QUADS: STABILIZING NVFP4 REINFORCEMENT LEARNING FOR MOE VIA QUANTIZATION-ERROR ALIGNMENT ACROSS DUAL SIDES

Anonymous authors
Paper under double-blind review

ABSTRACT

Rollout generation is a major bottleneck in Reinforcement Learning (RL) for Mixture-of-Experts (MoE) Large Language Models, motivating low-precision rollout acceleration such as FP8. As an emerging low-precision format, NVFP4 combines fine-grained scaling for accuracy preservation with native W4A4 FP4 GEMMs for higher throughput than FP8. However, we find that directly applying NVFP4 to MoE RL rollout is impractical. NVFP4 rollout with BF16 training collapses after roughly 150 steps, accompanied by rapidly growing rollout–trainer log-probability gaps. Through error analysis and operand-isolated experiments, we identify an alignment asymmetry: shared weight QDQ improves cross-engine probability agreement and supports stable RL, whereas activation QDQ lowers the measured agreement and is not helpful to avoid RL collapses. This is because weights can be synchronized across engines, while activations are recomputed online and matching their nominal precision is insufficient for reliable alignment. Therefore, to stabilize NVFP4 RL for MoE, we propose **QU**antization-error **A**lignment across **D**ual **S**ides (**QUADS**). On the trainer side, we introduce Asymmetric Quantization-Aware Training fake-quantizing weights while keeping activations unquantized for better alignment. On the rollout side, Partial Residual Compensation (PRC) corrects high-error activation channels while preserving native W4A4 GEMMs. In our MoE RL experiments on several benchmarks, QUADS achieves BF16-level accuracy, improving average pass@1 by 21.49 points over naive NVFP4 RL, and delivers at least 26% higher rollout throughput than FP8 across the evaluated scenarios.

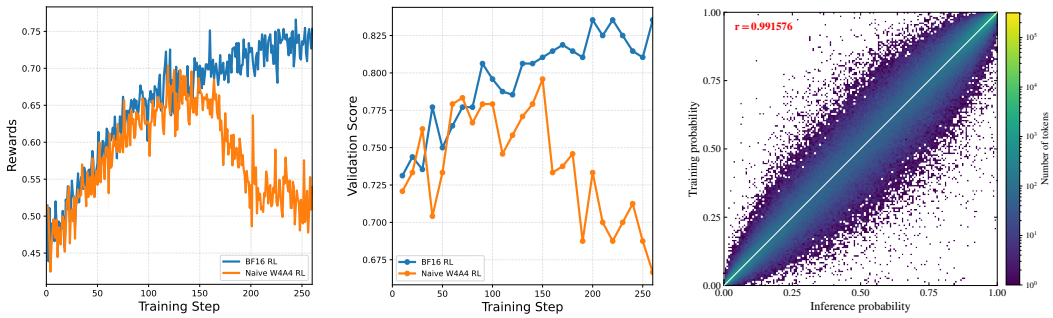
1 INTRODUCTION

Modern RL pipelines decouple autoregressive *rollout* on an inference engine from gradient computation on a separate trainer (Sheng et al., 2024; Wang et al., 2025). Rollout generation still dominates end-to-end wall-clock time—often exceeding 70% of each training step—and the imbalance grows as chain-of-thought traces become longer, making rollout acceleration a first-order priority for policy-gradient methods such as GRPO (Shao et al., 2024; Li et al., 2026; Xi et al., 2026; Qiu et al., 2026). The challenge is that rollout precision is not merely a serving detail: token-level log-probabilities produced by the rollout engine determine importance ratios in the trainer, so numerical bias can directly corrupt the policy gradient.

A common acceleration strategy is to run rollout inference at reduced precision. At the FP8 tier, the resulting *training–inference precision mismatch* can often be controlled with truncated importance sampling (TIS) (Yao et al., 2025), adaptive clipping, or unified FP8 precision flows (Li et al., 2026; Xi et al., 2026; Qiu et al., 2026; Zhou et al., 2026), so FP8 accelerates rollout while keeping the log-probability gap small enough for stable optimization.

NVIDIA Blackwell introduces a more aggressive target. NVFP4 enables native *W4A4* general matrix multiplications (GEMMs) on FP4 Tensor Cores, with peak GEMM throughput up to $\sim 2\times$ higher than FP8 and $\sim 4\times$ over BF16 (Alvarez et al., 2025; Jarmusch & Chandrasekaran, 2025; NVIDIA, 2025). However, NVFP4 is less forgiving than FP8, since its E2M1 core uses a coarser quantization

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.



(a) Training Reward (b) Validation Score (c) Rollout–Trainer Prob Diff

Figure 1: Naive NVFP4 collapse in RL. **(a)** Mean training reward: naive NVFP4 W4A4 RL (orange) peaks near step 150 and then collapses, whereas the BF16 RL (blue) continues to improve. **(b)** Validation score shows the same collapse pattern. **(c)** Token probabilities from the NVFP4 rollout engine and the BF16 trainer. The deviations from the identity line reveal discrepancies between the two engines.

grid than FP8 E4M3. In our setting, NVFP4 W4A4 rollout with BF16 training produces the collapse and probability discrepancy in Figure 1.

To understand which errors can be aligned, we separately quantize weights and activations during rollout (Section 3.1). Figures 3 and 4(a) reveal a strong operand asymmetry: with W4A16 rollout, trainer-side weight QDQ improves token-probability agreement and supports stable RL training; with W16A4 rollout, activation QDQ lowers the measured probability correlation and training still collapses. Thus, matching the nominal activation precision is insufficient to ensure cross-engine alignment, even when weights remain in BF16. This motivates treating NVFP4 RL as a dual-side alignment problem: align synchronized weights on the trainer and reduce activation quantization error on the rollout side.

Inspired by the observation, we propose QUantization-error Alignment across Dual Sides (QUADS) for NVFP4 RL. On the training side, Asymmetric Quantization-Aware Training (AQAT) fake-quantizes weights while keeping learner activations in BF16 (Section 3.2). On the rollout side, Partial Residual Compensation (PRC) improves the alignment with the trainer, while maintaining most of the W4A4 throughput advantage (Section 3.3). Finally, these components recover BF16-level accuracy in full-parameter MoE GRPO: our QUADS pipeline reaches 72.86% average pass@1 across four held-out benchmarks, matching the 73.15% BF16 baseline and avoiding the severe degradation of naive NVFP4 training (51.37%). At the same time, it improves rollout throughput by at least 26% over FP8 across the evaluated scenarios.

Our main contributions are:

- **NVFP4 RL failure diagnosis.** Ablations show that precision alignment for activation insteadly aggravates the RL collapse, which reveals weights and activations need to be treated asymmetrically.
- **Asymmetric Quantization-Aware Training.** On the trainer side, we propose AQAT to reduce weight mismatch without amplifying cross-engine activation drift.
- **Partial Residual Compensation.** On the rollout side, PRC improves the alignment with the trainer, while preserving native W4A4 GEMMs and closing the remaining log-probability bias.
- **NVFP4 RL empirical validation for MoE.** On the RL training for MoE models of different sizes, our QUADS matches BF16-level benchmark accuracy and maintains a significant throughput advantage.

2 Related Work

2.1 Reinforcement Learning for Large Language Models

RLHF and its verifiable-reward variant, RLVR, are standard LLM post-training stages: PPO bounds updates with a clipped surrogate, while GRPO removes the value model and normalizes advantages

within a sampled response group (Schulman et al., 2017; Shao et al., 2024). Longer chain-of-thought traces make rollout on a separate inference engine the dominant cost, motivating system-level acceleration of the RL loop (Yao et al., 2025; Li et al., 2026; Xi et al., 2026). MoE models gain capacity from sparse, input-dependent expert selection (Shazeer et al., 2017); DeepSeek-V3 activates a subset of parameters per token and uses RL in post-training (DeepSeek-AI, 2024). GSPO stabilizes MoE RL with sequence-level importance ratios and clipping (Zheng et al., 2025b), and HybridFlow and ROLL handle the distributed execution of RL training and generation (Sheng et al., 2024; Wang et al., 2025).

2.2 Low-Precision Inference and RL Rollout

Quantization is one part of the efficiency techniques surveyed by Wan et al. (2024). Inference precision has moved from INT8 (Li et al., 2026) through FP8 (Zhao et al., 2025; Xi et al., 2026; Qiu et al., 2026) to FP4. FP8 E4M3 has 256 representable levels, is native on Hopper and Blackwell Tensor Cores, and is the default for high-throughput serving. NVFP4 uses an E2M1 core with block-scaled E4M3 scaling and targets $\sim 4\times$ GEMM throughput over BF16 on Blackwell FP4 Tensor Cores (Alvarez et al., 2025; NVIDIA, 2025; Jarmusch & Chandrasekaran, 2025). Four Over Six lowers NVFP4 error by adaptive block scaling (Cook et al., 2025), and ReQAT applies reasoning-oriented FP4 QAT to weights, activations, and KV caches, using trace alignment and selective entropy minimization to protect low-entropy decisions (Lee et al., 2026). For RL rollout, QAT inserts fake quantization–dequantization into the learner (Gu et al., 2026). Weight-only QAT is direct because both engines share the QDQ-aligned weight and the straight-through estimator passes gradients through rounding. QeRL quantizes only NVFP4 weights and keeps activations in BF16, but runs FP16 GEMMs instead of FP4 Tensor Cores (Huang et al., 2025). Symmetric W4A4 QAT fake-quantizes both operands and can amplify engine drift on the coarse E2M1 grid (Section 3.2), whereas asymmetric W4A16 QAT with inference-side Partial Residual Compensation supports stable full-parameter MoE RL on native W4A4 FP4 GEMMs.

2.3 Training–Inference Mismatch in RL

Separate rollout and training engines create a per-token log-probability gap that corrupts importance ratios and policy gradients (Qi et al., 2025; Yao et al., 2025). R3 replays rollout expert routes to remove MoE routing inconsistency (Ma et al., 2025), PR2 predicts how the router changes during off-policy updates before replaying those routes (Dong et al., 2026), and Zheng et al. (2025a) study this discrepancy together with policy staleness. QUADS targets weight and activation quantization under NVFP4. On the finer FP8 and INT8 grids, QuRL quantizes the actor for generation while keeping full-precision updates and adaptive clipping (Li et al., 2026); Jet-RL uses one FP8 flow for training and rollout by also quantizing the learner (Xi et al., 2026); FP8-RL adds truncated importance sampling (Qiu et al., 2026); QaRL uses rollout-aware QAT (Gu et al., 2026); and AIS sets clipping thresholds from the observed ratio distribution (Zhou et al., 2026). Their log-probability gaps remain tractable, and none targets native W4A4 FP4 Tensor Core inference, where E2M1 has far fewer representable values than E4M3 and importance-sampling corrections alone are insufficient.

3 Methods

Naive NVFP4 W4A4 rollout with BF16 training fails because the training–inference mismatch destabilizes optimization. Figure 2 summarizes the dual-side response. Section 3.1 shows that shared weight QDQ improves cross-engine agreement, whereas activation QDQ does not reliably align the engines or stabilize training. Section 3.2 therefore fake-quantizes trainer weights to FP4 while keeping learner activations in BF16, and Section 3.3 compensates the remaining rollout activation error.

3.1 FP4 RL Mismatch Error Analysis

Figure 1 shows the symptom: W4A4 rollout with a BF16 learner collapses within roughly 150 steps, while BF16 rollout remains stable. For a fixed checkpoint θ and token sequence, the per-token gap between $\pi_\theta^{\text{infer}}$ and $\pi_\theta^{\text{train}}$ is attributed to weight quantization, activation quantization, and numerical differences between the decoupled engines (Qi et al., 2025; Yao et al., 2025). This attributes error

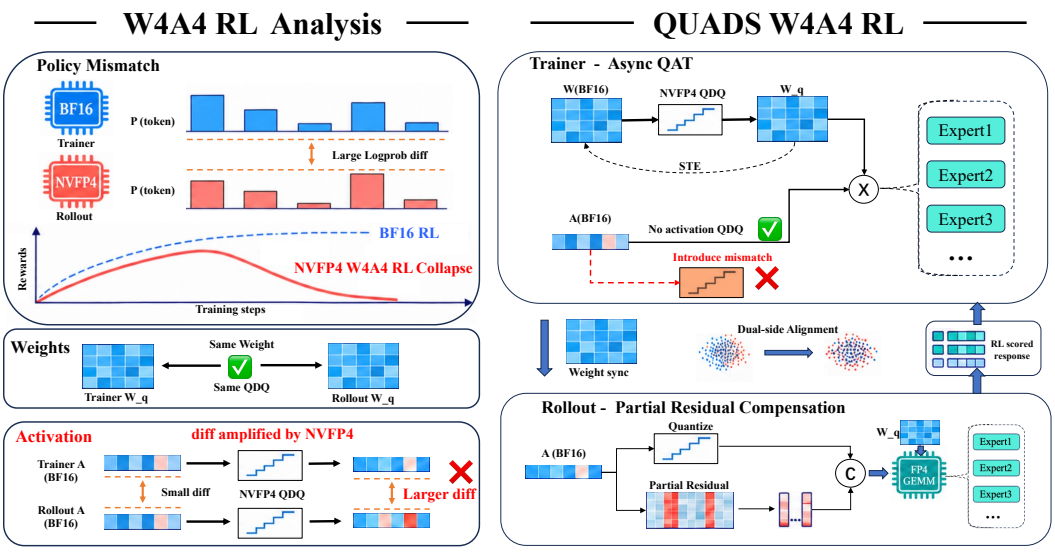


Figure 2: Overview of QUADS. Left: NVFP4 W4A4 rollout induces a training–inference policy mismatch; shared weight QDQ aligns weights, whereas activation QDQ can amplify differences between the two engines. Right: QUADS combines trainer-side W4A16 QAT, which fake-quantizes weights while retaining BF16 activations, with rollout-side Partial Residual Compensation to reduce activation quantization error.

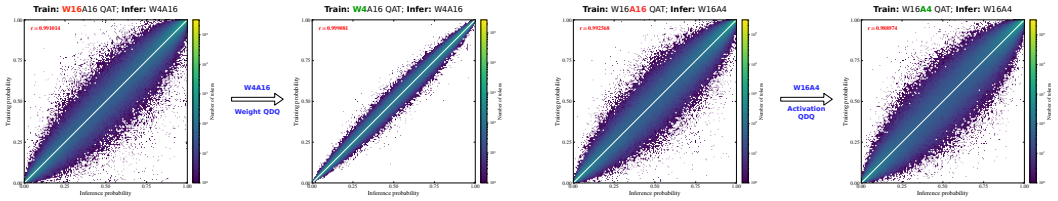
sources; it is not an exact sum of log-probabilities, and interactions through later layers are not separated. The question is which term standard quantization–dequantization (QDQ) can reduce, and which remains exposed in true W4A4 rollout.

Layer-wise source of the mismatch. Consider one linear or MoE projection with BF16 training output $\mathbf{Y}_{\text{train}} = \mathbf{W}_{\text{train}}\mathbf{X}_{\text{train}}$. For this comparison, $\mathbf{W}_{\text{infer}} = \mathbf{W}_{\text{train}}$. The inference engine recomputes activations through a different numerical path before applying NVFP4 quantization. Let Q_W and Q_A denote the hardware-aligned NVFP4 weight and activation QDQ operators. The weight error, the engine drift before activation quantization, and the activation quantization error are $\Delta\mathbf{W}_q = Q_W(\mathbf{W}_{\text{infer}}) - \mathbf{W}_{\text{infer}}$, $\boldsymbol{\eta} = \mathbf{X}_{\text{infer}} - \mathbf{X}_{\text{train}}$, and $\Delta\mathbf{X}_q = Q_A(\mathbf{X}_{\text{infer}}) - \mathbf{X}_{\text{infer}}$. The inference output is $\mathbf{Y}_{\text{infer}} = (\mathbf{W}_{\text{train}} + \Delta\mathbf{W}_q)(\mathbf{X}_{\text{train}} + \boldsymbol{\eta} + \Delta\mathbf{X}_q)$. Subtracting the training output and taking the Frobenius norm defines the layer-wise forward mismatch

$$\begin{aligned}\Delta\mathbf{Y} &\triangleq \|\mathbf{Y}_{\text{infer}} - \mathbf{Y}_{\text{train}}\|_F \\ &= \|\mathbf{W}_{\text{train}}(\boldsymbol{\eta} + \Delta\mathbf{X}_q) + \Delta\mathbf{W}_q\mathbf{X}_{\text{train}} + \Delta\mathbf{W}_q(\boldsymbol{\eta} + \Delta\mathbf{X}_q)\|_F\,,\end{aligned} \tag{1}$$

where the three summands are the activation error, the weight error, and the second-order product. Weights are synchronized across engines, so a matched weight QDQ can eliminate the difference between their effective weights, although the quantization distortion $\Delta\mathbf{W}_q$ itself remains nonzero. Activations, in contrast, are recomputed: applying the same activation QDQ to $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{train}} + \boldsymbol{\eta}$ need not produce matching values.

Operand-isolated QDQ alignment. Figure 3 holds rollout precision fixed within each comparison and adds only the corresponding trainer-side QDQ. With W4A16 rollout, weight QDQ raises the token-probability correlation from $r = 0.991014$ to $r = 0.999081$. With W16A4 rollout, activation QDQ lowers it from $r = 0.992568$ to $r = 0.988974$. Matching the activation quantization format therefore does not improve cross-engine agreement in this comparison.



(a) Weight QDQ Alignment (b) Activation QDQ Alignment

Figure 3: Asymmetric effects of trainer-side QDQ on cross-engine token-probability agreement. **(a)** With W4A16 rollout fixed, weight QDQ on the trainer side improves probability correlation. **(b)** With W16A4 rollout fixed, activation QDQ lowers the measured correlation.

Training stability under operand-isolated alignment. Figure 4(a) carries the same comparison into RL training. W4A16 rollout with a W4A16 QAT trainer continues to improve and stays close to the FP8 reference. W16A4 rollout with a W16A4 QAT trainer still collapses, with reward turning downward around step 120, even though the weights remain in BF16. Activation-side QDQ is therefore insufficient to stabilize that configuration. These diagnostics use W4A16 or W16A4 rollout; the proposed pipeline uses W4A4 rollout, and Appendix A compares asymmetric and symmetric QAT in that setting.

3.2 Asymmetric Quantization-Aware Training for the Trainer

Figures 3 and 4(a) show that weight QDQ improves agreement and stabilizes weight-only RL, whereas activation QDQ does not. For native W4A4 rollout we therefore fake-quantize only the trainer weights to FP4 and keep learner activations in BF16. Appendix A compares this choice with symmetric W4A4 QAT at the same rollout precision.

Forward and backward of Asymmetric QAT. For each linear or MoE projection, let $\mathrm{QDQ}_W(\cdot)$ denote the FP4 weight fake-quantization operator whose scale layout and rounding match the rollout kernel. With no activation fake quantization on the learner, the forward pass and the straight-through estimator (STE) backward pass are

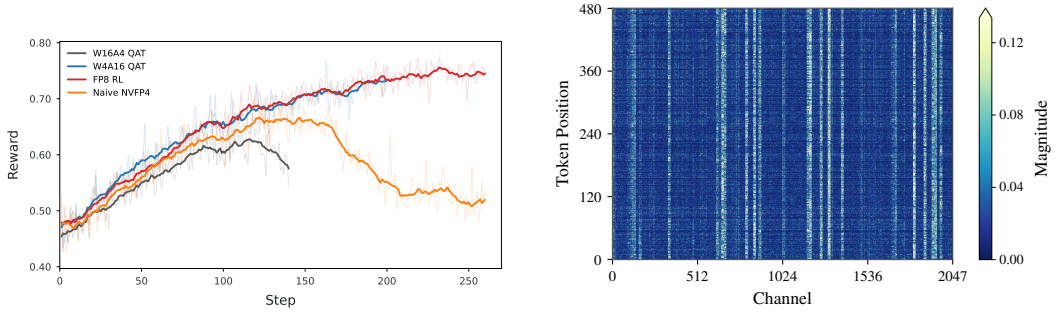
$$\begin{aligned} \widehat{\mathbf{W}} &= \mathrm{QDQ}_W(\mathbf{W}), & \mathbf{Y}_{\mathrm{qat}} &= \widehat{\mathbf{W}}\mathbf{X}_{\mathrm{train}}^{\mathrm{BF16}}, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}} &\approx \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}}, & \frac{\partial \mathcal{L}}{\partial \mathbf{X}_{\mathrm{train}}} &= \widehat{\mathbf{W}}^{\top}\frac{\partial \mathcal{L}}{\partial \mathbf{Y}_{\mathrm{qat}}}. \end{aligned} \tag{2}$$

The optimizer still updates BF16 master weights, but every learner forward pass sees the same discretized weights used by rollout. Below, $\Delta\mathbf{Y}^{\mathrm{pre}}$ and $\Delta\mathbf{Y}^{\mathrm{post}}$ are local mismatch norms before and after the corresponding trainer-side QDQ. The comparison holds layer inputs and engine drift fixed; inserting QAT throughout a network can also change upstream activations.

Why weights should be quantized on the learner. In the weight-only setting, activations remain BF16, so $\Delta\mathbf{X}_q = 0$, while engine drift $\boldsymbol{\eta}$ may remain nonzero. Synchronized parameters satisfy $\mathbf{W}_{\mathrm{infer}} = \mathbf{W}_{\mathrm{train}}$. Writing $\widehat{\mathbf{W}} = Q_W(\mathbf{W}_{\mathrm{train}})$, the local mismatch before trainer-side weight QDQ is $\|\Delta\mathbf{W}_q\mathbf{X}_{\mathrm{train}} + \widehat{\mathbf{W}}\boldsymbol{\eta}\|_F$. If the trainer-side quantizer Q_W^{train} matches rollout in scale layout and rounding, then $\|Q_W(\mathbf{W}_{\mathrm{infer}}) - Q_W^{\mathrm{train}}(\mathbf{W}_{\mathrm{train}})\|_F = 0$, and the mismatch after alignment is $\|\widehat{\mathbf{W}}\boldsymbol{\eta}\|_F$. Weight QDQ removes the explicit cross-engine weight-difference term, rather than the quantization distortion $\Delta\mathbf{W}_q$ itself. This does not guarantee a strict decrease in the total norm for every input, since error terms can cancel. Figures 3(a) and 4(a) provide the empirical evidence: weight alignment improves probability agreement and supports stable training in the evaluated weight-only configuration.

Why activations should remain unquantized on the learner. In the activation-only setting, weights remain BF16 and only the activation term in Eq. (1) is active, so the mismatch before learner-side activation QDQ is $\|\mathbf{W}_{\mathrm{train}}(\boldsymbol{\eta} + \Delta\mathbf{X}_q)\|_F$. After both engines apply Q_A, the comparison becomes $\|\mathbf{W}_{\mathrm{train}}[Q_A(\mathbf{X}_{\mathrm{train}} + \boldsymbol{\eta}) - Q_A(\mathbf{X}_{\mathrm{train}})]\|_F$, because Q_A is applied to already-different

activations. Unlike synchronized weights, the inputs to activation QDQ differ across engines. Small input differences can cross quantization boundaries, so a shared activation quantizer need not align their outputs and can widen the gap. Figures 3(b) and 4(a) show the corresponding probability disagreement and training collapse under activation-only alignment. For the target W4A4 rollout setting, Figure 8(a) further shows that asymmetric W4A16 QAT yields a smaller measured MoE-input mismatch than symmetric W4A4 QAT. Together, these observations support retaining BF16 learner activations and compensating rollout-side activation error separately.



(a) Operand-isolated RL training.

(b) Activation residual magnitude.

Figure 4: Operand-isolated training and the NVFP4 activation residual. **(a)** W4A16 QAT (blue) pairs W4A16 rollout with weight QAT and tracks FP8 RL (red). W16A4 QAT (gray) pairs W16A4 rollout with activation QAT but still collapses. Naive NVFP4 (orange) uses W4A4 rollout with a BF16 trainer. **(b)** Persistent vertical bands show that the activation residual is concentrated on a subset of channels.

3.3 Activation Error Compensation for Rollout

Weight-only trainer QAT leaves the activation term in Eq. (1) uncorrected, so we compensate that rollout-time FP4 error on the inference side. Section 4.3 (Figure 6) evaluates the remaining gap to BF16 and the benefit of this compensation. For an inference-side BF16 activation $\mathbf{X} \in \mathbb{R}^{d\times n}$, with channels indexed by rows and tokens by columns, let Q_A denote the primary NVFP4 quantization–dequantization operator. We retain $\Delta\mathbf{X}_q = Q_A(\mathbf{X}) - \mathbf{X}$ and write the signed correction residual as $\mathbf{R} = \mathbf{X} - Q_A(\mathbf{X}) = -\Delta\mathbf{X}_q$. The residual is not uniform across hidden dimensions. Block scales follow the local dynamic range, so some high-magnitude channels create large quantization errors while other channels remain much smaller. Figure 4(b) shows the high-residual channels as persistent vertical bands, which motivates additional computation only where $\|\Delta\mathbf{X}_q\|$ is large.

Targeted Partial Residual Compensation. We quantize the signed residual on a selected channel set $\mathcal{S}$ using a second NVFP4 quantization–dequantization operator Q_R, with scales computed from the selected residual values rather than reused from the primary activation quantization. Writing $Q_R(\mathbf{R})$ with zeros outside the selected channels, the effective reconstructed activation is

$$\tilde{\mathbf{X}} = Q_A(\mathbf{X}) + \mathbf{M}_{\mathcal{S}} \odot Q_R(\mathbf{R}), \tag{3}$$

where $\mathbf{M}_{\mathcal{S}}$ is a binary channel mask broadcast over tokens and $\odot$ denotes element-wise multiplication. Residual magnitudes determine which channels are selected, but the correction retains the residual signs. The reconstruction error is $\tilde{\mathbf{X}} - \mathbf{X} = -(1-\mathbf{M}_{\mathcal{S}}) \odot \mathbf{R} + \mathbf{M}_{\mathcal{S}} \odot (Q_R(\mathbf{R}) - \mathbf{R})$: selected channels replace the primary residual with the second-stage quantization error, and unselected channels keep their original error. This helps when the second stage represents the selected residuals more accurately, but it does not guarantee a smaller distance to another engine's activations. Eq. (3) describes the correction algebraically; it does not quantize the reconstructed sum again. For $\widehat{\mathbf{W}} = Q_W(\mathbf{W})$, the corresponding projection can be written as

$$\tilde{\mathbf{Y}} = \widehat{\mathbf{W}} Q_A(\mathbf{X}) + \widehat{\mathbf{W}}_{:,\mathcal{S}} \left[Q_R(\mathbf{R})\right]_{\mathcal{S},:}. \tag{4}$$

Both terms use FP4-quantized operands with their associated scales, preserving the W4A4 computation path without a BF16 residual branch.

Table 1: Pass@1 accuracy (%) on four benchmarks. **BF16 RL** is the full-precision baseline; **FP8 RL** applies FP8 quantization on both sides; **Naive NVFP4 RL** uses NVFP4 rollout with BF16 training and no alignment; **QUADS** is our proposed dual-side alignment pipeline in NVFP4 RL. QUADS recovers average accuracy close to BF16 for both models, with benchmark-specific differences, while naive NVFP4 RL has lower scores on all four benchmarks. The best result in each column within each model block is shown in bold.

METHOD	LIVECODEBENCH	HMMT 2025	AIME 2024	AIME 2025	AVERAGE
Qwen3.5-35A3B					
BF16 RL	63.05	**66.04**	**83.54**	80.00	**73.15**
FP8 RL	**65.27**	63.54	82.91	79.79	72.88
Naive NVFP4 RL	24.25	48.33	66.67	66.25	51.37
QUADS RL	64.79	62.92	83.33	**80.42**	72.86
Qwen3.5-122A10B					
BF16 RL	**79.82**	**80.83**	92.08	84.58	84.33
Naive NVFP4 RL	60.42	66.67	86.67	77.92	72.92
QUADS RL	78.86	80.42	**92.92**	**86.25**	**84.61**

Online per-channel selection. Because RL updates change the policy, activation statistics are non-stationary, and we choose $\mathcal{S}$ online on each forward pass. For channel j, the score on the current token batch is $s_j = \|\mathbf{R}[j,:]\|_2$, and $\mathcal{S} = \mathrm{TopK}_m(\{s_j\}_{j=1}^{d})$ returns the indices of the m largest scores for a target ratio of $k\%$. The count $m = |\mathcal{S}| \approx kd/100$ must be a multiple of 16 for the NVFP4 residual GEMM block, so the target percentage is subject to this constraint.

4 EXPERIMENTS

We evaluate QUADS on GRPO training of Qwen3.5-35A3B and Qwen3.5-122A10B MoE models (Qwen Team, 2026) under NVFP4 W4A4 rollout. We first describe the experimental setup (Section 4.1), followed by the main RL training results (Section 4.2), the training dynamics of QAT and QUADS (Section 4.3), and efficiency analysis (Section 4.4).

4.1 EXPERIMENTAL SETUP

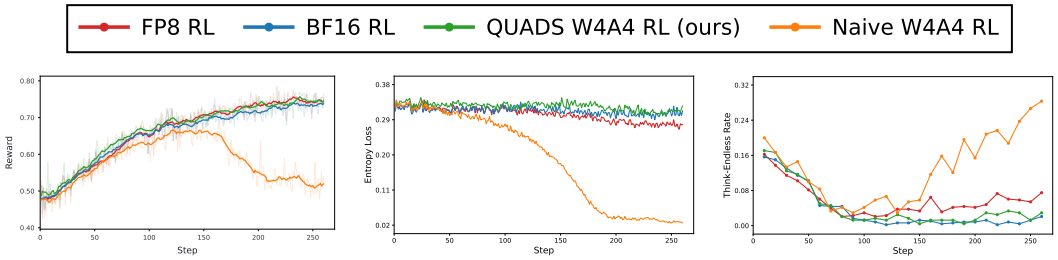
Experiment Configuration. Training and rollout run on separate pools of NVIDIA Blackwell GPUs, using Megatron-LM (Shoeybi et al., 2019) and SGLang (Zheng et al., 2023), with weights synchronized at each checkpoint refresh. We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024), sampling N=16 responses per prompt with a global batch size of 256, for 260 steps on Qwen3.5-35A3B and 150 steps on Qwen3.5-122A10B. QUADS uses a residual channel selection ratio of 20% in the main performance and throughput experiments. Training uses a mixture of mathematical reasoning and code generation prompts. We evaluate pass@1 on AIME 2024, AIME 2025, and HMMT 2025, and on LiveCodeBench (Jain et al., 2024), and report the arithmetic mean across the four benchmarks. Parallelism, optimizer settings, importance sampling, routing replay, and the dataset protocol are given in Appendix C.

Baselines. We investigate four configurations: **BF16 RL**, with BF16 training and rollout; **FP8 RL**, with FP8 training and W8A8 rollout; **Naive NVFP4 RL**, with BF16 training and W4A4 rollout; and **QUADS**, combining Asymmetric QAT with Partial Residual Compensation W4A4. BF16 provides the high-precision reference, while FP8 provides a low-precision reference for both accuracy and rollout throughput.

4.2 OVERALL PERFORMANCE AND TRAINING DYNAMICS

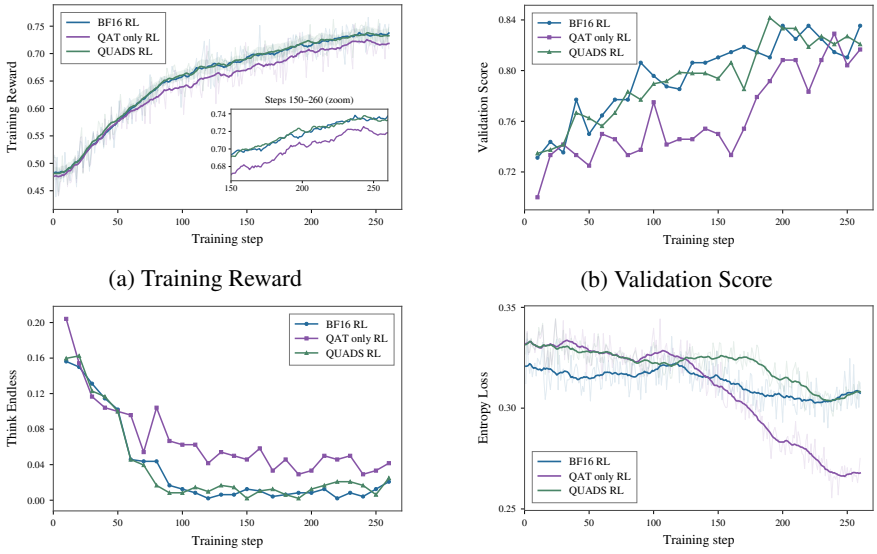
Benchmark Performance. QUADS recovers average performance close to the BF16 and FP8 references (Table 1). For Qwen3.5-35A3B, its average pass@1 is 72.86%, compared with 51.37% for Naive NVFP4 RL, an improvement of 21.49 percentage points. The gap to BF16 is 0.29 percentage

points, and the gap to FP8 is 0.02 percentage points. The largest recovery over Naive NVFP4 RL occurs on LiveCodeBench, from 24.25% to 64.79%. Against BF16, QUADS is higher on LiveCodeBench and AIME 2025, 0.21 percentage points lower on AIME 2024, and 3.12 percentage points lower on HMMT 2025. For Qwen3.5-122A10B, QUADS achieves an average pass@1 of 84.61%, compared with 84.33% for BF16 and 72.92% for Naive NVFP4 RL.



(a) Training Reward (b) Entropy Loss (c) Think-Endless Rate

Figure 5: Training dynamics of Qwen3.5-35A3B over 260 steps. (a) Training reward; (b) coding-task entropy loss; (c) think-endless rate. Per-benchmark validation curves, HMMT 2025 response length, and gradient norm are in Appendix D.



(a) Training Reward (b) Validation Score

(c) Think Endless (d) Entropy Loss

Figure 6: Training dynamics of weight-only QAT and QUADS against BF16 RL. (a) Training reward; (b) validation score; (c) think-endless rate; (d) entropy loss.

Training Dynamics. Figure 5 tracks training reward, coding-task entropy loss, and think-endless rate. BF16 RL and FP8 RL have increasing reward. BF16 maintains relatively stable entropy after an early decrease; FP8 shows a gradual entropy decline and a later increase in think-endless rate. QUADS tracks both baselines on reward and remains close to BF16 on entropy and think-endless rate. Naive NVFP4 RL follows them early, then its reward falls in the second half of training, its entropy drops toward zero, and its think-endless rate rebounds. Appendix D shows the same runs on each held-out benchmark, where BF16 and FP8 improve steadily and naive NVFP4 declines after an early rise, together with HMMT 2025 response length and gradient norm. There, naive NVFP4 response length rebounds and FP8 response length increases later, while QUADS stays closer to BF16; gradient norms remain on a comparable scale across the four runs, aside from isolated spikes.

Together with the mismatch analysis in Section 3.1, these observations support improved training stability in the evaluated setting.

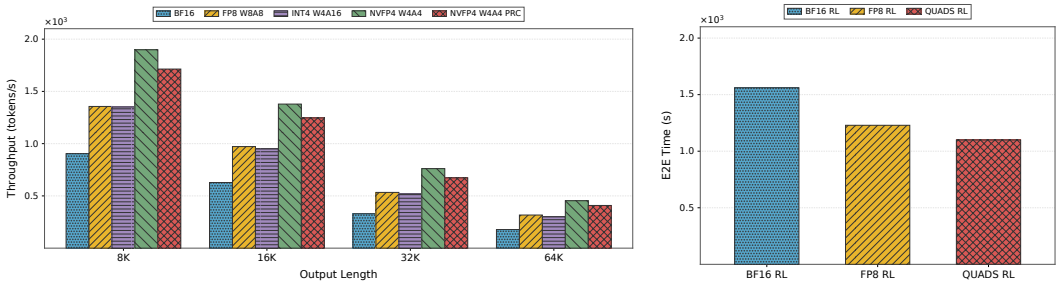
4.3 Training Dynamics of QAT and QUADS

Figure 6 compares BF16 RL with two W4A4 rollout runs: weight-only W4A16 QAT, and QUADS. QAT alone does not show the late reward collapse of naive NVFP4 RL in Figure 5. Its curves in Figure 6 nevertheless remain far from BF16 RL: reward grows more slowly, the validation score stays below BF16 for most of training, the think-endless rate remains higher after the early decline, and entropy loss drops in the second half of training. That gap makes weight-only QAT unfavorable for practical training. QUADS stays close to BF16 on the same four curves, including reward over the later steps, so the full method is still required.

The layer-wise mismatch comparison is reported in Appendix A, and the comparison of PRC alone, W4A16 QAT alone, and their combination is reported in Appendix B.

4.4 Efficiency Analysis

Figure 7 reports rollout throughput and the time for one RL training step. On Qwen3.5-35A3B, we compare BF16, FP8 W8A8, INT4 W4A16, plain NVFP4 W4A4, and NVFP4 W4A4 with Partial Residual Compensation (PRC) at output lengths of 8K, 16K, 32K, and 64K (Figure 7a). The plain NVFP4 comparison isolates the throughput cost of PRC; the BF16 and FP8 comparisons measure the remaining throughput advantage. Input length is fixed at 8K. PRC achieves 1,713.9, 1,246.4, 672.7, and 408.5 tokens/s at 8K, 16K, 32K, and 64K outputs, respectively. It retains 88.4–90.4% of plain NVFP4 throughput, a reduction of 9.6–11.6%. Its throughput is at least 26% higher than FP8 W8A8 across these output lengths, is 1.90–$2.29\times$ that of BF16, and exceeds INT4 W4A16 at all four lengths. Figure 7b shows the average time for one RL training step on Qwen3.5-122A10B. The BF16 and FP8 step times are $1.42\times$ and $1.12\times$ the QUADS step time, respectively.



(a) Rollout throughput (↑). (b) Single-step RL time (↓).

Figure 7: Efficiency comparison. (a) Qwen3.5-35A3B rollout throughput across output lengths and precision configurations; PRC denotes Partial Residual Compensation. (b) Qwen3.5-122A10B average time for one RL training step.

5 Conclusion

Native W4A4 FP4 GEMMs raise rollout throughput over FP8, but pairing NVFP4 rollout with BF16 training destabilizes MoE GRPO on Blackwell. Weight QDQ improves cross-engine agreement and supports stable weight-only training, whereas activation QDQ does not prevent collapse in the evaluated activation-only setting. QUADS therefore aligns trainer weights with asymmetric W4A16 QAT and reduces rollout activation error by Partial Residual Compensation, while keeping native W4A4 GEMMs. On full-parameter MoE GRPO, it reaches BF16-level accuracy, improves average pass@1 by 21.49 percentage points over naive NVFP4 RL, and retains at least 26% higher rollout throughput than FP8. Future work includes more adaptive residual-channel selection and extending this alignment principle to other RL algorithms and architectures.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing. We have reviewed all AI-assisted text. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, Ronny Krashinsky, and Kyle Aubrey. Introducing NVFP4 for efficient and accurate low-precision inference. NVIDIA Technical Blog, 2025. URL https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/.

Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, Keith Wyss, Mahdi Nazemi, Asit Mishra, Carlo del Mundo, Tijmen Blankevoort, and Song Han. Four Over Six: More Accurate NVFP4 Quantization with Adaptive Block Scaling. *arXiv preprint arXiv:2512.02010*, 2025. doi: 10.48550/arXiv.2512.02010. URL https://arxiv.org/abs/2512.02010.

DeepSeek-AI. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437*, 2024. doi: 10.48550/arXiv.2412.19437. URL https://arxiv.org/abs/2412.19437.

Daize Dong, Junlin Chen, Haolong Jia, Jiang Liu, Jiawei Wu, Huanwei Di, Jialian Wu, Zhengzhong Liu, Zicheng Liu, Emad Barsoum, Dimitris N. Metaxas, and Hongyi Wang. PR2: Predictive Routing Replay for MoE-Based LLM Reinforcement Learning. *arXiv preprint arXiv:2606.00395*, 2026. doi: 10.48550/arXiv.2606.00395. URL https://arxiv.org/abs/2606.00395.

Hao Gu, Hao Wang, Jiacheng Liu, Lujun Li, Qiyuan Zhu, Bei Liu, Binxing Xu, Lei Wang, Xintong Yang, Sida Lin, Sirui Han, and Yike Guo. QaRL: Rollout-aligned quantization-aware RL for fast and stable training under training–inference mismatch. *arXiv preprint arXiv:2604.07853*, 2026.

Wei Huang, Yi Ge, Shuai Yang, Yicheng Xiao, Huizi Mao, Yujun Lin, Hanrong Ye, Sifei Liu, Ka Chun Cheung, Hongxu Yin, Yao Lu, Xiaojuan Qi, Song Han, and Yukang Chen. QeRL: Beyond efficiency–quantization-enhanced reinforcement learning for LLMs. *arXiv preprint arXiv:2510.11696*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv preprint arXiv:2403.07974*, 2024. doi: 10.48550/arXiv.2403.07974. URL https://arxiv.org/abs/2403.07974.

Aaron Jarmusch and Sunita Chandrasekaran. Microbenchmarking NVIDIA's blackwell architecture: An in-depth architectural analysis. *arXiv preprint arXiv:2512.02189*, 2025.

Janghwan Lee, Sihwa Lee, Jinseok Kim, Yongjik Kim, Jieun Lim, Jinwook Oh, and Jungwook Choi. ReQAT: Achieving Full-Precision Reasoning Accuracy with 4-bit Floating-Point Quantization-Aware Training. *arXiv preprint arXiv:2606.15682*, 2026. doi: 10.48550/arXiv.2606.15682. URL https://arxiv.org/abs/2606.15682.

Yuhang Li, Reena Elangovan, Xin Dong, Priyadarshini Panda, and Brucek Khailany. QuRL: Efficient reinforcement learning with quantized rollout. *arXiv preprint arXiv:2602.13953*, 2026.

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. doi: 10.48550/arXiv.2510.11370. URL https://arxiv.org/abs/2510.11370.

NVIDIA. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025.

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.

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.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv preprint arXiv:1701.06538*, 2017. doi: 10.48550/arXiv.1701.06538. URL https://arxiv.org/abs/1701.06538.

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. doi: 10.48550/arXiv.2409.19256. URL https://arxiv.org/abs/2409.19256.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2019. doi: 10.48550/arXiv.1909.08053. URL https://arxiv.org/abs/1909.08053.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient Large Language Models: A Survey. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=bsCCJHbO8A.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, Zichen Liu, Haizhou Zhao, Dakai An, Lunxi Cao, Qiyang Cao, Wanxi Deng, Feilei Du, Yiliang Gu, Jiahe Li, Xiang Li, Mingjie Liu, Yijia Luo, Zihe Liu, Yadao Wang, Pei Wang, Tianyuan Wu, Yanan Wu, Yuheng Zhao, Shuaibing Zhao, Jin Yang, Siran Yang, Yingshui Tan, Huimin Yi, Yuchi Xu, Yujin Yuan, Xingyao Zhang, Lin Qu, Wenbo Su, Wei Wang, Jiamang Wang, and Bo Zheng. Reinforcement Learning Optimization for Large-Scale Learning: An Efficient and User-Friendly Scaling Library. *arXiv preprint arXiv:2506.06122*, 2025. doi: 10.48550/arXiv.2506.06122. URL https://arxiv.org/abs/2506.06122.

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.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient RL framework secretly brings you off-policy RL training. Notion Blog, 2025. URL https://fengyao.notion.site/off-policy-rl.

Chenggang Zhao, Zhean Xu, Liang Zhao, Jiashi Li, Chenhao Xu, Anyi Xu, Shengyu Liu, Kexing Zhou, and Kuai Yu. DeepGEMM: Clean and efficient FP8 GEMM kernels with fine-grained scaling. GitHub, 2025. URL https://github.com/deepseek-ai/DeepGEMM.

Chujie Zheng, Kai Dang, Bowen Yu, Mingze Li, Huiqiang Jiang, Junrong Lin, Yuqiong Liu, Hao Lin, Chencan Wu, Feng Hu, An Yang, Jingren Zhou, and Junyang Lin. Stabilizing Reinforcement Learning with LLMs: Formulation and Practices. *arXiv preprint arXiv:2512.01374*, 2025a. doi: 10.48550/arXiv.2512.01374. URL https://arxiv.org/abs/2512.01374.

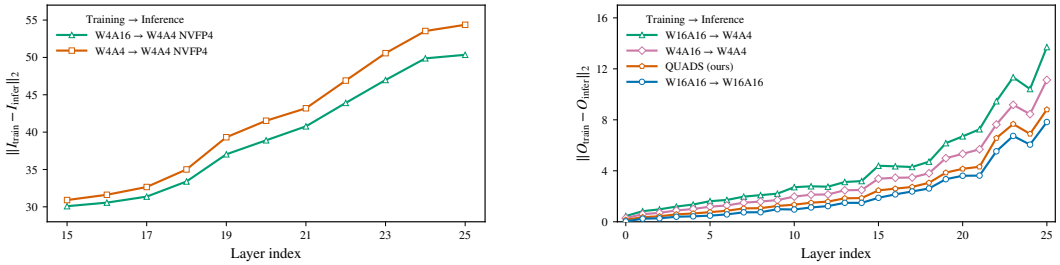
Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group Sequence Policy Optimization. *arXiv preprint arXiv:2507.18071*, 2025b. doi: 10.48550/arXiv.2507.18071. URL https://arxiv.org/abs/2507.18071.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient Execution of Structured Language Model Programs. *arXiv preprint arXiv:2312.07104*, 2023. doi: 10.48550/arXiv.2312.07104. URL https://arxiv.org/abs/2312.07104.

Jiajun Zhou, Wei Shao, Lingchao Zheng, Yuwei Fan, and Ngai Wong. AIS: Adaptive importance sampling for quantized RL. *arXiv preprint arXiv:2605.13907*, 2026.

A Layer-wise Mismatch Analysis

We measure layer-wise MoE activation mismatch between the training and inference engines on matched requests (Figure 8). Aside from the BF16 reference in Figure 8(b), rollout in this comparison is W4A4. Figure 8(a) compares two trainer-side settings. Symmetric W4A4 QAT fake-quantizes both weights and activations, while asymmetric W4A16 QAT fake-quantizes weights only. The asymmetric setting has the smaller MoE-input gap, so matching the nominal activation precision on both sides does not produce a smaller measured discrepancy. Figure 8(b) compares MoE-output mismatch under four train–infer settings. The gap is largest with no QAT. W4A16 QAT reduces it by aligning weights to the quantized grid. Adding Partial Residual Compensation further closes the distance toward the BF16 baseline.



(a) MoE-input mismatch (b) MoE-output mismatch

Figure 8: Layer-wise train–infer mismatch on matched requests. **(a)** MoE-input mismatch. **Green**: asymmetric W4A16 QAT with NVFP4 W4A4 inference. **Orange**: symmetric W4A4 QAT with NVFP4 W4A4 inference. **(b)** MoE-output L_2 distance. **Green**: no QAT (W16A16 train / W4A4 infer). **Pink**: W4A16 QAT. **Orange**: W4A16 QAT with Partial Residual Compensation. **Blue**: BF16 baseline (W16A16 train / W16A16 infer).

B Individual Component Comparison

Figure 9 compares naive NVFP4 RL, Partial Residual Compensation (PRC) alone, W4A16 QAT alone, and the combination of the two. PRC alone raises average pass@1 from approximately 51.4% to 63.2%, and QAT alone raises it to 71.1%. Both are above naive W4A4 RL and both remain below BF16 RL, whose average on the same four benchmarks is 73.15% (Table 1). Only the combination reaches that level, with an average pass@1 of 72.9%, and it is the highest of the four settings on every benchmark.

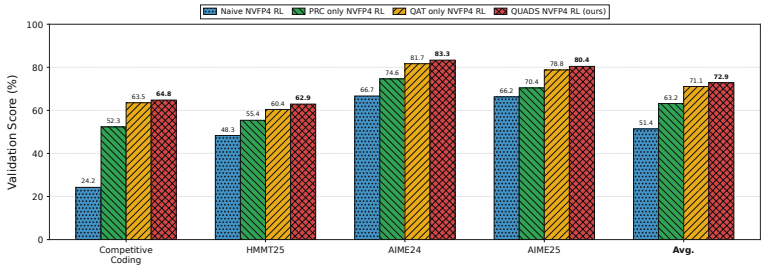


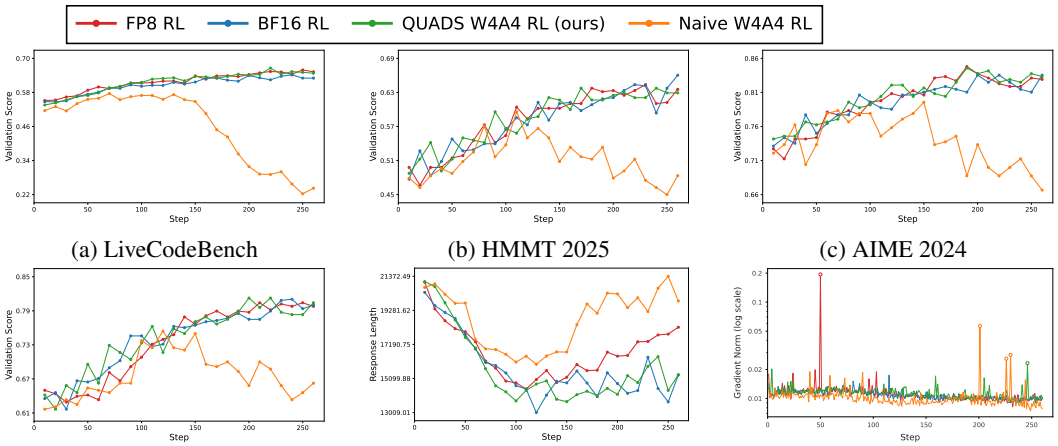
Figure 9: Pass@1 of naive NVFP4 RL, PRC alone, W4A16 QAT alone, and the combination of PRC and QAT on four held-out benchmarks.

C Experimental Details

For Qwen3.5-35A3B, training uses tensor-model parallelism TP=2, expert-model parallelism EP=16, context parallelism CP=1, pipeline parallelism PP=1, sequence parallelism SP=1, and data parallelism DP=1. SGLang uses the `trtllm_mha` attention backend with TP=2 and DP=1. We apply truncated importance sampling (TIS) (Yao et al., 2025) to the per-token importance ratios that enter the GRPO surrogate, and we use R3 (Ma et al., 2025) to replay the expert-routing decisions recorded during rollout in the training forward pass. The RL runs use a mini-batch size of 128, a learning rate of $1e{-}6$, and AdamW with $\beta_1{=}0.9$, $\beta_2{=}0.95$, and $\epsilon{=}1e{-}15$. The maximum prompt and generation lengths are 3,072 and 32,768 tokens. Positive and negative clipping ratios are 0.27 and 0.2, with a dynamic PG loss strategy whose cumulative clip range is 90, and sample-averaged loss. Qwen3.5-35A3B is evaluated every 10 steps. The mathematical training mixture combines O4-Mini, verified prover data, and DeepMath, with Chinese-language entries filtered out; the code data consists of filtered programming tasks. LiveCodeBench evaluation uses Python submissions and a unified evaluation service.

D Additional Training Dynamics

Figure 10 reports the per-benchmark validation curves, HMMT 2025 response length, and gradient norm for the same Qwen3.5-35A3B runs as Figure 5. BF16 RL and FP8 RL improve steadily on LiveCodeBench, HMMT 2025, AIME 2024, and AIME 2025. Naive NVFP4 RL follows them early and then declines on these benchmarks; its HMMT 2025 response length rebounds, while FP8 response length increases later and QUADS stays closer to the shorter BF16 responses. Gradient norms stay on a comparable scale across the four runs, aside from isolated spikes.



(a) LiveCodeBench (b) HMMT 2025 (c) AIME 2024

(d) AIME 2025 (e) HMMT 2025 Response Length (f) Gradient Norm

Figure 10: Additional training dynamics of Qwen3.5-35A3B over 260 steps, for the runs in Figure 5. (a)–(d) Validation score on LiveCodeBench, HMMT 2025, AIME 2024, and AIME 2025; (e) HMMT 2025 response length; (f) gradient norm.