ROLLOUT-GUIDED QAT FOR FP4 REINFORCEMENT LEARNING OF MOE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

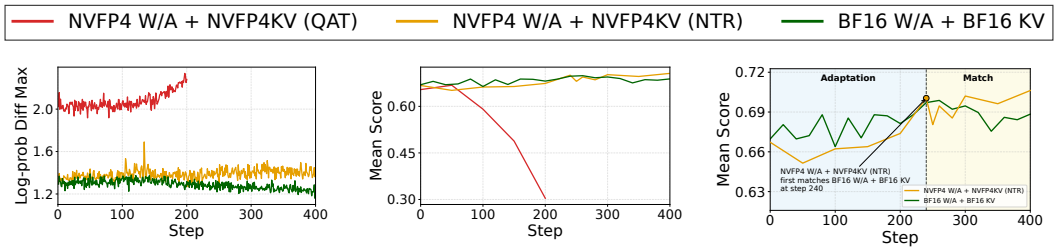
Reinforcement learning (RL) for post-training large language models (LLMs) incurs substantial computation and memory overhead during rollout generation, which motivates low-precision rollout for efficient RL training. However, existing FP4 RL methods suffer from a key limitation: they primarily optimize quantization accuracy on the training and rollout paths independently rather than directly reducing the discrepancy between the two quantized execution paths. In this work, we propose `NTR` (**N**earest-**T**o-**R**ollout), an FP4 quantization framework for RL training of Mixture-of-Experts (MoE) language models that addresses the limitation of existing FP4 RL methods. `NTR` incorporates rollout-guided quantization-aware training that uses rollout-side quantization outcomes to guide training-side FP4 rounding decisions, directly reducing train-rollout discrepancy. Moreover, `NTR` adopts an efficient quantization-information caching scheme that selectively retains mantissa and scale information from deeper layers to reduce the storage and communication overhead introduced by rollout guidance. We evaluate `NTR` on four large-scale MoE language models across reasoning, coding, and long-horizon RL tasks. Our results demonstrate that `NTR` enables joint FP4 weight/activation and FP4 KV-cache rollout with RL performance comparable to BF16 rollout, while achieving up to 5.4$\times$ rollout speedup and strong final FP4 performance compared with post-hoc FP4 quantization of BF16-trained policies.

1 INTRODUCTION

Reinforcement learning (RL) has become an important post-training paradigm for improving the reasoning, coding, and other complex capabilities of Large Language Models (LLMs) (Zhao et al., 2023; Zhang et al., 2025). Despite its effectiveness, RL training incurs substantial computation and memory overhead due to the repeated generation of long trajectories during rollout (Wang et al., 2025). Low-precision rollout (Zhuge et al., 2026; Yao et al., 2026; Li et al., 2026; Xi et al., 2026; Huang et al., 2025) provides a promising solution by quantizing the policy model during generation to reduce rollout cost. In particular, aggressive FP4 quantization (W4A4) provides greater opportunities to accelerate rollout generation and reduce its memory footprint. However, the coarse quantization space of FP4 also introduces large numerical discrepancies between the rollout and training execution paths, which can result in policy mismatch and destabilize RL training. Such inconsistency is particularly challenging for Mixture-of-Experts (MoE) language models (Shazeer et al., 2017), where numerical differences may further interact with expert routing and amplify the mismatch between rollout and training policies (Zheng et al., 2025a). To mitigate this issue, several recent FP4 RL methods (Zhuge et al., 2026; Mak et al., 2026) improve the numerical consistency between low-precision rollout and training through quantization-aware training or quantization-error compensation.

Despite these efforts, existing FP4 RL methods suffer from a fundamental issue: **Misalignment between quantization accuracy and train-rollout discrepancy.** Existing methods primarily improve quantization accuracy with respect to the high-precision representation on the training and rollout paths independently. However, minimizing the quantization error on each execution path does not directly minimize the discrepancy between the two quantized execution paths. As a consequence, even when both paths achieve small quantization errors, their quantized values can still diverge substantially, leading to large policy mismatch, degraded RL performance, or even training collapse.

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) Train-Rollout Log-Prob Diff (b) Test Score on Terminal-Bench (c) Low-Precision Adaptation

Figure 1: Training dynamics of Qwen3.8-Flash-Next under FP4 rollout. Conventional FP4 QAT exhibits increasing train-rollout discrepancy and degraded test score on Terminal-Bench 2.1, while NTR maintains stable alignment and enables the policy to progressively adapt to FP4 rollout.

In this paper, we propose NTR (**N**earest-**T**o-**R**ollout), an FP4 quantization framework for RL training of MoE language models. NTR differs from existing FP4 RL approaches in two key aspects. ❶ **Rollout-Guided Quantization-Aware Training:** NTR directly reduces local quantized train-rollout discrepancy by using rollout-side quantization outcomes to guide training-side FP4 rounding decisions. ❷ **Efficient Quantization-Information Caching:** To reduce the storage and communication overhead introduced by rollout guidance during RL training, NTR only retains the mantissa and scale information from selected deeper layers.

Figure 1 highlights the training dynamics under aggressive FP4 rollout (NVFP4 W/A + NVFP4 KV) with NTR. FP4 QAT exhibits an increasingly large train-rollout policy mismatch as training proceeds, resulting in a sharp degradation in test score on Terminal-Bench. In contrast, NTR effectively controls the train-rollout discrepancy and maintains stable RL training. More interestingly, once the discrepancy is properly controlled, the policy progressively adapts to FP4 rollout: although NTR initially suffers from quantization-induced performance degradation, the performance gap relative to BF16 rollout gradually diminishes during training and can eventually be closed or even reversed.

We compare NTR with representative FP4 rollout baselines, including vanilla QAT and QUADS, as well as post-hoc FP4 quantization methods, including vanilla NVFP4, 4over6, and H-Scale. To demonstrate the generalizability of NTR, we evaluate it on four MoE language models—Qwen3.5-35B-A3B, Qwen3.5-122B-A10B, Qwen3.8-Flash-Next, and Qwen3.8-2.4T-A95B—across reasoning, coding, and long-horizon RL tasks. We highlight three of our findings:

- NTR enables joint FP4 W/A and FP4 KV rollout while preserving BF16-level performance, maintaining training dynamics comparable to BF16 rollout in terms of log-probability difference and training reward, and achieving comparable or even better final task performance.
- NTR retains the efficiency advantage of FP4 rollout with limited algorithm overhead, achieving up to $5.4\times$ rollout speedup over BF16 rollout.
- NTR NTR achieves stronger final FP4 task performance than pipelines that first train with BF16 rollout and apply post-hoc FP4 quantization.

2 Related Work

Reinforcement Learning for MoE Language Models: Modern LLM post-training increasingly relies on reinforcement learning (RL), with GRPO (Shao et al., 2024) being widely adopted for reasoning models. For Mixture-of-Experts (MoE) models (Shazeer et al., 2017), RL training is particularly sensitive to train–rollout inconsistency because small numerical differences can alter router scores and activate different experts (Zheng et al., 2025a). R3 (Ma et al., 2025) mitigates this routing mismatch by replaying rollout routing decisions during training, while PR2 (Dong et al., 2026) further accounts for router evolution across policy updates. GSPO (Zheng et al., 2025b) instead improves stability through sequence-level importance ratios and clipping. Meanwhile, the large parameter capacity of MoE models introduces substantial memory, communication, and rollout costs, motivating efficient distributed training and low-precision rollout (Wang et al., 2025).

Low-Precision Reinforcement Learning: Recent works have explored rollout with low-precision weight to reduce the generation cost of reinforcement learning for LLMs. For MoE language

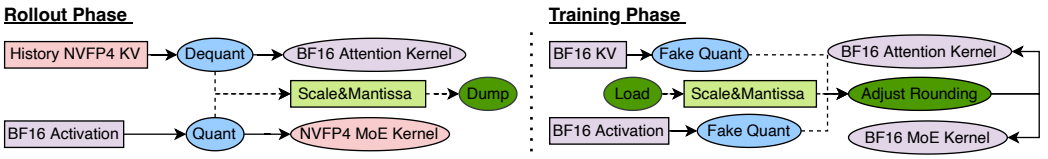


Figure 2: Overview of NTR for NVFP4 RL Training.

models, only routed experts are quantized while other modules still in high precision to ensure the generation quality (Zhuge et al., 2026; Yao et al., 2026; Yu et al., 2026). Most existing low-precision rollout methods focus on 8-bit quantization, such as FP8 or INT8. For example, QuRL (Li et al., 2026) mitigates the mismatch between quantized rollout and high-precision training through adaptive clipping and update-aware quantization. Jet-RL (Xi et al., 2026) instead aligns the numerical precision of training and rollout by adopting a unified FP8 computation flow. More recently, a few works have extended rollout quantization to 4-bit precision with quantization-aware training (QAT) on the training side for better train–rollout consistency. For example, QeRL (Huang et al., 2025) mainly considers 4-bit weight quantization with higher-precision activations. QUADS (Zhuge et al., 2026) mitigates train–rollout mismatch through asymmetric QAT and rollout-side activation compensation, whereas Rollout-ResQ (Mak et al., 2026) attributes the dominant degradation to rollout activation underflow and augments FP4 rollout with a sparse residual correction. QaRL (Gu et al., 2026) explicitly aligns the training-side forward computation with the quantized rollout by adopting matched low-precision kernel during training and rollout. However, even under matched low-precision execution, small differences between the training and rollout activations can still fall on opposite sides of an FP4 rounding boundary and be mapped to different codewords. In contrast, NTR records rollout-side FP4 quantization outcomes and uses them to directly guide the corresponding training-side rounding decisions, explicitly targeting this remaining rounding-outcome mismatch.

3 Method: Nearest-To-Rollout (NTR)

Figure 2 provides an overview of NTR. At a high level, NTR is an FP4 quantization framework for efficient RL training of MoE language models that directly aligns the quantized computation paths used for rollout generation and training. Specifically, during rollout generation, NTR records the quantization outcomes of FP4 routed experts's activation and FP4 KV states. During the subsequent quantization-aware training (QAT) phase, the recorded rollout-side information is used to guide the corresponding training-side rounding decisions. To reduce the overhead of rollout guidance, NTR further adopts an efficient quantization-information caching scheme. It retains only the mantissa and scale information from selected deeper layers to reduce storage and communication costs. In the following, we describe rollout-guided quantization-aware training and efficient quantization-information caching in detail. The pseudocode of NTR is provided in Section A.1.

3.1 Rollout-Guided Quantization-Aware Training

Motivation: Low-precision rollout introduces numerical discrepancy between the training and rollout execution paths. Simply combining FP4 quantization-aware training (QAT), where FP4 fake quantization is applied during the forward pass while the backward pass remains in BF16, with FP4 rollout generation can therefore lead to substantial policy divergence and degraded or even unstable RL training (Zhuge et al., 2026). For paired training- and rollout-side activations corresponding to the same trajectory, token position, and quantization site, we characterize their quantized discrepancy as

$$\mathcal{D}_{\text{act}} = \left\| Q^{\text{train}}_{\text{FP4}}(X_{\text{train}}) - Q^{\text{rollout}}_{\text{FP4}}(X_{\text{rollout}}) \right\|_F , \tag{1}$$

where X_{train} and X_{rollout} denote the corresponding activations produced during training and rollout generation, respectively, and $Q^{\text{train}}_{\text{FP4}}$ and $Q^{\text{rollout}}_{\text{FP4}}$ denote the FP4 quantization procedures on the two execution paths. We use $\mathcal{D}_{\text{act}}$ to characterize local train–rollout discrepancy.

Existing methods such as QUADS (Zhuge et al., 2026) and Rollout-ResQ (Mak et al., 2026) mitigate this discrepancy indirectly by improving the quantization accuracy of each execution path. Specifically, they primarily reduce the quantization error with respect to the corresponding high-precision representation, e.g., $\mathcal{L}_{\text{error}}(X) = \|X - Q(X)\|_F$. However, minimizing the quantization errors of the training and rollout paths independently does not directly minimize their cross-path discrepancy.

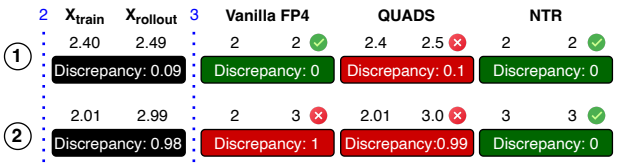
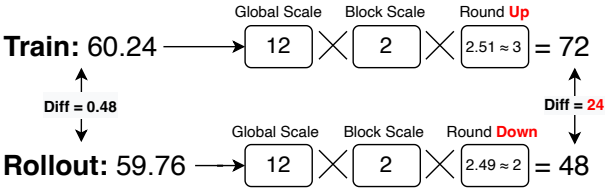
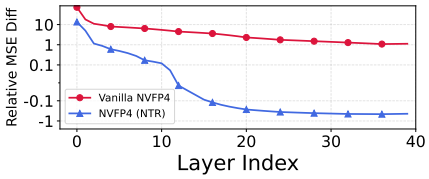


Figure 3: Two Examples of Train–Rollout Discrepancy under Different FP4 RL Methods.

To illustrate this issue, we provide two examples of QUADS in Figure 3. QUADS retains the BF16 training activation and compensates for the FP4 rollout quantization error using a residual $R(X) = X - Q(X)$. In example ❶, the training and rollout activations are 2.40 and 2.49, respectively. Vanilla FP4 maps both values to 2, yielding zero quantized discrepancy. In contrast, QUADS retains 2.40 on the training side and reconstructs the rollout activation as 2.50, reducing the per-path quantization error but increasing the train–rollout discrepancy to 0.10. In example ❷, QUADS similarly reduces the rollout-side quantization error but leaves a discrepancy of 0.99, close to the discrepancy of 1.0 under vanilla FP4. These examples demonstrate that improving per-path quantization accuracy does not necessarily reduce train–rollout discrepancy.



(a) Discrepancy Amplification by Inconsistent FP4 Rounding



(b) Relative Excess Discrepancy

Figure 4: Problem Analysis of FP4 RL Methods on Qwen3.5-35B-A3B.

Key Design: The key idea of NTR is to reduce the additional train–rollout discrepancy introduced by inconsistent FP4 rounding. A key source of this discrepancy is that small differences between the BF16 training and rollout activations can be substantially amplified when they are rounded to different FP4 codewords. As illustrated in Figure 4(a), the original training and rollout activations, 60.24 and 59.76, differ by only 0.48. After applying the same global and block scales, however, their normalized values, 2.51 and 2.49, fall on opposite sides of the rounding boundary and are rounded to 3 and 2, respectively. Consequently, the discrepancy is amplified from 0.48 in BF16 to 24 after FP4 quantization. We further quantify this quantization-induced discrepancy in Figure 4(b) using the relative excess train–rollout MSE, $(D_{\mathrm{FP4}} - D_{\mathrm{BF16}})/D_{\mathrm{BF16}}$, where D_{FP4} and D_{BF16} denote the train–rollout activation MSE under FP4 and BF16 computation, respectively. As shown, vanilla NVFP4 introduces substantial additional discrepancy across model layers.

Based on this observation, NTR uses the rollout-side quantization outcome to guide training-side rounding. For each captured training-side activation, let $\{q_-, q_+\}$ denote its two neighboring normalized FP4 E2M1 codewords under the rollout-side scale, and let q_{rollout} denote the exact FP4 codeword produced by the corresponding rollout-side activation. Instead of applying standard round-to-nearest (RTN), NTR selects

$$q_{\mathrm{NTR}} = \arg \min_{q \in \{q_-, q_+\}} |q - q_{\mathrm{rollout}}| . \tag{2}$$

Let $q_{\mathrm{RTN}} \in \{q_-, q_+\}$ denote the codeword selected by standard RTN under the same scale and candidate set. By construction,

$$|q_{\mathrm{NTR}} - q_{\mathrm{rollout}}| \leq |q_{\mathrm{RTN}} - q_{\mathrm{rollout}}| . \tag{3}$$

Therefore, relative to RTN, NTR does not increase the local quantized train–rollout discrepancy at each captured site under the same candidate set.

This local guarantee does not imply monotonic reduction of the full-network discrepancy in Equation (1) or of policy-level divergence after downstream nonlinear transformations. Rather, NTR reduces unnecessary amplification caused by inconsistent FP4 rounding. As illustrated in Figure 3, rollout-guided rounding eliminates the quantized discrepancy in both examples, while Figure 4(b) shows substantially reduced train–rollout discrepancy across model layers. The formulation above assumes access to the exact rollout-side FP4 codeword. In Section 3.2, we introduce a compressed caching scheme that approximates this guidance using only partial rollout-side code information.

3.2 MANTISSA-ONLY TRAIN-ROLLOUT COMMUNICATION

Motivation: Although rollout-guided quantization-aware training effectively reduces the train-rollout discrepancy, it requires preserving rollout-side quantization information and transferring it to the training engine, which introduces additional storage and communication overhead. During rollout generation, the quantization information must be transferred from GPU memory to host memory and stored until the corresponding trajectories are consumed by training. Before the QAT phase, the cached information is loaded and transferred back to GPUs for individual microbatches. As a result, directly preserving the complete rollout-side quantization information introduces additional GPU-to-CPU transfer, host-memory storage, disk I/O, and CPU-to-GPU transfer overhead. Therefore, reducing the amount of information communicated between rollout and training is critical for making rollout-guided QAT efficient in practice.

Key Design: The key idea of NTR is to communicate only the rollout-side quantization information necessary to determine the desired training-side rounding direction. As shown in Figure 5, for over 99% of the mismatched quantized values across layers, the training and rollout results differ by only one adjacent FP4 codebook entry. Because the two quantized values are overwhelmingly adjacent in the FP4 codebook, the training-side activation together with the rollout scale strongly constrains the candidate codewords. Consequently, a small amount of mantissa code information is typically sufficient to identify the desired neighboring rollout-side codeword. Therefore, communicating the complete quantized value is unnecessary. We further observe that rounding corrections toward lower FP4 codewords occur predominantly in deeper layers, indicating that rollout-side quantization information from these layers is more important for train-rollout alignment. Based on these observations, NTR communicates only the mantissa and scale information from the latter half of the model layers. During rollout generation, this compact information is cached and transferred to the training engine, where it is used to reconstruct a compact rollout-side reference and guide training-side quantization. This mantissa-only train-rollout communication substantially reduces the storage and communication overhead while preserving the effectiveness of NTR.

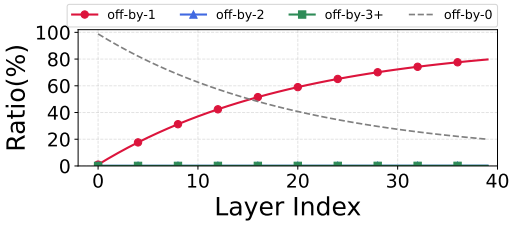


Figure 5: Difference in Mantissa Between Train and Rollout Activation on Qwen3.5-35B-A3B.

4 EXPERIMENT ANALYSIS

Baselines. We compare NTR against three groups of methods: (1) NVFP4 rollout methods, including vanilla NVFP4, NVFP4 with quantization-aware training (QAT), and QUADS (Zhuge et al., 2026) (Section 4.1). QaRL (Gu et al., 2026) and Rollout-ResQ (Mak et al., 2026) are also closely related to our setting, but we do not include direct numerical comparisons because their reported pipelines use different low-precision execution settings and we are not aware of publicly available end-to-end implementations that enable a controlled reproduction under our configurations. (2) MXFP4 rollout methods, including vanilla and QAT variants under W4A8 and W4A4 settings (Section 4.3); and (3) NVFP4 post-training quantization methods, including vanilla NVFP4, 4over6 (Cook et al., 2025), and H-Scale (Yu et al., 2026) (Section 4.4).

Models and Datasets. We evaluate NTR on four MoE language models: Qwen3.5-35B-A3B and Qwen3.5-122B-A10B (Qwen Team, 2026a), Qwen3.8-Flash-Next (Qiu et al., 2026) (125B total / 6B activated parameters), and Qwen3.8-2.4T-A95B (Qwen Team, 2026b). We consider three RL settings: reasoning, coding, and long-horizon tasks. For reasoning RL, we train Qwen3.5-35B-A3B and evaluate it on LiveCodeBench v6 (Jain et al., 2025), AIME 2024/2025 (Mathematical Association of America, 2024; 2025), and HMMT 2025 (Balunović et al., 2025). For coding RL, we train Qwen3.5-122B-A10B and Qwen3.8-Flash-Next, and evaluate them on DeepSWE v1.1 (Huang et al., 2026) and Terminal-Bench 2.1 (Merrill et al., 2026), respectively. For long-horizon RL, we train Qwen3.8-2.4T-A95B and evaluate it on GDPval (Patwardhan et al., 2025). We report sampled average pass@1 over 8 responses for AIME24, AIME25, and HMMT25, and over 10 responses for LiveCodeBench. For Terminal-Bench, DeepSWE, and GDPval, we report scores averaged over 10, 4, and 3 independent runs per task, respectively.

Table 1: Performance (↑) comparison of QAT, QUADS, and NTR under joint NVFP4 W/A/KV rollout setting on Qwen3.5-35B-A3B for reasoning RL tasks. The best performance is marked in bold. The performance gain compared to baselines is marked in green inside bracket.

		Qwen3.5-35B-A3B				
Rollout Config	Algo.	LiveCodeBench	AIME24	AIME25	HMMT25	Average
BF16 W/A + BF16 KV	Vanilla	**67.1**	83.8	**81.3**	67.5	74.9
NVFP4 W/A + NVFP4 KV	QAT	55.1	70.7	63.4	49.2	59.6
	QUADS	60.4	80.5	75.4	59.0	68.8
	NTR	66.4 (↑6.0)	**86.3** (↑5.8)	80.5 (↑5.1)	**70.0** (↑11.0)	**75.8** (↑7.0)

Implementation Details. For fair comparison, all methods use the same versions of VeRL (Sheng et al., 2025), Megatron (Shoeybi et al., 2019), and SGLang (Zheng et al., 2024), where VeRL coordinates asynchronous RL training, Megatron serves as the training backend, and SGLang performs rollout generation. We adopt GRPO (Shao et al., 2024) for policy optimization and additionally use token-level importance ratios between the training and rollout policies (Zheng et al., 2025a), with the clipping upper bound set to 5. We set `max_version_diff`=3 to bound the policy staleness introduced by asynchronous training. For rollout-guided quantization, rollout-side quantization records are paired with the corresponding training-side activations by trajectory, token position, and quantization site. Since the training policy can be newer than the policy that generated the rollout, NTR does not eliminate the underlying activation difference caused by policy staleness; instead, it reduces the additional discrepancy introduced when such differences are amplified by inconsistent FP4 rounding. Unless otherwise specified, we quantize only the routed experts in MoE layers and the KV cache in softmax-attention layers during rollout, while keeping all other modules in BF16. For Qwen3.5-35B-A3B, Qwen3.5-122B-A10B, Qwen3.8-Flash-Next, and Qwen3.8-2.4T-A95B, each RL step samples 256×16, 128×16, 64×16, and 64×8 trajectories, corresponding to batch sizes of 4,096, 2,048, 1,024, and 512, respectively. We set the maximum response length to 32K tokens for reasoning RL and the maximum context length to 256K tokens for coding and long-horizon RL. Qwen3.5-35B-A3B and Qwen3.8-Flash-Next are trained for 400 RL steps, while Qwen3.5-122B-A10B and Qwen3.8-2.4T-A95B are trained for 50 steps.

4.1 Overall Comparison

We first evaluate the performance of NTR under joint FP4 weight and KV cache quantization during rollout from three aspects: (1) performance across different benchmarks on the reasoning RL task, (2) performance on larger-scale MoE models under coding and long-horizon RL tasks, and (3) performance evolution throughout RL training. We compare NTR with baselines under the same FP4 rollout configuration and use BF16 rollout as the high-precision reference.

Performance on Different Benchmarks. We first evaluate NTR on Qwen3.5-35B-A3B across four reasoning benchmarks. As shown in Table 1, NTR consistently outperforms QAT and QUADS across all four benchmarks. In particular, NTR improves the average score from 68.8 to 75.8. The improvement is especially significant on HMMT25, where the score increases from 59.0 to 70.0. More importantly, NTR achieves an average score of 75.8, matching the BF16 rollout of 74.9, demonstrating that NTR can effectively recover the performance degradation introduced by joint FP4 weight and KV cache rollout while preserving BF16-level RL performance.

Table 2: Performance (↑) on larger-scale MoE LLMs, including Qwen3.5-122B-A10B and Qwen3.8-Flash-Next (125B-A6B) are trained on coding RL tasks and evaluated on DeepSWE and Terminal-Bench, respectively. Qwen3.8-2.4T-A95B is trained on long-horizon RL task and evaluated on GDPval. The best score in each row is marked in bold.

Model	BF16	QAT	QUADS	NTR
Qwen3.5-122B-A10B	**33.4**	28.8	29.1	33.0 (↑3.9)
Qwen3.8-Flash-Next	68.8	60.4	66.4	**70.6** (↑4.2)
Qwen3.8-2.4T-A95B	**90.3**	88.7	85.9	90.2 (↑1.5)

Performance on Larger-Scale Models. To examine the generalizability of NTR to larger-scale MoE models, we further evaluate it on Qwen3.5-122B-A10B, Qwen3.8-Flash-Next, and Qwen3.8-2.4T-A95B under coding and long-horizon RL settings. As shown in Table 2, NTR consistently outperforms QAT across all three models. In particular, on Terminal-Bench trained with Qwen3.8-Flash-Next, NTR improves the score from 66.4 to 70.6. Moreover, NTR matches BF16 rollout performance across all three models, achieving 33.0 vs. 33.4 on DeepSWE, 70.6 vs. 68.8 on Terminal-Bench, and 90.2 vs.

90.3 on GDPval. These results demonstrate that the effectiveness of NTR generalizes across different model scales and RL tasks while preserving the performance of high-precision BF16 rollout.

Performance Evolution Over Training Steps. Lastly, we examine the performance evolution of Qwen3.5-35B-A3B throughout RL training. As shown in Figure 7, we have two observations. (1) QAT and QUADS exhibits increasingly large log-probability differences as training proceeds. Such discrepancy is accompanied by clear signs of training collapse: the mean reward starts to decrease substantially in the later training stage, the entropy loss approaches zero, and the test scores on all reasoning benchmarks degrades sharply. In comparison, NTR effectively maintains stable training dynamics at a level comparable to the BF16 rollout. It continuously improves the reward and test score throughout training. (2) More importantly, we observe an interesting adaptation behavior under FP4 rollout with NTR. As illustrated in Figure 6, the test score of NTR is initially lower than that of the BF16 rollout due to accuracy degradation introduced by quantization. However, this performance gap gradually diminishes as RL training proceeds. After 120 training steps, NTR catches up with the BF16 reference and subsequently remains comparable to the BF16 reference, with higher point estimates at several later steps. This result suggests that, once the train–rollout discrepancy is properly controlled, the policy can progressively adapt to low-precision rollout, recovering the initial quantization-induced performance degradation and eventually matching the BF16 rollout.

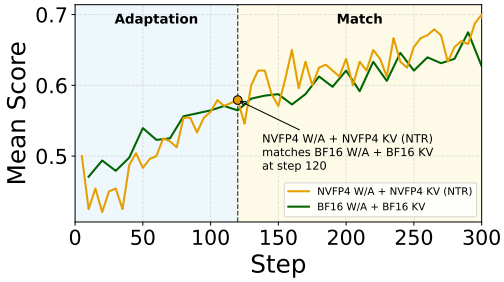
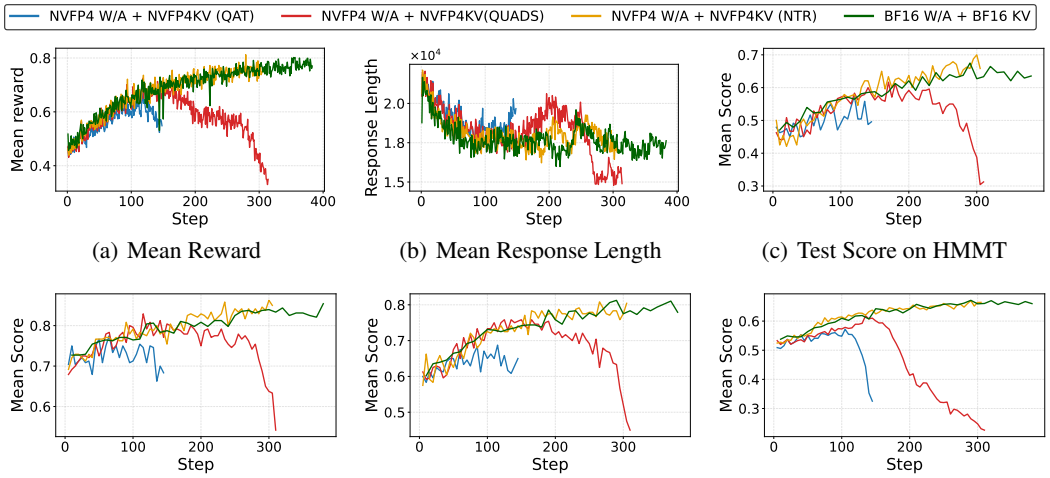


Figure 6: Adaptation for NVFP4 (W/A+KV) with NTR on HMMT25 during RL training.



(a) Mean Reward (b) Mean Response Length (c) Test Score on HMMT

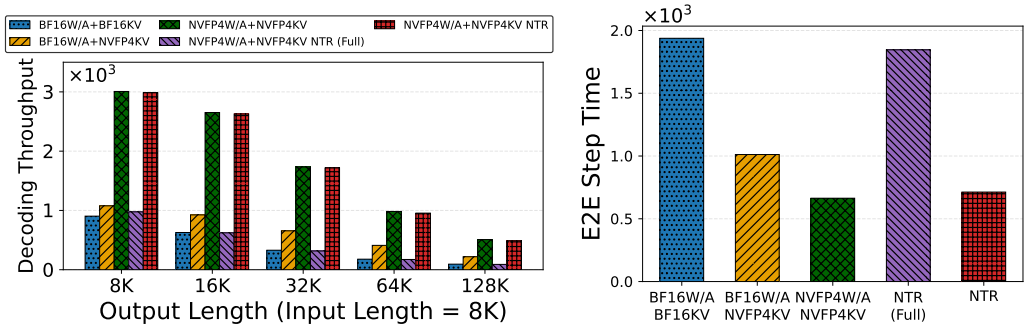
(d) Test Score on AIME24 (e) Test Score on AIME25 (f) Test Score on LCB.

Figure 7: Performance evolution, including maximum and minimum of log probability difference, mean reward, entropy loss, mean response length, and test scores, including HMMT, AIME24, AIME25, and LiveCodeBench of Qwen3.5-35B-A3B over training steps on reasoning RL tasks.

4.2 Efficiency Analysis

We next evaluate the efficiency of NTR on Qwen3.5-35B-A3B from two aspects: (1) rollout decoding throughput under different generation lengths, and (2) end-to-end RL training efficiency.

Rollout Efficiency. We first evaluate the decoding throughput of NTR under different output lengths. As shown in Figure 8(a), the full rollout-guided variant incurs substantial throughput degradation due to the overhead of storing and transferring complete rollout-side quantization information. In contrast, NTR maintains throughput close to vanilla FP4 rollout across different output lengths. For Qwen3.5-35B-A3B, retaining one-bit mantissa activation information for the latter 20 layers consumes $2048 \times 20 \times 1.5 = 61{,}440$ bits, or 7.5 KB, of cached quantization information per generated



(a) Decoding Throughput (b) E2E RL Training Step Time

Figure 8: Efficiency analysis of NTR on Qwen3.5-35B-A3B. Decoding throughput is measured using only the rollout engine (SGLang) on 4 GB200 GPUs. End-to-end RL training step time is measured on the reasoning RL tasks using 72 GB200 GPUs.

Table 3: Ablation of isolated NVFP4 weight and NVFP4 KV on Qwen3.5-35B-A3B under reasoning RL tasks. † marks results quoted from QUADS (Zhuge et al., 2026). The best performance is marked in bold. The performance gain compared to the QAT baseline is marked in green inside bracket.

ROLLOUT CONFIG	ALGO.	LIVECODEBENCH	AIME24	AIME25	HMMT25	AVERAGE
BF16 W/A + BF16 KV	Vanilla	67.1	**83.8**	81.3	67.5	74.9
NVFP4 W/A + BF16 KV	QAT†	63.5	81.7	78.8	60.4	71.1
	QUADS†	64.8	83.3	80.4	62.9	72.9
	NTR	67.1 (↑3.6)	83.1 (↑1.4)	**82.1** (↑3.3)	**69.4** (↑9.0)	**75.4** (↑4.3)
BF16 W/A + NVFP4 KV	QAT	65.3	82.3	79.2	65.8	73.2
	NTR	**68.1** (↑2.8)	83.0 (↑0.7)	80.7 (↑1.5)	67.3 (↑1.5)	74.8 (↑1.6)

token, which is 6 × smaller than caching full quantized activation. In particular, NTR achieves up to 5.4× higher decoding throughput at 128K output length compared with BF16 rollout. These results demonstrate that the proposed quantization-information caching preserves most of the decoding efficiency of FP4 rollout while enabling rollout-guided training.

End-to-End RL Training Efficiency. Lastly, we evaluate the end-to-end training overhead introduced by NTR. As shown in Figure 8(b), NTR increases the RL step time from 664 to 713, corresponding to only 7.4% overhead compared with vanilla joint FP4 rollout. To further understand the source of this overhead, Figure 9 reports the rollout-time breakdown of NTR. The model forward pass accounts for 86% of the rollout time, while weight-side and KV-side rollout guidance account for only 4% and 2%, respectively; the remaining 8% comes from KV dequantization. These results demonstrate that NTR introduces limited runtime overhead while retaining the efficiency advantage of FP4 rollout.

4.3 ABLATION STUDY

Isolated Weight and KV Quantization. We first examine the individual impact of FP4 weight and KV cache quantization. As shown in Table 3, in both FP4 weights and BF16 KV cache and F16 weights and FP4 KV cache settings, NTR consistently outperforms baselines methods and matches the performance of BF16 rollout. These results suggest that the joint FP4 configuration is substantially more challenging than either isolated setting, highlighting the importance of controlling the train–rollout discrepancy under joint FP4 rollout.

Performance Under Microscaling FP4 Formats. With recent adoption of MXFP4 MegaMoE kernels for efficient MoE inference, we further examine whether NTR generalizes to microscaling FP4 formats. As shown in Table 4, NTR improves the average score from 69.0 to 75.1 under W4A8 MXFP4 with FP4 KV cache, and from 67.2 to 73.5 under the more aggressive W4A4 MXFP4 setting. These results demonstrate that NTR remains effective across different FP4 formats.

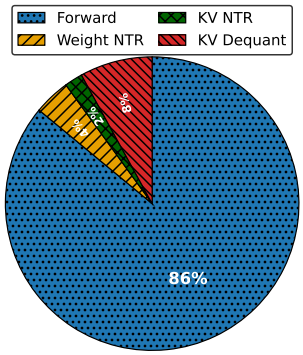


Figure 9: Rollout time breakdown for NTR.

Table 4: Performance (↑) comparison of stabilizing algorithms with different MXFP4 rollout configurations on Qwen3.5-35B-A3B under the reasoning RL task. The best performance is marked in bold. The performance gain compared to the QAT baseline is marked in green inside brackets.

ROLLOUT CONFIG	ALGO.	LIVECODEBENCH	AIME24	AIME25	HMMT25	AVERAGE
BF16 W/A + BF16 KV	Vanilla	**67.1**	**83.8**	**81.3**	67.5	74.9
W4A8 + MXFP4 KV	QAT	64.9	75.4	76.9	58.8	69.0
	NTR	**67.1** (↑2.2)	83.3 (↑7.9)	81.2 (↑4.3)	**68.7** (↑9.9)	**75.1** (↑6.1)
W4A4 + MXFP4 KV	QAT	60.2	74.8	73.2	60.7	67.2
	NTR	65.2 (↑5.0)	82.0 (↑7.2)	79.8 (↑6.6)	67.1 (↑6.4)	73.5 (↑6.3)

Table 5: Modular sensitivity study of NTR with FP4 MoE and FP4 KV on Qwen3.5-35B-A3B under reasoning RL tasks. `NTR (R-1bit-L20)` is the default config. The best performance is marked in bold. The performance gain compared to the QUADS baseline is marked in green inside brackets.

ROLLOUT CONFIG	ALGO.	LIVECODEBENCH	AIME24	AIME25	HMMT25	AVERAGE
BF16 W/A + BF16 KV	Vanilla	67.1	83.8	**81.3**	67.5	74.9
FP4 W/A + FP4 KV	QUADS	60.4	80.5	75.4	59.0	68.8
	NTR (R-4bit-L40)	67.0 (↑6.6)	86.1 (↑5.6)	80.1 (↑4.7)	**70.5** (↑11.5)	**75.9** (↑7.1)
	NTR (R-3bit-L40)	66.8 (↑6.4)	85.9 (↑5.4)	81.2 (↑5.8)	66.8 (↑7.8)	75.2 (↑6.4)
	NTR (R-2bit-L40)	**67.1** (↑6.7)	86.1 (↑5.6)	80.0 (↑4.6)	67.2 (↑8.2)	75.1 (↑6.3)
	NTR (R-1bit-L40)	66.4 (↑6.0)	**86.3** (↑5.8)	80.5 (↑5.1)	70.0 (↑11.0)	75.8 (↑7.0)
	NTR (R-1bit-L20)	66.4 (↑6.0)	**86.3** (↑5.8)	80.5 (↑5.1)	70.0 (↑11.0)	75.8 (↑7.0)
	NTR (R-1bit-L10)	66.1 (↑5.7)	82.7 (↑2.2)	**81.3** (↑5.9)	66.4 (↑7.4)	74.1 (↑5.3)
	NTR (R-1bit-L5)	65.8 (↑5.4)	81.2 (↑0.7)	79.7 (↑4.3)	66.9 (↑7.9)	73.4 (↑4.6)

Modular Sensitivity Study. Lastly, we evaluate the sensitivity of NTR to the amount of rollout-side quantization information retained for training. As shown in Table 5, retaining only one mantissa bit across all 40 layers achieves an average score of 75.8. Keeping this information for only the latter 20 layers incurs negligible degradation, while reducing the coverage to 10 and 5 layers lowers the average score to 74.1 and 73.4, respectively. These results show that NTR preserves strong performance with only compact mantissa information from a subset of deeper layers.

4.4 COMPARISON WITH POST-HOC FP4 QUANTIZATION

We finally compare the final FP4 policy performance obtained under two different training and quantization regimes. For the PTQ baselines, RL training is first completed with BF16 rollout, after which different PTQ algorithms are applied post hoc to the resulting policy. In contrast, NTR performs RL training directly with joint FP4 weight/activation and FP4 KV-cache rollout, allowing the policy to adapt to low-precision execution throughout training. As shown in Table 6, the post-hoc PTQ pipelines achieve average scores of 70.4, 71.0, and 71.4 with vanilla NVFP4, 4over6, and H-Scale, respectively, whereas NTR achieves 75.8 under FP4 rollout throughout RL training. This result demonstrates that incorporating low-precision execution into RL training with NTR can produce a strong final FP4 policy without relying on BF16 rollout followed by post-training quantization.

Table 6: Final FP4 policy performance (↑) under different RL training and quantization regimes on Qwen3.5-35B-A3B. For PTQ baselines, RL training is completed with BF16 rollout before applying post-hoc FP4 quantization. In contrast, NTR uses joint FP4 weight/activation and FP4 KV-cache rollout throughout RL training.

METHOD	LIVECODEBENCH	AIME24	AIME25	HMMT25	AVERAGE
BF16	67.1	83.8	81.3	67.5	74.9
Vanilla NVFP4	63.9	75.7	78.4	63.4	70.4
4over6	64.1	83.8	76.7	59.2	71.0
H-Scale	65.5	82.0	77.8	60.1	71.4
NTR	**66.4** (↑ 0.9)	**86.3** (↑ 2.5)	**80.5** (↑ 2.1)	**70.0** (↑ 6.6)	**75.8** (↑ 4.4)

5 CONCLUSION

In this paper, we presented NTR, an FP4 quantization framework for RL training of MoE language models. NTR combines rollout-guided QAT with efficient quantization-information caching to reduce train–rollout discrepancy and its associated overhead. Experiments across multiple MoE models and RL tasks show that NTR enables joint FP4 weight/activation and KV-cache rollout while preserving BF16-level performance and substantially improving rollout efficiency.

AI USE STATEMENT

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, supporting qualitative and thematic data analysis and interpreting results. We have not used generative AI tools for assisting in the writing of proofs and proposing or refining hypotheses, and generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, and cleaning and reformatting dataset are not applicable to this work. We have reviewed all AI-assisted work. For example, we checked LLM-generated research ideas for potential plagiarism through a manual literature survey and the LLM-generated code was verified for correctness by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. In *NeurIPS Datasets and Benchmarks Track*, 2025.

Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, and Song Han. Four Over Six: More accurate NVFP4 quantization with adaptive block scaling. *CoRR*, abs/2512.02010, 2025.

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. *CoRR*, abs/2606.00395, 2026.

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. In *ACL (Findings)*, pages 17051–17064. Association for Computational Linguistics, 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. *CoRR*, abs/2510.11696, 2025.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *CoRR*, abs/2607.07946, 2026.

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. In *ICLR*. OpenReview.net, 2025.

Yuhang Li, Reena Elangovan, Xin Dong, Priyadarshini Panda, and Brucek Khailany. QuRL: Efficient reinforcement learning with quantized rollout. *CoRR*, abs/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. *CoRR*, abs/2510.11370, 2025.

Hei Yi Mak, Shadan Golestan, Hoang Le, Mehran Taghian Jazi, Yunke Peng, Yaoyuan Wang, Yao Wang, Junsong Wang, Tianchi Hu, Fengchen He, Guipeng Hu, Tanzila Rahman, and Anandharaju Durai Raju. HiFloat4 format for end-to-end reinforcement learning post-training of large language models. *CoRR*, abs/2607.26515, 2026.

Mathematical Association of America. American invitational mathematics examination (AIME) 2024. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime, 2024.

Mathematical Association of America. American invitational mathematics examination (AIME) 2025. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime, 2025.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *CoRR*, abs/2601.11868, 2026.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks. *CoRR*, abs/2510.04374, 2025.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *CoRR*, abs/2608.30320, 2026.

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

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL `https://qwen.ai/blog?id=qwen3.8`.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR (Poster)*. OpenReview.net, 2017.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *EuroSys*, pages 1279–1297. ACM, 2025.

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

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. *CoRR*, abs/2506.06122, 2025.

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. *CoRR*, abs/2601.14243, 2026.

Jinghan Yao, Eric Tang, Sumanth Hegde, and Kourosh Hakhamaneshi. FP8 reinforcement learning in SkyRL: Preserving policy consistency across training and rollout. https://www.anyscale.com/blog/fp8-reinfinforcement-learning-in-skyrl, 2026. Accessed: 2026-09-02.

Hao Yu, Zheng Li, Dayiheng Liu, and Jianwei Zhang. H-Scale: hessian-guided scale refinement for NVFP4 sub-byte LLM inference. *CoRR*, abs/2608.28113, 2026.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, Yu Fu, Xingtai Lv, Yuchen Zhang, Sihang Zeng, Shang Qu, Haozhan Li, Shijie Wang, Yuru Wang, Xinwei Long, Fangfu Liu, Xiang Xu, Jiaze Ma, Xuekai Zhu, Ermo Hua, Yihao Liu, Zonglin Li, Huayu Chen, Xiaoye Qu, Yafu Li, Weize Chen, Zhenzhao Yuan, Junqi Gao, Dong Li, Zhiyuan Ma, Ganqu Cui, Zhiyuan Liu, Biqing Qi, Ning Ding, and Bowen Zhou. A survey of reinforcement learning for large reasoning models. *CoRR*, abs/2509.08827, 2025.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023.

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. *CoRR*, abs/2512.01374, 2025a.

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. *CoRR*, abs/2507.18071, 2025b.

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

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

A APPENDIX.

A.1 PSEUDOCODE OF NTR

Algorithm 1 summarizes NTR for quantized MoE inputs and KV states. The mantissa-only setting uses $b = 1$ in the latter half of the layers. Each record contains packed low-bit rollout codes and the corresponding rollout scales, indexed by trajectory, token, and site. During training, these records are used to construct a compact rollout-side reference for guiding the local FP4 rounding decision.

Algorithm 1 NTR: Rollout-Guided FP4 QAT

Require: Policy parameters W, prompts $\mathcal{D}$, reward function r, captured sites $\mathcal{S}$, bits $b = 1$
Ensure: Updated policy parameters W

for each RL iteration **do**
  Generate trajectories $\mathcal{T}$ and log-probabilities ℓ_{roll} from $\mathcal{D}$ using $Q_{\text{FP4}}(W)$
  During rollout, record $\mathcal{R} = \{(\text{Pack}(\text{low}_b(q_{\text{roll}})), s_{\text{roll}})\}$ at $\mathcal{S}$
  Compute advantages A from rewards $r(\mathcal{T})$
  for each policy-update minibatch $\mathcal{B} \subseteq \mathcal{T}$ **do**
    for each captured activation x encountered during the QAT forward **do**
      $(c, s) \leftarrow \text{UNPACK}(\mathcal{R}[\text{trajectory}, \text{token}, \text{site}])$
      $z \leftarrow \text{clip}(x/s, -6, 6)$ ▷ Normalize with the rollout scale
      $\widehat{q}_{\text{roll}} \leftarrow \text{RECONSTRUCT}(c, b, z)$
      $(q_-, q_+) \leftarrow \text{BRACKETFP4}(z)$
      $q^* \leftarrow \arg\min_{q \in \{q_-, q_+\}} |q - \widehat{q}_{\text{roll}}|$
      Use $\text{STE}(x, sq^*)$ in the remaining forward computation
    end for
    Compute $\mathcal{L}_{\text{RL}}$ from the QAT log-probabilities, ℓ_{roll}, and A on $\mathcal{B}$
    $W \leftarrow \text{OPTIMIZERSTEP}(W, \nabla_W \mathcal{L}_{\text{RL}})$
  end for
end for
return W

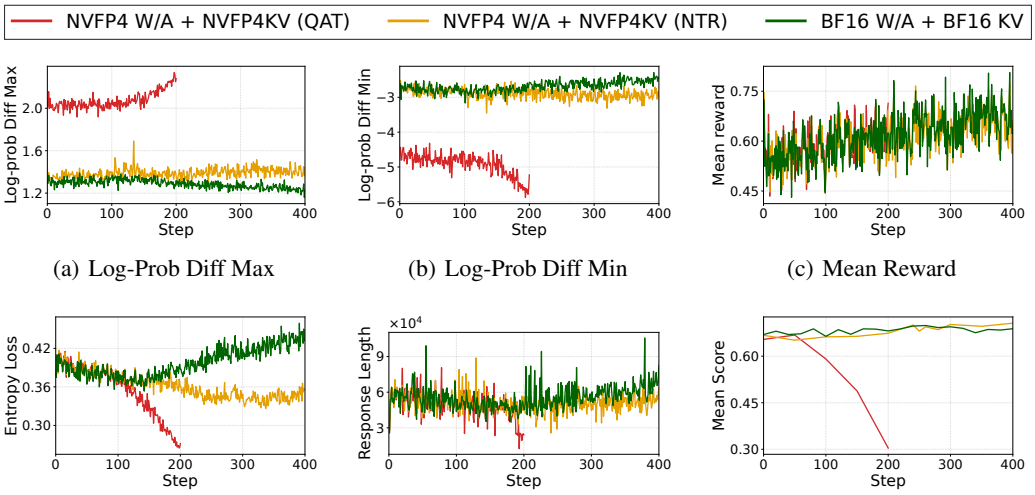
Here s combines block and token/global scales. RECONSTRUCT constructs a compact rollout-side reference by selecting the E2M1 magnitude nearest to $|z|$ whose low b code bits match c, while using the sign of the current training activation. Accordingly, the reconstructed value is used as a local guidance target rather than as an exact recovery of the original rollout-side FP4 codeword. BRACKETFP4 returns the adjacent E2M1 rounding candidates around z, and STE returns its second argument in the forward pass while passing gradients to its first argument.

A.2 TRAINING DYNAMICS ON QWEN3.8-FLASH-NEXT

Figure 10 complements the main-text analysis of coding RL on Qwen3.8-Flash-Next. We compare vanilla QAT and NTR under joint NVFP4 weight and KV-cache quantization with a BF16 rollout reference. The six panels show the same training diagnostics as in the main text, together with test scores on Terminal-Bench 2.1 (TB), over a window of 400 training steps.

Training–Rollout Consistency. In the recorded training steps, vanilla QAT exhibits larger log-probability differences than both NTR and the BF16 reference. Its maximum difference increases and its minimum becomes more negative toward the end of the recorded run. In contrast, NTR remains much closer to the BF16 reference in both diagnostics, with no comparable widening over the 400-step window. This behavior is consistent with the improved training–rollout alignment observed on reasoning RL tasks.

Reward, Entropy, and Response Length. Vanilla QAT nevertheless exhibits a pronounced entropy decrease after approximately step 100 and shorter responses near the end of its recorded trajectory. NTR shows a more gradual entropy decrease, maintains long responses, and improves reward on average. Its recorded TB test scores also increase over the training window. These observations support the training-stability trend in the main text.



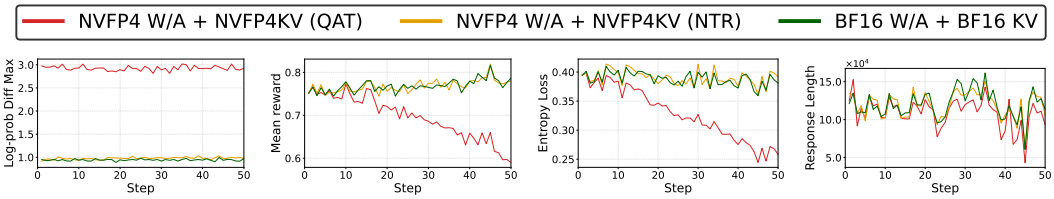
(a) Log-Prob Diff Max (b) Log-Prob Diff Min (c) Mean Reward

(d) Training Entropy Loss (e) Mean Response Length (f) Test Score on TB

Figure 10: Performance evolution of Qwen3.8-Flash-Next on coding RL tasks: maximum and minimum log-probability differences, mean reward, training entropy loss, mean response length, and test score on Terminal-Bench 2.1.

A.3 Training Dynamics on Qwen3.8-2.4T-A95B

Figure 11 complements the main-text comparison of long-horizon RL on Qwen3.8-2.4T-A95B. We compare vanilla QAT and NTR under joint NVFP4 weight and KV-cache quantization with a BF16 rollout reference. The four panels report the maximum log-probability difference, mean reward, training entropy loss, and mean response length over the 50-step training run.



(a) Log-Prob Diff Max (b) Mean Reward (c) Entropy Loss (d) Response Length

Figure 11: Training dynamics for Qwen3.8-2.4T-A95B (Max): (a) maximum log-probability difference, (b) mean reward, (c) training entropy loss, and (d) mean response length.

Training–Rollout Consistency. Over the 50 recorded steps, vanilla QAT keeps a substantially larger maximum log-probability difference, near 2.9, while NTR and the BF16 reference stay near 0.95. NTR remains close to the BF16 reference throughout the window. This behavior is consistent with the improved training–rollout alignment observed on reasoning and coding RL tasks.

Reward, Entropy, and Response Length. Vanilla QAT declines on the remaining diagnostics: mean reward falls from about 0.75 to about 0.59, training entropy falls from about 0.40 to about 0.26, and mean response length shortens from about 125K to about 93K tokens. NTR raises mean reward from about 0.75 to about 0.78, keeps entropy near the BF16 reference at about 0.38–0.40, and keeps responses in the 106K–134K range. These observations support the training-stability trend reported for long-horizon RL in the main text.

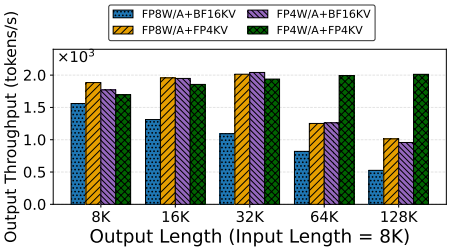


Figure 12: Qwen3.8-Flash-Next output throughput under four expert-weight/KV precision combinations. All settings use one GB200, BF16 PLE offload, 8K-token inputs, and 64 submitted requests. FP4 denotes NVFP4.

Table 7: Mean performance and standard deviation of NTR under joint NVFP4 W/A/KV rollout setting on Qwen3.5-35B-A3B for reasoning RL tasks with 5 different seeds.

ALGO.	LCB.	AIME24	AIME25	HMMT25
Vanilla BF16	**67.1**	83.8	**81.3**	67.5
NVFP4 W/A/KV (**NTR**)	66.4 ± 0.73	**86.3** ± 0.34	80.5 ± 0.38	**70.0** ± 0.51

Table 8: Performance with and without replaying routed experts (R3 (Ma et al., 2025)) from rollout sied to training side on Qwen3.5-35B-A3B for reasoning RL tasks.

ALGO.	LCB.	AIME24	AIME25	HMMT25	AVERAGE
With R3	66.4	86.3	80.5	70.0	75.8
Without R3	65.9	86.2	80.4	70.6	75.8

A.4 THROUGHPUT ANALYSIS ON QWEN3.8-FLASH-NEXT

Setup and Metric. Following Figure 8(a), we compare four routed-expert/KV precision combinations on Qwen3.8-Flash-Next: FP8/BF16, FP8/FP4, FP4/BF16, and FP4/FP4. All use the same checkpoint on one GB200, BF16 PLE/ngram embeddings offloaded to CPU pinned memory, and no multi-token prediction. The GPU memory fraction is 0.85, with automatic KV allocation. Each measurement submits 64 synthetic 8K-token prompts and generates exactly 8K–128K output tokens per request. We run three repetitions at 8K–128K with matched seeds and rotated configuration order.

Dependence on Output Length. In Figure 12, joint FP4 improves throughput over FP8/BF16 by $1.09\times$, $1.41\times$, $1.77\times$, and $2.43\times$ at 8K, 16K, 32K, and 64K outputs, respectively. In the single-run 128K supplement, joint FP4 reaches 2013 tokens/s versus 526 tokens/s for FP8/BF16 ($3.83\times$). The corresponding peak decoding batches are 64 and 7 requests, so this gain includes the concurrency enabled by KV capacity.

A.5 RESULTS ACROSS DIFFERENT RANDOM SEEDS

To examine the robustness of NTR to training randomness, we repeat the reasoning RL experiment on Qwen3.5-35B-A3B with five different random seeds. As shown in Table 7, NTR exhibits small variation across all four benchmarks, with standard deviations ranging from 0.34 to 0.73. The mean performance remains comparable to or higher than the BF16 rollout reference on AIME24 and HMMT25, while remaining close on LiveCodeBench and AIME25. These results demonstrate that the performance of NTR is stable across different random seeds.

A.6 EFFECT OF EXPERT-ROUTING REPLAY

Our main experiments replay the rollout-side routed experts during training following R3 (Ma et al., 2025), which reduces the routing mismatch between the rollout and training policies. To examine whether the effectiveness of NTR depends on such routing replay, we additionally evaluate NTR without R3 under the same joint NVFP4 W/A and KV rollout setting. As shown in Table 8, removing R3 results in nearly unchanged performance, with the average score remaining at 75.8. The differences on individual benchmarks are also small. These results indicate that the effectiveness of NTR does not rely on replaying rollout-side expert-routing decisions.