SCORE CENTERING STABILIZES OFF-POLICY REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) of large language models is notoriously sensitive to small differences between training and inference engines, often referred to as the training-inference mismatch (TIM). However, completely eliminating TIM is impractical, as it would come at a major cost to rollout efficiency. In this paper, we show that the instability of RL under TIM is primarily caused by *drift*: a persistent bias between training and inference engines that accumulates with every training step. We derive an additive *"score centering"* correction term that stabilizes RL under TIM by canceling drift. When training models from 0.6B to 30B parameters, score centering alone matches or outperforms methods based on importance sampling under quantization, with the gap growing as the mismatch becomes more severe. Because the correction is additive, score centering also composes with importance sampling – their composition outperforms pure importance sampling baselines in our staleness experiments.

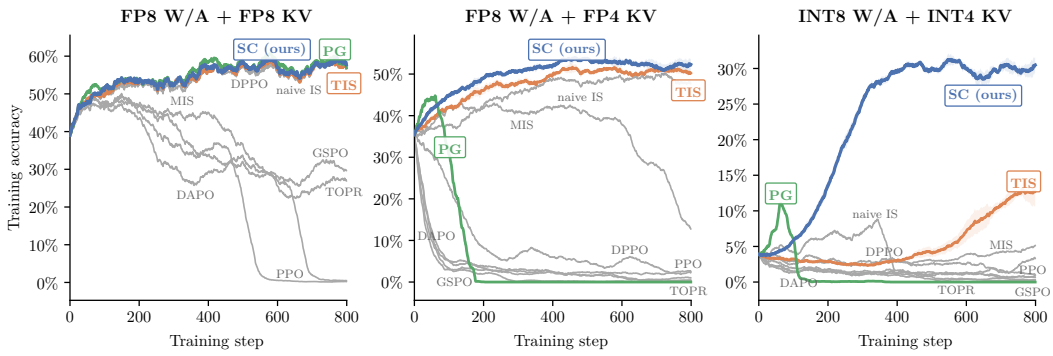

Figure 1: **Score centering outperforms importance sampling under heavy quantization.** We train Qwen3-30B-A3B-Base (Yang et al., 2025) on INTELLECT-2 math data (Prime Intellect Team et al., 2025) using RL under progressively stronger quantization of the sampler. We compare correction rules on a shared objective (Section 5.1). With mild quantization, naive policy gradient (PG) trains stably over 800 steps, and performs on par with correction methods. As quantization severity increases, training becomes less stable, and the gap between score centering (SC), truncated importance sampling (TIS), and other correction methods widens.

1 INTRODUCTION

Reinforcement learning is a crucial step in the training of today's large language models, as reflected by an increasingly large fraction of training compute allocated towards it (OpenAI, 2024; DeepSeek-AI, 2025; Khatri et al., 2026; Microsoft AI Team, 2026). This is typically achieved through algorithms such as PPO, GRPO, DAPO, and IcePop (Schulman et al., 2017; Shao et al., 2024; Yu et al., 2025; Ling Team, 2025), which are all based on the policy gradient method (Williams, 1992).

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.

RL training of LLMs via policy gradient consists of sampling rollouts, assigning a reward to each rollout, and computing gradients on the weighted rollouts. More formally, the policy gradient method computes the gradient of the expected reward over rollouts y from the policy p_θ as the expected score weighted by the reward R:

$$\nabla_\theta \mathbb{E}_{p_\theta}[R] = \mathbb{E}_{p_\theta}\big[R \underbrace{\nabla_\theta \log p_\theta(y)}_{\text{“score”}}\big]. \tag{1}$$

Crucially for practitioners, computing the right side of Equation (1) requires two separate forward passes through the model: one forward pass of the sampler (inference engine) to generate the rollouts and a second forward pass of the trainer to compute gradients on the generated rollouts. In theory, the trainer and sampler represent the same model, so their outputs should be identical. However, in practice, this is rarely the case. Most RL codebases admit small numerical differences between training and inference engines, resulting in the *training-inference mismatch* (TIM), which can degrade training performance (Khatri et al., 2026) or even lead to complete reward collapse (Liu et al., 2025b). By contrast, supervised fine-tuning generally has stable training dynamics using samples from a different model or a human annotator, with no adjustments required.

In this work, we investigate the reasons behind the sensitivity of RL to the training-inference mismatch. By analyzing the policy gradient update, we show that it has a *drift* term, equal to 0 if and only if training and sampling policies match. Based on this finding, we introduce a novel method to stabilize RL training under TIM, called **score centering**.As we show in experiments with Qwen3-0.6B on Countdown and Qwen3-30B-A3B-Base on the math subset of INTELLECT-2 data, score centering can match or outperform state-of-the-art techniques for training stabilization based on importance sampling. Due to the orthogonality of the two approaches, score centering can be combined with importance sampling for further gains.

2 Background and Related Work

2.1 Sources of Training-Inference Mismatch

The vanilla policy gradient identity in Equation (1) only holds when the policy for computing the gradients is the same policy that produced the rollouts. In practice, however, rollouts are generated from a **sampler** q_θ, while gradients are computed through a **trainer** p_θ. Hence, training-inference mismatch occurs when $q_\theta \neq p_\theta$.

There are many sources of TIM, each with different severity. In the worst case, the training and inference engines run at different precisions or are based on two independent codebases, each of which can have different (hard-to-detect) bugs that result in subtly different outputs (Liu et al., 2025a; FlashInfer contributors, 2026). In a less severe case, even if both engines are correct and use the same GPU kernels, they might still produce different outputs due to non-associativity of floating point operations.[1] Since sampling is autoregressive, while training is typically parallelized over the sequence dimension, the training and inference engines might call the same kernels with different input shapes, changing the order of reductions, resulting in small floating point differences. Batch-invariant kernels eliminate the dependency on input shapes, but require extensive engineering work and result in worse GPU utilization (He & Thinking Machines Lab, 2025). Switching both engines from bf16 to fp16 precision also greatly reduces the mismatch (Qi et al., 2025), but does not eliminate it entirely.

Finally, even if the trainer and the generator use identical batch-invariant kernels, TIM can occur due to update staleness. Maximizing GPU utilization during RL requires disaggregating training and inference engines, running them asynchronously (Fu et al., 2025), and serving rollouts with continuous batching (Yu et al., 2022). This means that a single training batch, or even different subsequences of a single rollout, can be generated from different model checkpoints (Piché et al., 2025). Staleness becomes most severe in long-context environments such as agentic coding, where some episodes may finish in minutes, while others might last hours or even days (Hou et al., 2026). From the perspective of hardware utilization, it is highly desirable to algorithmically stabilize training with stale rollouts, rather than trying to eliminate staleness entirely (Fu et al., 2025).

[1] For example, in `fp4_e2m1` precision, $(0.5 + 0.5) + 2 = 3$, but $0.5 + (0.5 + 2) = 2$.

Our goal is to propose a method for stable RL training under TIM, to enable high hardware utilization while ensuring stable training.

2.2 Importance-Sampling Corrections

Assume that in Equation (1), the rollouts come from the sampler q_θ, rather than the trainer p_θ, where $q_\theta \neq p_\theta$. Then we can correct for the training-inference mismatch exactly using importance sampling (IS):

$$\nabla_\theta \mathbb{E}_{p_\theta}[R] = \mathbb{E}_{p_\theta}\left[R \nabla_\theta \log p_\theta(y)\right] = \mathbb{E}_{q_\theta}\left[\frac{p_\theta(y)}{q_\theta(y)} R \nabla_\theta \log p_\theta(y)\right]. \tag{2}$$

Equation (2) is written for full sequences; in practice, the ratio is typically applied per token (Zheng et al., 2025). Either way, importance sampling corrects for TIM at the cost of increased variance. The weighted score gets multiplied by the importance ratio $r = \frac{p_\theta(y)}{q_\theta(y)}$, which can take arbitrarily large values on rare tokens, inflating the variance of the gradient estimate (Ionides, 2008; Owen, 2013). Since training with raw importance ratios is unstable, the methods used in practice bound the importance ratio, which inevitably introduces bias.

Table 1 (in Section B.2) shows that most popular correction methods are based on importance sampling and fall onto a simple grid, depending on the region where the importance ratio gets clipped or masked. A notable outlier is DPPO, which uses a masking region based on binary total variation – but its objective still uses an IS ratio, just like every other method in the grid. Going beyond the table, Ye et al. (2026) inject learnable perturbations into the trainer's hidden states and use the perturbed policy as the numerator of the importance ratio – reducing the heavy tail of the ratios instead of clipping them – but still relying on importance sampling. In contrast, *score centering* (a method that we introduce in Section 4) uses *no* importance ratios, no masking, and no clipping: it is an additive correction term that is fundamentally different from every method in this grid.

3 Why RL Is Sensitive to Mismatch

To understand why RL is so sensitive to TIM, it helps to understand what makes RL different from Supervised Fine-Tuning (SFT). As we noted earlier, SFT is stable under a far more severe mismatch – training on data generated completely offline by a different model, with no importance sampling correction (Hinton et al., 2015; DeepSeek-AI, 2025). In other words, SFT is not sensitive to staleness or the choice of kernels at all.

Crucially, policy gradient reduces to SFT on a task with binary rewards when the advantages are $+1/0$ (i.e. the rewards are not centered or normalized) and the training is completely offline (i.e. the sampler never gets updated).[2] Therefore, we can consider the **reward mode** and **staleness** to be the two key differences between SFT and RL.[3] We want to determine which of these two mechanisms is the one that results in increased sensitivity to TIM.

We test this using a toy setup of training with intentional policy mismatch. We train Qwen3-1.7B (Yang et al., 2025) on Countdown (Gandhi et al., 2024), and artificially perturb the weights of the sampler by a small amount compared to the trainer, to create controlled TIM. We provide further justification behind this setup in Section 5 as well as test more realistic sources of TIM.

[2] In practice, advantages almost always include negative values, for example, because of group centering (Shao et al., 2024). In addition, even if training admits some level of staleness, it is never completely offline.

[3] Note that shifting the rewards by a constant does not change the expected gradient on-policy, and group centering only rescales it by $1 - 1/G$ for a group of G rollouts. The reason to center rewards is that it reduces the gradient variance (Greensmith et al., 2004; Liu et al., 2025c). However, this doesn't hold off-policy: SFT with $+1/0$ converges to a maximum likelihood estimator (Bishop, 2006), while SFT with $0/-1$ rewards is degenerate: the loss becomes unbounded and can result in model collapse (Ren & Sutherland, 2025).

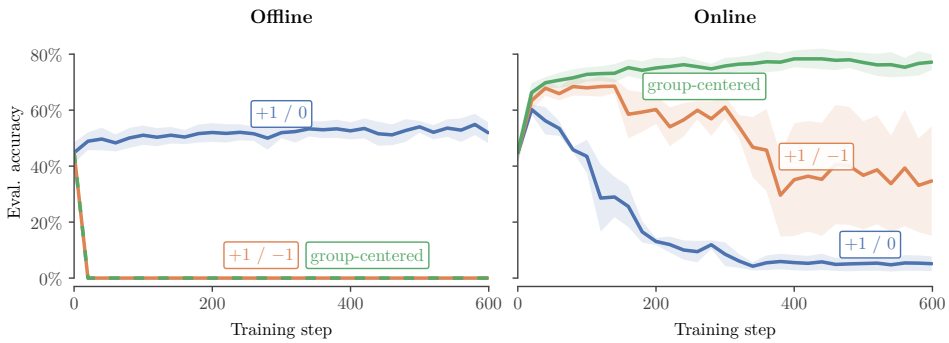

Figure 2: Offline vs. online training of Qwen3-1.7B on Countdown under three reward modes. Offline training is *only* stable with $+1/0$ rewards, while online training under TIM is *least* stable with $+1/0$ rewards.

Figure 2 shows that in this setup, offline training is only stable with $+1/0$ (non-negative) rewards, while the opposite holds for online training – group-centered rewards (Shao et al., 2024), which mix positive and negative values within a group, are the *most* stable, while $+1/0$ rewards – that were most stable for offline training – are *least* stable for online training. We therefore hypothesize that there are two separate mechanisms at play: one making offline training with *negative* rewards unstable, and another one making online training with *positive* rewards unstable.

The main focus of this paper is online reinforcement learning, therefore we do not explore the instability of offline training with negative rewards further. Ren & Sutherland (2025) argue that the main mechanism behind this instability is the unboundedness of negative rewards in combination with distribution sharpening. Logprobs are unbounded from below, therefore during offline training with negative rewards, they can diverge to $-\infty$. This effect is absent from online training where only high probability tokens get sampled. This asymmetry is well documented: training on positive samples alone (i.e. distillation or rejection finetuning) is completely stable even offline (Hinton et al., 2015; DeepSeek-AI, 2025), while negative gradients can drag down the probability of correct responses (Deng et al., 2025) – even though on-policy, they carry useful learning signal (Zhu et al., 2025).

In online training, we observe the opposite behavior to offline training – using only positive rewards is the *least* stable setting. We hypothesize therefore that the instability must arise from the online nature of the training process. Note that online training with $+1/0$ rewards under TIM is itself a form of distillation: the trainer is fit to successful rollouts from the sampler, a biased copy of itself, whose weights are refreshed from the trainer after every step. The only difference from offline distillation is that the teacher moves with the student, so this is where the instability must come from.

4 Score Centering

Notation. The update from a rollout y sums its token scores, each weighted by the rollout's reward (or advantage) R. We examine the expected contribution of one token y_t, conditional on its prefix $y_{<t}$. The sampler q produces the token y_t; v indexes the vocabulary. p_v and q_v are the next-token probabilities of v under the trainer p and the sampler q, $s_v = \nabla_\theta \log p_v$ is its score, and $\bar{s} = \sum_v q_v\, s_v$ is the expected score under the sampler. $\mathbb{E}_q$, $\mathbb{E}_p$ are over the rest of the rollout after the prefix.

Consider RL training with vanilla policy gradient in an environment with a constant $+1$ reward. In theory, there should be no learning – the reward is constant, so there is no "signal" coming from the environment. And indeed, on-policy the expected gradient is zero at every prefix: $\mathbb{E}_p[R\, s_{y_t}] = R\, \mathbb{E}_p[s_{y_t}] = R \cdot 0$, since $\mathbb{E}_p[s_{y_t}] = \sum_v p_v \nabla_\theta \log p_v = \nabla_\theta \sum_v p_v = \nabla_\theta 1 = 0$ for any distribution.

Under TIM, however, the expected gradient is nonzero: the token is sampled from the sampler q, while the score is computed through the trainer p, so $\bar{s} = \mathbb{E}_q[s_{y_t}] \neq 0$ in general. Using the covariance identity,[4] we can decompose the expected policy gradient update at a prefix into a "drift" and a "signal" term:

$$\underbrace{\mathbb{E}_q[R\, s_{y_t}]}_{\text{policy gradient update}} = \underbrace{\mathbb{E}_q[R]\ \bar{s}}_{\text{"drift"}} + \underbrace{\mathrm{Cov}_q(R,\ s_{y_t})}_{\text{"signal"}} \tag{3}$$

Drift is an artifact of TIM that acts as distillation toward the sampler. The drift term carries no information about which rollouts were successful: it depends on the rewards only through their mean $\mathbb{E}_q[R]$. Only the covariance term sees which token led to which reward. What the drift term does instead is determined by its direction $\bar{s}$: the expected score is the negative gradient of the cross-entropy (SFT) loss with the sampler as the teacher, so vanilla policy gradient distills the trainer toward the sampler at every prefix, scaled by the expected reward at that prefix. This is purely an artifact of TIM: on-policy, the trainer already matches the sampler, so $\bar{s} = 0$ and the drift term vanishes. Distillation toward a *fixed* teacher is harmless – it converges to the maximum likelihood fit of that teacher. The sampler, however, is not fixed: it is a biased copy of the trainer (e.g. quantized or stale), so each step pushes the trainer toward the sampler; the trainer's weights are then synced back to the sampler, and the error compounds in a feedback loop instead of converging. This explains why online training with all-positive rewards is the least stable setting in Figure 2. Dong et al. (2026) likewise identify systematic bias as the root cause of instability under TIM, and show it compounds through a positive feedback loop.

Group centering does not remove drift. Group centering (Shao et al., 2024) makes the advantages sum to zero over the rollouts of a prompt, but drift arises at individual prefixes, and the expected advantage at a given prefix is not zero: a prefix that is likely to lead to a correct answer has positive expected advantage, and a prefix that already contains a mistake has negative expected advantage. Drift is therefore nonzero precisely at the prefixes that carry learning signal – the trainer is pulled toward the sampler after prefixes with positive expected advantage and pushed away from it after prefixes with negative expected advantage, regardless of whether the tokens in question affect the reward. Group centering does shrink drift relative to $+1/0$ rewards, which is consistent with it being the most stable online setting in Figure 2, but it does not eliminate it.

Score centering removes drift. These observations directly motivate our method, called *score centering*: even under TIM, we want the expected score to be zero at every prefix, as it is on-policy. We achieve this simply by subtracting from each score the expected score under the sampler, replacing s_{y_t} by the centered score $\tilde{s}_{y_t}$:

$$\tilde{s}_{y_t} = s_{y_t} - \bar{s}. \tag{4}$$

The expected policy gradient update with score centering then becomes:

$$\begin{aligned} \mathbb{E}_q[R\,\tilde{s}_{y_t}] &= \cancel{\mathbb{E}_q[R]\ \mathbb{E}_q[\tilde{s}_{y_t}]}^{\mathbb{E}_q[\tilde{s}_{y_t}] = \bar{s} - \bar{s} = 0} + \mathrm{Cov}_q(R, \tilde{s}_{y_t}) \\ &= \mathrm{Cov}_q(R, \tilde{s}_{y_t}) \\ &= \mathrm{Cov}_q(R, s_{y_t}) \quad \longleftarrow \quad \begin{array}{l}\text{only difference to on-policy update} \\ \text{is the covariance sampling distribution}\end{array} \end{aligned} \tag{5}$$

The centered score as defined in Equation (4) has zero mean under the sampler because both expectations are taken under the same distribution – the drift gets canceled exactly at every prefix, even off-policy. Equation (5) shows that the expected update with score centering equals the expected on-policy update up to the distribution over which the covariance is measured – both on-policy and under TIM with importance sampling, the covariance is measured under the training distribution p_θ, while score centering measures it under the sampling distribution q_θ. In the absence of TIM, i.e. when $p_\theta = q_\theta$, score centering is a no-op, since the correction term in Equation (4) is exactly zero.

[4] $\mathrm{Cov}(A, B) = \mathbb{E}[AB] - \mathbb{E}[A]\,\mathbb{E}[B] \ \Rightarrow\ \mathbb{E}[AB] = \mathbb{E}[A]\,\mathbb{E}[B] + \mathrm{Cov}(A, B)$

Score Centering vs Importance Sampling. Importance sampling, just like score centering, also cancels drift exactly. However, importance sampling uses a random multiplicative correction term, meaning that it increases the gradient variance, and its value depends on the sampled token. For rare tokens, the importance ratio can be very large, which is why almost all practical implementations clip or mask large importance ratios in some way (Table 1), reintroducing drift. In contrast, score centering is an *additive* correction term that is *deterministic* given the prefix – it does not depend on the sampled token, and it can be evaluated exactly. Since score centering and importance sampling correct for TIM in independent ways, the two methods can be composed with each other, as described in Section A.2.

Relation to classical baselines. Subtracting a zero-mean quantity from the policy gradient is a classical variance-reduction idea: reward baselines (Williams, 1992) and score-function control variates (Ranganath et al., 2014) both rely on the identity $\mathbb{E}_p[s_{y_t}] = 0$. Score centering is a bias correction rather than a variance reduction: the subtracted vector is deterministic given the prefix and is chosen to shift the mean of the update, whereas on-policy a reward baseline changes the variance but not the mean. Off-policy, a reward baseline equal to the expected reward at the prefix (an exact value function) would also cancel drift in expectation; score centering obtains the same expected update without a critic. To our knowledge, no prior method subtracts the expected score itself: on-policy it is zero, and in classical off-policy RL it requires an intractable expectation over the action space, so off-policy methods rely on importance sampling instead. RL training of LLMs under TIM is unusual in that the expected score is both nonzero and computable exactly, as a sum over the next token's logprobs.

4.1 Implementation

Storing the sampler's full next-token distribution is prohibitively expensive. We therefore log only its top-k logprobs ($k = 128$) and model the tail with the trainer's distribution, rescaled to match the sampler's tail mass. We implement score centering as a scalar loss:

$$L = -R \left(\log p_{y_t} - \sum_{v \in H} \mathrm{sg}\left[q_v - \rho\, p_v\right] \log p_v \right), \tag{6}$$

where ρ is the ratio of the sampler's to the trainer's tail mass and sg denotes stop-gradient. Using $k = 128$ or even $k = 32$ matches full score centering in every setting we tested (Section A.4). The code snippet below illustrates a minimal implementation for a single token:

```
import jax.numpy as jnp
from jax.lax import stop_gradient

def score_centering_loss(train_logp, samp_logp, topk_ids, sampled_token, advantage):
    train_head_logp = train_logp[topk_ids]
    tail_mass_ratio = (1 - jnp.exp(samp_logp).sum()) / (1 - jnp.exp(train_head_logp).sum())
    head_prob_residual = jnp.exp(samp_logp) - tail_mass_ratio * jnp.exp(train_head_logp)
    logp_correction = (stop_gradient(head_prob_residual) * train_head_logp).sum()
    return -advantage * (train_logp[sampled_token] - logp_correction)
```

In Section A, we provide the full derivation and a generalized implementation of score centering with support for importance weights.

5 Experiments

5.1 Setup

We compare score centering against the correction methods in Table 1 by training Qwen3-0.6B-Instruct on Countdown (Gandhi et al., 2024) and Qwen3-30B-A3B-Base on the math subset of the INTELLECT-2 dataset (Prime Intellect Team et al., 2025).

Shared objective. Existing methods often bundle several techniques together: for example, DAPO (Yu et al., 2025) combines asymmetric clipping with dynamic sampling , while IcePop (Ling

Team, 2025) combines MIS between training and inference engines with PPO clipping between policy versions. To compare each correction method in isolation, we use the same objective (REINFORCE with group-centered rewards) throughout, with each correction being applied on top of this objective in isolation. All methods share the same sampler, trainer, optimizer and advantages, with one SGD step per batch. All importance ratios use the sampler's logged probabilities. Table 1 shows the correction rules and hyperparameters, together with implementation notes. The learning rate, batch sizes, and compute costs are listed in Section B.

Mismatch settings. In general, we struggled to find natural setups with TIM severe enough to observe statistically significant differences between the strongest-performing methods within a handful of GPU hours. We believe that this is a consequence of the accumulation of drift during training (Section 4): under mild TIM, it takes more training steps for sufficient drift to accumulate, and until it does, the best-performing methods overlap. We show this in Section 5.2: under small weight noise, the best-performing methods overlap, making it impossible to draw comparisons, while as TIM increases, methods collapse earlier and the performance gap between them grows. Since we cannot afford to ablate correction methods with statistical significance over very long training runs, we deliberately amplify TIM instead, and use these results as a proxy for large-scale training under milder TIM. For this reason, in Section 5.3 we test intentionally severe quantization and staleness settings. Notably, prior works disagree on just how unstable RL under TIM really is: Qiu et al. (2026) find fp8 rollouts stable using only token-level importance sampling without matched numerics, while Microsoft AI Team (2026) see runs diverge despite trying to minimize TIM and using bf16 sampling. We believe both findings are consistent with drift accumulating over training: to observe instability within short training runs, TIM must be severe, while mild TIM can destabilize training too, but it takes more training steps.

5.2 Synthetic Weight Noise

For fast initial experimentation, we found it useful to study TIM induced by adding an artificial offset to the weights of the sampler compared to the weights of the trainer. Namely, we set $\theta_{\text{sampler}} = \theta_{\text{trainer}} + \Delta\theta$, where $\Delta\theta$ was sampled at the beginning of training from an isotropic Gaussian distribution and held fixed during training. While this is the least realistic setup we have studied, it resulted in the fastest collapse and separation between methods, and allowed for a continuous scale of TIM, making it an indispensable tool for initial experiments.

Across three noise scales in Figure 3, we find Score Centering (SC), Truncated Importance Sampling (TIS) and Masked Importance Sampling (MIS) to perform the best, along with their compositions: MIS + SC and TIS + SC. We describe in Section A.2 how score centering can be composed on top of importance sampling methods. Under the most severe noise, only score centering and score centering composed with MIS or TIS trained stably. Figure 3 also shows drift accumulating over training – methods that collapse do so earlier under larger mismatch. For example, DPPO collapses at roughly step 160, 80 and 20 across the three noise scales, and TIS trains stably under the smallest noise but collapses at roughly step 180 and 40 under the larger two.

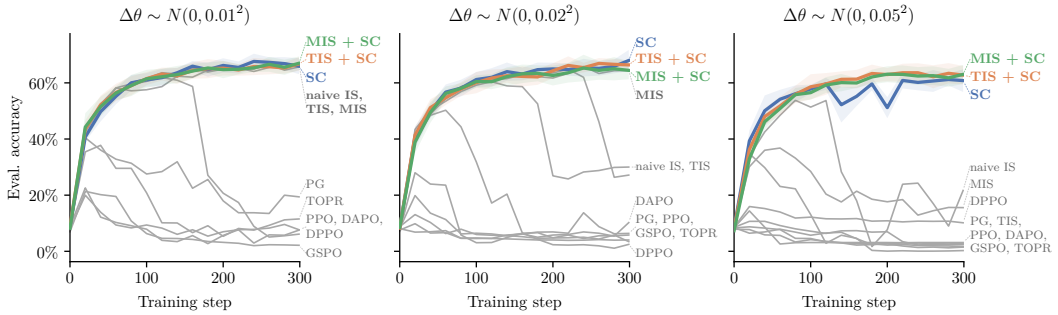

Figure 3: Qwen3-0.6B on Countdown with Gaussian noise added to the sampler weights. More noise causes earlier collapse. Only score centering (alone or composed with TIS / MIS) trains stably under the largest noise.

5.3 Quantization and Staleness

Next, we replace artificial weight noise with two more realistic sources of TIM: quantization and staleness. We deliberately made both sources of TIM severe (Section 5.1) to observe the separation between different correction methods within hundreds of training steps using short sequence lengths. In the quantized setting, we quantize only the sampler (weights, activations, and KV cache); the trainer uses bf16 precision. The staleness setting is intentionally severe too: the inference engine gets updated *only* every 64 steps, rather than having a *maximum* staleness of 64. In practice, it would be preferable to update the inference engine as frequently as possible (Piché et al., 2025) and to apply the same quantization scheme to both the sampler and the trainer (Xi et al., 2026).

In the quantized sampler setting (Figure 4), we see similar results to Figure 3 – score centering, alone and composed on top of TIS / MIS, performs best. However, under the staleness setting, score centering composed with TIS / MIS performs best, outperforming vanilla score centering. We attribute this to score centering measuring the covariance under the sampler rather than the trainer (Equation 5): TIS partially corrects the sampling distribution, and score centering removes the remaining drift. We therefore recommend composing score centering with an importance-sampling correction whenever the trainer can move far from the sampler between syncs. PPO and DAPO survive staleness but collapse under quantization and weight noise. This is consistent with their clip being designed for ratios that arise from policy movement: staleness produces such ratios, while numerical error does not.

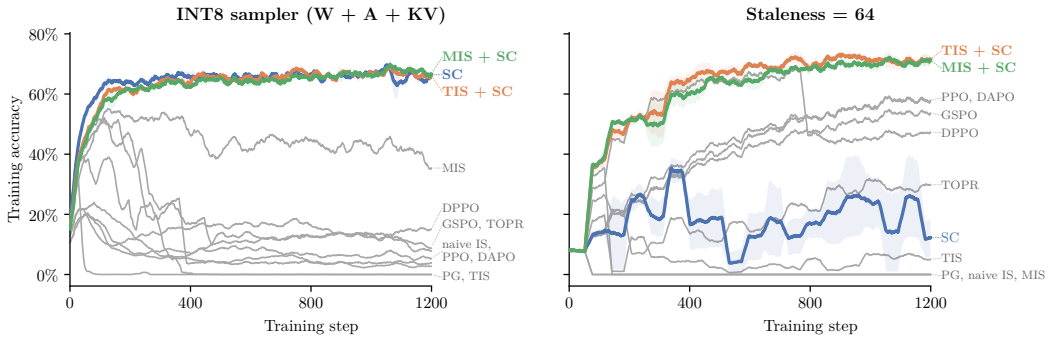

Figure 4: Qwen3-0.6B on Countdown with an int8 sampler (left) and a sampler updated only every 64 steps (right). Under quantization, both score centering alone and composed perform best; under staleness, score centering composed with TIS / MIS dominates.

5.4 Scaling to 30B

Finally, we verify that our results hold at a larger scale by training Qwen3-30B-A3B-Base on INTELLECT-2 math under three levels of sampler quantization (Figure 1). Since Qwen3-30B-A3B is a mixture-of-experts model, the sampler and the trainer can also disagree on expert routing; we do not replay router indices, although doing so would be preferable in practice (Ma et al., 2025). With an FP8 sampler, even uncorrected policy gradient trains stably, reaching 58% training accuracy. Once the KV cache is quantized to FP4, policy gradient collapses within 200 steps and MIS collapses late in training, while score centering (52%) and TIS (51%) stay stable. With an INT8 sampler and an INT4 KV cache, score centering reaches 30%, TIS 12%, and every other method ends below 5%.

6 Conclusion

This paper makes two contributions. The first is an explanation of why RL training of LLMs is so sensitive to training-inference mismatch. Under TIM, the policy gradient update contains a drift term that acts as distillation toward the sampler. Because the sampler is a biased copy of the trainer and is periodically synced back to it, this bias compounds in a feedback loop: the more severe

the TIM, the earlier training collapses. Prior work has observed that this bias compounds over training (Dong et al., 2026; Microsoft AI Team, 2026). We show that canceling drift alone, with no importance ratios, is sufficient to stabilize training under severe quantization, which establishes drift as the cause of the instability. This also explains why offline distillation from a completely different model is stable, while online RL collapses under small numerical differences: what matters is not the size of the mismatch, but whether the teacher is fixed or tracks the student.

Our second contribution is score centering, a practical method that cancels drift by subtracting the expected score. It is an additive correction term with no hyperparameters, it can be expressed as a scalar loss (Equation 6), and it composes with importance-sampling methods such as TIS and MIS. Exact score centering requires the sampler's full next-token distribution; we therefore introduce an approximation that needs only the sampler's top-k logprobs and matches the performance of exact score centering in all of our experiments (Section A.4). Under mild TIM, score centering matches importance-sampling methods; under severe quantization, it is the only method that trains stably, and under severe staleness its composition with TIS or MIS performs best.

Limitations. Score centering cancels drift, but the remaining update measures the covariance between reward and score under the sampler rather than the trainer (Equation 5). This mismatch matters under severe staleness, where score centering performs best when composed with importance sampling (Section 5.3). Additionally, our headline results come from deliberately severe mismatch on short sequences, used as a proxy for long training under milder mismatch.

REFERENCES

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Wenlong Deng, Yi Ren, Muchen Li, Danica J. Sutherland, Xiaoxiao Li, and Christos Thrampoulidis. On the effect of negative gradient in group relative deep reinforcement optimization. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.18830.

Yiming Dong, Kun Fu, Haoyu Li, Xinyuan Zhu, Yurou Liu, Lijing Shao, Jieping Ye, and Zheng Wang. Probing RLVR training instability through the lens of objective-level hacking. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=KlGj06E8Wa.

FlashInfer contributors. fix(attention): handle extreme negative logits in masked softmax, August 2026. URL https://github.com/flashinfer-ai/flashinfer/pull/4401.

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. *arXiv preprint arXiv:2505.24298*, 2025. URL https://arxiv.org/abs/2505.24298.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024. URL https://arxiv.org/abs/2404.03683.

Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530, 2004.

Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026. URL https://arxiv.org/abs/2607.07508.

Edward L. Ionides. Truncated importance sampling. *Journal of Computational and Graphical Statistics*, 17(2):295–311, 2008. doi: 10.1198/106186008X320456.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FMjeC9Msws.

Nicolas Le Roux, Marc G. Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy REINFORCE: Stable and efficient reinforcement learning for LLMs. *arXiv preprint arXiv:2503.14286*, 2025. URL https://arxiv.org/abs/2503.14286.

Ling Team. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *arXiv preprint arXiv:2510.18855*, 2025. URL https://arxiv.org/abs/2510.18855.

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

Liyuan Liu, Feng Yao, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. FlashRL: 8bit rollouts, full power RL, 2025b. URL https://fengyao.notion.site/flash-rl.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025c. URL https://arxiv.org/abs/2503.20783.

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.

Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

MiniMax. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025. URL https://arxiv.org/abs/2506.13585.

Sagnik Mukherjee, Lifan Yuan, Pavan Jayasinha, Dilek Hakkani-Tür, and Hao Peng. Do we need Adam? Surprisingly strong and sparse reinforcement learning with SGD in LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=z31fdV4WRu.

OpenAI. OpenAI o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Art B. Owen. *Monte Carlo Theory, Methods and Examples*. 2013. URL https://artowen.su.domains/mc/.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. PipelineRL: Faster on-policy reinforcement learning for long sequence generation. *arXiv preprint arXiv:2509.19128*, 2025.

Prime Intellect Team, Sami Jaghouar, Justus Mattern, Jack Min Ong, Jannik Straube, Manveer Basra, Aaron Pazdera, Kushal Thaman, Matthew Di Ferrante, Felix Gabriel, Fares Obeid, Kemal Erdem, Michael Keiblinger, and Johannes Hagemann. INTELLECT-2: A reasoning model trained through globally decentralized reinforcement learning. *arXiv preprint arXiv:2505.07291*, 2025. URL https://arxiv.org/abs/2505.07291.

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. URL https://arxiv.org/abs/2510.26788.

Penghui Qi, Xiangxin Zhou, Zichen Liu, Tianyu Pang, Chao Du, Min Lin, and Wee Sun Lee. Rethinking the trust region in LLM reinforcement learning. *arXiv preprint arXiv:2602.04879*, 2026. URL https://arxiv.org/abs/2602.04879.

Zhaopeng Qiu, Shuang Yu, Jingqi Zhang, Shuai Zhang, Xue Huang, Jingyi Yang, and Junjie Lai. FP8-RL: A practical and stable low-precision stack for LLM reinforcement learning. *arXiv preprint arXiv:2601.18150*, 2026. URL https://arxiv.org/abs/2601.18150.

Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *Artificial Intelligence and Statistics (AISTATS)*, 2014.

Yi Ren and Danica J. Sutherland. Learning dynamics of LLM finetuning. In *International Conference on Learning Representations*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

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. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024. URL https://arxiv.org/abs/2409.19256.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696.

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.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. On the rollout-training mismatch in modern RL systems. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=8MHqvb4lK9. Blog version: https://fengyao.notion.site/off-policy-rl.

Chenlu Ye, Xuanchang Zhang, Yifan Hao, Zhou Yu, Ziji Zhang, Abhinav Gullapalli, Hao Chen, Jing Huang, and Tong Zhang. Adaptive layerwise perturbation: Unifying off-policy corrections for LLM RL. *arXiv preprint arXiv:2603.19470*, 2026. URL https://arxiv.org/abs/2603.19470.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

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. URL https://arxiv.org/abs/2507.18071.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. *arXiv preprint arXiv:2506.01347*, 2025. URL https://arxiv.org/abs/2506.01347.

A Implementing Score Centering

Notation (as in Section 4). At a fixed prefix, y_t is the sampled token and v a vocabulary token; p_v, q_v are its trainer and sampler probabilities, $s_v = \nabla_\theta \log p_v$ its score, and $\bar{s} = \sum_v q_v\, s_v$ the expected score under the sampler.

A.1 Top-k Approximation

Full score centering, as we have described it thus far, is prohibitively expensive for most practical applications. Computing the score centering correction term (an expectation of scores over the sampler's per-token distribution) as described in Equation (4) requires storing the full output logprobs for every sampled token. For instance, Qwen3 models have a vocabulary size of 152K, so storing the full logprobs for just a single token in fp32 precision requires around 608KB of memory. Assuming a batch of 1024 rollouts, each with a sequence length of 32K, storing the full logprobs would require a prohibitive 20TB of memory. For this reason, we only store a top-k approximation of the sampler's distribution throughout our experiments, where $k = 128$. However, rather than merely computing the expected score over the top-k distribution, we still try to approximate the sampler's full distribution, using logprobs from the trainer to fill in the tail. We used $k = 128$ as a conservative default value; Figure 5 shows that $k = 128$ and even $k = 32$ perform on par with full score centering across all of our experimental settings.

The question is then how to reconstruct the sampler's full distribution from only its top-k logprobabilities. Let $\hat{q}$ denote our approximation of the sampling distribution. We take top-k logprobabilities from the sampler and model the tail using logprobs from the trainer, rescaled so that the entire distribution sums to one. Denoting the set of top-k tokens as head H and the remaining tokens as tail T:

$$\hat{q}_v = \begin{cases} q_v & v \in H \\ \rho\, p_v & v \in T \end{cases} \qquad \text{where} \quad \rho = \frac{1 - \sum_{v \in H} q_v}{1 - \sum_{v \in H} p_v}. \tag{7}$$

The scalar ρ is the ratio of the sampler's tail mass to the trainer's tail mass – it rescales the trainer's tail so that $\hat{q}$ sums to one.

To compute the expected score under this reconstructed $\hat{q}$ distribution, there is no need to sum over the whole vocabulary. Instead, we compute the expectation over the head taken from q exactly, and compute the expectation over the tail taken from p as zero minus the expectation over the head, taking advantage of the fact that the expected score over the full vocabulary sums to zero:

$$\mathbb{E}_{\hat{q}}[s_{y_t}] = \sum_{v \in H} q_v\, s_v + \rho \sum_{v \in T} p_v\, s_v = \sum_{v \in H} q_v\, s_v + \rho \left(\underbrace{\mathbb{E}_p[s_{y_t}]}_{=0} - \sum_{v \in H} p_v\, s_v \right) = \sum_{v \in H} (q_v - \rho\, p_v)\, s_v. \tag{8}$$

As a result, the centering term only involves the k head tokens, even though the modeled tail covers the whole vocabulary.

The cost of top-k score centering is negligible: on matched hardware, runs with $k = 128$ finish within 1% of the wall-clock time of the baseline methods for both the 0.6B and 30B models. Our custom JAX sampler computes the top-k logprobs as part of decoding; vLLM and SGLang expose top-k logprobs natively, but we have not measured their overhead.

A.2 Composing Score Centering with Importance Sampling

Score centering subtracts a correction term equal to the expected score. Importance sampling methods like TIS or MIS work by reweighting the scores using clipped / masked importance ratios. Therefore, when composing score centering on top of importance sampling, we compute the expectation of the *weighted* scores rather than raw scores. We can think of this as IS first reweighting the scores, and then score centering being applied on top of the reweighted scores.

Let $r_v = p_v/q_v$ denote the importance ratio and $w_v = f(r_v)$ the weight assigned by the IS method being composed, for example $f(r) = \min(r, 2)$ for TIS. Score centering subtracts the expectation of the *weighted* score:

$$R\left(w_{y_t}\, s_{y_t} - \mathbb{E}_q[w_{y_t}\, s_{y_t}]\right), \qquad \mathbb{E}_q[w_{y_t}\, s_{y_t}] = \sum_v q_v\, w_v\, s_v. \tag{9}$$

Just like vanilla score centering, the expected update under a constant reward is exactly zero – subtracting the expected weighted score $\mathbb{E}_q[w_{y_t}\, s_{y_t}]$ cancels drift by construction, regardless of the weighting function $f(r)$.

The top-k reduction in Equation (8) also applies here. On the modeled tail the sampler probability is only known through $\hat{q}$, so the weight is computed under $\hat{q}$ as well: $\hat{w}_v = f(p_v/\hat{q}_v)$, which equals w_v on the head. Since $p_v/\hat{q}_v = p_v/(\rho\, p_v) = 1/\rho$ is constant on the tail, the centering term keeps the same head-only form, with a scalar α in place of ρ:

$$\mathbb{E}_{\hat{q}}[\hat{w}_{y_t}\, s_{y_t}] = \sum_{v \in H} (q_v\, w_v - \alpha\, p_v)\, s_v, \qquad \text{where} \quad \alpha = \rho\, f(1/\rho). \tag{10}$$

Vanilla score centering is just a special case of $f = 1$, giving $\alpha = \rho$. As a sanity check, vanilla importance sampling $f(r) = r$ gives $\alpha = 1$ and $q_v\, w_v = p_v$, so the centering term vanishes – exact importance sampling already has zero expected weighted score, so there is nothing left to center.

A.3 Loss Formulation

Score centering can be efficiently implemented in autograd frameworks by expressing it inside a scalar loss function. Since the expected score is a weighted sum of per-token scores, $\bar{s} = \sum_v q_v\, \nabla_\theta \log p_v$, we can express score centering as a loss by weighting the trainer's logprobs with detached sampler probabilities. For policy gradient this becomes:

$$L = -R\left(\log p_{y_t} - \underbrace{\sum_v \mathrm{sg}\left[q_v\right] \log p_v}_{\text{centering term}}\right), \tag{11}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient, i.e. we do not differentiate through the sampling probabilities. Differentiating recovers the centered update: $-\nabla_\theta L = R\,(s_{y_t} - \bar{s}) = R\,\tilde{s}_{y_t}$. The generalized top-$k$ version, composed on top of arbitrary importance weights, follows the same pattern, requiring only computation over the head tokens:

$$L = -R\left(\mathrm{sg}\left[w_{y_t}\right] \log p_{y_t} - \sum_{v \in H} \mathrm{sg}\left[q_v\, w_v - \alpha\, p_v\right] \log p_v\right). \tag{12}$$

The JAX code below implements Equation (12), with `weight_fn` specifying $f(r)$ (default: vanilla SC). As in Section 4.1, `train_logp` spans the vocabulary and `samp_logp` contains the sampler's top-k logprobs. The additional scalar `samp_token_logp` is the sampler's logprob for the sampled token, even if it falls outside the head. Tail masses are floored at `eps` for numerical stability.

```python
import jax.numpy as jnp
from jax.lax import stop_gradient

def score_centering_loss(train_logp, samp_logp, topk_ids, sampled_token,
                         samp_token_logp, advantage, weight_fn=lambda r: 1.0, eps=1e-6):
    train_head_logp = train_logp[topk_ids]
    head_weights = weight_fn(jnp.exp(train_head_logp - samp_logp))
    train_tail_mass = jnp.maximum(1 - jnp.exp(train_head_logp).sum(), eps)
    samp_tail_mass = jnp.maximum(1 - jnp.exp(samp_logp).sum(), eps)
    tail_mass_ratio = samp_tail_mass / train_tail_mass
    tail_scale = tail_mass_ratio * weight_fn(1 / tail_mass_ratio)
    head_prob_residual = jnp.exp(samp_logp) * head_weights
    head_prob_residual -= tail_scale * jnp.exp(train_head_logp)
    logp_correction = (stop_gradient(head_prob_residual) * train_head_logp).sum()
    sampled_ratio = jnp.exp(train_logp[sampled_token] - samp_token_logp)
    weighted_logp = stop_gradient(weight_fn(sampled_ratio)) * train_logp[sampled_token]
    return -advantage * (weighted_logp - logp_correction)

tis_weight = lambda r: jnp.minimum(r, 2.0)
mis_weight = lambda r: jnp.where((r >= 0.5) & (r <= 5.0), r, 0.0)
```

Pass `weight_fn=tis_weight` or `weight_fn=mis_weight` to compose score centering with TIS or MIS, respectively. As in Equation (12), both the sampled importance weight and the centering coefficients are detached.

A.4 Top-k Ablation

In Figure 5, we compare top-k score centering against full score centering across all of our experimental settings. Both $k = 32$ and $k = 128$ perform on par with full score centering in every setting. The tail model of Equation (7) only has to account for the sampler probability mass outside the top-k head, which we log at every step. With $k = 128$, the head covers more than 99.9% of the sampler's mass on average in every setting except INT8 W/A + INT4 KV at 30B, where it covers 99.45% on average and 95.8% in the worst batch – the setting where the tail is most distorted by quantization and where Figure 5 shows that $k = 32$ and $k = 128$ still match full score centering.

B Experimental Details

B.1 Training Setup

Unless otherwise noted, we use REINFORCE with group-centered rewards:

$$J_{\text{base}}(\theta) = \frac{\sum_i A_i \sum_{t=1}^{L_i} \log p_\theta(y_{i,t} \mid y_{i,<t})}{\sum_i L_i}, \qquad A_i = R_i - \bar{R}_{\text{group}}, \qquad R_i \in \{-1, +1\}. \quad (13)$$

Here i indexes rollouts in the batch, L_i is the rollout length, and $\bar{R}_{\text{group}}$ is the mean reward of completions for the same prompt.

Following Mukherjee et al. (2026), we use SGD across all of our experiments, with a fixed learning rate of 10^{-2}. We found this learning rate stable across all of our experiments, and performing on par with AdamW, while saving up to 240GB of memory. Each batch samples 8 completions per prompt and takes a single optimizer step. We use 64 prompts per batch (512 sequences) with a maximum sequence length of 512 for Qwen3-0.6B-Instruct on Countdown, and 16 prompts per batch (128 sequences) with a maximum sequence length of 1024 for Qwen3-30B-A3B-Base on INTELLECT-2 math. Longer sequences would increase TIM (Xi et al., 2026) but make multi-seed comparison of many baselines too expensive.

B.2 Correction Methods

All of the IS-based methods we compare fall onto a simple grid (Table 1). They differ in whether the correction is applied per token or per sequence, where the importance ratio gets clipped or masked,

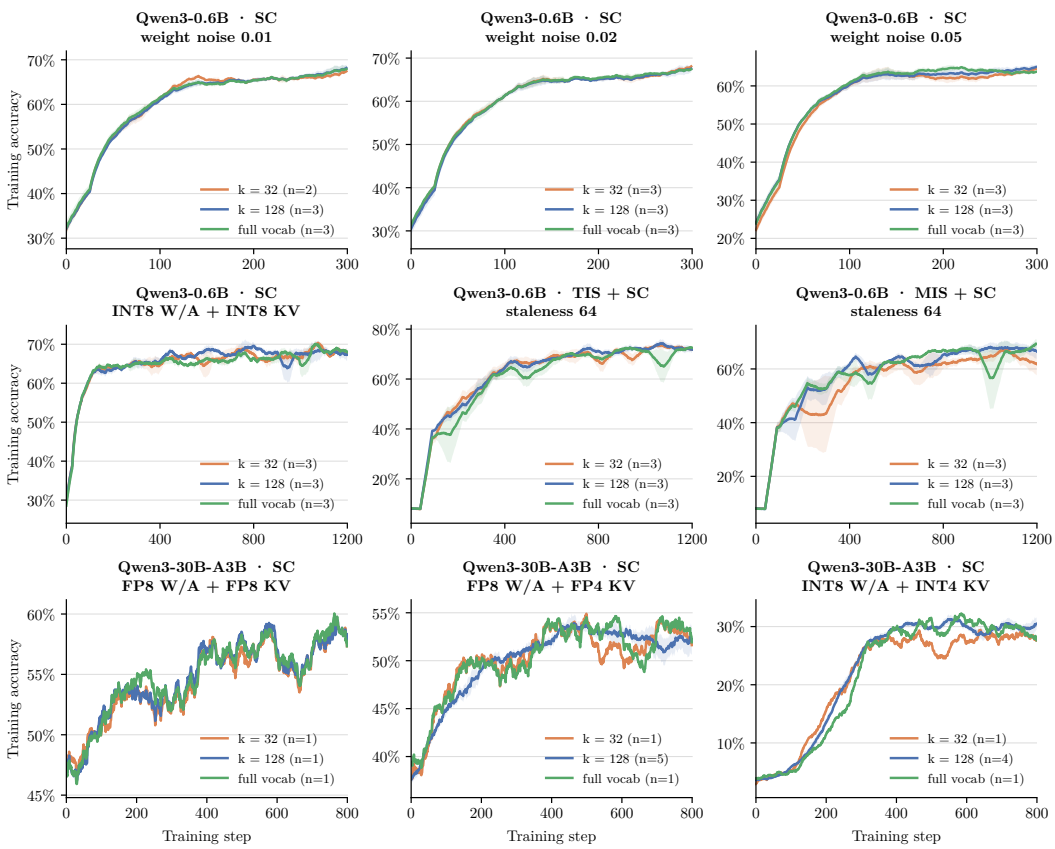

Figure 5: Top-k ablation of score centering. Using $k = 128$ or $k = 32$ sampler logprobs with the tail model of Equation (7) matches full-vocabulary score centering across all tested settings. The number of seeds for each curve is printed in the legend.

and which advantage signs are corrected. DPPO uses a masking region based on binary total variation instead of the importance ratio, but its objective still uses IS weights. We use default parameters from the original papers or `verl` without further tuning. All importance ratios are computed against the sampler's logged probabilities. GSPO uses the geometric mean of the token ratios, as in the original paper.

We apply each correction on top of the same objective, rather than reproducing the full recipe from each paper (Section 5.1). The last column of Table 1 provides implementation details for each method. PG uses Equation (13) without any correction. For score centering, we use the top-k approximation with $k = 128$ (Section A).

B.3 Compute Resources

Across all figures, shading represents ± 1 standard error. Figures 2 and 3 report accuracy on held-out evaluation prompts, while Figures 1, 4 and 5 report training accuracy, i.e. the mean reward of the sampled rollouts, smoothed with a centered moving average over training steps. Table 2 lists the number of seeds per method and the approximate compute required to reproduce each figure. For the 0.6B and 1.7B models, we ran 3 seeds across every experiment for every correction method (Figures 2 to 4). However, since training the 30B model (Figure 1) was much more expensive, we initially ran every method only with a single seed, and then added more seeds only for those methods that performed best (since these were the comparisons we cared about the most). In the FP8 KV panel every method is single-seed, while in the FP4 KV and INT4 KV panels score centering uses 5 and 4 seeds, TIS 4 and 3, MIS 3 and 3, and vanilla IS 2 and 1, respectively. All remaining methods

Table 1: IS-based correction methods represented as a grid, with implementation notes in the last column. `Level` determines whether the statistic is computed per token or per sequence. `Outside` specifies what happens beyond the `Low...High` region. `Sign` specifies which advantage signs are corrected: `both` applies the weight to every token; `neg` only to tokens with negative advantage (positive-advantage tokens keep weight 1); and `ppo` applies the PPO pessimistic clip, i.e. the minimum of the weighted and clipped objectives, so the clip only binds in the direction that would increase the objective. Dual clipping follows `verl`'s PPO implementation: negative-advantage contributions with ratio above 3 are dropped (whole responses for GSPO). ^{P} denotes the paper default and ^{V} the `verl` default (Sheng et al., 2024).

Method	Level	Stat	Low	High	Outside	Sign	Implementation notes
Naive IS (Williams, 1992)	`token`	`ratio`	0	∞	`-`	`both`	—
TIS^{V} (Yao et al., 2025) CISPO (MiniMax, 2025)	`token`	`ratio`	0	2.0	`clip`	`both`	CISPO's dynamic sampling and length penalty omitted.
MIS (IcePop)^{P} (Ling Team, 2025)	`token`	`ratio`	0.5	5.0	`drop`	`both`	Separate PPO surrogate between policy versions omitted.
PPO$^{\mathrm{P,V}}$ (Schulman et al., 2017)	`token`	`ratio`	0.8	1.2	`drop`	`ppo`	Dual clipping at 3 (`verl` default).
DAPO$^{\mathrm{P,V}}$ (Yu et al., 2025)	`token`	`ratio`	0.8	1.28	`drop`	`ppo`	Dual clipping at 3; dynamic sampling and overlong reward penalty omitted.
GSPO$^{\mathrm{P,V}}$ (Zheng et al., 2025)	`seq`	`ratio`	0.9997	1.0004	`drop`	`ppo`	Token gradients summed rather than response-averaged; added dual clipping at 3.
TOPR^{P} (Le Roux et al., 2025)	`seq`	`ratio`	0	1.0	`clip`	`neg`	—
DPPO^{P} (Qi et al., 2026)	`token`	`tv`	0	0.2	`drop`	`ppo`	No importance-ratio clipping.

are single-seed. Similarly, in Figure 5, the $k = 128$ results for the 30B model are transferred from Figure 1, which is why they are multi-seed; $k = 32$ and full score centering are single-seed to reduce computational cost.

Table 2: Seeds per curve and approximate compute required to reproduce our figures. Experiments primarily ran on nodes with 8× NVIDIA H100 SXM GPUs.

Experiment	Figure	Seeds	Unique runs	Approx. H100-hours
Online vs. offline	Figure 2	3	18	190
Synthetic weight noise	Figure 3	3	108	280
Quantization and staleness	Figure 4	3	72	1,120
30B math	Figure 1	1–5	47	3,750
Top-k ablation	Figure 5	see legend	51	840
Total			**296**	**6,180**

C Disclosure of LLM Use

We used large language models (LLMs) to improve writing clarity, find related work, implement experiments, and assist in formulating the top-k approximation. The authors take full responsibility for the final content of this paper.