DORA: A Scalable Asynchronous Reinforcement Learning System for Language Model Training

Tianhao Hu*, **Xiangcheng Liu***, **Yuchun Miao***, **Youshao Xiao***, **Hongyu Zang**, **Yang Zheng**
Xuan Huang, **Jinrui Ding**, **Yufei Zhang**, **Yu Yang**, **Yi-Kai Zhang**, **Yueqing Sun**
Chengcheng Han, **Xiandi Ma**†, **Qi Gu**†, **Wei Wang**, **Yerui Sun**, **Yuchen Xie**, **Xunliang Cai**
Meituan Longcat Team, *Equal contribution, †Corresponding authors
{xiaoyoushao, maxiandi02, guqi03}@meituan.com

Abstract

Reinforcement learning (RL) has become a critical paradigm for large-scale post-training of LLMs in industrial settings, yet it faces a structural *long-tail dilemma*: rollout efficiency is bottlenecked by the longest trajectories, which are often the most valuable for RL training. Existing approaches alleviate this dilemma at the cost of either system overhead (e.g., re-prefill in partial-rollout methods) or algorithmic compromises (e.g., discarded long trajectories in replication-based methods). Both approaches implicitly assume that all rollout instances must synchronize around a single policy version, creating an unavoidable batch barrier. We propose **DORA** (**D**ynamic **OR**chestration for **A**synchronous Rollout), which breaks this assumption by maintaining multiple policy versions concurrently within the rollout cluster to solve the skewed generation problem without algorithmic compromise. DORA combines three mechanisms: *multi-version streaming training* that decouples trajectory completion from batch barrier, a centralized *load-balancing orchestrator* that re-partitions resources across versions, and nearly *zero-re-prefill migration* that transfers KV-Cache directly across same-version instances. Experiments on open-source benchmarks show that DORA achieves up to $2.12\times$ end-to-end throughput improvement and $8.2\times$ rollout-stage acceleration over synchronous training while preserving convergence parity. **In real-world production deployment** with thousands of accelerators, DORA achieves up to $6.2\times$ rollout speedup, training a 560B-parameter MoE model (i.e., **LongCat-Flash-Thinking**) competitive with state-of-the-art open-source LLMs.

1 Introduction

Reinforcement Learning (RL) has become a pivotal paradigm for LLM post-training, leveraging test-time scaling [29] to advance complex reasoning and agentic capabilities [2, 19, 25, 31, 33]. The RL training loop sequentially cycles through rollout (trajectory generation), experience preparation (reward and reference computation), and model training. Among these stages, rollout accounts for 50%–80% of the total step duration and represents the fundamental training bottleneck [32, 34, 35]. As industrial deployments scale to thousands of accelerators, optimizing rollout efficiency directly determines the overall training cost and turnaround time.

This rollout bottleneck is fundamentally driven by a ***long-tail dilemma*** inherent to complex reasoning tasks: a direct conflict between algorithmic value and hardware efficiency. In mathematics and coding domains, response lengths follow a highly skewed distribution where the 99th-percentile output can exceed the median by over an order of magnitude (Figures 1 and 2). The dilemma arises because these exceptionally long trajectories carry the highest information density—encapsulating the intricate chain-of-thought (CoT) reasoning steps that are the primary source of emergent capabilities [19, 8]—

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

making them indispensable for RL training. However, because decoding is inherently memory-bound, we cannot simply accelerate them by scaling compute, nor can we discard them without catastrophic algorithmic degradation. Consequently, under standard synchronous training, the entire batch is held hostage by a small fraction of these most valuable, yet longest trajectories, leaving the majority of devices completely idle.

To address this dilemma, asynchronous training has been proposed, with current efforts primarily focusing on two directions. *Replication-based methods* [7, 38] shorten rollout duration by oversampling and dropping the in-flight long trajectories once enough complete. This *discards* precisely the chain-of-thought trajectories highlighted above, and the resulting *length-biased* distribution distorts the advantage estimation in group-relative methods such as GRPO [24]. *Partial-rollout methods* [30, 34, 6, 41] segment long trajectories at each weight update and resume them under the new policy. On the system side, every update invalidates the KV-Cache and forces a *full re-prefill* that grows dramatically with context length (Figure 3) and is further amplified in MoE architectures. On the algorithmic side, a trajectory now stitches multiple policy versions, potentially downgrades the rollout quality, and departs from the standard RL formulation, risking model performance degradation. Across both directions, existing approaches **alleviate the long-tail dilemma, but at the cost of either additional system overhead or algorithmic compromises**.

We argue that these tradeoffs stem from a common implicit assumption—***single-version rollout***, i.e., the rollout instances only serve a single policy version. Under this assumption, in-flight long-tailed trajectories must either complete before the next policy update or be sacrificed at the update—leaving no room for solutions that avoid both system overhead and algorithmic compromises.

In this paper, we depart from this regime through **DORA** (**D**ynamic **OR**chestration for **A**synchronous Rollout), which embodies a ***multi-version rollout*** paradigm where multiple policy versions coexist within the rollout instances, **resolving the long-tail dilemma without incurring significant system overhead or algorithmic compromises**. At its core, each trajectory is generated entirely under the policy version active at its dispatch, so that long-tailed trajectories run to completion under their original version while new requests proceed under the latest version.

Realizing this design, however, raises three system-level challenges, each addressed by a corresponding mechanism. **Version coexistence.** Maintaining multiple policy versions concurrently requires breaking the synchronous batch barrier so that completed trajectories can flow into training without waiting for the slowest ones. DORA achieves this through *multi-version streaming training*, which dispatches and collects trajectories at the granularity of individual requests across versions, with a sliding window that bounds policy staleness. **Resource fragmentation.** As trajectories of older versions complete, their rollout instances become progressively underutilized while the latest version is over-subscribed. DORA addresses this through a centralized *load-balancing orchestrator*, which continuously re-partitions data-parallel groups across versions in proportion to their pending workloads and migrates requests to rebalance the cluster. **Migration overhead.** Naively migrating a request across instances would re-trigger the prefill phase, which is costly in long-context and MoE settings. DORA avoids this entirely: since the trajectory is generated under the consistent policy version across multiple RL steps, its KV-Cache states are mathematically equivalent across any instance hosting that version, enabling nearly *zero-re-prefill migration* via direct cross-instance KV-Cache transfer. Together, these mechanisms enable DORA to deliver **substantial efficiency gains while maintaining standard RL convergence.** Our main contributions are summarized as follows:

• **Multi-Version Streaming Training.** We propose *multi-version streaming training*, a new asynchronous paradigm that maintains multiple policy versions concurrently within the rollout instance, eliminating the long-tail bubble without introducing significant system overhead or algorithmic compromises.

• **Dynamic Orchestration.** We design a centralized *load-balancing orchestrator* that dynamically repartitions data-parallel groups across versions in proportion to their pending workloads and migrates requests to rebalance the cluster, eliminating resource fragmentation across coexisting policy versions.

• **Nearly Zero Re-prefill KV-Cache Reuse.** We design *zero re-prefill migration* that transfers KV-Cache directly across same-version instances with negligible communication costs, eliminating prefill recomputation during request migration—especially beneficial for long-context reasoning and agentic scenarios.

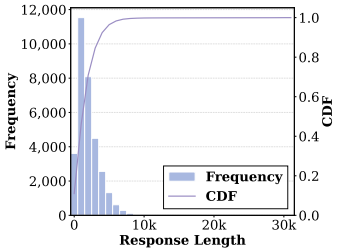


Figure 1: Response length distribution on the DAPO-Math-17K dataset.

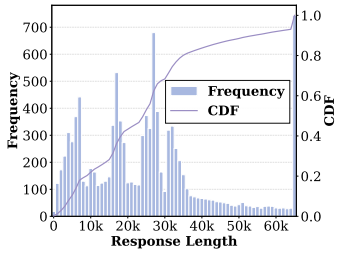


Figure 2: Response length distribution in production. Stacked bar in the end of x-axis reflects overlong truncation.

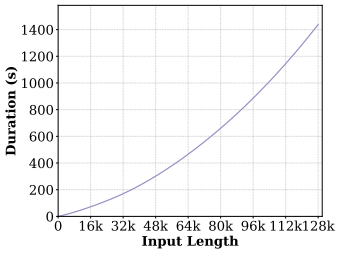


Figure 3: Prefill duration of a ∼500B MoE model with expert parallelism size 128 on non-CUDA accelerators.

• **Extensive Evaluation and Real-World Deployment.** On open-source benchmarks, DORA achieves up to 2.12× end-to-end and 8.2× rollout-stage speedup over synchronous training while preserving model convergence. *Production deployment* with thousands of accelerators further yields up to 6.2× rollout speedup, training a ∼500B-parameter MoE model competitive with state-of-the-art open-source LLMs.

2 Preliminaries

2.1 Asynchronous RL Training

RL post-training for LLMs proceeds in iterative steps, each consisting of three stages: *rollout* (sampling responses from the current policy), *experience preparation* (computing rewards and references), and *model training*. In synchronous training, a step cannot begin until all trajectories of the previous step are complete, enforcing a strict batch barrier between rollout and training. To overlap rollout and training, asynchronous methods relax this barrier and allow the training samples to mix trajectories generated under different behavior policy versions.

We take GRPO [24], a variant of PPO, as a representative algorithm. Given a prompt x, G trajectories $\{y_i\}_{i=1}^{G}$ are sampled per prompt and used to update the policy π_θ via a clipped importance-weighted objective with group-relative advantages $\hat{A}_i$ shared across all tokens of trajectory y_i [8, 37]. Let $v(\cdot)$ denote the version index of a policy and K a configurable upper bound on staleness. For trajectory y_i under behavior policy π_{w_i} updating training policy π_θ, asynchronous training requires:

$$v(\theta) - v(w_i) \;\le\; K, \tag{1}$$

which is the standard condition for convergence guarantees in asynchronous optimization [16, 39]. The asynchronous GRPO objective replaces the single behavior policy with the per-trajectory π_{w_i} in the importance ratio:

$$\mathcal{J}_{\text{async}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{L_i}\sum_{t=1}^{L_i}\min\Big(r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}_\varepsilon\big(r_{i,t}(\theta)\big)\hat{A}_{i,t}\Big)\right], \quad r_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}\mid\cdot)}{\pi_{w_i}(y_{i,t}\mid\cdot)}, \tag{2}$$

subject to Equation 1. Throughout the paper, we denote the rollout batch size (number of prompts dispatched per step) as RBS and the training batch size (number of trajectories consumed by the training stage) as TBS.

2.2 Long-Tail Dilemma

The core objective of asynchronous RL is to resolve the massive hardware idleness caused by long-tailed generation, which manifests at the system level through two compounding factors:

Workload skewness. In long-context reasoning workloads, response lengths L_i follow a heavily long-tailed distribution. As shown in Figures 1 and 2, the 99th-percentile output exceeds the median by over an order of magnitude on both an open-source benchmark and a production workload. Because decode is memory-bound, this tail cannot be flattened by adding compute. Prefill cost compounds

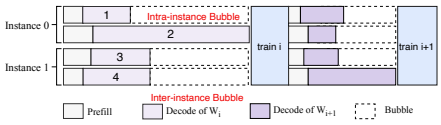


Figure 4: Skewed bubbles in synchronous training: intra-node bubble (idle slots within a device) and inter-node bubble (faster instances waiting for the slowest).

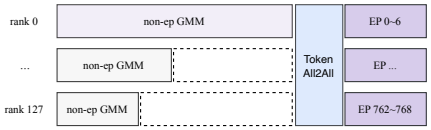


Figure 5: Non-EP GMM workloads become unbalanced under long-tailed inputs (expert parallelism size 128).

this issue: as shown in Figure 3, prefill duration grows dramatically with input length, making any technique that re-triggers prefill mid-step (e.g., re-prefill after weight updates) increasingly expensive in long-context regimes.

Hardware-level idleness. The rollout phase processes RBS prompts per step, where each request incurs a compute-bound prefill and a memory-bound decode, with concurrency capped by accelerator memory occupied by model weights and KV-Cache. The objective is to minimize total step duration:

$$\min\ T_{\text{train}} + T_{\text{Prefill}} + T_{\text{Decode}} \ \Rightarrow\ \min\ T_{\text{train}} + T_{\text{Prefill}} + \overbrace{\tau \max_j \underbrace{\max_{i \in \text{Device}_j} \{L_i\}}_{\text{intra-node bubble}}}^{\text{inter-node bubble}} \tag{3}$$

where τ is the time per output token (TPOT, approximately constant under fixed batch size for rollout engine). Combined with the long-tailed distribution above, this $\max\max$ structure in the rollout creates two forms of device idleness illustrated in Figure 4: an *intra-node bubble*, where completed slots on a device sit idle while a long-tailed request continues without device saturation, and an *inter-node bubble*, where faster instances wait for the slowest one. In MoE architectures, this skew further propagates into non-EP layers, where the slowest rank stalls the entire EP group (Figure 5).

While long-tail trajectories bottleneck rollout efficiency, their rich learning signals make them indispensable for RL. This *long-tail dilemma* exposes a fundamental conflict between hardware utilization and algorithmic integrity. Consequently, existing asynchronous methods cannot mitigate rollout bubbles without incurring significant re-prefill overhead or algorithmic sampling bias (Section 1, Appendix A). The remainder of this paper introduces a system architecture that natively resolves this tension, eliminating bubbles without incurring significant system overhead or sacrificing algorithmic fidelity.

3 DORA Design

3.1 System Overview

Without loss of generality, we present DORA using a disaggregated architecture, although it readily extends to the colocated architecture. As motivated in Section 2.2, our goal is to resolve the long-tail dilemma without paying either significant system overhead or additional algorithmic cost. DORA achieves this through three interlocking mechanisms:

(i) Multi-version streaming training (Section 3.2) maintains multiple policy versions concurrently on rollout instances, enabling trajectory-level streaming that eliminates both intra-node and inter-node bubbles. Each trajectory is generated end-to-end under a single policy version, and the staleness across versions is bounded by a configurable window.

(ii) Dynamic resource orchestration (Section 3.3) resolves the resource fragmentation inherent in multi-version rollout by dynamically re-partitioning DP groups and migrating requests, while preserving every sampled trajectory and respecting the staleness bound throughout.

(iii) KV-Cache reuse (Section 3.4) turns the single-policy-per-trajectory design into a system-level advantage: because every token of a trajectory is generated under the same policy version, its KV-Cache states are mathematically equivalent across any instance hosting that version, enabling nearly zero-re-prefill migration during request relocation.

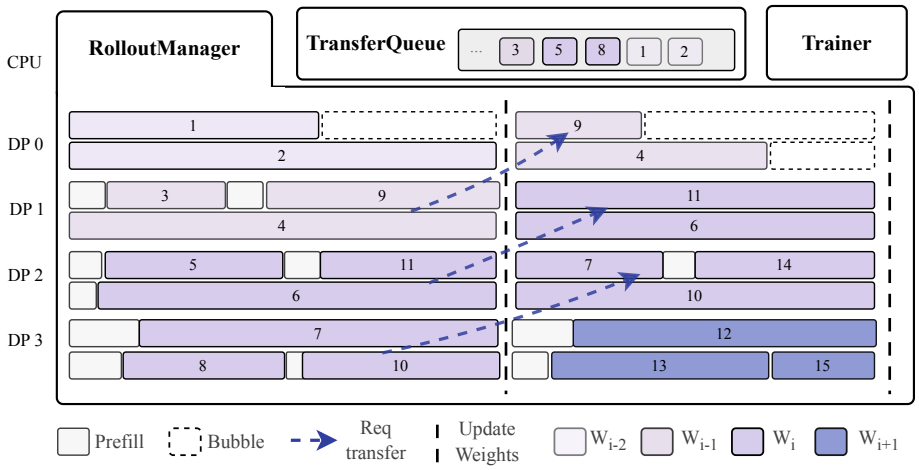


Figure 6: The execution timeline of DORA's multi-version streaming training system.

As shown in Figure 6, these mechanisms cooperate at runtime through four cooperating components. A `RolloutManager` dispatches prompts to rollout instances, tagging each prompt with a policy version so that the entire trajectory is generated under that version. Completed trajectories stream into an asynchronous `TransferQueue` equipped with staleness monitoring. The `Trainer` consumes TBS samples for experience preparation and model training, then synchronizes the latest weights with rollout instances. A `Load-balancing orchestrator` monitors per-version workloads and triggers resource re-partitioning and request migration as needed, preserving the intermediate execution state. These components run on different nodes and coordinate via Remote Procedure Call (RPC), while workers running on accelerators execute the actual tasks.

Together, these mechanisms eliminate the long-tail bubble in rollout while preserving every sampled trajectory and bounding policy staleness—resolving the long-tail dilemma without significant system overhead or algorithmic compromise.

3.2 Multi-version Streaming Training

Trajectory-level streaming. DORA eliminates the synchronous barrier by streaming completed trajectories directly to training without waiting for straggling trajectories. During the rollout phase, we maintain multiple versions of policy weights in the rollout instances, where each Data Parallel (DP) group hosts a single version of the policy weights. At the onset of each step, we overprovide the rollout prompts, where $RBS > TBS$ generation requests are dispatched to the rollout instances. Training begins as soon as TBS samples are collected; unfinished long-tailed trajectories continue under their original policy version and flow into subsequent steps, ensuring the long trajectories are not abandoned or block the training. As illustrated in Figure 6, the Rollout and training processes execute non-blockingly; only after a training iteration concludes does the `Trainer` notify the `RolloutManager` to synchronize the latest weights.

Multi-version policy management. The key insight is that maintaining multiple policy versions concurrently allows long-tailed trajectories to continue under their original version while the training proceeds with completed trajectories. As illustrated in Figure 6, each prompt is tagged with a version w_j upon dispatch, ensuring $a_t \sim \pi_{w_j}(\cdot \mid s_t)$ for every token—so each trajectory is generated end-to-end under a single policy version without algorithmic modifications. For example, in Figure 6, Trajectory 4 (a long-tail request under version w_1) spans two training steps while Trajectories 1–3 complete and stream into training during Step 1. The system proceeds to Step 2 with updated weights w_2 without waiting for Trajectory 4 to complete, which continues under its original version w_1 in a dedicated DP group. This allows the legacy-version requests to execute in parallel, thereby eliminating the inter-node bubble and fully utilizing all rollout instances.

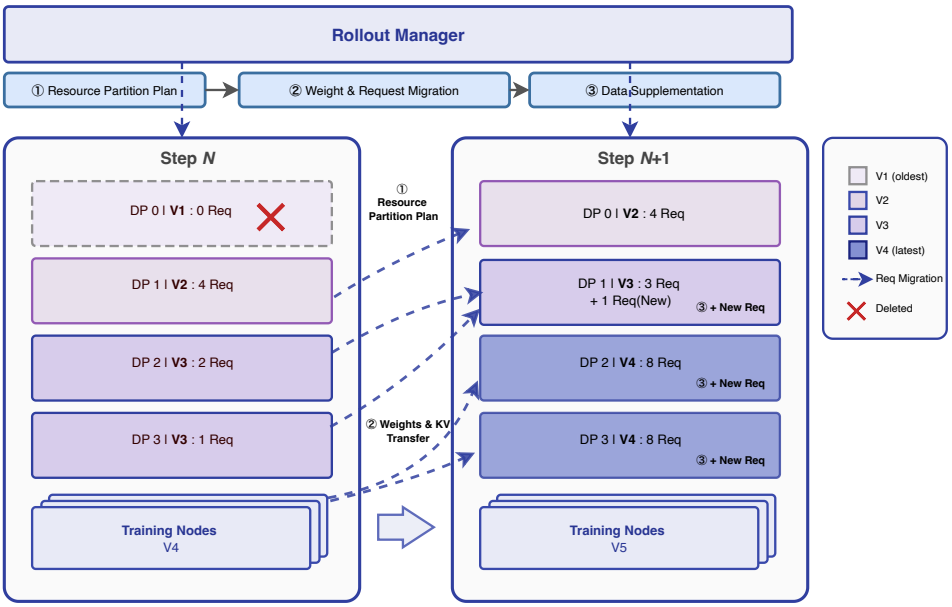


Figure 7: The workflow of the Dynamic Resource Orchestration.

Sliding-window staleness control. Active versions are managed through a sliding window $W = \{w_j, \ldots, w_{j-K+1}\}$ of size $|W| \leq K$. The window advancement follows a strict protocol to control the staleness. The window slides forward only when all trajectories from the oldest version w_{j-K+1} have been collected and forwarded to training. This provides a deterministic upper bound on policy staleness. Our theoretical analysis (Appendix D) further establishes that, under bounded per-update policy drift, the distribution-mismatch penalty in the multi-version policy-improvement bound grows at most linearly with K. The staleness bound K serves as an explicit control knob for the convergence–throughput tradeoff: a smaller K yields more on-policy data at the cost of rollout efficiency; a larger K increases throughput with controlled convergence impact.

Remaining challenges. While multi-version streaming preserves every sampled trajectory, bounds policy staleness, and partially alleviates both bubble types, it introduces two second-order efficiency challenges that motivate the subsequent mechanisms:

(i) Resource fragmentation. We observe that the legacy version's pending requests decrease monotonically as trajectories complete, yet its allocated resources remain fixed, leading to notable device underutilization. This motivates the dynamic orchestration described in Section 3.3.

(ii) Re-prefill overhead. Migrating requests across DP groups naively re-triggers the full prefill phase—prohibitive for long-context scenarios (64k–128k tokens). This motivates the KV-Cache reuse mechanism in Section 3.4, which exploits the mathematical equivalence guaranteed by single-version trajectory generation.

3.3 Dynamic Resource Orchestration

Under multi-version streaming training, the pending request count of each legacy version w decreases monotonically as trajectories complete, yet the physical resources (DP groups) assigned to w remain fixed. This implies that static resource allocation leads to progressive underutilization—a legacy version's DP groups may each serve only one or two residual requests, while the latest version, which carries the majority of new prompts, is under-provisioned. Therefore, DORA requires proactively rebalancing workloads while simultaneously controlling staleness.

To resolve this resource fragmentation, DORA employs a centralized orchestrator that dynamically re-partitions resources across model versions. The orchestrator maintains real-time metrics—active request counts per version, KV-Cache utilization, and generation progress—and supports three re-

balancing triggers: (1) *update-driven*, mandatory upon the completion of each training step to promote the new policy version; (2) *utilization-based*, activated when KV-Cache pressure exceeds a threshold, avoiding costly eviction-induced recomputation; and (3) *temporal-based*, periodic execution to prevent orphan requests from lingering in legacy versions.

As illustrated in Figure 7, each re-balancing cycle executes three coordinated operations:

- **Resource partitioning plan.** The orchestrator assesses the distribution of active and pending requests across all maintained versions W to produce a migration plan. It computes the target DP group count for each version $w \in W$ proportional to its current workload, preventing resources from being stranded on legacy versions with dwindling tasks. This addresses the inter-instance data skewness identified in Section 2.2.
- **P2P weight update and request migration.** Once the migration plan is determined, the orchestrator generates a mapping from the current partition to the target partition. For each version whose allocation changes, it leverages P2P weight transfers to rescale the DP groups—decrease the legacy versions and increase the latest one. Active requests on re-assigned nodes are migrated to their new DP groups with execution states fully preserved via KV-Cache reuse (Section 3.4). Notably, no trajectory is abandoned during this process.
- **Staleness-aware data supplementation.** To control data staleness within the configured bound, the orchestrator prioritizes the latest policy version for proactive data injection, dispatching supplemental prompts until the RBS is fully met. This strategy maximizes sample freshness by ensuring that the majority of new trajectories are generated using the most recent model weights. Subsequently, to maintain high-watermark utilization across the cluster, the orchestrator performs opportunistic data injection following the request migration phase. Legacy versions are only supplemented with sufficient prompts to fill their residual idle slots. This tiered injection approach effectively saturates all rollout instances while preventing the over-production of stale trajectories, striking an optimal balance between hardware occupancy and algorithmic freshness.

This optimization cycle exemplifies algorithm-system co-design: **Proportional Resource Partitioning** resolves the resource fragmentation inherent in multi-version rollout; **Request Migration** preserves trajectory execution state without abandoning any sampled trajectory; and **Staleness-Aware Data Supplementation** ensures high hardware occupancy within the staleness bound.

3.4 KV-Cache Reuse

Note that request migration across DP groups naively re-triggers the prefill phase. The re-prefill cost scales with the growing context length and is further amplified in MoE architectures [13, 8], where it causes workload imbalance across non-MoE layers as shown in Section 2.2. However, DORA's single-policy-per-trajectory design enables a powerful system-level optimization: since all tokens in a trajectory are generated by the same policy version π_w, the KV-Cache states are *mathematically equivalent* across any physical instance hosting version w. This equivalence enables cross-instance KV-Cache transfer that completely avoids re-prefill. Methods that mix multiple policy versions within a single trajectory forfeit this optimization: each weight update forces a full re-prefill of all ongoing trajectories, up to the output length. When the `Load-Balancing Orchestrator` triggers a resource re-allocation, DORA executes a coordinated two-phase state transfer:

- **Metadata forwarding.** Request metadata (request ID, generation state, decoded token count, and version tag) is transmitted via lightweight RPC. This control-plane transfer is negligible in both latency and bandwidth.
- **KV-Cache data transfer.** The voluminous KV Cache data—often comprising tens of gigabytes for long-context and MoE settings—is transferred using high-performance collective communication primitives, fully exploiting the available interconnect bandwidth.

Locality-aware scheduling. To minimize transfer volume, the orchestrator prioritizes re-assigning requests back to their original ranks when possible—preserving data locality and avoiding physical migration entirely. Only requests that must relocate due to version transitions incur transfer costs.

Hierarchical memory management. To alleviate VRAM pressure from aggregated requests during long-context training, DORA temporarily offloads KV-Caches to host memory [20], freeing device

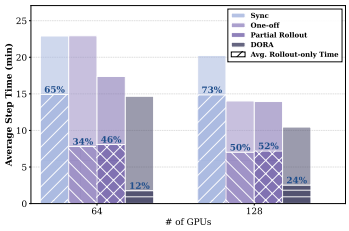


Figure 8: Average RL step time for Dense-32B across training paradigms at 64 and 128 GPUs.

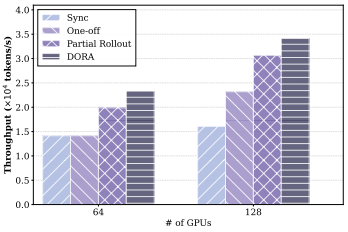


Figure 9: End-to-end throughput for Dense-32B across paradigms at 64 and 128 GPUs.

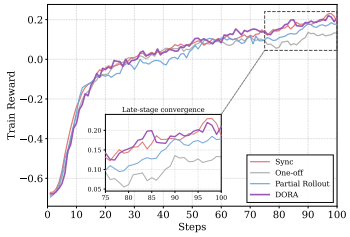


Figure 10: Training rewards for Dense-32B across training paradigms at 128 GPUs.

memory for active computations while preserving state for deferred generation. This hierarchical management safeguards system efficiency even under extreme long-tailed workloads.

By eliminating re-prefill from request migration, KV-Cache reuse closes the last source of system overhead introduced by multi-version coexistence: long-tailed trajectories now traverse legacy DP groups, get migrated under load-balancing, and continue generation—all without the prefill recomputation that would otherwise grow dramatically with context length.

4 Experiments

4.1 Experimental Setup

We evaluate DORA on a 16-node H800 cluster (128 GPUs) for open-source experiments and on a production cluster of non-CUDA accelerators for large-scale evaluation. Two model scales are used: Qwen2.5-32B [21] for dense architectures and a 560B-parameter MoE model for production scale. We compare DORA against three representative paradigms spanning the constraint–efficiency landscape: (1) *Synchronous* (all constraints satisfied, batch-barrier limited); (2) *One-step off-policy* (overlaps stages but does not eliminate rollout bubbles); and (3) *Partial rollout in the colocated model placement* (eliminates bubbles by relaxing single-version generation, requiring algorithmic corrections). All baselines are implemented in the same in-house framework on identical hardware. Full hardware specs, software stack, dataset, and training hyperparameters are in Appendix B.

4.2 Training Performance

Rollout Acceleration. Figure 8 shows that DORA delivers consistent gains at both 64 and 128 GPUs, reducing rollout-only time by up to **8.2**× and delivering an end-to-end speedup of up to **1.93**×. These gains address the long-tail dilemma identified in Section 1: long-tailed trajectories continue under their original policy versions in dedicated DP groups, while trajectory-level streaming removes inter-instance waiting and new requests fill released capacity to eliminate intra-instance idle slots. Compared with Partial Rollout, the strongest long-tail-mitigating baseline, DORA remains **1.18**× faster end-to-end, owing to its zero-re-prefill migration (Section 3.4), which avoids the prefix recomputation incurred when unfinished trajectories resume after weight updates.

End-to-End Throughput. The step-time gains translate into proportional throughput improvements across both cluster scales. As shown in Figure 9, on 64 GPUs, DORA reaches **23,327 tokens/s**, a **1.65**× improvement over synchronous training and **1.17**× over partial rollout. The same pattern holds on 128 GPUs, where DORA achieves **34,135 tokens/s**, yielding **2.12**× and **1.11**× speedups over the two baselines, respectively. Notably, this throughput reflects the trajectories actually consumed by training. Both baselines waste accelerator time—one-step off-policy through long-tail idleness, partial rollout through re-prefill at each weight update—whereas DORA's multi-version streaming and zero-re-prefill migration convert all rollout duration into effective throughput.

Model Convergence. To assess whether DORA's efficiency gains affect convergence, we track the mean training reward over 100 steps on 72 GPUs (Figure 10). For a controlled comparison, both DORA and Partial Rollout use the same staleness bound, $K=3$. DORA closely follows the synchronous baseline throughout training and reaches a comparable late-stage reward, as highlighted

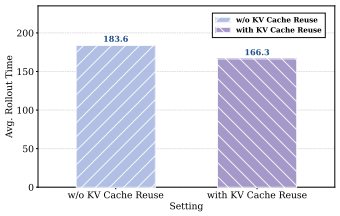


Figure 11: Average rollout time with and without KV-Cache reuse on Dense-32B.

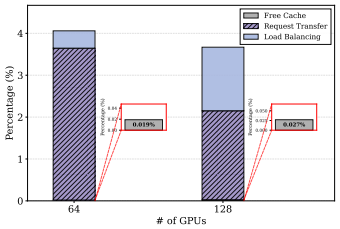


Figure 12: Breakdown of DORA's system overhead at 64 and 128 GPUs.

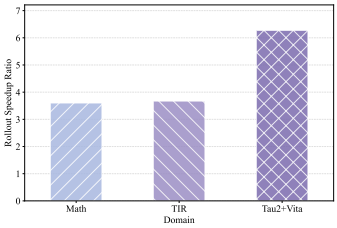


Figure 13: Production rollout time on a 560B MoE model (64K-token limit).

in the inset. In contrast, One-step off-policy and Partial Rollout exhibit lower late-stage rewards. These results indicate that DORA's multi-version streaming preserves convergence under bounded staleness while improving system efficiency.

4.3 Ablation Study

We conduct module-level ablations to quantify the contributions of KV-Cache reuse and dynamic orchestration. As shown in Figure 11, under the Dense-32B setting, enabling KV-Cache reuse reduces the average rollout time from 183 s to 166 s, corresponding to a **9%** reduction. To isolate dynamic orchestration, we retain multi-version streaming and all other settings but disable the orchestration loop on the 560B-parameter MoE model. The resulting variant requires **2.30**× the rollout time of the synchronous baseline, demonstrating that dynamic orchestration is essential for translating multi-version streaming into practical efficiency gains. Notably, we ablate dynamic orchestration as a whole because its resource partitioning, request migration, and staleness-aware scheduling are interdependent rather than standalone optimizations.

4.4 Overhead Analysis

We quantify three sources of overhead introduced by DORA's dynamic orchestration and KV-cache reuse: P2P load balancing, request transfer, and free cache. As shown in Figure 12, load balancing—including request monitoring, resource repartitioning, and P2P weight synchronization—accounts for only 0.4% and 1.5% of execution time on 64 and 128 GPUs, respectively. Request-transfer overhead decreases from 3.6% to 2.1% at scale, as higher throughput amortizes the migration cost. Free cache is negligible ($<$0.03% in both settings), indicating virtually no critical-path overhead from hierarchical memory management. Overall, the total overhead decreases slightly from approximately 4.0% to 3.6% when scaling from 64 to 128 GPUs. Thus, across the evaluated scale-up, DORA's aggregate orchestration overhead remains bounded and even decreases slightly, while enabling the throughput improvements reported above.

4.5 Production Deployment

We deploy DORA on 4,096 accelerators to train a $\sim$500B-parameter MoE model with response lengths of up to 64K tokens. Because evaluating every baseline at this scale is prohibitively expensive, we compare DORA against the well-tuned synchronous system used in production. As shown in Figure 13, DORA achieves a **3.6**× rollout speedup on mathematical and tool-integrated reasoning workloads and up to **6.2**× on agentic workloads from Tau2-bench [3] and Vita [11]. The larger gains on agentic workloads are consistent with DORA's design: longer and more skewed responses create larger rollout bubbles, allowing multi-version streaming and dynamic orchestration to recover more otherwise idle capacity.

Beyond system efficiency, we further evaluate the downstream capabilities of the model trained with DORA. As shown in Table 1, DORA-based RL substantially improves the SFT initialization across all three Tau2-bench domains, with particularly pronounced gains on Retail (70.7 to 88.6) and Telecom (47.6 to 99.3). Notably, the resulting model achieves competitive performance against frontier reasoning models, approaching Claude Opus 4.5 Thinking on Retail and surpassing both Claude Opus 4.5 Thinking and GPT-5.2 Thinking xhigh on Telecom. These results further demonstrate

DORA's practical effectiveness in supporting large-scale RL training with strong downstream task performance.

Table 1: Task performance of DORA-based RL and frontier models on Tau2-bench.

Method	Tau2-Retail	Tau2-Airline	Tau2-Telecom
SFT (cold start)	70.7	72.75	47.6
+ DORA RL	**88.6**	**76.5**	**99.3**
Claude Opus 4.5 Thinking	88.9	–	98.2
GPT-5.2 Thinking xhigh	82.0	–	98.7

4.6 Comparison with an Open-Source RL Framework

To control for implementation differences, all preceding baselines use the same in-house framework. We further compare DORA with Slime, a representative open-source LLM RL framework, under identical model, hardware, and training settings. As shown in Table 2, DORA achieves **34.0K tokens/s**, compared with 13.0K and 19.3K tokens/s for Slime's synchronous and Partial Rollout modes, corresponding to **2.6**$\times$ and **1.76**$\times$ improvements, respectively. Notably, while Partial Rollout alleviates long-tail waiting, it incurs repeated re-prefill overhead when switching policy versions. In contrast, DORA preserves single-policy trajectory generation and enables KV-Cache reuse across request migrations, avoiding this overhead. These results further validate DORA's efficiency advantages beyond our in-house implementation.

Table 2: Throughput comparison on Qwen2.5-Dense-32B under the same experimental configuration.

Framework	Paradigm	Throughput	DORA Speedup
Slime (v0.3.0)	Synchronous	13K tok/s	2.6$\times$
Slime (v0.3.0)	Partial rollout	19.3K tok/s	1.76$\times$
DORA (ours)	**Multi-version streaming**	**34K tok/s**	**– (ref.)**

5 Conclusion and Limitations

We present DORA, a scalable asynchronous RL system that resolves the long-tail dilemma in large-scale LLM post-training. By embodying a *multi-version rollout* paradigm, where multiple policy versions coexist and each trajectory is generated end-to-end under a single version, DORA eliminates the long-tail bubble without introducing algorithmic corrections. Furthermore, this single-policy generation yields KV-Cache equivalence, enabling zero-re-prefill migration during request relocation. Experiments demonstrate up to $8.2\times$ rollout speedup and $2.12\times$ end-to-end acceleration over synchronous training, validated by large-scale industrial deployments. Despite these significant efficiency gains, DORA has limitations that merit further investigation. Algorithmically, the staleness bound K requires manual configuration, relying on PPO's clipping to mitigate off-policy bias; incorporating adaptive staleness control or explicit delay compensation could further optimize the convergence–throughput tradeoff. Experimentally, our evaluations were conducted within a controlled in-house framework, and MoE scaling was validated primarily on production data and cluster. Future efforts will address this by benchmarking directly against public systems such as veRL [27] and AReaL [6], alongside extensive evaluations on more open-source MoE models.

Acknowledgements

We are grateful to the anonymous reviewers and the area chair for their insightful comments and suggestions. This work was primarily supported by the Meituan LongCat team. The work of Yuchun Miao was supported by the National Natural Science Foundation of China under Grant 625B2134.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 22–31. PMLR, 2017.

[2] Anthropic. Introducing claude opus 4.5, 2025. URL https://www.anthropic.com/news/claude-opus-4-5.

[3] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025.

[4] Pritam Damania, Shen Li, Alban Desmaison, Alisson Azzolini, Brian Vaughan, Edward Yang, Gregory Chanan, Guoqiang Jerry Chen, Hongyi Jia, Howard Huang, et al. Pytorch rpc: Distributed deep learning built on tensor-optimized remote procedure calls. *Proceedings of Machine Learning and Systems*, 5: 219–231, 2023.

[5] Dong Du, Shulin Liu, Tao Yang, Shaohua Chen, and Yang Li. Ulorl: An ultra-long output reinforcement learning approach for advancing large language models' reasoning abilities. *arXiv preprint arXiv:2507.19766*, 2025.

[6] Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

[7] Wei Gao, Yuheng Zhao, Dakai An, Tianyuan Wu, Lunxi Cao, Shaopan Xiong, Ju Huang, Weixun Wang, Siran Yang, Wenbo Su, et al. Rollpacker: Mitigating long-tail rollouts for fast, synchronous rl post-training. *arXiv preprint arXiv:2509.21009*, 2025.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[9] Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. Asyncflow: An asynchronous streaming rl framework for efficient llm post-training. *arXiv preprint arXiv:2507.01663*, 2025.

[10] Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*, 2025.

[11] Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. *CoRR*, abs/2509.26490, 2025.

[12] Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

[13] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[14] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 267–274, 2002.

[15] 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 ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[16] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in neural information processing systems*, 28, 2015.

[17] Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, et al. Deepcoder: A fully open-source 14b coder at o3-mini level. *Notion Blog*, 1, 2025.

[18] Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. *arXiv preprint arXiv:2410.18252*, 2024.

[19] OpenAI. Introducing openai o1, 2024. URL https://openai.com/o1/.

[20] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Transactions on Storage*, 2024.

[21] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1889–1897. PMLR, 2015.

[23] ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

[24] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[25] Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *CoRR*, abs/2511.22570, 2025.

[26] Guangming Sheng, Yuxuan Tong, Borui Wan, Wang Zhang, Chaobo Jia, Xibin Wu, Yuqi Wu, Xiang Li, Chi Zhang, Yanghua Peng, et al. Laminar: A scalable asynchronous rl post-training framework. *arXiv preprint arXiv:2510.12633*, 2025.

[27] 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*, pages 1279–1297, 2025.

[28] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[29] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[30] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[31] Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chengcheng Han, Chenhui Yang, Chi Zhang, et al. Introducing longcat-flash-thinking: A technical report. *arXiv preprint arXiv:2509.18883*, 2025.

[32] Meituan LongCat Team, Bei Li, Bingye Lei, Bo Wang, Bolin Rong, Chao Wang, Chao Zhang, Chen Gao, Chen Zhang, Cheng Sun, et al. Longcat-flash technical report. *arXiv preprint arXiv:2509.01322*, 2025.

[33] Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chen Gao, Chen Zhang, Chengcheng Han, et al. Longcat-flash-thinking-2601 technical report. *arXiv preprint arXiv:2601.16725*, 2026.

[34] Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, et al. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm training. *arXiv preprint arXiv:2505.24034*, 2025.

[35] Youshao Xiao, Zhenglei Zhou, Fagui Mao, Weichang Wu, Shangchun Zhao, Lin Ju, Lei Liang, Xiaolu Zhang, and Jun Zhou. An adaptive placement and parallelism framework for accelerating rlhf training. *arXiv preprint arXiv:2312.11819*, 2023.

[36] Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, et al. Deepspeed-chat: Easy, fast and affordable rlhf training of chatgpt-like models at all scales. *arXiv preprint arXiv:2308.01320*, 2023.

[37] 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. *arXiv preprint arXiv:2503.14476*, 2025.

[38] Yiqi Zhang, Huiqiang Jiang, Xufang Luo, Zhihe Yang, Chengruidong Zhang, Yifei Shen, Dongsheng Li, Yuqing Yang, Lili Qiu, and Yang You. Sortedrl: Accelerating rl training for llms through online length-aware scheduling. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

[39] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International conference on machine learning*, pages 4120–4129. PMLR, 2017.

[40] Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, et al. Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation. *arXiv preprint arXiv:2504.15930*, 2025.

[41] Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Related Work

To accelerate reinforcement learning (RL) post-training for large language models, various distributed training systems and asynchronous strategies have been proposed. Previous RL training systems [36, 35, 12, 27] primarily focus on model placement and scheduling within a synchronous training paradigm. Synchronous training cycles sequentially through rollout, experience preparation, and model training within each step. While this provides clean algorithmic semantics by enforcing a strict batch barrier between rollout and training, it suffers from severe batch-barrier idle time—especially in long-context and complex reasoning scenarios where the generation length is highly skewed. To alleviate this rollout bottleneck, recent efforts have shifted towards asynchronous and semi-asynchronous paradigms, which can be broadly categorized into three main directions.

Replication-based (Oversampling) Methods. Replication-based or oversampling methods [7, 38] attempt to shorten the rollout duration by over-provisioning generation prompts. Specifically, they dispatch a larger number of rollout requests than the required training batch size and simply discard the in-flight long trajectories once enough completed responses are collected. While this approach effectively reduces the long-tail latency, it fundamentally compromises data integrity. The discarded trajectories often contain the lengthy chain-of-thought reasoning steps that are crucial for developing emergent agentic capabilities. Furthermore, the resulting length-biased distribution distorts advantage estimation, particularly in group-relative algorithms like GRPO [24], whereas DORA preserves every sampled trajectory without algorithmic distortion.

One-step Off-policy Methods. K-step off-policy methods [17, 40, 18, 10, 9]—most commonly one-step off-policy—overlap the rollout and training stages temporally. They achieve this by allowing the current step's rollout to use behavior policy weights from the previous iteration while the training stage updates the target policy. Although this design successfully hides the pipeline bubble between the rollout and training stages, it does not address the fundamental duration of the rollout phase itself. Both intra-node bubbles (idle slots within a device) and inter-node bubbles (faster instances waiting for the slowest) persist under highly skewed workload distributions. In contrast, DORA utilizes a multi-version streaming mechanism that completely breaks the synchronous batch barrier, directly resolving these intra- and inter-node inefficiencies.

Partial-rollout Methods. Partial-rollout methods [30, 34, 5, 6, 41] mitigate the long-tail issue by segmenting lengthy responses into smaller chunks at each weight update. When a new policy version becomes available, ongoing trajectories are resumed and continued under the latest weights. This approach effectively eliminates device idleness but introduces two significant challenges. First, from an algorithmic perspective, a single trajectory is now stitched together from multiple distinct policy versions. This departs from the standard RL formulation and necessitates complex algorithmic corrections, such as masking earlier segments during loss computation [30] or applying decoupled PPO objectives [6] to maintain convergence. Second, from a system perspective, every weight update invalidates the KV-Cache for the ongoing trajectories, forcing a full re-prefill of the context. This re-prefill overhead grows dramatically with context length and is particularly prohibitive in MoE architectures. DORA bypasses both challenges by ensuring each trajectory is generated end-to-end under a single policy version, preserving standard algorithmic semantics and enabling zero-re-prefill migration.

Concurrently with our work, several systems [23, 26] have explored multi-version streaming training concepts. However, these systems rely on two-tier CPU relay architectures for weight management and employ significantly different mechanisms for workload orchestration and staleness control compared to DORA's centralized load-balancing orchestrator and zero-re-prefill migration.

B Experimental Setup

Testbed. Our experiments are conducted on a cluster consisting of 16 nodes, each equipped with 8 NVIDIA H800 GPUs. Intra-node communication is facilitated by NVLink with a bandwidth of 400 GB/s, while inter-node connectivity is provided by 8×400 Gbps network interfaces. Additionally, our production cluster employs non-CUDA accelerators, each providing approximately 60 GB of available device memory.

Models and Metrics. Our experiments use Qwen2.5-32B [21] for dense architectures and a around 500B open-source MoE model for MoE architectures. We measure *end-to-end throughput* (tokens/s), calculated as the total tokens (prompts and responses) processed per second, and *average step time* (min), which represents the wall-clock time per RL iteration. All reported numbers are averaged over five RL iterations after the warm-up phase to reflect steady-state performance.

Datasets. We utilize the "DAPO-Math-17k" dataset for training, with the maximum input and output sequence lengths set to 2K and 30K tokens, respectively.

Baselines and Implementation. We compare DORA against three representative RL training paradigms: (1) *Synchronous (All-Colocated)*, which satisfies all constraints but suffers from batch barriers; (2) *One-step off-policy*, which satisfies all constraints with staleness $K{=}1$ but only overlaps pipeline stages without eliminating rollout bubbles; and (3) *Partial rollout* in All-colocated implementation similar to the work [30]. All baselines are implemented within the same in-house RL framework to ensure a controlled comparison under identical hardware and software configurations. Our RL system uses vLLM [15] as the inference engine, Megatron-LM [28] as the training backend, and extends torch RPC [4] with streaming primitives. The software environment includes CUDA-12.4, PyTorch-2.6.0, vLLM-0.8.5, and NCCL-2.28.

Training Configurations. For the RL algorithm, we follow the setting of DAPO [37], a variant of GRPO. Each rollout consists of a prompt batch size of 512, with 16 responses sampled per prompt, resulting in a global training batch size of 8,192. Each training iteration involves 16 update steps with a micro-batch size of 512. To stress-test efficiency under realistic long-tailed generation, we select an intermediate checkpoint where the mean response length is 2.4K tokens and the maximum reaches 30K.

C Case Study: Empirical Validation of Multi-Version Streaming

This part presents a detailed empirical case study that validates the algorithmic properties of DORA's multi-version streaming training. Using rollout logs spanning 22,820 records (365,120 responses) over training steps 20–100 with staleness bound $K{=}3$, we examine three modes: *DORA*, *Partial Rollout*, and *Synchronous* training. Each record contains 16 responses scored by a binary reward model ($+1/-1$).

C.1 Observation 1: Bounded Staleness Incurs Nearly Zero Quality Degradation

A natural concern with asynchronous training is that stale data degrades learning. We perform a controlled analysis to disentangle staleness from confounding variables.

Naïve observation. Aggregating across all records, staleness-0 data achieves a 58.9% pass rate versus 55.6% for staleness-1, an apparent 3.3 percentage point gap.

Controlled analysis. This gap is entirely explained by *selection bias*: harder problems produce longer responses, which take more time to generate and thus receive higher staleness labels. Table 3 stratifies records by difficulty (pass rate bin) and compares staleness-0 versus staleness-1 *within each stratum*:

Table 3: Pass rate (%) by staleness, stratified by problem difficulty (DORA, $K{=}3$). The staleness gap vanishes after controlling for difficulty, confirming zero quality degradation.

Difficulty Bin	**s=0 (%)**	**n (s=0)**	**s=1 (%)**	**n (s=1)**	**Δ (%)**
Hard (PR <0.25)	10.93	187	11.27	667	$+0.34$
Medium (0.25–0.5)	34.42	146	34.39	580	-0.03
Medium-Easy (0.5–0.75)	60.67	184	60.05	625	-0.62
Easy (PR >0.75)	86.55	441	86.55	1263	$+0.00$

We further verify this at the per-response level using Partial Rollout's generation segments (Table 4), which provides 141,632 responses with explicit version labels:

Mechanism. Figure 14(b) confirms the causal pathway: staleness-1 responses have a median token count of 1,495 versus 1,015 for staleness-0 (47% longer). Longer generation implies harder problems;

Table 4: Per-response staleness analysis (Partial Rollout, n=141,632). Within each difficulty bin, staleness-0 and staleness-1 responses perform identically.

Pass Rate Bin	**s=0 (%)**	**s=1 (%)**	**Δ (%)**	**n (total)**
$[0, 0.125)$	6.25	6.25	0.00	14,384
$[0.125, 0.25)$	15.43	15.28	−0.15	17,776
$[0.25, 0.5)$	34.04	33.86	−0.17	25,712
$[0.5, 0.75)$	60.18	59.85	−0.33	28,880
$[0.75, 0.875)$	78.37	78.63	+0.26	19,344
$[0.875, 1.0]$	91.37	91.28	−0.10	35,520

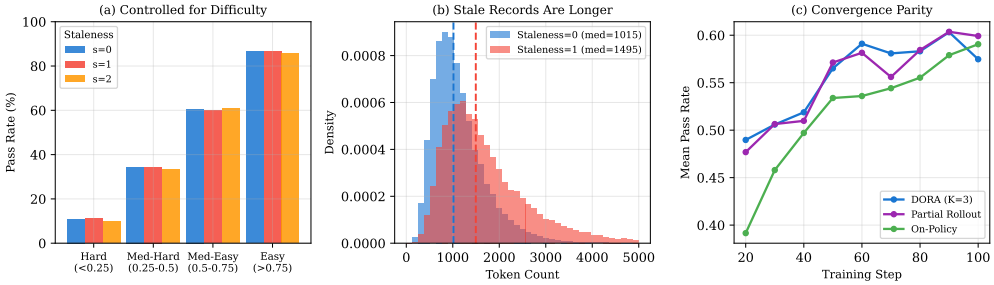


Figure 14: (a) After controlling for problem difficulty, the staleness gap vanishes ($\Delta < 0.6\%$ across all bins). (b) Selection bias mechanism: stale records are systematically longer (harder). (c) All three paradigms converge to the same pass-rate range (0.57–0.60).

under DORA's asynchronous scheduling, these naturally complete after a weight update and receive a higher staleness label. The *on-policy* improvement rate is only 0.113%/step (measured from on-policy data over steps 50–100), so the theoretical maximum penalty for K=3 is merely $3 \times 0.113 = 0.34\%$, well below sampling noise. This validates that DORA's sliding-window staleness control (Section 3.2) is sufficient for convergence parity.

C.2 Observation 2: Long-Tail Trajectories Carry the Highest Training Signal

DORA's trajectory-level streaming preserves every sampled trajectory (Section 3.2), in contrast to replication-based methods that discard long-tail samples once enough short ones complete. We validate the importance of this design by measuring the GRPO gradient signal, quantified as within-group reward variance, across difficulty levels:

Table 5: Reward variance (gradient signal strength for GRPO) by difficulty. Medium-difficulty problems with the highest variance are also the longest and most likely to be stale, i.e., exactly the trajectories that replication-based methods discard first.

Difficulty	**n**	**Reward Var.**	**Avg Length**	**Avg Staleness**
Hard (1–3/16 correct)	1,007	0.409	3,755	0.99
Medium-Hard (4–7/16)	868	**0.939**	3,624	1.01
Medium-Easy (8–11/16)	950	**1.001**	3,418	0.96
Easy (12–14/16)	1,206	0.617	3,025	0.90
Very Easy (15–16/16)	721	0.250	2,711	0.87

The medium-difficulty records (pass rate 0.25–0.75) exhibit the highest reward variance ($\sim$1.0), providing the strongest gradient signal. These records are simultaneously the longest (3,400–3,800 chars) and the most stale (staleness 0.96–1.01). In a replication-based system that terminates generation once TBS short trajectories are collected, these high-signal samples would be the first to be discarded. DORA preserves all of them via multi-version streaming, ensuring the integrity of advantage estimation in GRPO.

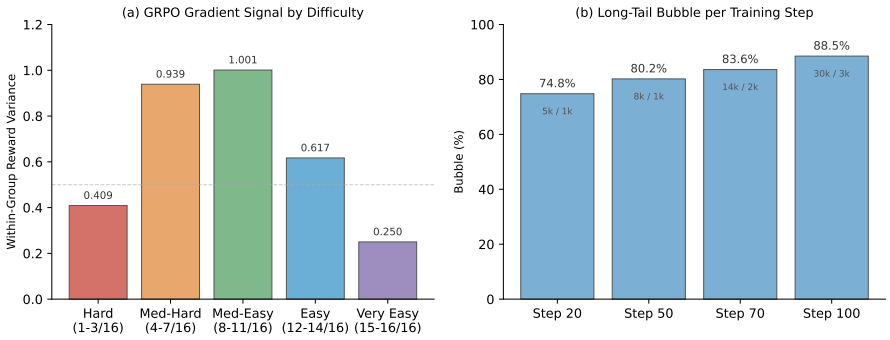


Figure 15: (a) GRPO gradient signal peaks at medium difficulty (pass rate 0.25–0.75). (b) Under synchronous training, the inter-node bubble exceeds 75% across all steps and grows to 88.5% at step 100 as responses lengthen.

C.3 Observation 3: Quantifying the Long-Tail Bubble

We directly measure the inter-node bubble that DORA could theoretically eliminate from the view of the request workloads. For each training step, we compute the fraction of decode time wasted waiting for the longest trajectory:

$$\text{Bubble} = \frac{\max_i L_i - \overline{L_i}}{\max_i L_i}$$

As shown in Figure 15(b), the bubble exceeds 74% at step 20 and grows to **88.5%** at step 100 (max response: 30,720 tokens *vs.* mean: 3,536). The max-to-median token ratio reaches **28**×, consistent with the order-of-magnitude skew reported in Section 2.2. This quantifies the efficiency opportunity that DORA captures through multi-version streaming: by allowing the longest trajectories to continue under their legacy version while new requests run on the latest policy model, DORA could theoretically eliminate this 75–89% bubble entirely from the viewpoint of engineering only.

C.4 Summary

This case study empirically characterizes DORA's multi-version streaming training along three axes. Two concern its algorithmic design principles, and one quantifies the efficiency opportunity that motivates the design.

- **Bounded staleness (design principle).** After controlling for problem difficulty, staleness ≤ 3 produces nearly zero measurable quality degradation ($\Delta < 0.6\%$ across all strata); the apparent staleness gap is entirely explained by selection bias.
- **Preserving every sampled trajectory (design principle).** The longest, most stale trajectories carry the highest GRPO gradient signal (reward variance $\sim$1.0). Discarding them, as replication-based methods do, removes the most valuable training data.
- **Quantifying the long-tail bubble (motivation).** Under synchronous training the inter-node bubble reaches 75–89%, confirming that the inefficiency DORA targets is substantial.

Together, these observations support DORA's design goal of resolving the long-tail dilemma without introducing algorithmic corrections.

D Policy Improvement Bounds for Multi-Version Rollouts

We analyze the policy-improvement property of DORA's multi-version trajectory sampling and show how its maximum-staleness constraint controls the resulting distribution-mismatch term.

D.1 Setup and Notation

Let π_θ denote the current policy being optimized and $\{\pi_{w_i}\}$ the behavior-policy versions that generated the trajectories in a training batch. Let α_i be the proportion of trajectories generated by π_{w_i}, where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Each trajectory is generated entirely under one behavior version.

We follow the standard infinite-horizon discounted setting with $\gamma \in (0, 1)$ and bounded rewards. All policies share the same environment, reward function, and initial prompt distribution. For any policy π, define

$$J(\pi) := \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \tag{4}$$

and the normalized discounted state distribution

$$d_\pi(s) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid \pi). \tag{5}$$

Let $A^\pi(s, a)$ denote the advantage function under π. We assume common action support: whenever $\pi_\theta(a \mid s) > 0$, $\pi_{w_i}(a \mid s) > 0$ for every included behavior version and every state. Thus, the importance ratio $\pi_\theta(a \mid s)/\pi_{w_i}(a \mid s)$ is well defined.

For each behavior version, the performance-difference lemma [14, Lemma 6.1] gives

$$J(\pi_\theta) - J(\pi_{w_i}) = \frac{1}{1-\gamma}\mathbb{E}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_\theta(\cdot|s)}}\left[A^{\pi_{w_i}}(s, a)\right] \tag{6}$$

$$= \frac{1}{1-\gamma}\mathbb{E}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_{w_i}(\cdot|s)}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{w_i}(a \mid s)} A^{\pi_{w_i}}(s, a)\right]. \tag{7}$$

In Eq. (7), the action expectation is now taken under the behavior policy. Following the TRPO surrogate construction [22], we define a local surrogate by evaluating the same importance-weighted advantage under the behavior-state distribution $d_{\pi_{w_i}}$:

$$L_i(\pi_\theta) := \frac{1}{1-\gamma}\mathbb{E}_{\substack{s\sim d_{\pi_{w_i}}\\ a\sim\pi_{w_i}(\cdot|s)}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{w_i}(a \mid s)} A^{\pi_{w_i}}(s, a)\right]. \tag{8}$$

This replacement is an approximation rather than an equality: $L_i(\pi_\theta)$ uses $d_{\pi_{w_i}}$ in place of d_{π_θ}. The resulting state-distribution mismatch is retained explicitly in the analysis below.

D.2 Policy Improvement under Multi-Version Trajectory Sampling

Theorem 1 (Multi-version policy-improvement bound) *Define*

$$C_{\theta,i} := \sup_s \left|\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[A^{\pi_{w_i}}(s, a)\right]\right|. \tag{9}$$

Then

$$J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}) \geq \sum_i \alpha_i L_i(\pi_\theta) - \frac{2\gamma}{(1-\gamma)^2}\sum_i \alpha_i C_{\theta,i}\mathbb{E}_{s\sim d_{\pi_{w_i}}}\left[D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s)\right], \tag{10}$$

where $D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s)$ is the total variation distance between their action distributions at state s.

Proof 1 *For each behavior version i, define*

$$g_{\theta,i}(s) := \mathbb{E}_{a\sim\pi_{w_i}(\cdot|s)}\left[\frac{\pi_\theta(a \mid s)}{\pi_{w_i}(a \mid s)} A^{\pi_{w_i}}(s, a)\right]. \tag{11}$$

By importance sampling,

$$g_{\theta,i}(s) = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[A^{\pi_{w_i}}(s, a)\right], \qquad |g_{\theta,i}(s)| \leq C_{\theta,i}. \tag{12}$$

Using Eqs. (7) and (8),

$$J(\pi_\theta) - J(\pi_{w_i}) - L_i(\pi_\theta) = \frac{1}{1-\gamma} \sum_s \left(d_{\pi_\theta}(s) - d_{\pi_{w_i}}(s)\right) g_{\theta,i}(s)$$
$$\geq -\frac{C_{\theta,i}}{1-\gamma} \left\| d_{\pi_\theta} - d_{\pi_{w_i}} \right\|_1 . \tag{13}$$

By Achiam et al. [1, Supplementary Appendix 10.1, Lemma 3, Eq. (25)], the normalized discounted state distributions of any two policies π' and π satisfy

$$\|d_{\pi'} - d_\pi\|_1 \leq \frac{2\gamma}{1-\gamma} \mathbb{E}_{s\sim d_\pi} \left[D_{\mathrm{TV}}(\pi', \pi; s) \right] . \tag{14}$$

Applying Eq. (14) with $\pi' = \pi_\theta$ and $\pi = \pi_{w_i}$ in Eq. (13) gives

$$J(\pi_\theta) - J(\pi_{w_i}) \geq L_i(\pi_\theta)$$
$$- \frac{2\gamma C_{\theta,i}}{(1-\gamma)^2} \mathbb{E}_{s\sim d_{\pi_{w_i}}} \left[D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \right] . \tag{15}$$

Because $\alpha_i \geq 0$, multiplying Eq. (15) by α_i preserves its direction. Summing the resulting inequalities over all behavior versions gives

$$\sum_i \alpha_i \left[J(\pi_\theta) - J(\pi_{w_i}) \right]$$
$$\geq \sum_i \alpha_i L_i(\pi_\theta) - \frac{2\gamma}{(1-\gamma)^2} \sum_i \alpha_i C_{\theta,i} \mathbb{E}_{s\sim d_{\pi_{w_i}}} \left[D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \right] . \tag{16}$$

The left-hand side of Eq. (16) satisfies

$$\begin{aligned} \sum_i \alpha_i \left[J(\pi_\theta) - J(\pi_{w_i}) \right] &= \left(\sum_i \alpha_i \right) J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}) \\ &= J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}), \end{aligned} \tag{17}$$

where the second equality uses $\sum_i \alpha_i = 1$. Substituting Eq. (17) into Eq. (16) therefore gives

$$J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}) \geq \sum_i \alpha_i L_i(\pi_\theta)$$
$$- \frac{2\gamma}{(1-\gamma)^2} \sum_i \alpha_i C_{\theta,i} \mathbb{E}_{s\sim d_{\pi_{w_i}}} \left[D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \right] ,$$

which is Eq. (10).

D.3 Policy Improvement under Bounded Staleness

Let $v(\theta)$ and $v(w_i)$ denote the update indices of the current policy and behavior version, respectively. DORA enforces

$$0 \leq v(\theta) - v(w_i) \leq K. \tag{18}$$

Here $v(\theta) - v(w_i)$ counts every policy transition from the behavior snapshot to the current learner iterate, including the final in-flight learner update. To relate this version difference to policy divergence, suppose that every pair of consecutive policies along this update path satisfies

$$\sup_s D_{\mathrm{TV}}(\pi^{(j+1)}, \pi^{(j)}; s) \leq \delta. \tag{19}$$

The path includes the current optimization iterate π_θ as its final endpoint.

Corollary 1 (Staleness-controlled policy-improvement bound) *Under Eqs. (18) and (19),*

$$J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}) \geq \sum_i \alpha_i L_i(\pi_\theta)$$
$$- \frac{2\gamma K \delta}{(1-\gamma)^2} \max_i C_{\theta,i}. \tag{20}$$

Proof 2 *For a behavior version π_{w_i}, let $\ell_i = v(\theta) - v(w_i)$. By Eq. (18), $\ell_i \le K$. Denote the policies along this path by $\pi^{(0)} = \pi_{w_i}, \dots, \pi^{(\ell_i)} = \pi_\theta$. Applying the triangle inequality along its ℓ_i-step update path gives, for every state s,*

$$\begin{aligned} D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) &\le \sum_{j=0}^{\ell_i - 1} D_{\mathrm{TV}}(\pi^{(j+1)}, \pi^{(j)}; s) \\ &\le \ell_i \delta \le K\delta. \end{aligned} \tag{21}$$

It follows that

$$\begin{aligned} &\sum_i \alpha_i C_{\theta,i} \mathbb{E}_{s\sim d_{\pi_{w_i}}} D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \\ &\quad \le K\delta \sum_i \alpha_i C_{\theta,i} \le K\delta \left(\max_i C_{\theta,i}\right) \sum_i \alpha_i = K\delta \max_i C_{\theta,i}, \end{aligned} \tag{22}$$

where the last equality uses $\sum_i \alpha_i = 1$. Since $2\gamma/(1-\gamma)^2 > 0$, Eq. (22) implies

$$\begin{aligned} &-\frac{2\gamma}{(1-\gamma)^2} \sum_i \alpha_i C_{\theta,i} \mathbb{E}_{s\sim d_{\pi_{w_i}}} D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \\ &\quad \ge -\frac{2\gamma K\delta}{(1-\gamma)^2} \max_i C_{\theta,i}. \end{aligned} \tag{23}$$

Applying Eq. (23) to the right-hand side of Theorem 1 gives the complete inequality chain

$$\begin{aligned} J(\pi_\theta) - \sum_i \alpha_i J(\pi_{w_i}) &\ge \sum_i \alpha_i L_i(\pi_\theta) - \frac{2\gamma}{(1-\gamma)^2} \sum_i \alpha_i C_{\theta,i} \mathbb{E}_{s\sim d_{\pi_{w_i}}} D_{\mathrm{TV}}(\pi_\theta, \pi_{w_i}; s) \\ &\ge \sum_i \alpha_i L_i(\pi_\theta) - \frac{2\gamma K\delta}{(1-\gamma)^2} \max_i C_{\theta,i}, \end{aligned}$$

which is Eq. (20).

D.4 Discussion and Design Implications

Corollary 1 connects DORA's system-level staleness control to its policy-improvement guarantee. Under the per-update drift condition, limiting each behavior policy to at most K updates behind the current policy bounds the distribution-mismatch penalty by

$$\frac{2\gamma K\delta}{(1-\gamma)^2} \max_i C_{\theta,i}. \tag{24}$$

Consequently, if the weighted surrogate improvement satisfies

$$\sum_i \alpha_i L_i(\pi_\theta) \ge \frac{2\gamma K\delta}{(1-\gamma)^2} \max_i C_{\theta,i}, \tag{25}$$

then Eq. (20) guarantees

$$J(\pi_\theta) \ge \sum_i \alpha_i J(\pi_{w_i}). \tag{26}$$

Thus, maximum staleness converts a system-level scheduling constraint into a sufficient condition for policy improvement: whenever the weighted surrogate improvement covers the worst-case staleness penalty, the current policy has return no lower than the weighted average return of the behavior policies represented in the training batch.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction sections of the paper clearly state the research objectives, methodologies, key findings, and conclusions, providing a coherent overview of the core contributions of the work. Moreover, the paper complements these claims with detailed experimental setups, results, and analyses that are consistent with the statements made in the abstract and introduction, ensuring that the readers have a comprehensive understanding of the work's scope and contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed the limitations of our work via the algorithm (lack of adaptive staleness control) and experiment aspects (lack of more extensive MoE experiments and open–source framework due to resource limitations).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We have provided a complete theoretical proof in Appendix **??**.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in the appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have provided the fully open-source benchmarks on the open-source dataset and open-source models, which are reproducible.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in the supplemental material?

Answer: [No]

Justification: Our code implementation is currently confidential, but our idea and justifications are clear for reproduce.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental data, together with all parameter-related details, are provided in Appendix B. The data were obtained from the open-source community, and—except for the parameters specific to DORA—the algorithmic hyperparameters were kept consistent with those used in the corresponding open-source experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: While standard error bars may not be featured on the primary system performance graphs (such as throughput), we provide average results to support its core claims. Additionally, Appendix C.2 provides a controlled statistical analysis of staleness, concluding that performance gaps are "well below sampling noise".

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper explicitly details the computing resources utilized for the experiments in Section 4.1 and Appendix B. We specify using a 16-node cluster where each node is equipped with 8 NVIDIA H800 GPUs, noting communication bandwidths like 400 GB/s for NVLink and 8×400 Gbps for inter-node connectivity. Furthermore, we describe the production cluster as using non-CUDA accelerators with approximately 60 GB of device memory each (The specific device type is currently confidential according to our policy). Finally, execution time is documented via the "average step time (min)" metric, which measures the wall-clock time per RL iteration.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: The research conducted in this paper adheres to the NeurIPS Code of Ethics. The study focuses on improving the systems-level efficiency of reinforcement learning for large language models and does not involve human subjects, crowdsourcing, or the collection of sensitive personal data. All experiments were performed using standard open-source benchmarks and datasets, and the authors have ensured transparency in reporting computational resources and theoretical assumptions.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper has discussed potential societal impacts primarily through the lens of computational efficiency and environmental sustainability in subsection **??**. By significantly improving the throughput and acceleration of large language model (LLM) training (up to 2.12x end-to-end improvement), the work directly reduces the energy consumption and carbon footprint associated with large-scale industrial AI deployments. While we do not explicitly detail "negative" societal impacts in a dedicated section, we acknowledge potential risks by adhering to the NeurIPS Code of Ethics and focuses on systems-level optimizations that enhance the accessibility and efficiency of machine learning.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper presents DORA, a systems-level optimization for the reinforcement learning training process of Large Language Models (LLMs). The primary contribution is a technical infrastructure for improving training throughput and resource orchestration rather than the release of a new pre-trained model or a novel high-risk dataset. Our experiments were conducted using existing open-source benchmarks, such as DAPO-Math-17K, and established open-source models. As the work does not involve the release of new models or data with a high risk for misuse beyond what is already standard in the open-source community, specific additional safeguards for release are not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper utilizes existing open-source assets for its experimental evaluation, and these are properly credited through citations and explicit mentions in the text. The authors use the DAPO-Math-17K dataset, which is cited and described in Section 4.1. The experiments involve widely recognized open-source architectures, such as Qwen and

large-scale MoE models, with citations provided for the respective original works. The paper builds upon and compares against established methods like GRPO and PPO, citing the foundational papers that introduced these techniques.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The code and system implementation for DORA are currently proprietary and are not being released at this time.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This paper does not involve human subjects, clinical trials, or the collection of sensitive personal data.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.