KV4-RL: UNLOCKING FASTER ASYNCHRONOUS ROLLOUTS FOR RLVR WITH INT4 KV CACHE QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Asynchronous rollout improves the efficiency of reinforcement learning with verifiable rewards by mitigating long-tail generation. However, sustaining higher concurrency compresses the same context-processing work into fewer decode steps, increasing the per-step cost of KV cache processing. We characterize this *KV concentration* and show that KV cache quantization provides substantially greater acceleration than weight-only or weight-activation quantization under the resulting workload. We further find that INT4 KV quantization can degrade learning despite importance correction. To address this, we propose *KV4-RL*, which combines calibration-free INT4 KV quantization with selective INT8 protection of attention sinks and recent context. This bounded protection reduces sampler mismatch while retaining efficient execution over the compressed cache. In Qwen3-8B-Base GRPO training, KV4-RL achieves $1.80\times$ rollout and $1.27\times$ full-iteration speedup over BF16 across training, with rollout speedup reaching $1.97\times$ as responses lengthen. Under BF16 evaluation, final-policy mean accuracy across six math benchmarks remains within 0.5 percentage points of BF16-rollout training for both Qwen3-4B-Base and Qwen3-8B-Base.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become a standard approach to training reasoning models, with methods such as GRPO learning from multiple responses generated for each prompt (Shao et al., 2024; Yu et al., 2026; Zheng et al., 2025). Generating these *rollouts* is a major training bottleneck because reasoning responses can span thousands of tokens and vary substantially in length (Fu et al., 2025; Zhou et al., 2026). In synchronous rollout, shorter responses finish early and leave the engine decoding only a few long responses, producing a prolonged long-tail phase with low hardware utilization. Recent asynchronous and partial-rollout systems mitigate this inefficiency by carrying unfinished responses across policy updates and replenishing the engine with new requests (Kimi Team et al., 2025; Fu et al., 2025; Zhou et al., 2026; Lu et al., 2025). Their efficiency gain comes primarily from sustaining substantially higher decode concurrency.

We show that this higher concurrency fundamentally changes the workload seen by the generation engine. For a fixed set of responses, scheduling does not change the total number of generated tokens or the total context positions that attention must process. Asynchronous rollout instead processes this work in fewer decode steps by advancing more responses together. The result is a substantially heavier KV workload at each step, which we call *KV concentration*. We formalize this effect and show that response-length variability further amplifies it by keeping long responses resident in the active batch for more steps. In a Qwen3-4B-Base rollout trace under matched hardware and memory budgets, asynchronous rollout sustains $7.82\times$ higher mean resident KV volume than synchronous rollout and incurs $4.52\times$ longer decode steps.

This workload shift has an important implication for quantized rollout. Prior work has primarily accelerated rollout by quantizing model weights or activations (Liu et al., 2025; Huang et al., 2026; Li et al., 2026), with recent work also considering FP8 KV caches (Qiu et al., 2026). Under sustained high concurrency, however, weight traffic is amortized across a wider batch while attention continues to process the context of every active response. Our roofline analysis and kernel measurements

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.

show that KV traffic becomes the dominant execution burden and that KV quantization provides substantially larger decode acceleration than the tested weight-only and weight-activation alternatives. KV compression provides an additional systems benefit by increasing the amount of context that fits in memory, which can further raise sustainable concurrency when KV capacity limits request admission.

Aggressive KV quantization, however, introduces a learning challenge. We find that INT4 KV rollout can noticeably degrade the learned policy even when truncated importance sampling (TIS) is used to correct sampler mismatch, whereas INT8 KV rollout remains close to BF16 training. This creates a tension between execution efficiency and rollout fidelity. Improving quantization fidelity uniformly across the entire cache would also distribute additional metadata and reconstruction overhead across the same large context whose cost we seek to reduce.

We address this tension with **KV4-RL**, a practical INT4 KV cache design for asynchronous RL rollout. KV4-RL keeps the dominant cache on a simple calibration-free INT4 path while selectively retaining a small, fixed-size region in INT8. Guided by shared-prefix conditional KL, we protect attention-sink and recent positions, which sharply reduces sampler mismatch while bounding the higher-precision storage and computation per response. This simple protection recovers the learning improvement missed by unprotected KV4 while retaining most of its execution benefit. Across Qwen3-4B-Base and Qwen3-8B-Base GRPO training, KV4-RL preserves final-policy accuracy under BF16 evaluation while substantially accelerating rollout generation.

Our contributions are summarized as follows:

- We identify and characterize *KV concentration* as a structural consequence of asynchronous rollout. By concentrating context processing into fewer decode steps, asynchronous scheduling shifts the rollout workload toward KV cache traffic. We show that this makes KV cache quantization a particularly effective acceleration target compared with the tested weight-only and weight-activation alternatives.
- To our knowledge, we present the first study of sub-8-bit KV cache quantization for RLVR rollout. We identify a learning-stability problem with uniform INT4 KV quantization that persists despite TIS, and propose *KV4-RL*, which uses selective INT8 protection to reduce sampler mismatch while retaining most of the INT4 execution benefit.
- We validate KV4-RL in full-parameter GRPO training on Qwen3-4B-Base and Qwen3-8B-Base. Compared with BF16 rollout, KV4-RL preserves final-policy mean accuracy within 0.5 percentage points across six math benchmarks, while achieving up to $1.97\times$ rollout speedup and $1.80\times$ cumulative rollout acceleration on Qwen3-8B-Base.

2 PRELIMINARIES

RLVR and GRPO. Reinforcement learning with verifiable rewards (RLVR) optimizes an autoregressive policy π_θ using rewards obtained by verifying generated responses (Shao et al., 2024; Yu et al., 2026; Zheng et al., 2025). Group Relative Policy Optimization (GRPO) samples G responses per prompt and assigns each response y_i the group-normalized advantage $\hat{A}_i = (R_i - \bar{R})/(s_R + \delta)$, shared across its tokens, where $\bar{R}$ and s_R are the group reward mean and standard deviation. Let $h_{i,j}$ denote the prompt and prefix preceding token $y_{i,j}$. GRPO uses the PPO-style clipped surrogate (Schulman et al., 2017)

$$\mathcal{S}_{i,j}(\theta) = \min\Big\{ r_{i,j}\hat{A}_i,\ \mathrm{clip}(r_{i,j}, 1-\epsilon, 1+\epsilon)\hat{A}_i \Big\}, \qquad r_{i,j} = \frac{\pi_\theta(y_{i,j} \mid h_{i,j})}{\pi_{\mathrm{old}}(y_{i,j} \mid h_{i,j})}, \tag{1}$$

where π_{old} denotes the frozen pre-update policy used as the reference for the current optimization step. Following DAPO (Yu et al., 2026), we use token-level averaging over collected responses and further augment the objective with KL regularization toward the initial reference policy.

Asynchronous rollout. Each RLVR training iteration collects T prompt groups of G responses before updating the policy. In synchronous rollout, all TG responses must finish, causing the active batch to contract as shorter responses complete and resulting in a long-tail phase (Fu et al., 2025; Zhou et al., 2026). Asynchronous systems relax this synchronization to improve utilization (Fu et al.,

2025; Lu et al., 2025; Li et al., 2027). We use *asynchronous rollout* to refer specifically to quota-based partial collection with prefix carry (Kimi Team et al., 2025; Zhou et al., 2026): generation stops once T complete groups are collected, while unfinished responses retain their prefixes for subsequent iterations. After each policy update, their KV caches are rebuilt by re-prefilling the retained prefixes under the new policy before generation resumes. Thus, a single response may contain tokens generated under multiple policy versions. Fig. A1 in Appendix B.1 illustrates the procedure.

Quantization and rotations. Quantization represents tensors using low-bit codes and associated scale/offset parameters. In b-bit affine quantization, a value is approximated as $\hat{x} = \Delta c + \beta$ with $c \in \{0, \dots, 2^b - 1\}$, where $\Delta > 0$ is the scale and β is an offset. Group-wise quantization shares these parameters across a block of values, so outliers can enlarge the represented range and reduce the effective resolution for the remaining values. Hadamard rotations mitigate this sensitivity by spreading large coordinates across channels before quantization (Ashkboos et al., 2024). For a normalized Hadamard matrix H with $HH^\top = I$, jointly rotating queries and keys preserves the unquantized attention scores since $(QH)(KH)^\top = QK^\top$. Quantization is then applied in the rotated basis.

Quantized rollout and sampler mismatch. Quantized rollout accelerates generation by reducing the precision of model weights (W), activations (A), or the KV cache (Liu et al., 2025; Huang et al., 2026; Li et al., 2026; Liu et al., 2024; Hooper et al., 2024; Lin et al., 2025; Qiu et al., 2026), while the learner remains in BF16. We denote four-bit weights with 16-bit activations as W4A16 and a four-bit KV cache as KV4; KV-only quantization retains BF16 weights and activations. Even at identical policy parameters, reduced precision can perturb next-token probabilities (Yao et al., 2025; Li et al., 2026; Xi et al., 2026; Zhuge et al., 2026), introducing sampler mismatch that can compound with policy staleness in asynchronous rollout.

Let $q_{i,j}$ denote the rollout probability of token $y_{i,j}$ when generated. Token-level truncated importance sampling (TIS) (Yao et al., 2025; Li et al., 2026) accounts for this mismatch by reweighting the policy surrogate:

$$c_{i,j} = \min\left\{C,\ \frac{\pi_{\text{old}}(y_{i,j} \mid h_{i,j})}{q_{i,j}}\right\}, \quad \widehat{\mathcal{J}}_{\text{TIS}}(\theta) = \frac{1}{N}\sum_{i,j} c_{i,j}\, \mathcal{S}_{i,j}(\theta). \tag{2}$$

where $C \geq 1$ is the truncation threshold and N is the number of response. Thus, $r_{i,j}$ captures the policy update from π_{old} to π_θ, whereas $c_{i,j}$ accounts for the mismatch between the rollout sampler and π_{old}. For carried tokens, $q_{i,j}$ corresponds to the policy version that generated each token. Further discussion of related work is provided in Appendix A.

3 KV Concentration in Asynchronous Rollout

Asynchronous scheduling improves rollout efficiency by sustaining high concurrency, while also reshaping the decode workload seen by the generation engine. In this section, we show that it concentrates substantially more KV work into each decode step and examine how this shift changes the relative benefits of weight, activation, and KV cache quantization.

3.1 Asynchronous Rollout Concentrates the Decode Workload

As described in Section 2, asynchronous rollout sustains high concurrency through an oversubscribed request pool, quota-based collection, and prefix carry. For a fixed set of completed responses with generation lengths L_i, scheduling does not change the total $\sum_i L_i$ tokens that must be generated. Higher concurrency instead processes more tokens at each decode step, completing the same generation work in fewer sequential steps.

The same observation applies to attention over the generated contexts. A response with prompt length P and generated length L accesses $Q(P, L) = \sum_{g=0}^{L-1}(P + g) = PL + L(L-1)/2$ logical KV positions during decoding. Since this quantity depends on the response rather than its scheduling, the total KV work $\sum_i Q(P_i, L_i)$ is unchanged for a fixed response set. Processing this work in

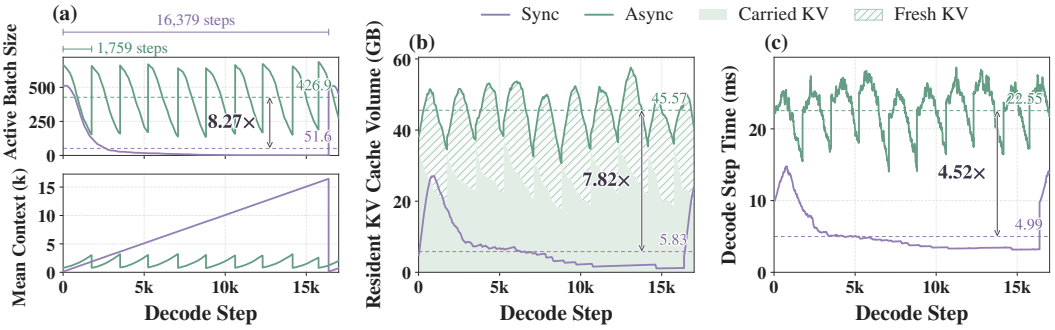

Figure 1: KV concentration at Qwen3-4B-Base checkpoint 300 on H200 (DP2×TP2) with matched KV budgets and concurrency limits. (a) Active batch and mean context. (b) Nominal resident KV volume per GPU. (c) Decode-step latency. Brackets mark collection intervals and dashed lines average both replicas over twelve rounds. Details appear in Appendix C.

fewer decode steps therefore increases the average KV work per step. We refer to this effect as *KV concentration*.[1]

In standard causal decoding, each active response attends to all cached positions in its retained prefix. The resulting per-step KV work is therefore proportional to the total cached context of the active responses. Let $\mathcal{R}_t$ denote the active responses at decode step t and $B_t = |\mathcal{R}_t|$. Response i has cached context length $\ell_i(t)$, including its prompt and generated prefix. The mean context length is $\bar{\ell}_t = B_t^{-1} \sum_{i \in \mathcal{R}_t} \ell_i(t)$, which gives the nominal resident KV volume $V_t = \kappa B_t \bar{\ell}_t$, where κ denotes the KV bytes per cached position for the given model, precision, and parallelism. For $s \in \{\text{sync}, \text{async}\}$, we define the long-run decode-step averages as $\overline{\ell}_s = \lim_{D\to\infty} D^{-1} \sum_{t<D} \bar{\ell}_t^s$ and $\overline{V}_s = \lim_{D\to\infty} D^{-1} \sum_{t<D} V_t^s$.

To isolate the effect of sustained concurrency, we consider *continuous refill*. Each completed response is replaced before the next decode step so that $B_t = B$, while unfinished prefixes persist across collection boundaries. The synchronous counterpart starts a wave of B responses together and waits for all of them to finish before starting another. Its wave duration is therefore $M = \max_{1\le i\le B} L_i$ decode steps.

Proposition 1 (Mean resident context under continuous refill)**.** *For fixed $P, B \in \mathbb{N}_{>0}$, let $L_i \overset{\text{i.i.d.}}{\sim} \mathcal{D}$ on $\mathbb{N}_{>0}$ with $\mu = \mathbb{E}[L]$ and $\sigma^2 = \mathrm{Var}(L) < \infty$. Under the schedules above, the long-run averages exist almost surely with $\overline{V}_{\text{async}} = \kappa B\, \overline{\ell}_{\text{async}}$ and satisfy*

$$\overline{\ell}_{\text{async}} = P + \frac{\mu - 1}{2} + \frac{\sigma^2}{2\mu}, \qquad \frac{\overline{V}_{\text{async}}}{\overline{V}_{\text{sync}}} = \frac{\mathbb{E}[M]}{\mu} \ge 1. \tag{3}$$

Proposition 1 formalizes this concentration under continuous refill. The ratio $\mathbb{E}[M]/\mu$ captures how synchronous execution spreads the same KV work across its long-tail steps, whereas asynchronous execution maintains a full batch. The mean resident context also contains an additional $\sigma^2/(2\mu)$ term. Longer responses remain active for more decode steps and are therefore overrepresented in the running batch, reflecting the classical inspection-paradox length bias (Ross, 1996). Thus, even at a fixed mean response length, greater response-length variability increases the KV burden sustained by asynchronous rollout. Appendix D provides the proof and relates the continuous-refill model to finite quota-based collection.

Fig. 1 shows the same effect in a Qwen3-4B-Base rollout trace. In panel (a), synchronous rollout gradually collapses into a long tail where only a few long responses remain active. Asynchronous rollout instead begins with higher concurrency and substantial carried context, then terminates collection before concurrency approaches zero. Across the observation window, its mean active batch is $8.27\times$ larger and each collection interval requires $8.35\times$ fewer decode calls per engine.

[1]This accounting follows complete responses across quota boundaries. Prefill and prefix reconstruction contribute additional work outside the decode-step accounting.

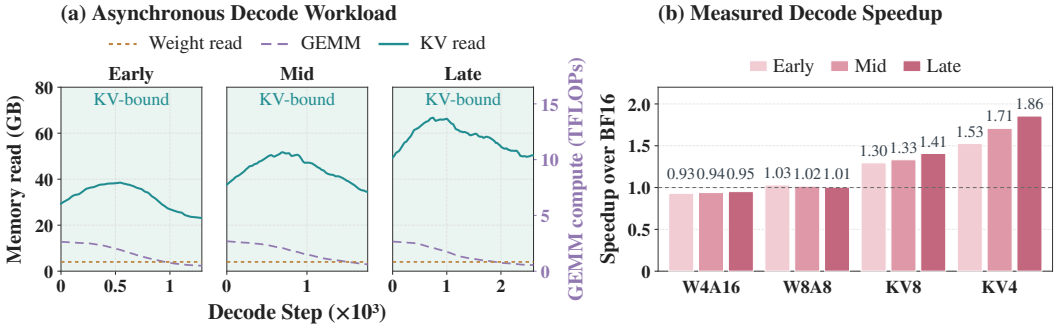

Figure 2: Asynchronous rollout workload and effect of different type of quantizations at Qwen3-4B-Base checkpoints 100/300/500 (Early/Mid/Late) on H200 with TP2. (a) Memory and compute demands with axes aligned by idealized service time. (b) Decode-forward speedup over BF16 at fixed workloads. Measurement details appear in Appendix C.

These scheduling differences translate directly into the resident KV volume in Fig. 1(b). Synchronous execution reaches its longest contexts only after most responses have completed, whereas asynchronous execution processes long contexts while maintaining a much wider active batch. Its mean resident KV volume is consequently $7.82\times$ larger. Panel (c) shows a corresponding $4.52\times$ increase in mean decode-step latency, which limits the throughput gain from higher concurrency to $1.83\times$. Further acceleration therefore requires reducing the cost of this concentrated per-step workload.

3.2 Implications for Quantized Rollout

Asynchronous scheduling and quantization act on complementary aspects of rollout generation. The former improves concurrency, while the latter reduces the cost of processing each decode step. However, the KV concentration identified in Section 3.1 changes which component of that step dominates. Previous quantized rollout methods have primarily targeted weights or activations (Liu et al., 2025; Huang et al., 2026; Li et al., 2026). Under sustained high concurrency, weight traffic is amortized across a wider batch, whereas attention continues to process the context of every active response. This makes the KV cache an increasingly attractive quantization target.

Fig. 2 (a) compares the resulting hardware demands at early, middle, and late checkpoints of the same Qwen3-4B-Base training run. We align the memory and compute axes using peak HBM bandwidth and BF16 compute throughput so that equal heights correspond to equal idealized service times. KV cache reads dominate the resulting service demand throughout generation. As training progresses, longer responses further increase KV traffic while weight traffic and GEMM work remain at similar levels across checkpoints. The imbalance therefore grows as the rollout workload evolves.

Fig. 2 (b) tests whether this cost structure translates into measured acceleration. We replay representative asynchronous decode workloads from each checkpoint under W4A16, W8A8, KV8, and KV4. All precision configurations process the same workloads, and the timings include quantization, dequantization, and tensor-parallel communication. Despite these costs, KV quantization consistently provides the largest gains. KV4 improves decode speed by $1.53\times$, $1.71\times$, and $1.86\times$ at the early, middle, and late checkpoints, respectively, while KV8 achieves 1.30–$1.41\times$. In contrast, W4A16 and W8A8 remain close to or below BF16 performance.

These fixed-workload measurements isolate the execution benefit of quantization. During end-to-end rollout, KV compression provides a second benefit by increasing the amount of context that fits within the same HBM budget. When response growth makes KV capacity the limiting factor for concurrency, this additional capacity allows the engine to sustain a larger active batch. KV quantization can therefore improve rollout throughput through both lower per-step cost and higher feasible concurrency.

4 Faster Asynchronous Rollout with INT4 KV Caches

Section 3.2 shows that KV cache quantization can substantially accelerate asynchronous decoding, but using aggressive low-bit KV caches during RL training introduces additional requirements in

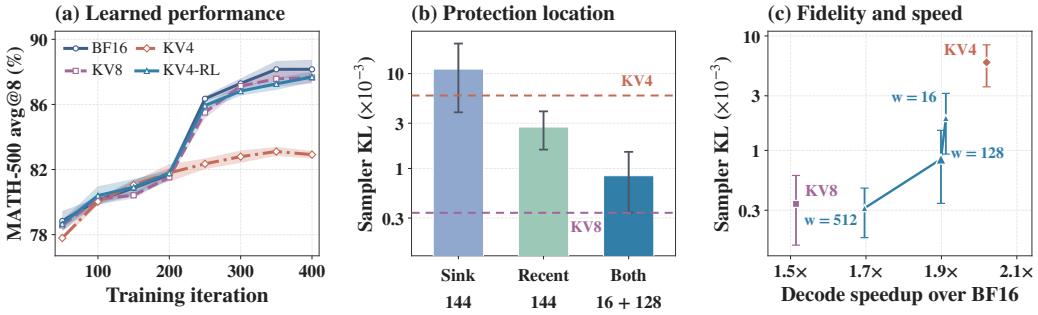

Figure 3: Selective INT8 protection. (a) 8B learning with BF16 evaluation. Shading shows standard deviation over three evaluation seeds. (b) Allocations of 144 protected positions with KV4 and KV8 reference lines. (c) Measured decode speedup and sampler KL with sink 16 and recent width w. Panels (b) and (c) use the Qwen3-4B-Base mid-training checkpoint at iteration 300. Error bars are 95% response-bootstrap intervals for KL.

both execution and learning. We address these with *KV4-RL*, which combines calibration-free INT4 quantization for efficient execution (Sec. 4.1) with selective INT8 protection to control sampler mismatch (Sec. 4.2).

4.1 INT4 KV Cache Quantization for Practical Acceleration

KV quantization for RL rollout must remain efficient under repeated policy updates. A quantizer calibrated to one policy may become stale as the policy and its KV statistics evolve, making repeated calibration undesirable in the training loop. At the same time, the resident KV cache is processed at every decode step, so even modest reconstruction overhead can offset the benefit of compression. We therefore use a calibration-free quantizer whose parameters are derived directly from each newly produced KV vector and whose reconstruction remains simple during attention.

Specifically, we apply affine INT4 quantization independently to each 128-dimensional token-head vector. The scale Δ and offset m are computed online and stored in FP16 alongside the packed four-bit codes. Sharing one pair of reconstruction parameters across the head keeps the attention computation simple. For a quantized key $\hat{k} = \Delta c + m$, the dot product can be written as $q^\top \hat{k} = \Delta(q^\top c) + m \sum_j q_j$, so the scale and offset corrections remain outside the channel reduction. Finer grouping improves quantization fidelity, but requires more metadata and additional separately scaled reductions throughout the cache. We therefore retain head-wise grouping, which requires 4.25 bits per element including metadata.

The resulting coarse grouping makes outlier control particularly important. Following rotation-based KV quantization (Jia et al., 2026), we apply a normalized Hadamard rotation to keys after RoPE and the same rotation to queries. Since the rotation is orthogonal, it preserves the unquantized attention scores while spreading large coordinates before quantization. The transformation is fixed and requires no calibration. Keys are stored directly in the rotated basis, while each query is rotated into the same basis before attention. In our storage-format probes, rotating K reduces mean conditional KL by over two orders of magnitude at both tested checkpoints, while additional V rotation provides no consistent benefit. We therefore rotate Q and K only, and refer to this calibration-free head-wise INT4 representation for all KV positions as KV4. Appendix E.3 reports the group-size, rotation, and FP4 ablations.

4.2 Reducing Sampler Mismatch with Selective INT8 Protection

Aggressive KV quantization introduces a second challenge beyond execution efficiency. Fig. 3 (a) compares Qwen3-8B-Base training with BF16, KV8, and KV4 rollout under the same asynchronous GRPO recipe with TIS. KV8 closely tracks BF16, whereas KV4 follows the early learning trajectory but fails to sustain its later improvement, ending 5.24 points below BF16 on MATH-500 avg@8. Since all checkpoints are evaluated with the same BF16 sampler, this gap reflects differences in the learned policies. Thus, TIS alone does not fully compensate for the sampler perturbation introduced by KV4.

To quantify this perturbation, we compare quantized and BF16 execution on the same policy checkpoint and prefixes. For BF16-generated prefixes $\mathcal{H}$, we define the shared-prefix conditional KL

$$\mathcal{D}_{\text{sampler}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} D_{\text{KL}}(p_{\text{BF16}}(\cdot \mid h) \,\|\, q_{\text{quant}}(\cdot \mid h)) \,, \tag{4}$$

with probabilities measured before temperature scaling or sampling truncation. Holding both the policy and prefixes fixed isolates the distributional change caused by quantized execution. The contrast between KV4 and KV8 suggests a practical target: reduce the mismatch of KV4 toward the KV8 regime while preserving its execution advantage.

Improving fidelity uniformly across the cache would undermine this goal. Smaller quantization groups require more reconstruction parameters and partial reductions, while calibration-based nonuniform schemes such as KVQuant (Hooper et al., 2024) add decoding or lookup overhead throughout the resident context. Since every decode step repeatedly processes this context, such refinements distribute their cost across all cached positions. We instead localize the additional precision to a bounded subset of positions and retain the simple INT4 path elsewhere. Motivated by the persistence of attention sinks (Xiao et al., 2024; Su & Yuan, 2025) and the importance of recent context in KV quantization (Liu et al., 2024; Duanmu et al., 2024), we store the first s positions and the most recent w positions in INT8, with all remaining positions in INT4. At most $s + w$ positions per sequence therefore use the higher-precision path.

Fig. 3 (b,c) shows that this simple allocation is surprisingly effective. With a fixed budget of 144 protected positions, splitting the budget between a 16-token sink and a 128-token recent window reduces sampler KL by $7.0\times$ relative to KV4 and outperforms assigning the full budget to either region alone. This configuration retains a $1.90\times$ decode speedup over BF16 while giving up only 6% of KV4 throughput. Increasing the recent window further approaches the KV8 mismatch level at additional cost. We use the 16/128 configuration as the default and refer to it as KV4-RL.

Most importantly, the reduction in sampler mismatch translates back to learning. KV4-RL recovers the later improvement missed by KV4 and reaches 87.68% MATH-500 avg@8, compared with 88.16% for BF16 rollout under the same TIS correction. Selective high-precision retention is not itself unique to quantized inference. Our finding is that, in quantized rollout, a small protected region can recover much of the learning loss caused by aggressive INT4 KV quantization while retaining most of its execution benefit. KV4-RL implements this design with separate INT4 and INT8 attention paths, restricting higher-precision storage and computation to the protected positions. Appendix E.4 reports additional learning results.

Kernel implementation. We implement KV4-RL in CUDA C++ with a paged INT4 cache and an INT8 side cache for the sink and recent window. The kernel widens packed 4-bit codes to 8-bit operands for integer Tensor Core operations, avoiding reconstruction of the full context in BF16. The INT4 path uses INT8 probability-value (PV) multiplication for efficiency, while the INT8 side path uses BF16 PV to improve fidelity by avoiding additional probability quantization. The two paths process disjoint context regions and merge their outputs using softmax normalization statistics. This keeps cache reads compressed while restricting the additional protection cost to a bounded number of positions. Implementation details are provided in Appendix B.3.

5 Experimental Results

Setup. We evaluate KV4-RL on Qwen3-4B-Base and Qwen3-8B-Base (Yang et al., 2025) using full-parameter GRPO on DAPO-Math-17k (Yu et al., 2026). Each run uses four H200 GPUs with DP2×TP2 rollout and a BF16 GRPO learner. We collect 128 groups of eight responses per iteration with a 16,384-token response cap. BF16, KV8, and KV4 serve as controls under the same asynchronous rollout and TIS recipe. Training runs for 600 iterations at 4B and 400 iterations at 8B.

All evaluations use BF16 sampling. We report avg@8 on MATH-500 (Lightman et al., 2024), OlympiadBench (He et al., 2024), and MinervaMath (Lewkowycz et al., 2022), and avg@32 on AIME 2024/2025 (Maxwell-Jia, 2024; yentinglin, 2025) and AMC 2023 (math-ai, 2025). Each endpoint is obtained from one training run and evaluated with three sampling seeds, so the reported standard deviations reflect evaluation variability rather than variation across training runs. Appendix C.1 and Appendix C.2 provide the full settings, with per-task pass@k results in Appendix E.4.

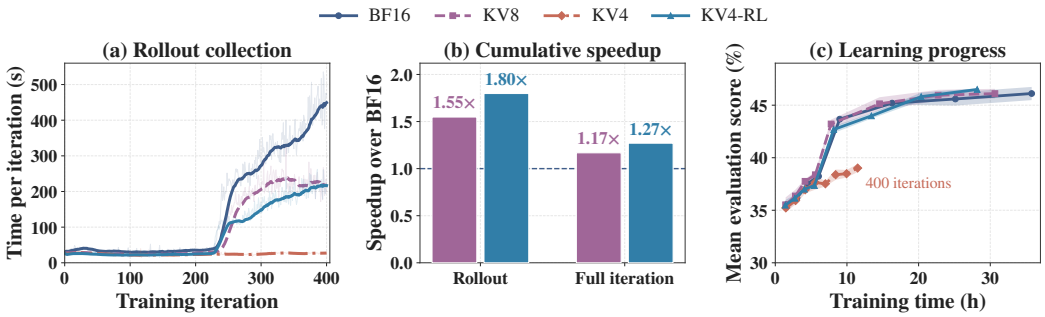

Figure 4: Acceleration on Qwen3-8B-Base. (a) Rollout time with 20-iteration means over faint raw traces. (b) BF16-to-method ratios of cumulative times. (c) Mean MATH-500 and AIME 2024/2025 accuracy versus cumulative iteration time with evaluation standard deviations. All runs complete 400 iterations. KV4 is omitted from (b) because its responses remain substantially shorter. Timing excludes initialization, evaluation, and checkpoint saving.

5.1 End-to-end Acceleration

As shown in Fig. 4, KV4-RL accelerates cumulative rollout by $1.80\times$ and full iterations by $1.27\times$ over BF16 across 400 iterations of Qwen3-8B-Base GRPO training. It reaches comparable evaluation accuracy in less wall-clock time while maintaining mean response lengths within 1% of BF16. The smaller end-to-end gain reflects the cost breakdown of the BF16 baseline, where rollout accounts for approximately 47% of iteration time. The remaining learner computation is largely unchanged by KV4-RL and therefore limits the full-iteration speedup.

KV4-RL also accelerates Qwen3-4B-Base, with $1.57\times$ rollout and $1.17\times$ full-iteration speedup over the measured continuation interval. The smaller gain is consistent with its shorter generated responses. Over the final 50 iterations, BF16 and KV4-RL responses average approximately 2.0k tokens at 4B and 5.5k at 8B. Shorter contexts leave less KV-dependent work for compression to reduce relative to other execution and collection costs. The same trend appears within the 8B run, where rollout speedup increases from $1.45\times$ over the first 100 iterations to $1.97\times$ over the final 50 as responses grow longer. The benefit of KV compression therefore depends strongly on the workload reached during training, which a measurement confined to the initial checkpoint would understate. Additional results and iteration-time breakdowns appear in Appendix E.

5.2 Performance and Learning Stability

Table 1 shows that KV4-RL remains within 0.5 percentage points of BF16 in mean accuracy across all six benchmarks at both model sizes and closely matches KV8. Selective protection is particularly important at 8B, where unprotected KV4 falls approximately 4.6 points below BF16 in mean accuracy, compared with only 0.8 points at 4B. This larger degradation coincides with the longer-context regime that also provides the greater acceleration opportunity. KV4-RL closes this gap while retaining most of the execution benefit of INT4 KV quantization.

Fig. 5 provides additional context for this difference. KV4-RL follows BF16 through the transition to longer responses and higher training rewards, while its reference-KL penalty settles at a similar level. In contrast, KV4 continues to improve its reward but does not exhibit the same later response-length growth, and its reference-KL penalty continues to rise. These trajectories indicate that its shorter runtime reflects a different learned policy and a reduced generation workload rather than execution acceleration alone. KV4-RL instead maintains learning and response-length trajectories close to BF16, allowing its timing gains to be compared under a substantially more similar workload.

6 Discussion and Limitations

The KV-concentration mechanism studied in this work originates from scheduling, while its magnitude depends on the model architecture. A rollout engine that sustains a wide active batch while carrying long prefixes across collection boundaries must retain the state of those active responses and repeatedly process the KV positions required by its attention pattern. We therefore expect the mechanism to

Table 1: Final Qwen3-Base benchmark accuracy (%) under BF16 evaluation at iteration 600 for 4B and 400 for 8B. Subscripts give standard deviations over three evaluation seeds. Mean averages the six tasks equally.

		avg@8			avg@32			
Model	**Rollout**	**MATH-500**	**OlympiadBench**	**MinervaMath**	**AIME24**	**AIME25**	**AMC23**	**Mean**
Qwen3-8B-Base	BF16	$88.2_{\pm 0.6}$	$56.3_{\pm 0.2}$	$51.9_{\pm 0.4}$	$29.5_{\pm 1.9}$	$20.7_{\pm 0.4}$	$67.9_{\pm 1.2}$	$52.4_{\pm 0.5}$
	KV8	$87.6_{\pm 0.4}$	$56.6_{\pm 0.4}$	$52.0_{\pm 0.6}$	$29.4_{\pm 0.6}$	$21.3_{\pm 0.5}$	$69.9_{\pm 1.1}$	$52.8_{\pm 0.4}$
	KV4	$82.9_{\pm 0.2}$	$50.1_{\pm 0.2}$	$51.2_{\pm 0.8}$	$18.5_{\pm 0.7}$	$15.6_{\pm 0.3}$	$68.4_{\pm 1.4}$	$47.8_{\pm 0.1}$
	KV4-RL	$87.7_{\pm 0.3}$	$56.8_{\pm 0.4}$	$51.8_{\pm 1.4}$	$29.7_{\pm 0.6}$	$22.1_{\pm 0.7}$	$68.9_{\pm 1.1}$	$52.8_{\pm 0.1}$
Qwen3-4B-Base	BF16	$83.0_{\pm 0.2}$	$50.1_{\pm 0.2}$	$48.0_{\pm 0.7}$	$19.1_{\pm 0.2}$	$16.5_{\pm 0.4}$	$65.2_{\pm 1.0}$	$47.0_{\pm 0.1}$
	KV8	$82.5_{\pm 0.1}$	$49.2_{\pm 0.5}$	$46.9_{\pm 0.5}$	$18.2_{\pm 0.6}$	$16.8_{\pm 1.0}$	$66.2_{\pm 0.3}$	$46.6_{\pm 0.3}$
	KV4	$81.7_{\pm 0.5}$	$49.1_{\pm 0.5}$	$48.0_{\pm 0.7}$	$16.8_{\pm 0.4}$	$14.1_{\pm 0.4}$	$67.6_{\pm 1.3}$	$46.2_{\pm 0.0}$
	KV4-RL	$82.4_{\pm 0.3}$	$48.8_{\pm 0.0}$	$49.1_{\pm 0.7}$	$18.2_{\pm 0.9}$	$13.6_{\pm 0.1}$	$67.5_{\pm 0.7}$	$46.6_{\pm 0.3}$

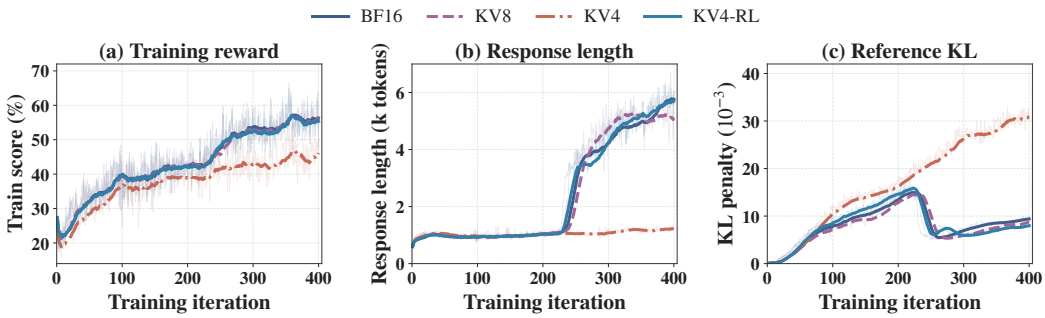

Figure 5: Training trajectories on Qwen3-8B-Base. (a) Mean correctness reward. (b) Mean completed-response length. (c) Logged reference-KL penalty against the initial policy. Thick curves show 20-iteration means with faint raw traces.

remain relevant beyond our colocated dense-model setting, although the point at which KV processing becomes dominant can shift substantially. In MoE models, sparse expert computation reduces the FFN cost without directly removing KV traffic, which may make cache processing relatively more important. Local and hybrid attention alter the picture more directly by restricting KV access in some or all layers, reducing the growth of KV traffic with context length. Determining these crossover points for MoE, hybrid-attention, and disaggregated rollout systems remains an important direction for future study.

The end-to-end results in Section 5.1 also expose a broader limitation of rollout-only acceleration. As rollout becomes faster, learner forward and backward computation occupies a larger fraction of the RLVR iteration and eventually limits further end-to-end gains. Further acceleration may therefore require complementary reductions in learner-side computation, including lower-precision training. Our experiments are additionally limited to the response-length horizon and staleness bounds considered here. Longer training horizons and more aggressive asynchronous schedules may alter both the KV workload and the interaction between sampler quantization and policy staleness, which we leave for future work.

7 Conclusion

In this work, we characterized *KV concentration* as a structural consequence of asynchronous rollout and showed that the resulting workload makes KV cache compression an effective target for further acceleration. Building on this observation, we proposed KV4-RL, which combines calibration-free INT4 KV quantization with selective INT8 protection to reduce sampler mismatch while preserving efficient execution. Across Qwen3-4B-Base and Qwen3-8B-Base GRPO training, KV4-RL substantially accelerates rollout while maintaining final-policy performance close to BF16 rollout. These results highlight the importance of jointly considering rollout scheduling, workload structure, and numerical precision when optimizing RLVR training.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research methodology and experimental design, implementation, translation, qualitative analysis, and interpretation of experimental results. We did not use generative AI tools to formulate mathematical claims, develop theoretical models or conceptual frameworks, provide essential steps for mathematical proofs, write proofs, generate synthetic datasets, or propose or refine research hypotheses. Dataset cleaning and reformatting are not applicable to this work.

We additionally used generative AI tools for brainstorming, information search, identifying relevant literature, creating or editing software code, preparing or revising scientific figures, and improving the presentation and wording of the manuscript. All AI-assisted outputs were reviewed by the authors. AI-assisted code was inspected, tested, and validated for correctness, and AI-assisted technical content was checked against the corresponding experiments, derivations, and source material. The authors take full responsibility for the final content of this work, including all text, claims, code, figures, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 4 describes KV4-RL and its selective-protection design. Appendix B provides implementation details for asynchronous collection, cache management, and the paged INT4/INT8 attention kernels. Appendix C documents the models, training configuration, evaluation protocol, workload traces, replay methodology, timing boundaries, and checkpoint-specific measurement procedures. Appendix D gives the complete proof and assumptions for the KV-concentration analysis, and Appendix E reports additional ablations and model-specific results. Source code and scripts are included in the supplementary material.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Roberto Castro, Andrei Panferov, Rush Tabesh, Oliver Sieberling, Jiale Chen, Mahdi Nikdan, Saleh Ashkboos, and Dan Alistarh. Quartet: Native FP4 training can be optimal for large language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 43552–43572. Curran Associates, Inc., 2025. doi: 10.52202/085713-1449. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3e11872d3bc9bfba34e1752fef87639c-Abstract-Conference.html.

Brian Chmiel, Maxim Fishman, Ron Banner, and Daniel Soudry. FP4 all the way: Fully quantized training of large language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 91130–91151. Curran Associates, Inc., 2025. doi: 10.52202/085713-3049. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8340b085045cf13f1f0b6c2c4cc0a89c-Abstract-Conference.html.

Dayou Du, Shijie Cao, Jianyi Cheng, Luo Mai, Ting Cao, and Mao Yang. BitDecoding: Unlocking tensor cores for long-context LLMs decoding with low-bit KV cache. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2026. arXiv:2503.18773.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. SKVQ: Sliding-window key and value cache quantization for large language models. In *Conference on Language Modeling (COLM)*, 2024.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A

challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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. In *International Conference on Learning Representations (ICLR)*, 2026.

Jinda Jia, Jisen Li, Zhongzhu Zhou, Jung Hwan Heo, Jue Wang, Tri Dao, Shuaiwen Leon Song, Ben Athiwaratkun, Chenfeng Xu, Tianyi Zhang, and Xiaoxia Wu. SAW-INT4: System-aware 4-bit KV-cache quantization for real-world LLM serving, 2026. URL https://arxiv.org/abs/2604.19157.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling Reinforcement Learning with LLMs, 2025. URL https://arxiv.org/abs/2501.12599v4.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 3843–3857. Curran Associates, Inc., 2022. doi: 10.52202/068431-0278. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/18abbeef8cfe9203fdf9053c9c4fe191-Abstract-Conference.html.

Li et al. QuRL: Low-precision reinforcement learning for efficient reasoning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2602.13953.

Haoyang Li, Sheng Lin, Fangcheng Fu, Yuming Zhou, Xiaodong Ji, Yanfeng Zhao, Lefeng Wang, Jie Jiang, and Bin Cui. StaleFlow: Staleness-aware data management for mitigating data skewness

in fully disaggregated RL post-training. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2027. arXiv:2601.12784.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

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

Tengxuan Liu et al. PM-KVQ: Progressive mixed-precision KV cache quantization for long-CoT LLMs. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.18610.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning (ICML)*, 2024.

Han Lu, Zichen Liu, Shaopan Xiong, Yancheng He, Wei Gao, Yanan Wu, Weixun Wang, Jiashun Liu, Yang Li, Haizhou Zhao, Ju Huang, Siran Yang, Xiaoyang Li, Yijia Luo, Zihe Liu, Ling Pan, Junchi Yan, Wei Wang, Wenbo Su, Jiamang Wang, Lin Qu, and Bo Zheng. Part II: ROLL flash – accelerating RLVR and agentic training with asynchrony, 2025.

math-ai. AMC 2023 Evaluation Dataset. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/math-ai/amc23. 40-question evaluation set from the 2023 American Mathematics Competitions.

Maxwell-Jia. AIME 2024 Dataset. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024. 30 problems from the 2024 American Invitational Mathematics Examination I and II.

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, 2026.

Sheldon M. Ross. *Stochastic Processes*. John Wiley & Sons, 2 edition, 1996. ISBN 9780471120629.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

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

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 *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Zunhai Su and Kehong Yuan. KVSink: Understanding and enhancing the preservation of attention sinks in KV cache quantization for LLMs. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=gIqb6zWZoO.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

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

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

yentinglin. AIME 2025. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/yentinglin/aime_2025. 30 problems from the 2025 American Invitational Mathematics Examination I and II.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

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, 2025. URL https://arxiv.org/abs/2507.18071.

Yuzhen Zhou, Jiajun Li, et al. APRIL: Active partial rollouts in reinforcement learning to tame long-tail generation. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.18521.

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, 2026.

A Related Work

Asynchronous and partial rollout. RL training systems improve accelerator utilization through flexible placement and execution of generation and policy optimization (Sheng et al., 2025; Fu et al., 2025; Lu et al., 2025). Asynchronous collection coordinates these operations while managing policy staleness (Fu et al., 2025; Li et al., 2027), and partial rollout allows unfinished responses to continue across updates (Kimi Team et al., 2025; Zhou et al., 2026). These mechanisms reduce waiting caused by long-tail generation. Our work examines the decode workload they sustain. We identify KV concentration as a consequence of processing accumulated contexts at higher concurrency and connect this workload to the benefits of KV-cache quantization. The scheduling mechanisms provide the execution setting for this analysis.

Quantized reinforcement learning. Prior work quantizes weights alone or together with activations to accelerate rollout with either low-rank adaptation or full-parameter policy updates (Huang et al., 2026; Liu et al., 2025; Li et al., 2026). Reduced precision also creates rollout–training mismatch, motivating importance correction and changes to policy optimization (Yao et al., 2025; Liu et al., 2025; Li et al., 2026). Other approaches align numerical execution across generation and training or compensate quantization error on both sides (Xi et al., 2026; Zhuge et al., 2026). FP8-RL additionally quantizes the KV cache within a broader low-precision RL stack (Qiu et al., 2026). Our focus is the execution benefit and learning behavior of sub-8-bit KV caches in asynchronous rollout. We observe degradation under uniform INT4 and use selective INT8 protection to reduce sampler mismatch while retaining acceleration. Training experiments evaluate the resulting policy quality under a common BF16 evaluator.

KV-cache compression and selective precision. Inference methods reduce KV storage through quantizers adapted to key and value distributions (Liu et al., 2024; Hooper et al., 2024) and rotations that reduce sensitivity to outliers (Ashkboos et al., 2024; Jia et al., 2026). Efficient attention implementations couple these representations with low-bit execution to realize practical speedups (Lin et al., 2025; Du et al., 2026; Jia et al., 2026). Selective preservation complements compression by retaining influential initial positions (Xiao et al., 2024; Su & Yuan, 2025) or recent context at higher precision (Liu et al., 2024; Duanmu et al., 2024). Mixed-precision policies have also been studied for long reasoning traces (Liu et al., 2026). KV4-RL builds on these encoding and protection principles in a setting where quantized samples drive repeated policy updates. Its evaluation therefore connects cache precision to both rollout efficiency and the quality of the learned policy, alongside shared-prefix measurements of sampler mismatch.

B Implementation Details

All experiments use an in-house RLVR framework with a tensor-parallel rollout engine and an FSDP2 learner. The implementation incorporates the core inference and training optimizations used in vLLM and verl (Kwon et al., 2023; Sheng et al., 2025), including direct reuse of vLLM's communication primitives and FlashInfer attention (Ye et al., 2025). These optimizations are enabled in the BF16 baseline as well as the quantized configurations. Precision changes are therefore evaluated within a common optimized execution stack.

Why a separate framework. Combining quota-based partial rollout with our paged INT4 attention requires coordinated control over response lifetime and cache execution. Prefixes carried across a policy update must be resumed with priority and used to rebuild both the packed KV history and its protected side cache. The custom cache representation also affects memory-budget accounting and attention dispatch within captured model forwards. Supporting this combination through verl and vLLM would require changes across both the collection layer and the engine's cache interfaces. We implement these components together to make their interaction explicit while retaining the serving and learning optimizations detailed below.

B.1 Collection and Policy Updates

Fig. A1 illustrates how collection proceeds across policy updates. Synchronous collection finishes every response assigned to the current iteration. Its request queue eventually empties and the remaining

long responses continue with progressively fewer companions. Partial collection instead maintains an oversubscribed pool and stops after enough complete prompt groups have become available. The collector selects the oldest T completed groups for training and retains any additional completed groups for later selection. Unfinished responses preserve their token prefixes and resume with priority when the next collection begins.

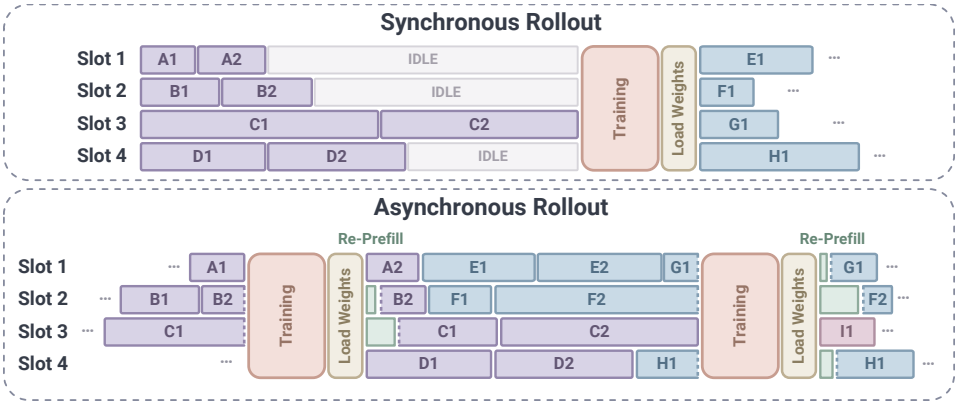

Figure A1: Synchronous collection and quota-based partial rollout. Blocks show generation work assigned to a finite number of execution slots. Synchronous collection waits through long-tail generation before updating the policy. Partial collection ends at a completion quota and carries unfinished prefixes across the update. Re-prefill reconstructs their KV caches before generation resumes. Slot placement is schematic and does not imply a fixed physical slot for a carried response.

In the evaluated configuration, the global quota is $T = 128$ groups with $G = 8$ responses per group. An oversubscription factor of $\beta = 1$ maintains up to $T(1 + \beta)$ groups across the rollout replicas. A group that reaches the age limit $\eta = 12$ must complete before collection can stop. This requirement can extend collection beyond the completion quota. Here asynchronous rollout refers to the lifetime of responses across updates. Generation and optimization alternate on the same four GPUs.

Each response retains the behavior log-probability recorded when each token was sampled. These probabilities remain attached to the trajectory when it crosses an update or waits in the completed-group pool. The learner evaluates the current pre-update BF16 policy on the collected tokens and forms the TIS correction in Eq. equation 2. The rollout configuration uses temperature one without top-k or nucleus truncation, so the recorded softmax probabilities correspond to its sampling distribution. After optimization, the updated weights are copied into the rollout engine. Re-prefill then rebuilds the carried prefixes under those weights while preserving the sampled tokens and their historical behavior probabilities.

B.2 Common Execution Optimizations

Scheduling and attention. The rollout engine stores KV entries in 16-token pages and admits requests according to available pages and the concurrency limit. Continuous batching reuses slots released by completed responses while queued requests remain. Chunked prefill shares a token budget with decode requests, and active-row compaction keeps completed requests out of subsequent model forwards. The BF16 baseline uses FlashInfer paged attention with its Tensor Core decode path (Ye et al., 2025). These choices provide the paged storage and dynamic batching mechanisms used in modern serving engines such as vLLM (Kwon et al., 2023).

Model execution and communication. The engine fuses QKV projections and the MLP gate/up projections. Compiled model execution reduces dispatch overhead and intermediate tensor allocations around attention. Multiple CUDA graphs cover different decode batch sizes, allowing the scheduler to select a captured forward as concurrency changes. Tensor-parallel communication reuses vLLM's custom all-reduce implementation with PyNCCL as a fallback. BF16 and quantized samplers share these scheduling and execution mechanisms. Precision-specific conversion work remains inside the timed forward when it is required during decoding.

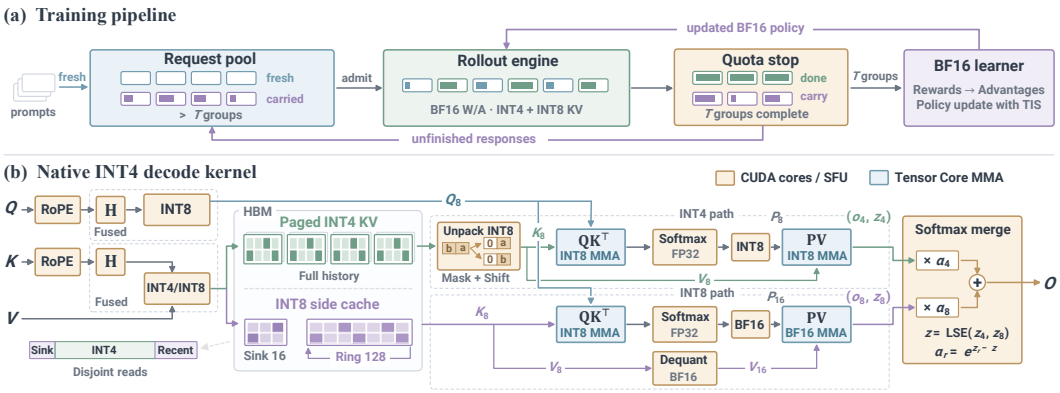

Figure A2: KV4-RL execution. (a) Completed prompt groups feed the BF16 learner while unfinished responses return to the request pool. (b) Paged INT4 storage and the INT8 side cache are read over disjoint context regions. Query preparation includes rotation and INT8 activation quantization. Packed INT4 codes are widened to eight-bit operands for integer MMA with INT32 accumulation. The protected path uses BF16 PV. CUDA-core operations and Tensor Core MMA have separate colors. INT4 in the kernel description denotes KV storage precision.

Learner execution. The learner updates all model parameters with FSDP2. BF16 forward and backward computation use FP32 master parameters and optimizer states. Padding is removed before variable-length attention, and token-budget microbatches balance the work of responses with different lengths. Fused projection and log-probability computation avoid retaining full-vocabulary logits for every training token. Activation checkpointing reduces the update's peak memory. The FSDP2 state-management and offloading utilities follow verl's implementation, while the rollout engine and phase controller support the combined partial-collection and cache-precision design.

Phase memory management. Before rollout, optimizer-only state and the frozen reference model are offloaded to make memory available for the KV pool. The updated learner parameters are streamed into the engine's tensor-parallel weight buffers. Before optimization, rollout allocations are released and the learner state is restored. Virtual-memory-backed allocations preserve addresses used by captured graphs across these transitions. The memory profiler reserves model and execution workspace before sizing the KV pool, including precision-dependent metadata and side-cache storage. Thus KV capacity is determined by the complete allocation rather than the packed payload alone.

B.3 Paged INT4 Attention with Selective Protection

Fig. A2 connects the training cycle to the cache representation and attention computation. The INT4 history provides compact storage for every context position. A separate INT8 cache preserves the sink and recent positions used by the protected attention path. Both representations are constructed online from the current policy's K/V tensors.

Cache encoding. Following the K-only rotation design of SAW-INT4 (Jia et al., 2026), a normalized Hadamard transform is applied to K and Q after RoPE. Their dot product is preserved before quantization. V stays in its original basis. Each 128-element group is encoded with an affine quantizer $x \approx \Delta c + m$, where $c \in \{0, \ldots, 15\}$ and both Δ and m are stored in FP16. The codes occupy half a byte per scalar and their metadata adds $4/128$ bytes per scalar. INT8 side entries use symmetric per-token/head scaling. They are computed from the source vectors before INT4 rounding, so protection does not attempt to recover information from four-bit codes.

Cache lifetime. The paged INT4 cache covers the full history. The side cache stores the first $s = 16$ positions and a ring buffer for the most recent $w = 128$ positions. A position leaving the ring already has its INT4 copy available and requires no historical requantization. The INT4 path skips the union of protected positions, including any overlap between the sink and recent regions. The default sink size aligns the beginning of the unprotected region with a 16-token page boundary in

this implementation. Side storage is reserved per request slot and remains bounded with respect to response length.

INT4 execution path. Query rotation and INT8 quantization are fused into query preparation. The attention kernel loads packed cache pages into shared memory and widens four-bit codes through mask and shift operations. Integer QK MMA accumulates into INT32. Quantization scales and affine offsets convert these accumulators into floating-point attention scores. Softmax is computed in FP32, after which the PV path forms an eight-bit probability operand with the required value-scale factors. A second integer MMA and affine correction produce the regional output. This path avoids materializing the entire KV history in BF16.

Protected execution and merging. The INT8 side path reuses the prepared query for integer QK. It dequantizes the protected values for BF16 PV with FP32 accumulation and avoids the probability quantization of the INT4 path. Protection therefore changes both the cache representation and the arithmetic applied to protected positions. Let o_r be the locally normalized output and z_r its log-normalizer for region $r \in \{4, 8\}$. The final output is

$$z = \log(e^{z_4} + e^{z_8}), \qquad o = e^{z_4 - z} o_4 + e^{z_8 - z} o_8. \tag{A1}$$

The implementation evaluates this expression with a stable log-sum-exp reduction. Empty regions receive zero weight. The same normalization identity combines context splits within each region. It is exact for the regional scores and outputs being merged, which already reflect their respective quantization paths.

Prefill and carried-prefix reconstruction use a separate BF16 attention path and populate the packed history and protected cache from the newly computed K/V. The compressed decode path then processes subsequent tokens. Consequently, a decode-forward measurement isolates a different portion of execution from the rollout time that also includes re-prefill.

C Experimental Setup and Measurement Protocols

Our experiments separate workload characterization from execution timing and learned-policy evaluation. Frozen-checkpoint traces expose the workload created by scheduling. Decode replay holds that workload fixed when comparing precision configurations. Training and common-sampler evaluation then measure the resulting rollout cost and policy quality. The protocols below specify the population and timing boundary used in each comparison.

C.1 Models and Training Configuration

We train Qwen3-4B-Base and Qwen3-8B-Base (Yang et al., 2025) on the English training split of `open-r1/DAPO-Math-17k-Processed` (Yu et al., 2026). Each problem is supplied without a chat template. We append the instruction "Let's think step by step and output the final answer within `\boxed{}`." Math-Verify assigns a correctness reward by comparing the extracted answer with the reference. The training scorer follows verl's boxed-answer verification procedure. We use full-parameter GRPO with group-normalized advantages and token-mean loss aggregation. Table A1 lists the shared recipe.

Policy optimization uses the clipped surrogate in Section 2 with reference-KL regularization. For negative advantages, dual clipping bounds the unclipped surrogate using coefficient 3.0. We retain the same optimization settings across cache-precision configurations. Training microbatches use an automatically profiled token budget with padding removed. Precision arms share the hardware and admission limit, while the allocator accounts for their different cache footprints when choosing the page count.

The workload and quantization-evolution results use one BF16-trained lineage per model. The 4B checkpoints are iterations 100/300/500 and the 8B checkpoints are iterations 100/250/400. Continued 8B runs retain the same training lineage. These checkpoint studies hold the policy fixed within each comparison and do not represent independent training repeats. The trace manifests record PyTorch 2.10.0 with CUDA 12.8. Kernel attribution for the measured precision configurations is given in Appendix C.4.

Table A1: Shared training configuration for the warmup-10 runs. Checkpoint iteration and training seed are specified with each comparison. The frozen trace and replay protocols below use explicit memory budgets instead of the training engine's automatic KV-pool sizing.

Setting	Value
Hardware per training run	4× NVIDIA H200 GPUs
Rollout parallelism	Two DP replicas with TP2 each
Learner	FSDP2 with BF16 compute and FP32 master weights
Prompt groups / responses per group	$T = 128$ / $G = 8$
Prompt / response token limits	1,024 / 16,384
Sampling temperature / top-p	1.0 / 1.0 with top-k disabled
Request-pool oversubscription / age limit	$\beta = 1$ / $\eta = 12$ iterations
Optimizer	AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$
Learning rate / warmup	10^{-6} / 10 updates then constant
Weight decay / gradient-norm limit	0.01 / 1.0
PPO epochs / global minibatch	One / all T prompt groups
Policy-ratio clipping / dual clip	$[0.8, 1.2]$ / 3.0
Reference-KL coefficient / estimator	0.001 / k_3
TIS correction	Token-level upper clip at $C = 5$
Gradient checkpointing / padding removal	Enabled / enabled
KV page size / maximum active requests	16 tokens / 1,024 per replica
KV4-RL protection	Sink 16 and recent 128 in INT8

Table A2: Evaluation tasks. Every task is included at the endpoint. OlympiadBench refers to the English text-only open-ended mathematics competition subset. MinervaMath is the OCWCourses university-level STEM set.

Task	Questions	k	Periodic	Dataset source
MATH-500	500	8	Yes	`HuggingFaceH4/MATH-500`
AIME 2024	30	32	Yes	`Maxwell-Jia/AIME_2024`
AIME 2025	30	32	Yes	`yentinglin/aime_2025`
AMC 2023	40	32	No	`math-ai/amc23`
OlympiadBench	674	8	No	`math-ai/olympiadbench`
MinervaMath	272	8	No	`math-ai/minervamath`

C.2 Policy Evaluation

We evaluate all trained policies with BF16 weights and BF16 KV caches in a common evaluator. This removes the test-time quantized sampler from the comparison of learned policies. MATH-500 (Lightman et al., 2024) is a 500-problem subset of MATH (Hendrycks et al., 2021). We also evaluate OlympiadBench (He et al., 2024) and MinervaMath (Lewkowycz et al., 2022). The competition sets comprise AIME 2024/2025 (Maxwell-Jia, 2024; yentinglin, 2025) and AMC 2023 (math-ai, 2025). Table A2 records the exact dataset distributions and sample counts.

Sampled evaluation uses temperature 0.6 and top-$p = 0.95$ with a 16,384-token response cap. Evaluation prompts allow up to 2,048 tokens and use the same boxed-answer instruction. Periodic evaluation covers MATH-500 and both AIME sets. Endpoint evaluation additionally covers the remaining three datasets.

Let c_i count correct answers among the k samples for question i. For n questions, we report $\text{avg@}k = (nk)^{-1} \sum_{i=1}^{n} c_i$ and $\text{pass@}k = n^{-1} \sum_{i=1}^{n} \mathbf{1}\{c_i > 0\}$. Thus avg@$k$ averages individual sampled answers and pass@k records whether at least one answer succeeds. An aggregate across the six tasks gives each task equal weight. Error bars across evaluation seeds use the sample standard deviation of per-seed scores at a fixed checkpoint. They measure sampling variation separately from variation across independently trained policies.

The evaluator uses Math-Verify for symbolic equivalence. OlympiadBench scoring preserves multiple-answer metadata and requires every answer. Numeric answers use the supplied absolute tolerance or 10^{-8} by default. MinervaMath numeric answers use relative tolerance 10^{-4} in the units requested by the question. These are the implemented grading protocols for the listed dataset distributions.

Standalone repeated evaluation uses TP1, whereas periodic evaluation uses the training run's BF16 evaluation layout. Comparisons retain the same evaluation protocol across precision arms. When periodic and standalone evaluations are displayed together, their sampling repetitions remain separate from training-seed repetitions.

The final checkpoints are iteration 600 for 4B and iteration 400 for 8B. Both comparisons use evaluation seeds 42/43/44. Each model uses the same evaluation sources across rollout configurations. The Mean column first averages all six tasks within each seed and then computes the mean and sample standard deviation across seeds.

C.3 Workload Traces

Both schedulers use four H200 GPUs with two TP2 replicas. The response quota and sampling settings match Table A1. The policy remains frozen, including across simulated update boundaries, so collection dynamics can be measured without concurrent changes in the learned policy. We pin the BF16 pool to 83,022 pages per GPU for 4B and 79,639 pages for 8B. Both models have $\kappa = 73{,}728$ nominal BF16 KV bytes per context position per GPU under TP2. These pools hold 1,328,352 and 1,274,224 positions respectively. The maximum active batch is 1,024 per replica for either scheduler.

We discard three collection rounds to form the carry pool before measurement. The 4B summaries use the following twelve rounds. The 8B summaries use the following six rounds at every checkpoint. The early 8B run contains twelve measured rounds, of which the first six form the matched comparison. Sampling uses seed 6101. Statistics combine pure-decode calls from both replicas and exclude prefill calls. Active batch and nominal resident volume are averaged over calls. Per-engine decode throughput is the total committed decode tokens divided by summed CUDA-event decode time. It therefore excludes time spent on prefill and collection outside these calls.

The resident-volume calculation is $V_t = \kappa \sum_{i \in \mathcal{R}_t} \ell_i(t)$ for the active sequences. It counts the nominal KV representation of their contexts and differs from total reserved VRAM or physically measured HBM traffic. The carried contribution is the context belonging to requests resumed from an earlier collection round. Fresh requests supply the remaining contribution. Slots may be reused or compacted without changing this classification.

Fig. 1 displays 17,000 decode calls from one 4B-middle replica. Its interval contains a complete synchronous round and the start of the next one. The first displayed synchronous and asynchronous collection intervals contain 16,379 and 1,759 calls respectively. Display smoothing averages at most 65 calls within a round and samples the resulting curve every eight calls. Batch and context sums are smoothed together so that their product remains the displayed resident volume. Means and reported ratios use unsmoothed calls across the full observation population. The displayed brackets therefore describe particular rounds rather than the average reduction in decode calls.

C.4 Roofline Comparison and Decode Replay

The workload comparison in Fig. 2(a) uses nominal BF16 weight bytes and linear-layer FLOPs alongside the traced KV volume. For Qwen3-4B with TP2, the linear projections and output head contribute $W = 4.022272$ GB of weights per GPU and $F = 4.022272$ GFLOPs per active token per GPU. We compare the demands $W/\mathcal{B}$, $FB_t/\mathcal{F}$ and $V_t/\mathcal{B}$ using peak bandwidth $\mathcal{B} = 4.8$ TB/s and dense BF16 throughput $\mathcal{F} = 989.5$ TFLOPs/s.[2] The aligned axes expose relative demands. They omit kernel inefficiency and communication, which are included in the subsequent execution measurements.

For each checkpoint, we divide its asynchronous trace into five equal-baseline-time phases and select a representative batch and context shape from each phase. Each shape uses uniform context length across its batch. Table A3 records the tested shapes. The comparison holds them fixed across precision configurations. It therefore measures execution speedup at fixed concurrency rather than the additional admission enabled by lower cache memory.

[2]H200 specifications: https://www.nvidia.com/en-us/data-center/h200/. The dense throughput is half the sparsity-enabled figure.

Table A3: Representative replay shapes (B, ℓ) in generation-phase order. Context lengths are in tokens. The five phases have equal weight in baseline decode time. The first 8B-late context was reduced from the sampled 3,463 to 3,461 tokens to fit the pinned page budget.

Checkpoint	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
4B–100	608/621	576/798	448/1,033	304/1,384	168/1,963
4B–300	640/874	576/1,170	464/1,506	336/1,968	200/2,627
4B–500	640/1,463	544/1,895	400/2,568	272/3,518	176/4,767
8B–100	624/547	592/676	496/879	352/1,134	192/1,529
8B–250	656/1,863	384/3,271	320/3,908	272/4,435	208/5,305
8B–400	368/3,461	256/4,712	224/5,529	200/6,196	176/6,972

Kernel configurations. W4A16 uses vLLM 0.18.0's Machete kernel at the tested batch sizes. FP8 W8A8 uses our schedule-tuned adaptation of vLLM's CUTLASS FP8 GEMM. A separate W8A8 configuration uses rotated INT8 activations. Integer KV8 and KV4 use our CUDA paged-attention kernels with BF16 linear layers. We also measure FlashInfer FA2 attention over an FP8 KV cache within the same engine. This last configuration tests a different cache format and attention path, rather than a separate vLLM end-to-end system. The integer KV4 path enables the 8-bit PV operand described in Appendix B.3.

Timing and aggregation. After warmup, we collect 90 CUDA-event samples per state in three within-process blocks. Timing covers the captured model forward with TP communication and the captured argmax operation. Conversion work required by the selected numerical path is included. Scheduler work and stochastic sampling outside the graph are excluded together with prefill and learning. If $\tau_{q,j}$ denotes mean latency for configuration q at state j, the aggregate is

$$S_q = \left(\sum_{j=1}^{5} \omega_j \frac{\tau_{q,j}}{\tau_{\mathrm{BF16},j}} \right)^{-1}, \qquad \omega_j = \frac{1}{5}. \tag{A2}$$

This is a baseline-time-weighted harmonic mean of state speedups. Replacing state means with medians changes every 4B aggregate by less than $0.006\times$ and every 8B aggregate by less than $0.002\times$. Repeated BF16 middle-checkpoint runs agree within 0.3% after aggregation. These repeats assess timing consistency within the replay protocol.

C.5 Shared-Prefix Fidelity and Protection Timing

For each fidelity checkpoint, we generate one BF16 response to each of 32 fixed prompts using temperature one. Eight positions span each response at generated-prefix indices $\mathrm{round}(j(L-1)/7)$ for $j = 0, \ldots, 7$. The resulting corpus has 256 states per checkpoint. We force every sampler along these same token histories and capture full-vocabulary logits at the selected positions. Both the BF16 reference and the quantized configurations run in the same TP2 engine. CPU analysis applies float64 log-softmax to the recorded logits before temperature scaling or sampling truncation and computes $D_{\mathrm{KL}}(p_{\mathrm{BF16}} \| q)$.

The reported mean gives each state equal weight. Confidence intervals use 10,000 bootstrap samples of the 32 response paths, retaining all eight positions of a selected path together. An independent BF16 process repeat at 4B–300 produces bit-identical reference logits and zero KL. This control measures the repeatability of the comparison within the same engine. The conditional KL remains a sampler diagnostic, while training measures the effect on learned performance.

Protection timing uses heterogeneous contexts to retain the variation encountered by the attention kernel. At 4B–300, we reconstruct 24 batch-context vectors from the midpoint of each measured round across both replicas. They contain 407–475 active responses with mean contexts of approximately 1.4k–1.7k tokens. At 8B, we construct 11 heterogeneous vectors from the mean and quantiles recorded in the iteration-250 traces. These are synthetic vectors because preemption prevents exact reconstruction from contiguous residency records. They contain 260–374 responses with mean contexts of approximately 3.3k–4.7k tokens. The 8B protection kernels and fidelity probe use the

Table A4: Timing scopes. Quantization can change both forward cost and the amount of work completed in a collection round. These measurements therefore answer different performance questions.

Measurement	Included work
Decode replay	Captured forward and TP communication at fixed states
Decode throughput	Committed tokens per summed GPU decode-service time
Rollout collection	Collection including prefill/re-prefill and reward scoring
Training iteration	Collection and learning with weight synchronization and phase switches

iteration-200 checkpoint. Thus its workload shapes and policy parameters come from different recorded iterations.

Each protection arm replays the same vectors as its BF16 reference with ten warmup calls and 90 timed calls per state. We use the median latency for each state and aggregate as $S_q^{\text{ragged}} = \sum_j \tau_{\text{BF16},j} / \sum_j \tau_{q,j}$. This weights states by their BF16 replay duration. These heterogeneous-state measurements have a different state population and aggregation from Eq. equation A2. Comparisons of protection overhead use the uniform-KV4 reference measured in this same heterogeneous-state suite.

C.6 Rollout and Training Time

GPU decode timing isolates execution once a batch is ready. Rollout collection time additionally includes fresh and carried-prefix prefill along with scheduling and output collection. In the training logs, the rollout timer also encloses reward evaluation performed by the collector. The iteration timer starts before switching to the rollout engine and ends after optimization. It includes parameter synchronization and both phase transitions. Logged training timers take the maximum across ranks. Evaluation and checkpoint saving occur outside that timer. Table A4 summarizes these boundaries.

Section 5 sums the iteration and rollout timers along the same checkpoint lineage used for policy evaluation. The 8B comparison contains exactly 400 updates per configuration. If a run restarts from a saved checkpoint, only the branch leading to the evaluated policy contributes to the total. Initialization and restart downtime are excluded together with evaluation and saving. These totals measure cumulative execution of the retained training iterations.

The reported speedup divides the sum of BF16 times by the corresponding sum for each sampler. It is not the mean of per-iteration ratios. The 8B full-run comparisons use iterations 1–400. Its early interval contains iterations 1–100 and the final evaluation interval contains 351–400. Response-length statistics use the same populations. Actual policies generate their own responses, so these end-to-end comparisons complement the fixed-workload replay rather than replacing it.

The iteration breakdown groups the logged rollout and policy-update timers with the sum of policy and reference log-probability timers. The remaining difference from the total iteration timer is assigned to other work. This residual includes phase transitions and work outside the three named groups. Component smoothing uses a trailing 20-iteration arithmetic mean and preserves their sum. Reported time shares divide unsmoothed component sums by the unsmoothed total. They describe the change in time allocation after quantization rather than a monotonic decrease in rollout share over training.

D Analysis of KV Concentration

This appendix proves Proposition 1 and relates its continuous-refill model to quota-based collection. The analysis concerns long-run averages over decode steps under a fixed distribution of response lengths. Prefill and policy updates are outside this clock.

D.1 Logical Work and Occupied Context

A response with prompt length P and generated length L visits cached contexts $P, P+1, \ldots, P+L-1$. Its total logical KV work is $Q(P, L) = PL + L(L-1)/2$. For fixed P, write $q = \mathbb{E}[Q(P, L)] = P\mu + (\mathbb{E}[L^2] - \mu)/2$. Finite second moment of L ensures that this expected work is finite. The

quantity counts context positions referenced during decoding. Physical memory traffic additionally depends on the attention implementation and cache representation.

Consider one slot that immediately replaces each completed response. A response of length l occupies the slot for l decode steps and visits each age $g \in \{0, \dots, l-1\}$ once. The limiting distribution obtained by inspecting occupied slot-steps therefore satisfies

$$\Pr(L^* = l) = \frac{l \Pr(L = l)}{\mu}, \qquad \Pr(g = a \mid L^* = l) = \frac{1}{l} \quad (0 \le a < l). \tag{A3}$$

These relations follow by dividing the visits contributed by complete responses by their total decode duration (Ross, 1996). They describe empirical time averages and do not require an aperiodic length distribution. Longer responses occur more frequently in the occupied batch because they remain there longer.

D.2 Proof of Proposition 1

For one continuously refilled slot, the renewal-reward theorem gives mean context q/μ. Equivalently, Eq. equation A3 gives mean generated prefix $\mathbb{E}[L(L-1)]/(2\mu)$. Averaging over the fixed number B of slots yields

$$\overline{\ell}_{\text{async}} = \frac{q}{\mu} = P + \frac{\mu - 1}{2} + \frac{\sigma^2}{2\mu}, \qquad \overline{V}_{\text{async}} = \frac{\kappa B q}{\mu}. \tag{A4}$$

Finite mean reward and cycle length ensure almost-sure convergence. An incomplete response at the observation boundary contributes a vanishing fraction of cumulative work in this limit.

A synchronous wave contains B responses and lasts $M = \max_{1 \le i \le B} L_i$ decode steps. The sum of resident volume over that wave is $\kappa \sum_{i=1}^{B} Q(P, L_i)$. Successive waves are independent cycles, so their expected work divided by expected duration gives

$$\overline{V}_{\text{sync}} = \frac{\kappa B q}{\mathbb{E}[M]}, \qquad \overline{B}_{\text{sync}} = \frac{B\mu}{\mathbb{E}[M]}, \qquad \frac{\overline{V}_{\text{async}}}{\overline{V}_{\text{sync}}} = \frac{\mathbb{E}[M]}{\mu} \ge 1. \tag{A5}$$

The reward and duration within a wave may be dependent. The renewal-reward theorem requires the ratio of their expectations rather than independence between them. Their expectations are finite under the proposition's assumptions. In addition, $M^2 \le \sum_{i=1}^{B} L_i^2$ ensures a finite second moment for the wave duration. The step-averaged synchronous context therefore also exists and equals $P + (\mathbb{E}[M^2] - \mathbb{E}[M])/(2\mathbb{E}[M])$, since every surviving response has context $P + t$ at wave step t. This proves the stated existence and identities.

D.3 Variable Prompts and Finite Collection

The mean context of an occupied slot should be distinguished from the unweighted mean of the context curve when batch size varies. For schedule s, define the slot-weighted context and step-averaged occupancy by

$$\ell_s^{\text{slot}} = \lim_{D \to \infty} \frac{\sum_{t<D} \sum_{i \in \mathcal{R}_t^s} \ell_i(t)}{\sum_{t<D} B_t^s}, \qquad \overline{B}_s = \lim_{D \to \infty} \frac{1}{D} \sum_{t<D} B_t^s. \tag{A6}$$

The first average samples occupied slot-steps. The main-text quantity $\overline{\ell}_s$ gives every decode step equal weight. They coincide when $B_t = B$, whereas a synchronous long-tail step receives disproportionately more weight in $\overline{\ell}_s$.

For an i.i.d. stream of pairs (P, L) with finite $\mathbb{E}[PL]$ and $\mathbb{E}[L^2]$, the work per generated token is

$$\ell^{\text{slot}} = \frac{\mathbb{E}[Q(P, L)]}{\mathbb{E}[L]} = \frac{\mathbb{E}[PL]}{\mu} + \frac{\mathbb{E}[L^2] - \mu}{2\mu}. \tag{A7}$$

The prompt term is weighted by response length. It reduces to $\mathbb{E}[P]$ when prompt and response lengths have zero covariance. Under continuous refill, Eq. equation A7 is also the mean resident context per slot.

The same accounting extends to prefix-preserving schedules that pause and resume responses. Suppose at most $H < \infty$ admitted responses remain unfinished at any time and every admitted response eventually completes. After n admissions, the unexecuted token count is bounded by $H \max_{j \le n} L_j$. The remaining logical KV work is bounded by $H \max_{j \le n} Q(P_j, L_j)$. Both bounds are $o(n)$ almost surely because the corresponding random variables have finite means. The strong law then gives Eq. equation A7 for the executed slot-steps. Whenever $\overline{B}_s$ exists, the resident-volume average follows as $\overline{V}_s = \kappa \overline{B}_s \ell^{\mathrm{slot}}$.

Actual quota-based collection has a finite request pool and admits requests subject to page availability. It can therefore lose concurrency within a round and does not satisfy constant occupancy. The proposition isolates the structural effect of continuous refill, while the measured traces establish its magnitude under these finite constraints. Our synchronous replicas collect 512 responses per round and both schedulers permit up to 1,024 active requests per replica. The two real schedulers consequently need not match the proposition's common-B comparison exactly. Policy changes can also alter response lengths across iterations. The fixed-stream identities describe how a given workload is arranged, and the frozen-checkpoint experiments hold policy parameters fixed when measuring that arrangement.

E Supplementary Results

We supplement the main-text results with two comparisons. The first examines how workload concentration and quantization gains change across training stages and model sizes. The second tests the location and precision of selective protection at fixed checkpoints. Both use the protocols in Appendix C.

E.1 Workload and Acceleration across Training

Fig. A3 extends the 4B-middle trace in Fig. 1 to earlier and later checkpoints of both models. The resident-KV ratio closely follows the active-batch ratio at each checkpoint. Their magnitudes vary substantially across training, even though the scheduler configuration remains fixed. Larger resident contexts also increase per-step cost, so decode-throughput gains are smaller than the gains in concurrency.

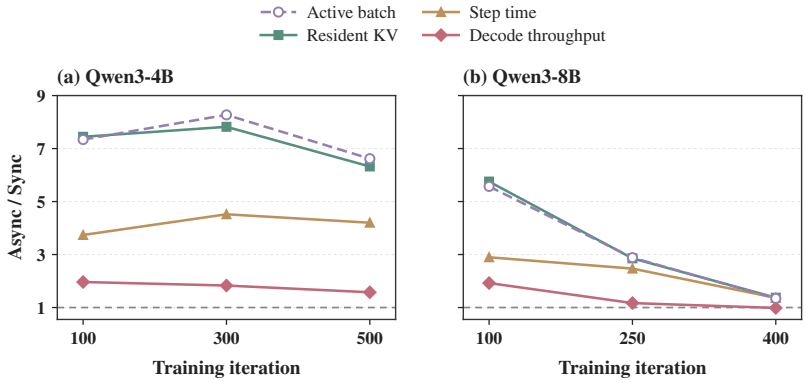

Figure A3: Evolution of workload concentration in Qwen3-4B-Base and Qwen3-8B-Base. Each point is an asynchronous-to-synchronous ratio at a frozen BF16 checkpoint. Batch size and nominal active KV volume are averaged over decode calls. Step time uses CUDA-event duration and throughput uses committed tokens per summed decode time. Both replicas contribute after three warmup rounds. The 4B results use twelve observed rounds and the 8B results use six. The dashed horizontal line marks equality between schedulers.

The late 8B checkpoint illustrates the capacity constraint. Its asynchronous engine holds an average of 91.7 GB of active KV per GPU against a nominal pool capacity of 93.9 GB. The synchronous workload is also substantial at 66.8 GB. Asynchronous scheduling therefore provides a smaller concurrency increase than at the early checkpoint, and its decode throughput is $0.98\times$ the synchronous

value. This result concerns decode-service throughput under the fixed page budget. It shows why the scheduling benefit cannot be extrapolated from the 4B-middle trace alone.

At the same time, longer contexts provide more opportunity for KV-cache compression. Fig. A4 applies the main-text replay protocol to all three 8B checkpoints. Uniform KV4 accelerates decode by $1.44\times$ at the early checkpoint and reaches $1.97\times$ at the late checkpoint. FP8 W8A8 provides at most $1.06\times$, while W4A16 remains slower than BF16. The benefit of KV precision thus persists beyond the 4B lineage even where the additional throughput from asynchronous scheduling has become small.

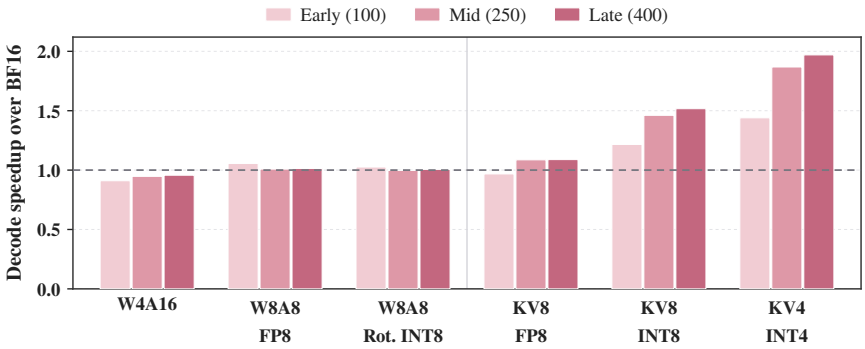

Figure A4: Measured Qwen3-8B-Base decode-forward speedup on H200 with TP2. Early/Mid/Late denote checkpoints 100/250/400. Each checkpoint uses five fixed shapes with equal baseline-time weights. The dashed line marks BF16 speed. W4A16 uses vLLM Machete and FP8 W8A8 uses our adaptation of vLLM's CUTLASS GEMM. FP8 KV8 uses FlashInfer FA2. Integer KV8 and KV4 use our paged-attention kernels. Appendix C.4 specifies all configurations including rotated INT8 W8A8. Uniform KV4 has no protected side cache here.

The two KV8 paths also show that storage precision alone does not determine execution speed. The FlashInfer FP8 path reaches 0.97–$1.09\times$ speedup in these states, compared with 1.22–$1.52\times$ for the integer path. The comparison changes both representation and attention arithmetic. It supports evaluating the implemented path when estimating the practical benefit of cache compression.

E.2 Location and Precision of Protection

Fig. A5 separates the effect of protection location from its execution cost. Panels (a,c) compare allocations of the same 144 protected positions against uniform KV4. At 4B–300, protecting sink 16 and recent 128 reduces mean KL by $7.0\times$ relative to uniform KV4. It also produces lower mismatch than assigning all 144 positions to either region. The same allocation reduces mean KL by $5.7\times$ at the 8B fidelity checkpoint. Both regions therefore contribute to the default protection in these comparisons.

The speed–mismatch plots in Fig. A5(b,d) show how this reduction relates to execution cost. The default retains approximately 94% of uniform-KV4 decode throughput at 4B and 98% at 8B in this suite. BF16 protection is slower than INT8 protection at the same positions. Their KL intervals overlap, with a slightly lower BF16-protection mean at 8B and a slightly higher mean at 4B. This comparison changes protected QK arithmetic along with storage precision. The observed trade-off supports INT8 protection as an efficient choice without requiring exact equivalence to BF16 protection.

The recent width controls how much context receives additional protection. Increasing it to 512 further reduces mismatch in both probes. Its execution cost is larger at 4B, while the 8B timings remain close to those of the default. Conversely, a 16-token recent window retains speed but leaves more mismatch. These results establish a useful protection range and explain the choice of a compact default. The training comparisons in Section 4 and Section 5 assess its effect on learning.

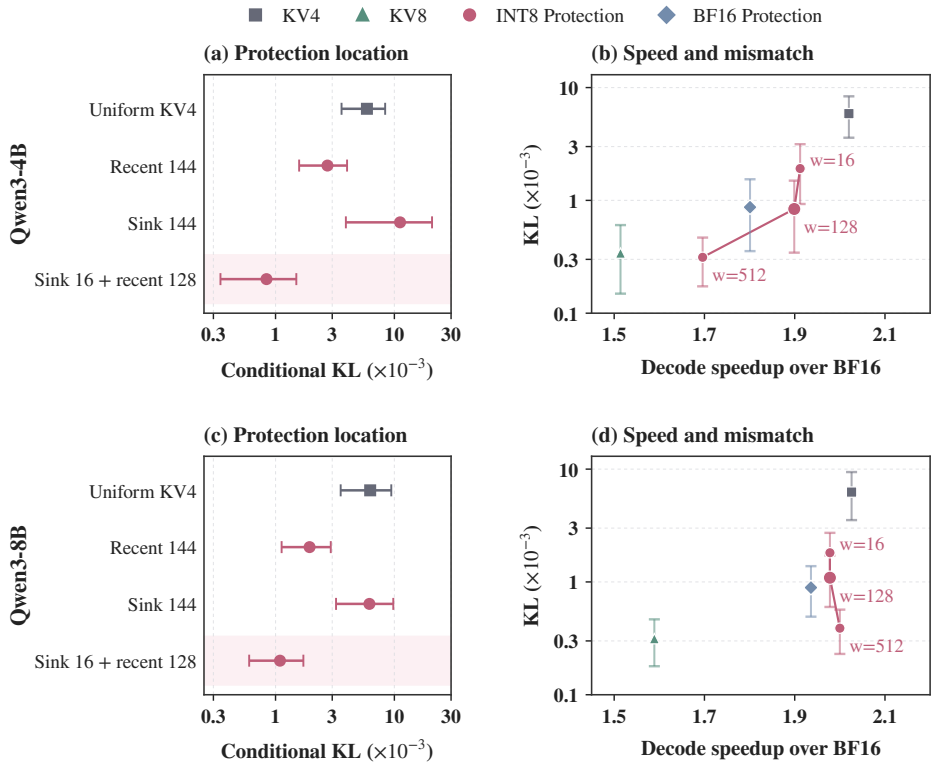

Figure A5: Location and cost of selective protection with actual attention kernels. Panels (a,c) compare INT8 allocations of 144 positions. Panels (b,d) plot decode speedup against shared-prefix $D_{\mathrm{KL}}(p_{\mathrm{BF16}} \| q)$. All KL axes are logarithmic. Connected INT8 points keep sink 16 and vary recent width w. The larger marker denotes $w = 128$. BF16 protection uses the same 16/128 positions and KV8 denotes uniform INT8. Error bars are 95% path-cluster bootstrap intervals over 32 responses. Speedup uses matched heterogeneous states. The 4B policy and traces are from iteration 300. The 8B policy is from iteration 200 with synthetic states matching iteration-250 trace statistics.

E.3 Quantization Format and Rotation

We examine the numerical choices underlying the INT4 representation with a controlled storage-quantization probe. It follows the same BF16-generated histories as the deployed-engine fidelity comparison and measures the conditional KL in Eq. equation 4. Prompt prefill remains BF16 and the forced continuation uses the quantized cache. The probe reconstructs the stored values and performs attention with BF16 operands. It therefore compares storage formats separately from the INT8 query and probability operands used by the deployed kernel. Its absolute KL values need not equal those in Fig. 3(b,c).

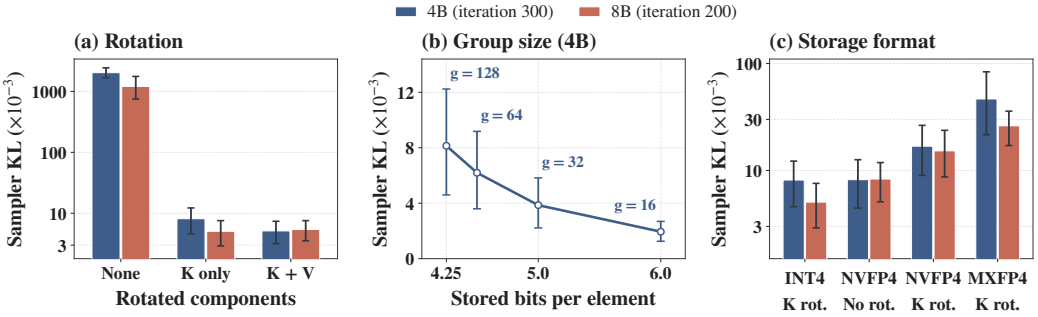

Figure A6: Storage-quantization design. (a) Rotation choices for affine INT4 with group size 128. (b) Group size g versus storage cost at 4B with K-only rotation. Bits include metadata. (c) INT4 and FP4 representations with rotation specified below each format. All probes use BF16 attention operands. Panels (a,c) use logarithmic KL axes. Error bars are 95% response-bootstrap intervals over 32 histories.

Fig. A6(a) shows that K rotation is important when a full head shares one affine scale. It sharply reduces mismatch at both checkpoints. Adding V rotation produces a smaller mean at 4B but does not give a consistent improvement across the probed histories and checkpoints. The response-paired geometric KL ratio for K-and-V rotation relative to K-only rotation is 0.89 with a 95% interval of $[0.55, 1.53]$ at 4B. At 8B–200, the corresponding ratio is 1.29 with interval $[0.63, 3.04]$. These comparisons support retaining K-only rotation in the deployed design.

As shown in Fig. A6(b), finer channel groups improve fidelity at a higher storage cost. Reducing the affine group size from 128 to 32 lowers the 4B mean KL from 8.14×10^{-3} to 3.86×10^{-3} while increasing storage from 4.25 to 5.00 bits per element. A further reduction to group size 16 lowers mean KL to 1.94×10^{-3} at 6.00 bits per element. Each group also requires its own scale and offset in the attention reduction. This comparison motivates examining selective protection as another use of additional precision. It does not establish a latency ordering between the deployed protection kernel and a separately optimized small-group kernel.

Fig. A6(c) compares INT4 with NVFP4 and MXFP4 representations. The NVFP4 probe uses E2M1 values with 16-element E4M3 scales and an FP32 global scale per token–head vector. This implementation uses 4.75 bits per element, with the global scale accounting for 0.25 bits per element. The MXFP4 probe uses 32-element E8M0 scales and occupies 4.25 bits per element. Without rotation, NVFP4 has a 4B mean KL close to head-grouped INT4 with K rotation. Adding K rotation raises NVFP4 mismatch at both checkpoints, showing that its effect depends on the representation. Thus the choice of INT4 rests on its head-level affine structure and validated integer execution rather than a universal fidelity advantage over FP4. No FP4 attention-kernel latency is inferred from this numerical probe.

E.4 Learning Trajectories and Endpoint Evaluation

Fig. A7 extends the MATH-500 comparison to both AIME benchmarks. Each precision configuration uses one Qwen3-8B-Base training run with seed 43 and ten warmup iterations. Training continues to iteration 400 under the same GRPO and TIS settings. Every checkpoint is evaluated in BF16. Each reported mean combines the evaluation recorded by the training run with standalone evaluations using

Table A5: Final Qwen3-Base pass@k scores (%) under BF16 evaluation at iteration 600 for 4B and 400 for 8B. Subscripts give standard deviations over three evaluation seeds. Mean averages the six tasks equally.

		pass@8			pass@32			
Model	**Rollout**	**MATH-500**	**OlympiadBench**	**MinervaMath**	**AIME24**	**AIME25**	**AMC23**	**Mean**
Qwen3-8B-Base	BF16	$96.5_{\pm 0.3}$	$73.8_{\pm 0.5}$	$70.7_{\pm 2.4}$	$66.7_{\pm 6.7}$	$52.2_{\pm 5.1}$	$97.5_{\pm 0.0}$	$76.2_{\pm 1.9}$
	KV8	$96.5_{\pm 0.1}$	$73.6_{\pm 0.3}$	$69.6_{\pm 1.5}$	$64.4_{\pm 3.8}$	$55.6_{\pm 1.9}$	$96.7_{\pm 1.4}$	$76.1_{\pm 0.6}$
	KV4	$92.3_{\pm 0.6}$	$66.3_{\pm 0.7}$	$69.0_{\pm 0.8}$	$56.7_{\pm 3.3}$	$37.8_{\pm 1.9}$	$93.3_{\pm 3.8}$	$69.2_{\pm 0.4}$
	KV4-RL	$96.2_{\pm 0.2}$	$74.2_{\pm 0.7}$	$71.3_{\pm 2.9}$	$64.4_{\pm 1.9}$	$57.8_{\pm 1.9}$	$96.7_{\pm 1.4}$	$76.8_{\pm 1.1}$
Qwen3-4B-Base	BF16	$92.5_{\pm 0.6}$	$66.8_{\pm 1.1}$	$63.8_{\pm 0.2}$	$45.6_{\pm 3.8}$	$42.2_{\pm 3.8}$	$95.0_{\pm 2.5}$	$67.7_{\pm 1.2}$
	KV8	$92.3_{\pm 0.6}$	$66.9_{\pm 0.6}$	$65.4_{\pm 1.0}$	$56.7_{\pm 8.8}$	$37.8_{\pm 5.1}$	$95.8_{\pm 3.8}$	$69.1_{\pm 1.6}$
	KV4	$92.0_{\pm 0.4}$	$65.8_{\pm 1.5}$	$64.8_{\pm 0.2}$	$44.4_{\pm 3.8}$	$41.1_{\pm 7.7}$	$93.3_{\pm 1.4}$	$66.9_{\pm 1.8}$
	KV4-RL	$92.4_{\pm 0.4}$	$66.6_{\pm 1.0}$	$65.7_{\pm 0.8}$	$48.9_{\pm 6.9}$	$41.1_{\pm 7.7}$	$95.0_{\pm 2.5}$	$68.3_{\pm 2.7}$

seeds 42 and 44. The standard deviation measures repeated evaluation at fixed policy parameters and does not estimate variation across independent training runs.

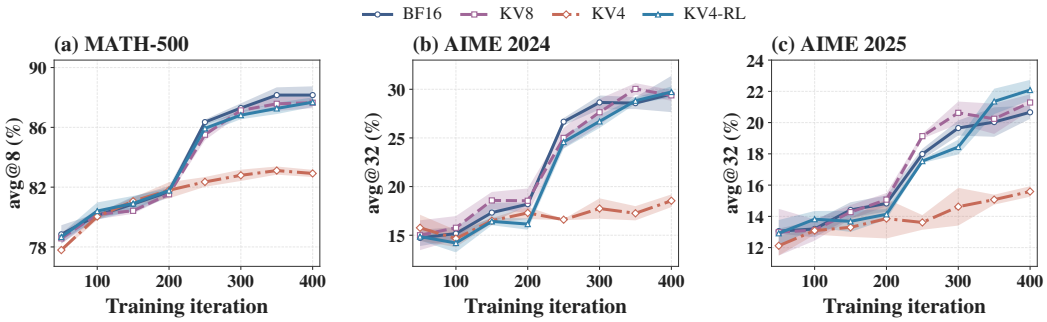

Figure A7: Learned performance under four rollout precisions. All curves use BF16 evaluation of Qwen3-8B-Base checkpoints from training seed 43. Shading shows mean ± one sample standard deviation over three evaluation seeds. MATH-500 uses avg@8 and both AIME benchmarks use avg@32.

KV4 follows the early improvement but falls behind as BF16 and the protected sampler improve after iteration 200. The pattern is visible on all three tasks. KV4-RL remains close to the BF16 curve on MATH-500 and AIME 2024, while its final AIME 2025 score is higher in this run. This supports recovery from the observed KV4 degradation without implying identical learned policies.

Table A5 complements the avg@k results in Table 1 with pass@k on the same six benchmarks for both model sizes. All scores use the BF16 evaluation protocol in Appendix C.2. Task means and sample standard deviations are computed over three evaluation seeds. For the Mean column, we first average the six tasks within each seed and then compute the mean and standard deviation across seeds. The 8B comparison combines in-training and standalone evaluation, while all three 4B repeats are standalone evaluations. Each model retains the same collection of evaluation sources across rollout precisions.

E.5 Acceleration and Learned Performance on Qwen3-4B-Base

Fig. A8 shows that KV4-RL lowers collection time while retaining generation lengths close to BF16. Across the 600 recorded updates, its cumulative rollout time decreases by $1.57\times$ ($8.65 \rightarrow 5.50$ hours) and total iteration time by $1.17\times$ ($19.80 \rightarrow 16.87$ hours). Mean response lengths are 1.45k tokens for BF16 and 1.46k for KV4-RL. The rollout advantage becomes larger late in training, reaching $1.72\times$ over the final 50 iterations when both configurations average approximately 2.0k tokens. This pattern is consistent with the growing benefit of reducing KV traffic as generation length increases.

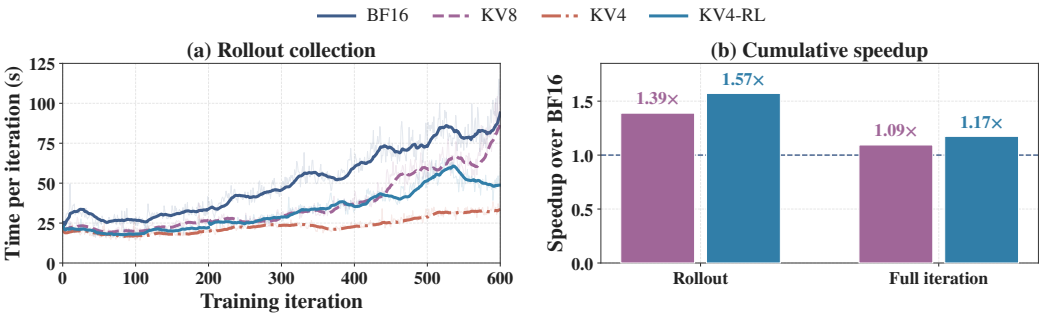

Figure A8: Qwen3-4B-Base timing over 600 iterations. (a) Rollout time with 20-iteration means within each source segment. (b) Ratios of summed BF16 timers to each method's timers. KV4 is omitted from (b) because its responses remain shorter.

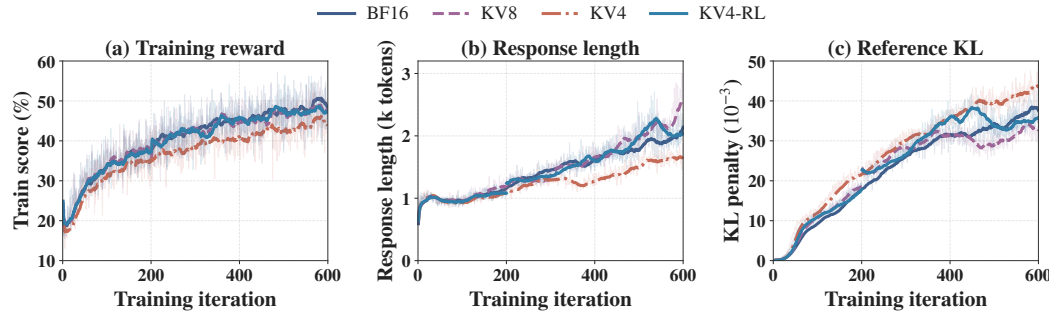

Figure A9: Qwen3-4B-Base training metrics over 600 iterations. (a) Correctness reward. (b) Mean completed-response length. (c) Logged KL penalty against the initial policy. Thick curves show 20-iteration means over faint raw traces.

The learning curves in Fig. A9 distinguish these savings from the shorter runtime of KV4. BF16 and the protected sampler show gradual response-length growth accompanied by increasing reward. KV4 also lengthens its responses at 4B but remains shorter and earns lower reward while its reference-KL penalty grows above the other configurations. Over the final 100 iterations, KV4-RL's mean reward is 1.1 percentage points below BF16 compared with a 4.6-point gap for KV4. Its late reference-KL penalty also remains closer to BF16. These measurements describe optimization on each policy's own samples and help explain why KV4's low collection time alone is insufficient evidence of comparable training efficiency.

E.6 Iteration-Time Breakdown and Remaining Learning Cost

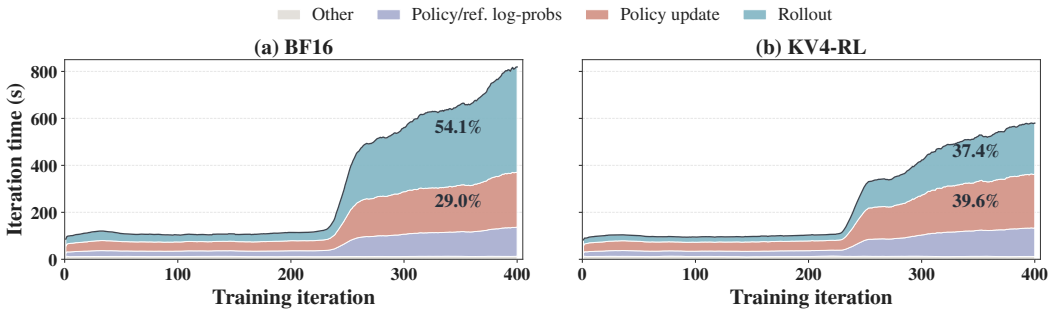

Figure A10: Iteration-time breakdown on Qwen3-8B-Base after 20-iteration averaging. Policy/ref. log-probs are separate learner forward passes. Other is the residual of the logged total. Percentages summarize the final 50 iterations.

Fig. A10 shows how rollout acceleration changes the allocation of training time. Over the final 50 iterations, KV4-RL reduces the rollout share from 54.1% under BF16 to 37.4%. The policy-update share consequently rises from 29.0% to 39.6% and becomes the largest individual component. Including the policy and reference log-probability passes, the learner accounts for 60.5% of iteration time. This change arises primarily from faster collection while the absolute cost of learning remains similar.

Our method targets rollout and retains BF16 learning. The remaining full-sequence forward and backward passes perform large matrix multiplications, motivating hardware-supported FP4 training (Castro et al., 2025; Chmiel et al., 2025) as a complementary acceleration direction. Quantization-aware training may help preserve update accuracy when paired with low-precision execution. Their effect on RLVR optimization remains to be evaluated, but the measured time allocation shows why learner efficiency deserves attention alongside further rollout acceleration.