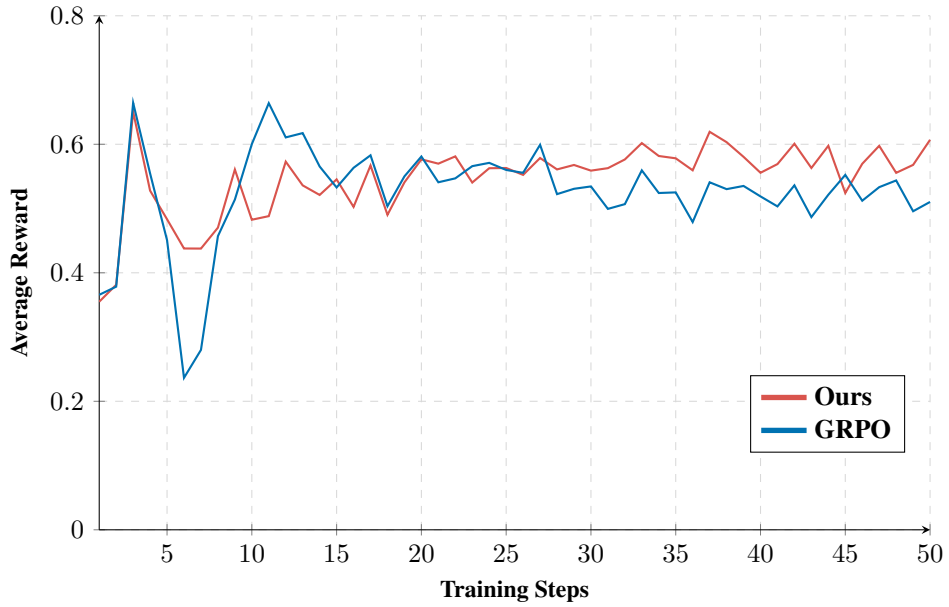


Figure 5: **Asynchronous-training comparison between P3O and GRPO under one optimizer step per rollout epoch.** Rollouts are generated by a stale policy while the learner continues updating, creating the off-policy lag discussed in the main text. In this one-step pipeline setting, P3O maintains a higher and more stable reward trajectory than GRPO across training. The corresponding two-step pipeline produces the same qualitative ordering and is omitted to avoid duplicating the same comparison with only a modest increase in late-training noise.

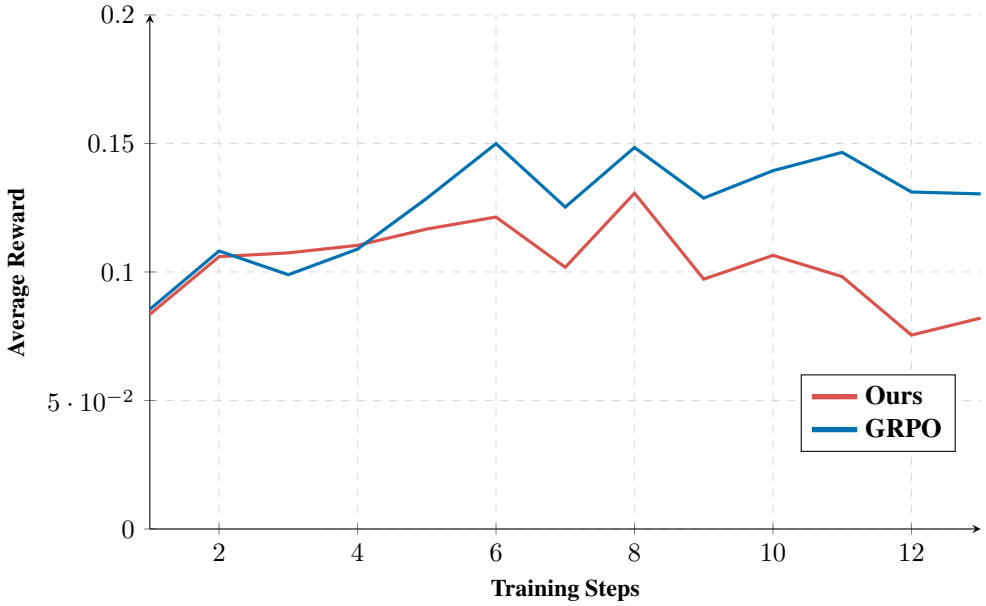
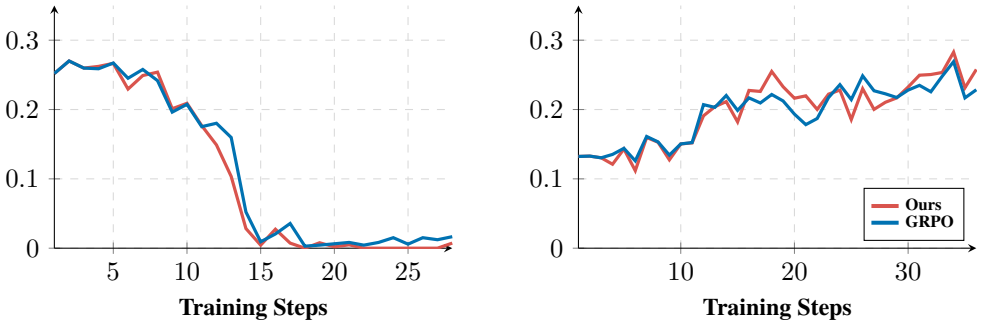


Figure 6: **Comparison of GRPO and P3O with respect to the mixing of off-policy data.** A rollout length of 4,096 tokens was used in this experiment. This experiment uses Qwen3-8B [28] to generate rollouts for training Qwen3-4B-Thinking-2507, as it is from a different model family. Data was mixed at a 50% ratio, meaning half of the rollouts were generated by the training model and half were generated by the separate policy.



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

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

Figure 7: **Temperature-robustness results for Qwen2.5-1.5B, corresponding to Fig. 2(c,d).** As in the main-text Qwen3-4B experiments, changing rollout temperature introduces a token-level distribution shift that degrades GRPO while P3O remains comparatively stable.

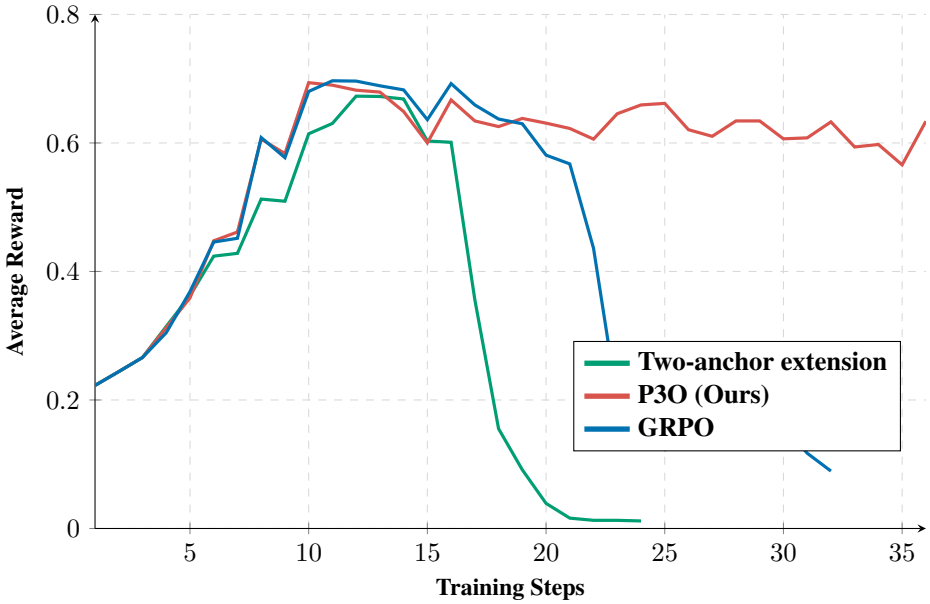


Figure 8: **Training curves for the two-anchor extension of P3O, P3O, and GRPO on Qwen3-4B-Thinking.** All runs use the DeepSeek dataset with temperature 1.2 and a rollout length of 4,096 tokens. The two-anchor extension peaks at a reward comparable to P3O and GRPO but undergoes a sharp collapse after step 16, dropping to near-zero reward by step 24 before the run terminates. P3O maintains stable, high reward throughout training, while GRPO shows partial instability in later steps. These results indicate that the two-anchor regularizer, though theoretically motivated, introduces training instability under the current hyperparameter regime.

hyperparameter setting. Addressing this instability—through tighter proximal resets, adaptive KL weighting, or learning-rate schedules—is left as future work.

Fig. 9 presents a complementary experiment at default rollout temperature, where the same three algorithm families are compared but GRPO is run with a larger clip ratio (ϵ=0.4).

E Benchmark results

Table 7 reports pass@1 for each checkpoint on all five held-out benchmarks.

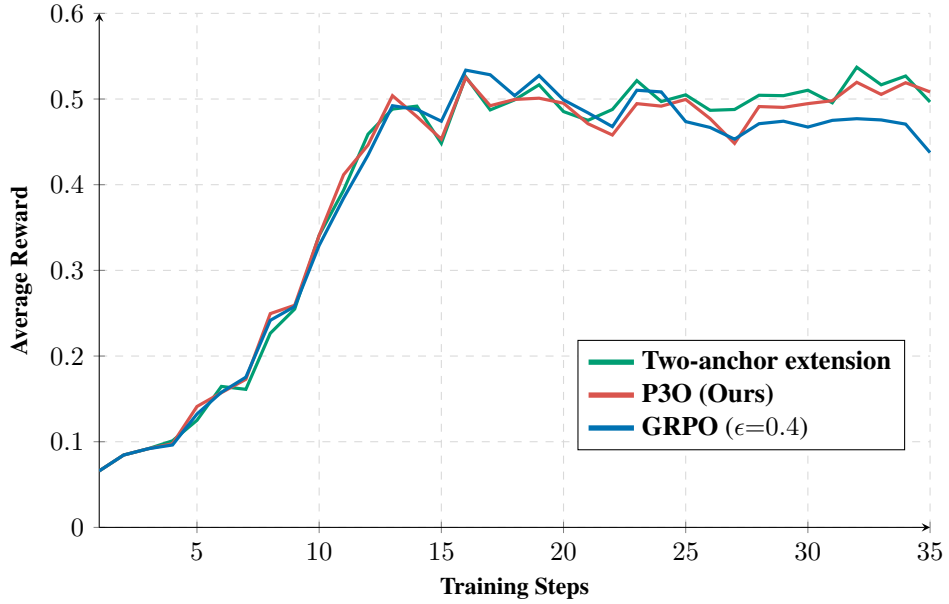


Figure 9: **Training curves for the two-anchor extension of P3O, P3O, and GRPO (ϵ=0.4) on Qwen3-4B-Thinking at default temperature.** All runs use the DeepSeek dataset with a rollout length of 4,096 tokens. Unlike the temperature-1.2 regime (Fig. 8), the two-anchor extension remains stable throughout training and achieves the highest final reward ($\approx$0.50), slightly outpacing both P3O and GRPO with ϵ=0.4. The standard GRPO baseline (default ϵ) reached only $\approx$0.19 reward in this setting and is excluded from the plot for clarity. Together with Fig. 8, these results suggest that the two-anchor extension's instability is sensitive to temperature: at higher rollout temperatures the dual-anchor regularizer can destabilize training, whereas at the default temperature it performs comparably to or better than P3O.

Training Method	**AIME24**	**AIME25**	**AIME26**	**AMO-Bench**	**AMC**
Baseline Model (4K tokens)	0.029	0.033	0.006	0.007	0.217
Baseline Model (16K tokens)	0.371	0.471	0.396	0.019	0.618
Clip Variants ($\epsilon \in \{0.2, 0.4, 0.6\}$, 4K tokens)					
GRPO (clip avg)	$\mathbf{0.176}^{\pm 0.039}$	$0.160^{\pm 0.123}$	$0.126^{\pm 0.087}$	$\mathbf{0.012}^{\pm 0.007}$	$0.381^{\pm 0.195}$
P3O	0.165	**0.183**	**0.160**	0.010	**0.493**
FP8 Variants (BF16 train + FP8 rollout, 16K tokens)					
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 P3O Iter 15	0.158	0.237	**0.179**	0.024	0.478
FP8 Rollout P3O Iter 30	**0.173**	**0.254**	0.175	**0.026**	**0.529**

Table 7: **Benchmark results (pass@1) across trained checkpoints.** Baseline rows report untrained Qwen3-4B-Thinking-2507 at two rollout lengths. Clip-variant GRPO shows mean $\pm$ std across $\epsilon \in \{0.2, 0.4, 0.6\}$. FP8 rows report individual checkpoints. **Bold**: best within each group per column.