# ECHO-2: A Large-Scale Distributed Rollout Framework for Cost-Efficient Reinforcement Learning

**Jingwei Song**[1,3*], **Meng Chen**[2,*], **Jie Xiao**[3,*], **Qingnan Ren**[3,*], **Jiaqi Huang**[3], **Yangshen Deng**[4], **Chris Tong**[3], **Wanyi Chen**[5], **Suli Wang**[6], **Zhisheng Chen**[7], **Ziqian Bi**[3], **Shuo Lu**[3], **Yiqun Duan**[3], **Xu Wang**[3], **Rymon Yu**[3], **Ween Yang**[3], **Lynn Ai**[3], **Eric Yang**[3], **Tianyu Shi**[8,†]

[1]The University of Hong Kong, [2]Fudan University, [3]Gradient, [4]University of Edinburgh, [5]Soochow University, [6]Technical University of Darmstadt, [7]University of the Chinese Academy of Sciences, [8]McGill University

songjingwei@connect.hku.hk, tianyu.shi3@mcgill.ca

[*]Equal contribution, [†]Corresponding author

## Abstract

Reinforcement learning (RL) is a critical stage in post-training large language models (LLMs), involving repeated interaction between rollout generation, reward evaluation, and centralized learning. Distributing rollout execution offers opportunities to leverage more cost-efficient inference resources, but introduces challenges in wide-area coordination and policy dissemination. We present ECHO-2, a distributed RL framework for post-training with remote inference workers and non-negligible dissemination latency. ECHO-2 combines centralized learning with distributed rollouts and treats bounded policy staleness as a user-controlled parameter, enabling rollout generation, dissemination, and training to overlap. We introduce an overlap-based capacity model that relates training time, dissemination latency, and rollout throughput, yielding a practical provisioning rule for sustaining learner utilization. To mitigate dissemination bottlenecks and lower cost, ECHO-2 employs peer-assisted pipelined broadcast and cost-aware activation of heterogeneous workers. Experiments on GRPO post-training of LLMs ranging from 4B to 32B parameters under real wide-area bandwidth regimes show that ECHO-2 significantly improves cost efficiency while preserving RL reward comparable to strong baselines.

## 1 Introduction

Reinforcement learning (RL) has become a central component of post-training for large language models (LLMs), driving improvements in reasoning, tool use, safety alignment, and preference optimization at scale [Team et al., 2025a, Guo et al., 2025, Shao et al., 2024]. While modern algorithms such as PPO [Schulman et al., 2017] and GRPO [Shao et al., 2024] have advanced stability and sample efficiency, the system design of LLM RL remains largely conventional: most pipelines co-locate learners and rollout workers inside a data center and run tightly coupled iteration cycles [Sheng et al., 2025b].

This design conflicts with the cost structure of contemporary RL workloads. Rollout generation dominates wall-clock time and often leaves the learner intermittently idle [Sheng et al., 2025b, Fu et al., 2025, He et al., 2025], yet rollouts—which consist primarily of forward passes and reward evaluation—are executed on the same expensive GPU clusters used for learning. Recent asynchronous systems [Noukhovitch et al., 2024, Fu et al., 2025, He et al., 2025, Zhong et al., 2025, Xiao et al., 2025] relax strict synchronization to improve utilization, but still assume rollout workers and learners

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

share an administrative domain with provisioned interconnects [Zhong et al., 2025]; they do not change the underlying cost structure. Distributed inference resources, in contrast, are abundant and cheap—geographically distributed cloud instances or opportunistic compute [Borzunov et al., 2023, Ryabinin et al., 2023]—and RL training is agnostic to where trajectories originate. However, naively extending centralized designs to such settings yields severe inefficiencies (training bubbles, overprovisioning) due to heterogeneous throughput, wide-area latency, and dynamic availability.

This motivates our research question: *how can we reduce the cost of RL post-training by offloading rollout generation to distributed inference resources, while keeping a centralized learner continuously utilized?*

We present ECHO-2, a distributed RL framework built on a simple architectural principle: *centralized learning with distributed rollouts.* Policy optimization runs on a small, stable set of data-center GPUs, while rollout generation is offloaded to a heterogeneous pool of Parallax [Tong et al., 2025] inference workers connected over wide-area networks. ECHO-2 achieves cost-efficient operation through two complementary mechanisms. First, it adopts a **bounded-staleness execution model** [Fu et al., 2025, He et al., 2025, Zhou et al., 2025]: the learner consumes rollouts whose policy lags by at most $S$ training steps, where $S$ is a user-specified staleness budget. This temporal slack absorbs wide-area latency and lets rollout generation, dissemination, and training overlap without stalling the learner. Second, ECHO-2 treats **policy dissemination** as an engineered primitive: workers organized into peer-forwarding chains immediately relay newly received snapshots and begin generating rollouts as soon as installation completes, leveraging aggregate fleet bandwidth to reduce tail broadcast latency. Together, these mechanisms turn $S$ from an artifact to be minimized into a *system-level control knob* that trades rollout cost against training stability.

This perspective reduces a hard synchronization constraint to a provisioning problem: given $S$, how much aggregate rollout capacity is required to saturate the learner? ECHO-2 answers this with a closed-form rule relating measurable training time, dissemination latency, and per-worker throughput to the required capacity. Evaluated on GRPO post-training of 4B and 8B models across distributed rollout pools and wide-area bandwidth regimes, ECHO-2 substantially reduces end-to-end training cost while matching the RL quality of strong centralized baselines.

**In summary, our contributions are:**

- **A distributed inference RL architecture for cost-efficient post-training.** We propose a system architecture that separates centralized learning from distributed rollout inference, enabling RL post-training to reduce cost by offloading rollout generation from data-center GPU clusters to distributed resources.
- **Overlap-aware execution and peer-assisted broadcast.** We design system mechanisms that enable overlapping rollout inference, policy dissemination, and training across distributed rollout workers and a centralized trainer via a simple provisioning rule. ECHO-2 bounds policy staleness by a user-specified budget $S$ and employs peer-assisted broadcast to reduce dissemination tail latency.
- **Three-plane disaggregation of rollout, learning, and data.** ECHO-2 cleanly decouples rollout inference, policy optimization, and data handling into independent execution planes, enabling flexible integration of new RL tasks.
- **End-to-end evaluation on LLM RL workloads.** Through extensive end-to-end experiments, we show that ECHO-2 substantially reduces the cost of RL post-training while maintaining learning quality, making large-scale RL more accessible under realistic resource constraints.

## 2 Related Work

**Asynchronous RL frameworks under controlled deployment.** Modern RL post-training frameworks predominantly assume *co-located* deployment, where learners and rollout workers share an administrative domain with provisioned high-bandwidth interconnects. VERL [Sheng et al., 2025b] supports both synchronous and asynchronous execution; AReaL [Fu et al., 2025] streams rollouts asynchronously to hide generation variance; and AReaL-Hex [Yan et al., 2025], StreamRL [Zhong et al., 2025], Laminar [Sheng et al., 2025a], and AsyncFlow [Han et al., 2025] extend this paradigm with heterogeneous-GPU support, disaggregated stream generation, and communication optimizations. While these systems improve learner utilization under bounded asynchrony, they largely rely

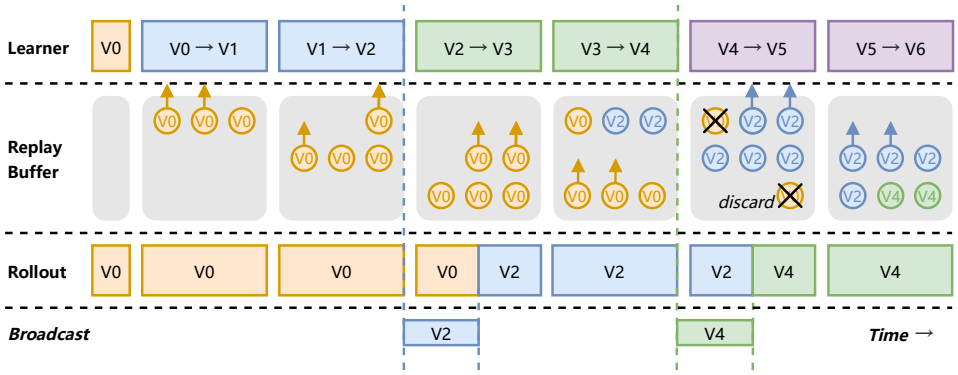



Figure 1: Asynchronous RL execution in ECHO-2. The system overlaps rollout generation, peer-assisted policy dissemination, and learner updates. Bounded staleness $S$ and publication period $\kappa$ decouple the distributed rollout pool from the centralized learner, enabling continuous training even under wide-area network latency.

on data-center-grade connectivity and do not address wide-area rollout execution, where policy dissemination latency becomes a first-order cost.

**Algorithm-level stabilization under staleness.** A complementary line of work studies how stale rollouts alter optimization dynamics rather than system throughput. Zheng et al. [2025] study how far off-policy RL can be pushed with stale data and propose M2PO, a trust-region method for large staleness; BAPO [Xi et al., 2025] stabilizes off-policy RL via balanced policy optimization with adaptive clipping; Song et al. [2026] show that stale rollouts in asynchronous GRPO induce a surrogate-gradient bias of order $O(S\eta)$ and derive a two-constraint stability rule coupling staleness $S$ and learning rate $\eta$; and GAC [Xu et al., 2026] identifies elevated consecutive-gradient cosine similarity as a signature of asynchronous instability and introduces gradient-alignment correction under bounded staleness. These works characterize or modify the surrogate objective and optimizer-level update dynamics under a fixed system, and are orthogonal to ECHO-2: they could be layered on top of our pipeline to extend the safe range of $S$.

**Decentralized RL.** Closer to our setting, INTELLECT-2 [Team et al., 2025b] demonstrates fully decentralized RL by deploying both training and rollout workers across a permissionless contributor network, with health-based orchestration, Shardcast checkpoint distribution, and freshness filtering. ECHO-2 instead targets a hybrid regime with centralized learning on a stable cluster and rollout inference distributed across heterogeneous wide-area workers. Its design questions are therefore different and largely orthogonal to permissionless deployment: how to expose staleness as a user-facing budget $S$, how to provision aggregate rollout capacity in closed form under dissemination overhead, and how to disseminate snapshots efficiently when the trainer uplink is the bottleneck. A detailed comparison with INTELLECT-2 is provided in Table 3 in Section A.

## 3 Cost-Efficient Distributed RL

### 3.1 Design Rationale

Rollout workers differ in throughput and availability, and wide-area dissemination incurs non-negligible variable latency. Strict on-policy execution would force one side to idle, erasing the cost advantage of cheaper compute. ECHO-2 instead exploits a known but underused observation: modern LLM RL objectives tolerate bounded policy lag without harming quality [Zheng et al., 2025, He et al., 2025, Zhong et al., 2025, Fu et al., 2025]. Rather than treating staleness as an artifact to minimize, we treat it as a *first-class budget* that makes distributed rollouts usable.

Concretely, rollout generation, dissemination, and training proceed concurrently; the learner consumes rollouts whose policy version is at most $S$ steps stale (user-specified) and publishes a snapshot every $\kappa$ steps. Section 3.3 turns this into a provisioning question: given $(S, \kappa)$, what aggregate throughput keeps the learner saturated? The answer is a closed-form rule in the measurable quantities $(T_{\text{train}}, T_{\text{bcast}}, R, \{\mu_i\})$. Crucially, $T_{\text{bcast}}$ is itself reducible: ECHO-2's peer-forwarding broadcast

(Section 4.2) lets workers relay snapshots and start generating immediately, so $S$ no longer merely masks network latency but controls the cost–quality trade-off.

### 3.2 Execution Model and Notation

The learner runs on a centralized cluster, performs one optimization step per $T_{\text{train}}$ seconds consuming $R$ trajectories, and publishes an immutable snapshot every $\kappa$ updates (larger $\kappa$ amortizes dissemination at the cost of freshness). Let $v_t$ denote the learner version at step $t$, and $v_x$ the snapshot version of the rollouts in step $t$'s batch. Define $\Delta(t) \triangleq t - x$ and $\Delta_{\max} \triangleq \max_t \Delta(t)$; the system is configured so that $\Delta_{\max} \leq S$. We write $T_{\text{bcast}}$ for the learner-visible time from publication until a newly published snapshot is installed and ready to generate rollouts (measured online).

Let $\mathcal{W}$ be the set of available workers. Worker $i$ has effective throughput $\mu_i$ (rollouts/sec, end-to-end including inference, reward, scheduling, and network) and unit cost $c_i$ (\$/hour). Its *unit throughput cost* is $\rho_i \triangleq c_i/\mu_i$.

### 3.3 Overlap Condition and Capacity Requirement

Continuous learner utilization requires that rollout generation and dissemination fit within one publication period of $\kappa$ training steps:

$$\kappa T_{\text{train}} \;\geq\; T_{\text{bcast}} + \frac{\kappa R}{\sum_{i\in\mathcal{A}} \mu_i}, \tag{1}$$

where $\mathcal{A} \subseteq \mathcal{W}$ is the active set. Rearranging gives the aggregate capacity requirement

$$\sum_{i\in\mathcal{A}} \mu_i \;\geq\; \mu_{\min}(\kappa) \;\triangleq\; \frac{\kappa R}{\kappa T_{\text{train}} - T_{\text{bcast}}}, \qquad \kappa T_{\text{train}} > T_{\text{bcast}}. \tag{2}$$

This collapses a heterogeneous worker pool into a single measurable throughput target.

**Linking $(S, \kappa)$.** Section B derives a conservative upper bound assuming no rollout under a new snapshot is generated until dissemination completes:

$$\Delta_{\max}^{\text{cons}} \;\leq\; \kappa + \left\lceil (T_{\text{bcast}} + R/\mu_{\text{pool}})/T_{\text{train}} \right\rceil - 1, \quad \mu_{\text{pool}} \triangleq \sum_{i\in\mathcal{A}} \mu_i.$$

In all our settings the overlap condition makes the ceiling term $\leq 2$, so $\kappa \leq S - 1$ suffices and we use $\kappa = S - 1$ by default. This is conservative: progressive dissemination and immediate rollout start (Section 4.2) typically yield smaller observed $\Delta(t)$. As an example, Figure 1 illustrates the case $S = 3, \kappa = 2$, in which the maximum 3-step staleness occurs at transitions $v_3 \to v_4$ and $v_5 \to v_6$.

### 3.4 Cost-Aware Provisioning and Scheduling

Given a candidate pool $\mathcal{W}$, ECHO-2 selects $\mathcal{A}$ minimizing cost subject to the capacity rule:

$$\min_{\mathcal{A}\subseteq\mathcal{W}} \sum_{i\in\mathcal{A}} c_i \quad \text{s.t.} \quad \sum_{i\in\mathcal{A}} \mu_i \geq \mu_{\min}(\kappa). \tag{3}$$

While Equation (3) is a knapsack-style problem, online operation favors a simple greedy: rank workers by increasing $\rho_i$ and activate the cheapest prefix whose cumulative throughput exceeds $\mu_{\min}(\kappa)$.

In practice, throughput and availability fluctuate. ECHO-2 treats provisioning as closed-loop control: each worker reports lightweight statistics yielding estimates $\mu_i(t)$ and an availability bit $a_i(t) \in \{0, 1\}$, with effective pool capacity

$$\mu_{\text{pool}}(t) \triangleq \sum_i a_i(t)\, \mu_i(t). \tag{4}$$

The scheduler targets $\mu_{\text{target}} = \gamma \mu_{\min}(\kappa)$ with $\gamma > 1$ to absorb variability; if $\mu_{\text{pool}}(t)$ persistently falls below (resp. exceeds with margin) this target, low-$\rho$ workers are activated (resp. expensive ones released).

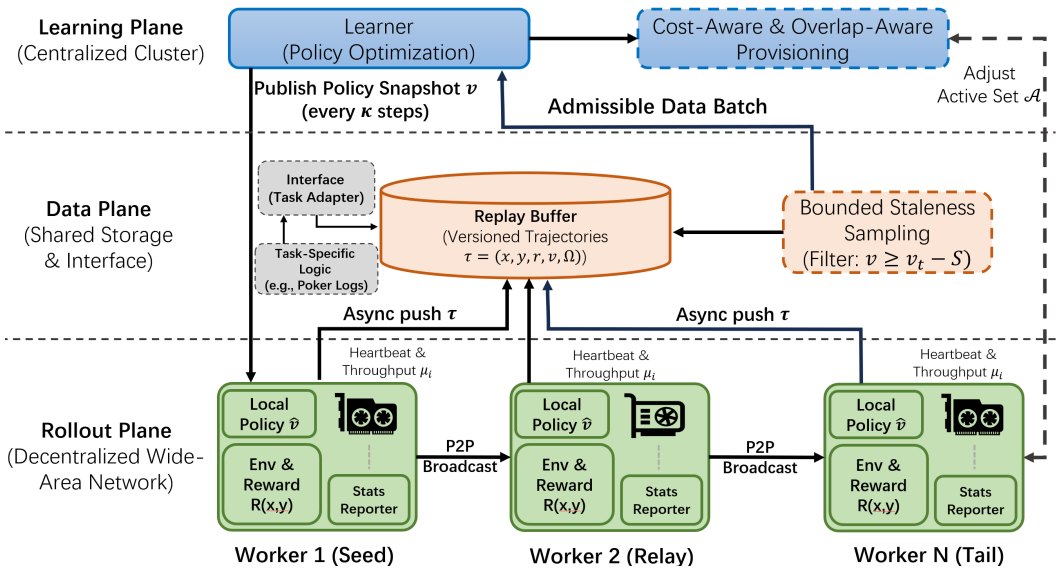



Figure 2: System Architecture of ECHO-2. The system adopts a three-plane decomposition for cost-efficient distributed RL. The centralized learning plane performs policy optimization using data sampled with a bounded staleness budget. The Data Plane provides a unified interface for task adaptation and manages versioned trajectory storage. The distributed rollout plane executes asynchronous generation across workers using pipelined broadcast. **For visual clarity, the rollout plane shows a single forwarding chain; ECHO-2 runs $N_{\text{ch}} \approx \lfloor B_0/B_w \rfloor$ such chains in parallel, with each worker maintaining one upstream parent and at most one downstream child.**

## 4 System Architecture and Implementation

This section describes how ECHO-2 realizes distributed and cost-efficient RL post-training with centralized learning and distributed rollouts. The design follows a three-plane decomposition: rollout, learning, and data, which are connected by versioned, immutable messages and a shared replay buffer; see Figure 2. We outline the workflow of ECHO-2 in Algorithm 1.

**Rollout Plane**: a distributed fleet of workers that repeatedly generates rewarded trajectories under a locally installed snapshot version $\hat{v}$ and pushes version-tagged results to the buffer. This plane is responsible for delivering effective throughput $\mu_i$ and forwarding policy snapshots.

**Learning Plane**: a centralized learner that consumes trajectories and performs a training step with two model updates. It enforces bounded staleness ($S$) when sampling data and publishes snapshots once every $\kappa$ learner updates.

**Data Plane**: task adapters for prompts, trajectory schemas, reward, and loss function design. This plane provides a task-agnostic interface so new workloads are integrated by swapping datasets and reward logic, without touching scheduling or infrastructure. Detailed API specifications and a case study on poker game alignment are provided in Appendix F.

### 4.1 Versioned Execution and Bounded Staleness

**Policy publication.** The learner publishes immutable policy snapshots once every $\kappa$ update steps (Algorithm 1 Line 15-17). Between two publications, the learner may perform multiple updates while workers continue generating rollouts under their most recent installed snapshot. The publication period $\kappa$ is chosen to respect the staleness budget $S$ (default $\kappa \leq S - 1$, and we use $\kappa = S - 1$ unless otherwise stated), which should be $\geq 2$ in ECHO-2.

**Rollout generation.** Each rollout worker maintains a local snapshot version $\hat{v}$. For each prompt $x$, the worker samples a response $y \sim \pi_{\hat{v}}(\cdot \mid x)$, computes reward $r = \mathcal{R}(x, y)$, and emits a trajectory $(x, y, r, \hat{v})$ into the buffer (Algorithm 1 Line 26-27). Reward computation is performed in the Rollout Plane. We reject items that violate the data format as a lightweight sanity check for data integrity.

**Replay buffer management.** The replay buffer stores version-tagged trajectories and supports selective sampling. At the learner update index $v_t$, only trajectories with bounded lag are admissible: $v \geq v_t - S$. Older trajectories are discarded. This enforces bounded staleness without imposing global synchronization, treating rollouts as a stream [Zhong et al., 2025].

Bounded staleness constrains data freshness but does not impose a fixed update schedule. The learner advances whenever sufficient eligible data are available. The parameter $S$ therefore bounds the maximum policy lag between rollout generation and training, providing temporal slack to absorb latency without modifying the underlying RL objective.

### 4.2 Peer-to-Peer Broadcast and Asynchronous Rollout Start

A naive push-to-all strategy (star topology) achieves minimal communication latency when the bandwidth between the training center and remote workers is abundant. Under wide-area constraints, however, both the learner uplink and the tail receivers become bottlenecks. ECHO-2 instead organizes workers into a peer-forwarding topology that disseminates snapshots under bandwidth constraints by leveraging the aggregate bandwidth of the rollout fleet.

A common wide-area regime is that the learner has a finite uplink budget $B_0$, while each worker is capped by a smaller per-node bandwidth $B_w$. ECHO-2 instantiates $N_{\text{ch}} \approx \lfloor B_0/B_w \rfloor$ parallel forwarding chains, each disseminating the full snapshot via chunked store-and-forward streaming. The learner streams snapshot chunks to a first-hop seed on each chain at rate $B_w$; each seed forwards to its downstream child upon receipt, and forwarding continues along the chain. After pipeline warm-up, dissemination approaches line-rate on each chain with minimal control overhead, since each worker maintains exactly one inbound and one outbound flow (fan-out $= 1$), avoiding multi-parent scheduling. Appendix C.3 details how chains are populated and reordered online based on workers' transfer speed and reliability statistics.

To acquire as many rollouts as possible, upon receiving any new chunks, a worker forwards them immediately (Algorithm 1, Line 22). Upon completing installation of the new snapshot, it switches its local version $\hat{v}$ and starts generating rollouts under the new version. A slow or failed relay would otherwise stall its downstream tail; ECHO-2 detects such incidents through the same per-worker statistics used for chain ranking and rebuilds the affected chain segment on the fly. Workers participating only in rollout generation are not affected by the broadcast stage faults, so a single failure causes bounded transient throughput loss rather than a pipeline stall (Appendix D).

### 4.3 Cost-Aware Scheduling over Distributed Resource Pools

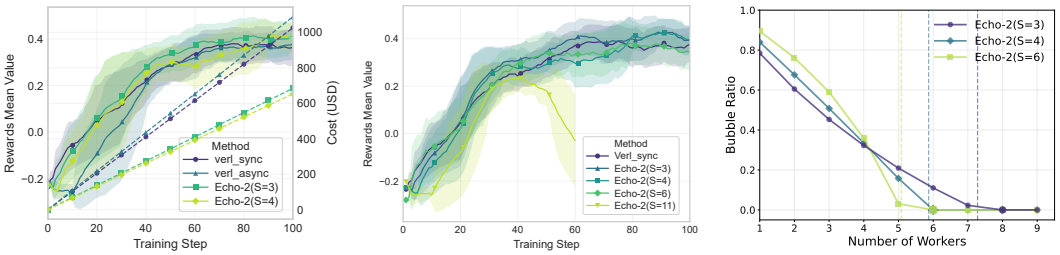
ECHO-2 instantiates the closed-loop scheduler of Section 3.4 as follows. Per-worker throughput $\mu_i(t)$ and availability $a_i(t)$ are derived from heartbeat statistics aggregated over a sliding window; $T_{\text{train}}$ and $T_{\text{bcast}}$ are calibrated offline at deployment time and re-measured only when the resource pool changes substantially. The safety factor is set to $\gamma = 1.1$ in our 4B/8B experiments, giving a $\sim$10% redundancy that absorbs typical short-term fluctuations and single-worker faults without reconfiguration (Section D). To prevent thrashing under transient noise, activation/release decisions are issued only after capacity deviates from $\mu_{\text{target}}$ for a sustained interval; the full update rule is given in Section C.2.1.

## 5 Experiments

We evaluate ECHO-2 in distributed RL post-training settings and ask three questions: (Q1) Does ECHO-2 reduce the cost required to reach a target RL quality compared with centralized pipelines? (Q2) Is RL quality robust to bounded staleness $S$ under wide-area distributed rollouts? (Q3) Do our overlap model and system mechanisms accurately predict and improve learner utilization under wide-area constraints?

### 5.1 Experimental Setup

**Models.** We post-train two base models, Qwen3-4B and Qwen3-8B [Team, 2025]. Unless otherwise stated, we use the same GRPO hyperparameters across all systems: global batch size 128, maximum




(a) Cost–quality on AIME24. (b) Effect of bounded staleness $S$. (c) Bubble ratio vs. rollout workers.

Figure 3: **Experimental results of ECHO-2 on Qwen3-8B. (a)** Cost-quality efficiency on AIME24 under a WAN deployment. Solid lines show reward curves smoothed by a moving average over training steps; shaded regions indicate the variance of the underlying raw per-step values. Dashed lines indicate estimated dollar costs computed from steady-state training time and public GPU rental prices (right y-axis). **(b)** Effect of staleness $S$ on RL stability. Performance remains robust for moderate staleness ($S \leq 6$), while excessive staleness ($S = 11$) leads to divergence. **(c)** Learner bubble ratio as a function of the number of rollout workers. Vertical dashed lines denote the theoretically predicted minimum worker counts.

generation length 8192, temperature 1.0, top-$p$ 0.95, and rollout multiplicity $n = 16$. We disable chain-of-thought prompting and report avg@64.

**Datasets.** We train on DAPO-Math [Yu et al., 2025] and evaluate on AIME24 [MAA, 2024], which has verifiable final answers. Unless otherwise noted, we report AIME accuracy as the primary RL quality metric. We additionally report results on a broader math benchmark suite in Appendix E.2.

**System deployment.** (1) *Learning plane.* The learner runs on $4\times$ A100 80GB GPUs for ECHO-2, and on $8\times$ A100 GPUs for the centralized baseline. We measure the steady-state per-update time $T_{\text{train}}$ as the median over five offline training steps. (2) *Rollout plane.* To simplify experimental validation, rollouts run on a distributed pool of RTX 5090 workers served by Parallax [Tong et al., 2025], a hardware-agnostic inference service.

**Network regimes.** To study the impact of bandwidth constraints, we cap the learner uplink budget to representative WAN regimes and cap each worker's download rate. In experiments, we identify $T_{\text{bcast}}$ as the elapsed time from publication until a target fraction $1/\gamma$ of active workers have installed the snapshot. This aligns the abstraction in Section 3 with the observed dissemination behavior and captures the practical effect of tail receivers.

**Baselines.** (1) CENTRALIZED-SYNC (VERL [Sheng et al., 2025b]): a synchronized pipeline in which rollouts are co-located with the learner on the same data-center GPUs. (2) CENTRALIZED-ASYNC (VERL-async [Sheng et al., 2025b]): a streaming asynchronous baseline within the data center (AReaL-style [Fu et al., 2025]) that overlaps rollout generation and learning while assuming high-bandwidth, low-latency connectivity.

**Cost and utilization metrics.** Dollar cost is the sum over GPU types of hourly rental price (Table 1) times GPU-hours used, with Google Cloud A100 representing data-center hardware and vast.ai RTX 5090 approximating distributed customer-grade resources. The learner *bubble ratio* is the fraction of wall-clock time the learner spends idle waiting for admissible rollouts. We measure the learner *bubble ratio* (idle fraction) as

Table 1: GPU rental prices (USD/hour, single GPU, collected on 2026-05-01).

| GPU | USD/hr | Platform |
|---|---|---|
| A100 80GB | \$3.06 | Google Cloud |
| RTX 5090 | \$0.35 | vast.ai |

$$\frac{T_{\text{idle}}}{T_{\text{idle}} + T_{\text{train,active}}},$$

where $T_{\text{idle}}$ is the waiting time caused by insufficient admissible rollouts.

### 5.2 Cost–Quality Efficiency

Given the discreteness and variance of AIME accuracy, we interpret cost–quality efficiency as the cumulative cost required to reach a target accuracy threshold under a representative WAN setting. Figure 3a plots the reward curves; the per-step training times ($T_{\text{train}}$) of VERL-sync, VERL-async,

and ECHO-2 with $S = 3/4$ are 1508.2s, 1582.3s, 1631.2s, and 1649.3s, respectively. Combining $T_{\text{train}}$ with rental prices under steady-state execution yields the linear cost curves shown as dashed lines (right $y$-axis).

For Qwen3-8B, ECHO-2 consistently dominates centralized pipelines: at matched AIME accuracy, it reduces cumulative cost by 33.3%–36.3%; at matched cost, it achieves comparable final accuracy. This gain arises because rollout generation is offloaded to cheaper distributed GPUs without stalling the centralized learner, as long as the overlap condition (Equation (1)) is satisfied. Qwen3-4B shows the same trend and leads to the same conclusion (Section E.1).

**Generalization to larger models.** We additionally validate ECHO-2 on QwQ-32B against Prime-RL [Team et al., 2025b], a recent decentralized-rollout system, under matched GRPO hyperparameters. Despite running rollouts on 76×RTX 5090 workers over a wide-area network (vs. Prime-RL's 10×H100 on an idealized intranet), ECHO-2 achieves a 4.6% end-to-end cost reduction with reward trajectories tracking Prime-RL throughout training. This isolates the contribution of our system mechanisms (overlap-aware provisioning, peer-forwarding broadcast, cost-aware activation) from the architectural shift to disaggregated rollout that Prime-RL also adopts. Notably, this saving is achieved *despite* ECHO-2 running over a wide-area network while Prime-RL enjoys an idealized intranet with unlimited trainer-to-worker bandwidth; under a fair WAN comparison, Prime-RL's centralized broadcast cost would scale with fleet size while ECHO-2's remains bounded (Section 5.5.1). Full setup and stability analysis are in Section E.3.

### 5.3 RL Quality under Bounded Staleness

We sweep the staleness budget $S$ to quantify the trade-off between training stability, final quality, and system efficiency. Specifically, we evaluate $S \in \{3, 4, 6, 11\}$ while keeping all other settings fixed. Recall that, in a stable pipelined ECHO-2, the observed rollout staleness is upper-bounded by $S$.

Figure 3b shows that moderate staleness does not materially degrade final quality: for $S \leq 6$, ECHO-2 achieves reward and accuracy within roughly 5% fluctuation of the synchronous baseline, with similar convergence trends and lower cost. In contrast, overly large staleness ($S = 11$) can lead to instability in standard GRPO, consistent with the intuition that stale data gradually deviates from the current policy distribution, and with the analysis of Song et al. [2026], which shows that the maximum stable learning rate shrinks roughly as $1/S$.

### 5.4 Validating the Overlap Condition

Equation (2) predicts a threshold behavior: as rollout capacity increases, learner bubbles should rapidly vanish once the system enters the feasible overlap region. We validate this prediction by sweeping the effective rollout-pool size. Figure 3c shows that, as the pool grows, bubble ratio drops consistently toward zero near the predicted threshold, confirming that the overlap model provides a practical provisioning rule. Larger $S$ shifts the transition to the left, showing that staleness acts as an explicit control knob that trades policy freshness for reduced rollout capacity.

### 5.5 Ablation Study

#### 5.5.1 Broadcast under Bandwidth Constraints

We evaluate policy dissemination latency under different broadcast strategies as the rollout fleet scales. We measure the learner-visible broadcast time $T_{\text{bcast}}$ as the elapsed time from snapshot publication until a target fraction $q$ of active workers have fully installed the snapshot and can start generating rollouts under the new version (Section 4.2). We use $q = 1/\gamma = 1/1.1 \approx 0.9$.

We compare three dissemination settings while sweeping the number of active rollout workers $N$: (i) STAR-UNLIMITED, an idealized push-to-all broadcast with no learner uplink cap, serving as a lower bound on $T_{\text{bcast}}$; (ii) STAR-LIMITED, push-to-all broadcast under realistic wide-area caps, with the learner uplink budget $B_0$ swept over $\{300, 600, 700, 800\}$ Mbps and a per-worker downlink cap $B_w = 100$ Mbps; and (iii) TREE-PIPELINED, ECHO-2's design, which constructs $N_{\text{ch}} \approx \lfloor B_0/B_w \rfloor$ parallel forwarding chains and uses chunked store-and-forward dissemination so that each worker relays data to its single downstream child upon receipt (Section 4.2).

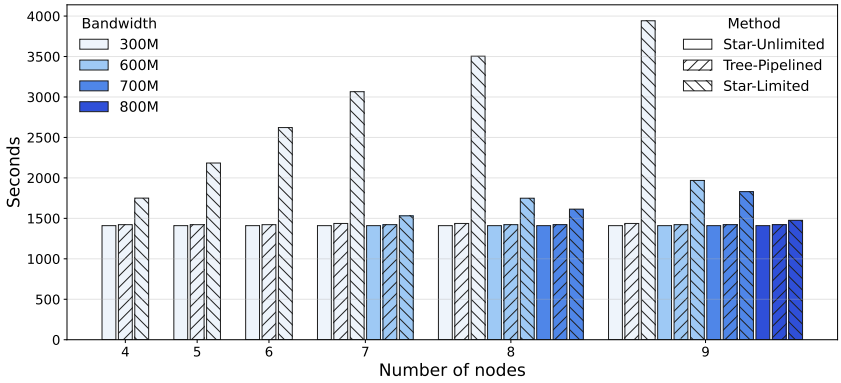



Figure 4: **Policy broadcast latency $T_{\text{bcast}}$ vs. rollout fleet size.** Comparison of three dissemination strategies across different numbers of nodes $N$. **Star-Limited** (with learner uplink $B_0 \in [300, 800]$ Mbps) suffers from increasing latency as the learner becomes the bandwidth bottleneck. **Tree-Pipelined** dissemination, by using chunked peer forwarding, maintains near-constant broadcast time and scales much more efficiently with fleet size, closely matching the idealized **Star-Unlimited** baseline.

Figure 4 shows that STAR-LIMITED suffers from rapidly growing $T_{\text{bcast}}$ as $N$ increases: with a fixed learner uplink budget $B_0$, the per-worker delivery rate degrades to $B_0/N$ once $N \cdot B_w > B_0$, making the learner the dominant bottleneck. In contrast, TREE-PIPELINED keeps $T_{\text{bcast}}$ within a small constant factor of STAR-UNLIMITED across the entire range, even under the same bandwidth caps. This is because each chain operates at line rate $B_w$ independently after pipeline warm-up, and the aggregate fleet bandwidth scales with $N$ rather than being capped by $B_0$.

### 5.5.2 Peer-Assisted Broadcast and Cost-Aware Provisioning

We isolate the contributions of ECHO-2's two mechanisms via two ablations: ECHO-2-NOP2P replaces peer-assisted broadcast with direct learner-to-worker dissemination, and ECHO-2-NOCOST replaces cost-aware provisioning with uniform random activation while still targeting the full-overlap objective. To make the cost ablation informative, we evaluate under a mixed-price rollout pool with heterogeneous worker costs, and report cost per step, dissemination latency, and inter-step learner waiting time.

Table 2: **Ablation summary.** We evaluate the impact of removing peer-assisted (P2P) broadcast and cost-aware provisioning (Cost) in ECHO-2. #Mach, $T_{\text{bcast}}$, and Wait denote the rollout fleet size, dissemination latency, and learner idle time, respectively.

| Method | #Mach | Cost/Step ↓ | $T_{\text{bcast}}$ (s) | Wait (s) |
|---|---|---|---|---|
| Full | 9 | 8.098 | 1437 | 0 |
| *w/o* P2P | 9 | 8.432 | 1830 | 131.9 |
| *w/o* P2P | 10 | 8.630 | 1872 | 84.1 |
| *w/o* Cost | 9 | 9.339 | 1437 | 0 |

Table 2 shows that removing peer-assisted broadcast inflates communication latency and bubble time, requiring additional machines—and thus higher cost—to suppress learner idle time. Disabling cost-aware provisioning likewise increases the cost to reach a given quality target by activating suboptimal workers. Together, these results show that both mechanisms are necessary for the end-to-end cost efficiency reported in Figure 3a. A task-agnostic data-plane use case is discussed in Section F.

## 6 Limitations and Future Work

ECHO-2 relies on the empirical robustness of modern LLM RL objectives to bounded policy lag. While moderate staleness preserves GRPO post-training quality in our experiments, we do not provide formal guarantees, and the safe range may depend on the task and reward signal. Developing theoretical staleness control remains future work. Our peer-assisted broadcast mitigates uplink bottlenecks, whereas our future work will include delta or quantized updates and cache-aware deployment. ECHO-2 focuses on centralized learning with distributed rollouts. Extending the design to multiple or geographically replicated learners is promising, but it introduces new challenges in

synchronization and policy consistency, and further validation across a wider range of model sizes (70B+) remains future work.

## 7 Conclusion

We presented ECHO-2, an RL framework for LLM post-training that separates centralized learning from distributed rollouts. By treating bounded staleness as a control knob, modeling overlap-based capacity, and activating workers on demand based on cost, ECHO-2 enables cost-aware provisioning under wide-area execution. With peer-assisted pipelined broadcast, ECHO-2 reduces dissemination overhead. Experiments on GRPO post-training of 4B, 8B and 32B models show that ECHO-2 significantly lowers training cost while preserving RL quality comparable to baselines.

## Acknowledgments and Disclosure of Funding

The GPU compute resources and financial support for this research were provided by Gradient Network.

**Competing interests.** J. Song, J. Xiao, Q. Ren, J. Huang, C. Tong, Z. Bi, S. Lu, Y. Duan, X. Wang, R. Yu, W. Yang, L. Ai, and E. Yang are affiliated with Gradient Network, which funded this work and develops the Parallax inference engine [Tong et al., 2025] used by the rollout workers in ECHO-2. The remaining authors declare no competing interests.

## References

Anthropic. Claude sonnet 4.5 system card. Technical report, Anthropic, 2025. URL `https://assets.anthropic.com/m/12f214efcc2f457a/original/Claude-Sonnet-4-5-System-Card.pdf`.

Daman Arora, Himanshu Singh, et al. Have llms advanced enough? a challenging problem solving benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7527–7543, 2023.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 558–568, 2023.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL `https://arxiv.org/abs/2507.06261`.

Jingxuan Fan, Sarah Martinson, Erik Y Wang, Kaylie Hausknecht, Jonah Brenner, Danxian Liu, Nianli Peng, Corey Wang, and Michael P Brenner. Hardmath: A benchmark dataset for challenging problems in applied mathematics. *arXiv preprint arXiv:2410.09988*, 2024.

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.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, Huazhong Ji, Wenjie Liu, Yu Huang, Yixiang Zhang, Chenyi Pan, Jing Wang, Xin Huang, Chunsheng Li, and Jianping Wu. Asyncflow: An asynchronous

streaming rl framework for efficient llm post-training, 2025. URL https://arxiv.org/abs/2507.01663.

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.

Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35406–35430, 2025.

MAA. American invitational mathematics examination (AIME), 2024. URL https://www.maa.org/math-competitions/aime.

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.

OpenAI. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. Swarm parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning*, pages 29416–29440. PMLR, 2023.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Yuxuan Tong, Borui Wan, Wang Zhang, Chaobo Jia, Xibin Wu, Yuqi Wu, Xiang Li, Chi Zhang, Yanghua Peng, Haibin Lin, Xin Liu, and Chuan Wu. Laminar: A scalable asynchronous rl post-training framework, 2025a. URL https://arxiv.org/abs/2510.12633.

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, 2025b.

Jingwei Song, Haofeng Xu, Jie Xiao, Chengke Bao, Jingwei Shi, Pengbin Feng, Yuhang Han, Weixun Wang, Eric Yang, and Tianyu Shi. Scaling laws for collapse in asynchronous grpo, 2026. URL https://arxiv.org/abs/2607.01083.

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*, 2025a.

Prime Intellect Team, Sami Jaghouar, Justus Mattern, Jack Min Ong, Jannik Straube, Manveer Basra, Aaron Pazdera, Kushal Thaman, Matthew Di Ferrante, Felix Gabriel, et al. Intellect-2: A reasoning model trained through globally decentralized reinforcement learning. *arXiv preprint arXiv:2505.07291*, 2025b.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Chris Tong, Youhe Jiang, Gufeng Chen, Tianyi Zhao, Sibian Lu, Wenjie Qu, Eric Yang, Lynn Ai, and Binhang Yuan. Parallax: Efficient llm inference service over decentralized environment. *arXiv preprint arXiv:2509.26182*, 2025.

xAI. Grok 4 model card. Technical report, xAI, 2025. URL https://data.x.ai/2025-08-20-grok-4-model-card.pdf.

Zhiheng Xi, Xin Guo, Yang Nan, Enyu Zhou, Junrui Shen, Wenxiang Chen, Jiaqi Liu, Jixuan Huang, Zhihao Zhang, Honglin Guo, Xun Deng, Zhikai Lei, Miao Zheng, Guoteng Wang, Shuo Zhang, Peng Sun, Rui Zheng, Hang Yan, Tao Gui, Qi Zhang, and Xuanjing Huang. Bapo: Stabilizing off-policy reinforcement learning for llms via balanced policy optimization with adaptive clipping, 2025. URL https://arxiv.org/abs/2510.18927.

Jie Xiao, Changyuan Fan, Qingnan Ren, Alfred Long, Yuchen Zhang, Rymon Yu, Eric Yang, Lynn Ai, and Shaoduo Gan. Echo: Decoupling inference and training for large-scale rl alignment on heterogeneous swarms. *arXiv preprint arXiv:2508.05387*, 2025.

Haofeng Xu, Junwei Su, Yukun Tian, Lansong Diao, Zhengping Qian, and Chuan Wu. Gac: Stabilizing asynchronous rl training for llms via gradient alignment control, 2026. URL https://arxiv.org/abs/2603.01501.

Ran Yan, Youhe Jiang, Tianyuan Wu, Jiaxuan Gao, Zhiyu Mei, Wei Fu, Haohui Mai, Wei Wang, Yi Wu, and Binhang Yuan. Areal-hex: Accommodating asynchronous rl training over heterogeneous gpus. *arXiv preprint arXiv:2511.00796*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? *arXiv preprint arXiv:2510.01161*, 2025.

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.

Runlong Zhou, Lefan Zhang, Shang-Chen Wu, Kelvin Zou, Hanzhi Zhou, Ke Ye, Yihao Feng, Dong Yin, Alex Guillen Garcia, Dmytro Babych, et al. Rlax: Large-scale, distributed reinforcement learning for large language models on tpus. *arXiv preprint arXiv:2512.06392*, 2025.

# A Detailed Comparison with INTELLECT-2

Table 3 summarizes the design contrast between INTELLECT-2 and ECHO-2 along four dimensions. The two systems target orthogonal sets of design questions: INTELLECT-2 focuses on infrastructure for permissionless contributor participation, while ECHO-2 focuses on cost-efficient hybrid execution under bounded staleness with a closed-form provisioning rule.

Table 3: Design contrast between INTELLECT-2 and ECHO-2.

| **Design dimension** | **INTELLECT-2** | **ECHO-2 (ours)** |
|---|---|---|
| Policy staleness | Empirical at fixed asynchrony | User-facing budget $S$; $\kappa$ from $S$ via Section B |
| Provisioning rule | Case-by-case fit check | Closed-form $\sum \mu_i \geq \mu_{\min}(\kappa)$ |
| Worker activation | Health/availability-based | Cost-aware via $\rho_i = c_i/\mu_i$ |
| Broadcast topology | Shardcast (CDN-style relay tier) | Peer-forwarding chains within fleet |

# B Worst-Case Staleness Bound under Overlap

This appendix derives a conservative upper bound on the maximum policy staleness $\Delta_{\max}$ in ECHO-2, and shows how the overlap condition tightens this bound.

## B.1 Execution Semantics and Conservative Assumption

We consider the following execution semantics, which intentionally model a worst-case scenario:

- Policy snapshots are published every $\kappa$ learner update steps, at the *end* of a training step.
- Training batches are formed at the *beginning* of each step.
- In the most conservative case, rollout workers generate no trajectories from a newly published snapshot until dissemination completes after $T_{\text{bcast}}$ time.
- After dissemination completes, rollouts from the new policy are generated at aggregated rate $\mu_{\text{pool}}$.

This model intentionally ignores progressive dissemination and early rollout start, and therefore upper-bounds the staleness that can occur in practice.

## B.2 Baseline Worst-Case Staleness Bound

Let $n$ denote the number of learner steps elapsed since a snapshot is published. By time $nT_{\text{train}}$, the number of rollouts generated from the new policy is at most

$$G(n) = \mu_{\text{pool}} \cdot \max\big(0,\ nT_{\text{train}} - T_{\text{bcast}}\big). \tag{5}$$

The earliest step at which at least $R$ new-policy rollouts are available satisfies

$$G(n) \ \geq \ R \quad \Rightarrow \quad n \ \geq \ \left\lceil \frac{T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}}}{T_{\text{train}}} \right\rceil. \tag{6}$$

At publication, the learner version advances by $\kappa$ relative to the previous published snapshot. Since no new-policy rollout can be consumed during the first $n-1$ steps after publication, the maximum staleness under this conservative model is

$$\Delta_{\max}^{\text{cons}} = \kappa + \left\lceil \frac{T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}}}{T_{\text{train}}} \right\rceil - 1. \tag{7}$$

### B.3 Tightening the Bound using the Overlap Condition

The baseline bound in Equation (7) depends on the rollout throughput $\mu_{\text{pool}}$. We now show that under the overlap condition, this dependence can be tightened.

Recall the overlap condition:

$$\kappa T_{\text{train}} \;\geq\; T_{\text{bcast}} + \frac{\kappa R}{\mu_{\text{pool}}}. \tag{8}$$

Rearranging yields

$$\frac{R}{\mu_{\text{pool}}} \;\leq\; T_{\text{train}} - \frac{T_{\text{bcast}}}{\kappa}. \tag{9}$$

Substituting Equation (9) into the numerator of Equation (7) gives

$$T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}} \;\leq\; T_{\text{train}} + \left(1 - \frac{1}{\kappa}\right) T_{\text{bcast}}. \tag{10}$$

Dividing both sides by $T_{\text{train}}$ and taking the ceiling,

$$\left\lceil \frac{T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}}}{T_{\text{train}}} \right\rceil \;\leq\; 1 + \left\lceil \left(1 - \frac{1}{\kappa}\right) \frac{T_{\text{bcast}}}{T_{\text{train}}} \right\rceil. \tag{11}$$

Substituting back into Equation (7) yields a tightened bound:

$$\Delta_{\max}^{\text{cons}} \;\leq\; \kappa + \left\lceil \left(1 - \frac{1}{\kappa}\right) \frac{T_{\text{bcast}}}{T_{\text{train}}} \right\rceil. \tag{12}$$

### B.4 Implication for $\kappa = 2$

For the common case $\kappa = 2$, the overlap condition implies $T_{\text{bcast}} < 2T_{\text{train}}$, and thus

$$0 < \frac{1}{2} \frac{T_{\text{bcast}}}{T_{\text{train}}} < 1.$$

Therefore,

$$\Delta_{\max}^{\text{cons}} \;\leq\; 3. \tag{13}$$

This bound corresponds to a worst-case execution in which new-policy rollouts become available only after dissemination completes. In practice, rollout workers begin generating trajectories as soon as they receive the update during dissemination, making the observed staleness typically smaller than this bound.

**Corollary: Single-Parameter Configuration.** Consider the configuration used by ECHO-2, in which the publication period is set to $\kappa = S - 1$. Substituting into the conservative bound yields

$$\Delta_{\max}^{\text{cons}} = S - 1 + \left\lceil \frac{T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}}}{T_{\text{train}}} \right\rceil - 1.$$

If the system satisfies the overlap condition and $T_{\text{bcast}}/T_{\text{train}} < 1$, which holds in all our experimental settings, then

$$\left\lceil \frac{T_{\text{bcast}} + \frac{R}{\mu_{\text{pool}}}}{T_{\text{train}}} \right\rceil \leq 2,$$

and therefore

$$\Delta_{\max}^{\text{cons}} \;\leq\; S.$$

This result justifies exposing $S$ as the sole staleness control parameter in ECHO-2.

**Algorithm 1** Execution of ECHO-2

**Shared:** replay buffer $\mathcal{B}$, worker pool $\mathcal{W}$, active set $\mathcal{A}$
**Learner state:** update index $v \leftarrow 0$
**Worker state:** each worker $i$ maintains local snapshot version $\hat{v}_i$
**Learning Plane:**
**while** training not converged **do**
  **if** scheduling tick or sustained capacity deviation **then**
    Estimate $\mu_{\text{pool}} = \sum_{i \in \mathcal{A}} a_i \mu_i$
    Compute $\mu_{\text{target}} = \gamma \cdot \mu_{\min}(\kappa)$ using Equation (2)
    Adjust $\mathcal{A}$ by activating/releasing workers based on $\rho_i$ (Section 4.3)
  **end if**
  **if** $\mathcal{B}$ has at least $R$ admissible trajectories with $v \geq v_t - S$ **then**
    Sample a batch from $\mathcal{B}$ subject to staleness version
    Perform one policy update (time $\approx T_{\text{train}}$)
    $v \leftarrow v + 1$
    **if** $v \bmod \kappa = 0$ **then**
      Publish snapshot with version $v$ and trigger dissemination (Section 4.2)
    **end if**
  **end if**
**end while**
**Rollout Plane, on each worker $i \in \mathcal{A}$ in parallel:**
**while** worker $i$ is active **do**
  Receive and forward snapshot chunks as a relay (Section 4.2)
  **if** a newer snapshot is fully installed **then**
    Update local version $\hat{v}_i \leftarrow v_{\text{new}}$
  **end if**
  Sample and generate $y \sim \pi_{\hat{v}_i}(\cdot \mid x)$ and compute reward $r = \mathcal{R}(x, y)$
  Push trajectory $(x, y, r, \hat{v}_i, \Omega)$ into $\mathcal{B}$
**end while**

# C ECHO-2 execution

## C.1 Overall Procedure

We illustrate the end-to-end execution model of ECHO-2 in Algorithm 1, which only includes Rollout Plane and Learning Plane since training process is transparent to Data Plane.

## C.2 Supplementary System Design

### C.2.1 Low-Frequency Adjustment

The scheduler maintains an active worker set $\mathcal{A}$ and monitors its aggregate throughput $\sum_{i \in \mathcal{A}} a_i \mu_i$. If capacity persistently falls below $\mu_{\text{target}}$, additional workers with low unit throughput cost $\rho$ are activated; if capacity exceeds the target by a sufficient margin, expensive workers are gradually released. This design ensures that the learner remains saturated whenever feasible, while avoiding frequent reconfiguration and unnecessary rollout cost.

## C.3 Forwarding-Chain Construction in Peer-Assisted Broadcast

This appendix details how ECHO-2 constructs and maintains the peer-forwarding chains used in Section 4.2. Given a learner uplink budget $B_0$ and a per-worker forwarding bandwidth $\hat{B}_w$, ECHO-2 instantiates

$$N_{\text{ch}} \approx \left\lfloor \frac{B_0}{\hat{B}_w} \right\rfloor$$

parallel chains. Each chain disseminates the full snapshot via chunked store-and-forward streaming; every active worker is assigned to exactly one chain in a given broadcast round and maintains one upstream parent and at most one downstream child.

**Worker scoring.** Among workers selected by cost-aware activation (Section 4.3), ECHO-2 ranks candidates by a dynamic peer score that combines transfer speed, success rate, and recent stability:

$$\mathrm{Score}_i = \mathrm{Decay}_i \left(0.5\,\mathrm{Speed}_i + 0.35\,\mathrm{Success}_i + 0.15\,\mathrm{Stability}_i\right).$$

The component scores are updated online from heartbeat statistics:

$$\begin{aligned}
\mathrm{Speed}_i &= 100 \cdot \frac{\log_{10}(\mathrm{avg_speed}_i) - 3.0}{5.0}, \\
\mathrm{Success}_i &= 100 \cdot \mathrm{success_rate}_i, \\
\mathrm{Stability}_i &= \begin{cases} 0, & \text{if consecutive failures} > 3, \\ 100 \cdot \min(\mathrm{succ_count}_i/10,\, 1)\,, & \text{otherwise}, \end{cases} \\
\mathrm{Decay}_i &= 1 - \min(\mathrm{elapsed}_i/600,\, 0.3)\,.
\end{aligned}$$

The $\mathrm{Speed}_i$ term saturates at 100 when the measured forwarding rate reaches the per-worker cap (e.g., 100 Mbps in our experiments). $\mathrm{Decay}_i$ down-weights stale measurements, so workers re-enter the ranking smoothly after periods of inactivity.

**Chain assembly.** The top-$N_{\text{ch}}$ workers under this ranking are designated as first-hop seeds, allowing them to receive snapshot chunks directly from the learner and start generating rollouts as early as possible. The remaining active workers are appended to the chain tails in rank order, in a simple round-robin manner across the $N_{\text{ch}}$ chains. The forwarding topology is therefore dynamic rather than static: workers admitted in later rounds are integrated into chains based on their current scores, and workers exhibiting degraded transfer quality are progressively deprioritized (seeds first, then moved toward chain tails, then dropped from the forwarding role entirely while remaining as rollout-only workers).

**Coefficient choices.** The weights $(0.5, 0.35, 0.15)$, the failure threshold of 3, the success saturation count of 10, and the decay window of 600 seconds are operational defaults rather than tuned hyperparameters; in our experiments ranking quality is dominated by $\mathrm{Speed}_i$ and the system is robust to moderate perturbations of these constants.

## D Fault Tolerance under Single-Worker Failure

ECHO-2's default deployment provisions a small redundancy buffer ($\gamma = 1.1$ in our experiments, see Section 4.3), which absorbs typical short-term throughput fluctuations without triggering any reconfiguration. To stress-test the system under more adversarial conditions, we additionally evaluate a no-redundancy setting ($\gamma = 1.0$) in which 8 rollout workers exactly saturate the learner. In this regime, the loss of a single worker corresponds to a $1/8 = 12.5\%$ capacity loss, which is large enough to expose any pipeline stall behavior.

**Recovery mechanism.** ECHO-2 continuously monitors per-worker liveness and progress through the same heartbeat statistics used for chain ranking (Appendix C.3). When effective capacity drops below $\mu_{\text{target}}$, the scheduler launches a replacement worker drawn from the standby pool on the same compute platform; the replacement retrieves the current snapshot from cloud storage and rejoins the active set. For broadcast-stage failures specifically, the affected forwarding chain is rebuilt: the failed worker is excised, its downstream tail is reattached to the nearest healthy parent, and the replacement is appended to the chain in rank order.

**Fault injection results.** Table 4 reports the impact of single-worker faults injected separately during broadcast and rollout stages. Across all four scenarios ECHO-2 recovers to full saturation within a bounded window (240–312 s, equivalent to roughly 0.15–0.20 of one training step at $T_{\text{train}} \approx 1500$ s for our 8B setting), with bubble-ratio increases below 3%.

**Why broadcast-stage faults appear more severe than rollout-stage faults.** At first glance the broadcast-stage drop-out result (16.45%) is larger than the rollout-stage drop-out result ($12.50\% = 1/8$). The gap reflects the cascading effect of a failed forwarder: when a relay $\mathcal{A}_{k,j}$ on chain $j$ disappears, every downstream worker on the same chain temporarily cannot install the new snapshot,

Table 4: Single-worker fault injection in a minimally saturated 8-worker deployment ($\gamma = 1.0$). Throughput drop is the peak instantaneous reduction; recovery time is measured from fault injection to the moment $\mu_{\text{pool}}$ returns to $\mu_{\text{target}}$; bubble-ratio increase is computed against the fault-free baseline.

| **Fault scenario** | **Throughput drop** | **Recovery time** | **Bubble-ratio increase** |
|---|---|---|---|
| Broadcast: 1 forwarder slows 0.5× | 8.94% | 312 s | 1.56% |
| Broadcast: 1 forwarder drops out | 16.45% | 312 s | 2.63% |
| Rollout: 1 worker slows 0.5× | 6.80% | 240 s | 0.97% |
| Rollout: 1 worker drops out | 12.50% | 240 s | 1.88% |

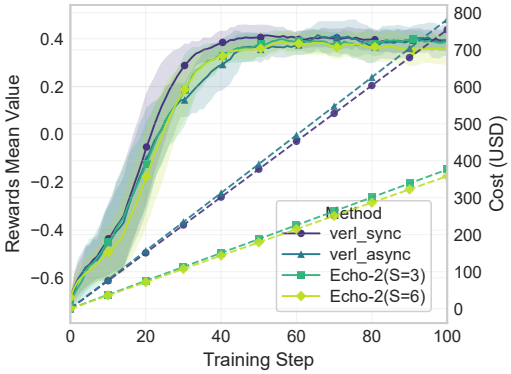



Figure 5: Cost–quality on AIME for Qwen3-4B.

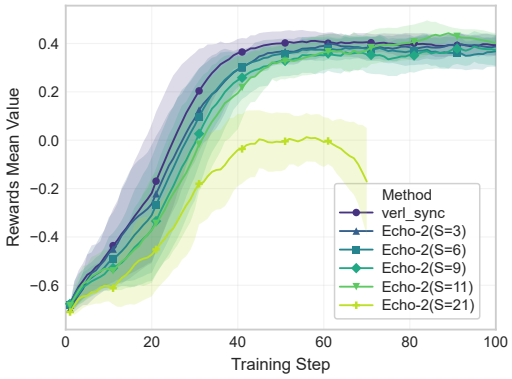



Figure 6: Effect of bounded staleness $S$ on RL quality in ECHO-2 for Qwen3-4B.

so the entire downstream segment falls back to generating rollouts under its previous policy version until the chain is rebuilt and the new snapshot propagates through. During this transient, the per-step trajectory throughput observed by the learner is reduced by more than the $1/8$ floor implied by losing a single worker. In contrast, a rollout-stage fault removes exactly one worker's contribution and has no effect on other workers' policy installation, so the peak drop matches the per-worker share. The same cascading effect explains why broadcast recovery (312 s) takes longer than rollout recovery (240 s): chain reconstruction must complete and the snapshot must traverse the rebuilt segment before downstream workers can resume generation under the new version, whereas a rollout-stage replacement only needs to download the snapshot once and start generating.

**Operating-regime caveat.** These results characterize the worst case under the no-redundancy setting ($\gamma = 1.0$). The default $\gamma = 1.1$ used in our main experiments absorbs single-worker faults entirely without requiring replacement, at the cost of a small ($\sim 10\%$) overprovisioning. We do not claim robustness to large-scale concurrent failures (e.g., $\geq 50\%$ unavailable workers), which fall outside the operating regime targeted by ECHO-2.

# E Supplementary Experiments

## E.1 Results of Qwen3-4B

In this section, we show cost-quality comparison for Qwen3-4B, which demonstrates similar performance to Qwen3-8B in Figure 3. Moreover, as shown in Figure 6, we also conduct empirical experiments of staleness for Qwen3-4B with standard GRPO.

## E.2 Wide-Range Benchmarks

Table 5 reports reward scores after RL post-training on 5 math reasoning benchmarks: AIME24 MAA [2024], OmniMath Gao et al. [2024], JEE Arora et al. [2023], HardMath Fan et al. [2024], and `IMO-answer-400` Luong et al. [2025]. We compare ECHO-2 ($S = 3$) with VERL under the same reward model and training configuration, using Qwen3-4B and Qwen3-8B as the base models. Across all datasets and both model scales, ECHO-2 maintains reward performance comparable to VERL,

Table 5: Reward scores after RL post-training on math reasoning benchmarks. AIME24 reports avg@64, JEE reports avg@8, and OmniMath / HardMath /IMO-A report avg@1 (i.e., Pass@1). IMO-A denotes `IMO-answer-400`. All results are reported under the same training configuration.

| Method | Model | AIME24 | OmniMath | JEE | HardMath | IMO-A | MEAN |
|---|---|---|---|---|---|---|---|
| initial | Qwen3-4B | 25.0 | 28.5 | 20.15 | 11.15 | 11.0 | 19.16 |
| | Qwen3-8B | 29.1 | 28.12 | 18.57 | 11.18 | 11.0 | 19.59 |
| VERL | Qwen3-4B | 45.78 | 40.65 | 36.82 | 24.17 | 23.25 | 34.13 |
| | Qwen3-8B | 47.92 | 41.92 | 32.31 | 25.33 | 29.0 | 35.30 |
| ECHO-2 | Qwen3-4B | 45.16 | 41.67 | 34.32 | 26.3 | 20.75 | 33.64 |
| | Qwen3-8B | 48.8 | 40.31 | 39.51 | 26.87 | 23.25 | 35.75 |

demonstrating that distributed rollouts with bounded staleness do not degrade RL optimization quality and provide a cost-efficiency opportunity, and ECHO-2 realizes it.

### E.3 Validation on a Larger Model and Comparison with Prime-RL

To assess whether ECHO-2's design generalizes beyond the 4B/8B regime reported in the main paper, and to compare directly against a recent decentralized-rollout system, we run ECHO-2 on a 32B-parameter base model and benchmark it against Prime-RL [Team et al., 2025b] under matched hyperparameters.

**Setup.** We post-train QwQ-32B with GRPO ($\epsilon = 0.2$, $\delta = 4$) on the TARGET-SHORT task from Team et al. [2025b] for 100 training steps, with $256 \times 16$ samples per rollout step. The average response length is approximately 4,000 tokens for both systems. All RL hyperparameters (clipping range, KL coefficient, batch size, sampling temperature) are matched across systems.

For training, both systems use 8×H100 GPUs. The two systems differ only in their rollout deployment: Prime-RL runs 10×H100 rollout workers on a single intranet node with effectively unlimited bandwidth between training and rollout, representing its idealized upper bound. ECHO-2 runs 76×RTX 5090 rollout workers connected over a realistic wide-area network, served by Parallax [Tong et al., 2025].

**Cost comparison.** Table 6 reports total dollar cost over the 100-step run, computed using the same public rental prices listed in Table 1 ($3.06/hr for A100, $0.35/hr for RTX 5090; we use $2.90/hr as a representative rate for H100 rollout workers, which is the lower end of common cloud quotes at the time of writing). Despite operating under the more constrained wide-area regime, ECHO-2 achieves a 4.6% end-to-end cost reduction relative to Prime-RL's idealized intranet deployment. The cost gap arises because ECHO-2's overlap-aware provisioning and cost-aware activation make commodity RTX 5090 workers a viable substitute for high-end H100 rollout GPUs without sacrificing learner saturation; the same substitution is not available to Prime-RL, whose architecture does not expose unit-throughput cost $\rho_i = c_i/\mu_i$ as a scheduling input.

Table 6: End-to-end cost comparison on QwQ-32B over 100 GRPO training steps with matched hyperparameters. Prime-RL is evaluated on an idealized intranet deployment; ECHO-2 is evaluated under realistic wide-area bandwidth.

| | Training GPUs | Rollout GPUs | Total cost ($) |
|---|---|---|---|
| Prime-RL | 8× H100 | 10× H100 | 1,914 |
| ECHO-2 (ours) | 8× H100 | 76× RTX 5090 | 1,826 |

**Interpreting the magnitude of the gap.** The 4.6% improvement over Prime-RL is substantially smaller than the 33–36% improvement ECHO-2 achieves over the centralized VERL baseline (Section 5.2), and the two numbers should not be conflated. The comparison against VERL measures the total cost saving from moving rollout generation off centrally managed data center GPUs onto distributed commodity workers and combines two independent gains: (i) the architectural shift to

disaggregated rollout execution and (ii) the system mechanisms (overlap-aware provisioning, peer-forwarding broadcast, cost-aware activation) that make this shift feasible under wide-area constraints. The comparison against Prime-RL, which already adopts disaggregated rollout, isolates the contribution of component (ii) alone. The $4.6\%$ figure is also conservative because Prime-RL is evaluated on an idealized intranet, which is the most favorable setting for any centralized-broadcast design; in a wide-area deployment with a finite uplink budget at the trainer, Prime-RL's broadcast cost would grow with rollout fleet size while ECHO-2's peer-forwarding cost remains bounded (Section 5.5.1).

**Training stability under chunked dissemination.** A natural concern at the 32B scale is whether the longer per-snapshot broadcast time interacts poorly with chunked policy dissemination and destabilizes optimization. Over the full 100-step run we observe no such effect: reward trajectories track the Prime-RL baseline closely throughout training, and final reward at step 100 matches the Prime-RL endpoint within run-to-run variance. This is consistent with the system design, which decouples *how* a snapshot is delivered from *which* snapshot version a rollout is tagged with: chunked dissemination only affects when a worker installs the new policy, while the staleness budget $S$ continues to bound the version gap consumed by the learner regardless of broadcast topology.

**Caveat on resource scope.** We were not able to evaluate at the 70B+ scale due to compute budget. The QwQ-32B result above is the largest model on which we have run ECHO-2 end-to-end. We expect the design to remain feasible at larger scales because $T_{\text{train}}$ also grows with model size, preserving the overlap region required by Equation (2); we leave a direct empirical study to future work.

# F Data Plane Implementation and Poker Game Alignment via Sandbox Integration

To demonstrate the versatility of ECHO-2's decoupled *Data Plane*, we extend our evaluation from static mathematical reasoning to a dynamic, interactive environment: **No-Limit Texas Hold'em**. This case study illustrates how ECHO-2 adapts to non-standard modalities (game logs and episodic returns) *without modifying* the underlying *Learning Plane* or *Rollout Plane*. Concretely, we only instantiate a task-specific **Data Plane Adapter** that (i) interfaces with a poker sandbox, (ii) standardizes raw logs into the canonical rollout schema, and (iii) materializes the additional metadata $\Omega$ (e.g., token masks and normalized advantages) required by GRPO, yielding the canonical record $\tau = (x, y, r, v, \Omega)$ consumed by the shared replay buffer and the learner.

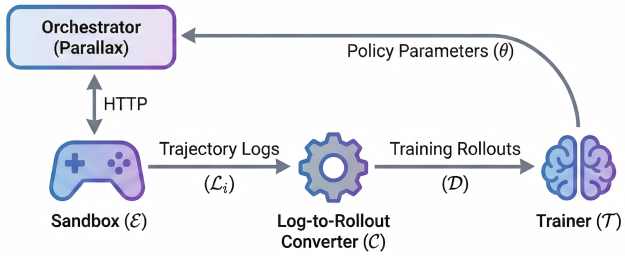



Figure 7: Overview of the Echo-2 Poker Game Alignment system. The Orchestrator (Parallax) interfaces with the Sandbox ($\mathcal{E}$) to generate Trajectory Logs ($\mathcal{L}_i$). The Log-to-Rollout Converter ($\mathcal{C}$) processes these logs into Training Rollouts ($\mathcal{D}$), which are then used by the Trainer ($\mathcal{T}$) to update the policy parameters ($\theta$), closing the iterative training loop.

## F.1 Data Plane Interfaces and Task Integration

The Data Plane defines the *task semantics* of ECHO-2 while preserving the versioned execution in Section 4.1 and the end-to-end loop in Algorithm 1. Concretely, it specifies how a workload is mapped to immutable, version-tagged trajectory records stored in the replay buffer: $\tau = (x,\ y,\ r,\ v,\ \Omega)$, where $(x, y)$ is the prompt-response pair, $r$ is the scalar reward, $v$ is the snapshot version used to generate $y$, and $\Omega$ is optional task metadata. The buffer $\mathcal{B}$ indexes $\tau$ by version and enforces bounded staleness.

A task is integrated by implementing a Data Plane adapter that (i) constructs prompts $x$, (ii) defines the reward function $\mathcal{R}$ used by rollout workers to produce $r = \mathcal{R}(x, y)$, and (iii) defines how $\Omega$ is materialized into learner-side training signals (e.g., masks and normalized advantages) under the chosen objective (e.g., GRPO with KL regularization).

### F.2 System Overview: A Data Plane Instantiation

We implement a specialized Data Plane Adapter that bridges the raw poker sandbox and ECHO-2's training interface. As shown in Figure 7, the pipeline consists of three phases: *Sandbox Interaction*, *Log Standardization*, and *Reward-Augmented Rollout Generation*. The adapter outputs a unified rollout tuple that can be consumed directly by the generic *Rollout Plane* and *Learning Plane*.

Let the policy be an autoregressive language model $\pi_\theta(\mathbf{y} \mid \mathbf{x})$, where $\mathbf{x}$ represents the serialized game context and $\mathbf{y}$ represents the agent's decision (betting action text). We denote a reference policy (for KL regularization) as $\pi_{\text{ref}}$.

**One-line task switching via sandbox adapters.** A key benefit of the decoupled Data Plane is that switching to a new interactive task only requires swapping the sandbox adapter configuration, while the Rollout/Learning Planes (and the replay schema $\tau = (x, y, r, v, \Omega)$) remain unchanged.

Unified Orchestration API (Poker ↔ MOBA)

```
# Task-specific: only the Data Plane adapter changes.
# Switching tasks is configuration-only:
# adapter.env = "moba://hok_v2" ; adapter.serializer = "
    moba_obs->prompt_v2"
adapter = SandboxAdapter(
    env="poker://nlhe_6max_v1",           # or "moba://dota2_v3
        ", "moba://hok_v2"
    serializer="envstate->chat_prompt_v2",
    action_renderer="action->text_v1",
    reward_spec="episodic_return",
    metadata_spec="grpo:turn_mask+adv_norm"
)

echo = Echo2(policy=pi_theta, reference=pi_ref)

echo.set(dataset=adapter.stream(mode="offline_or_live"))
echo.set(staleness=3)
echo.warmup()
echo.run()
```

### F.3 Phase 1: Environment Interaction (Sandbox → Raw Logs)

We deploy a sandbox environment $\mathcal{E}$ simulating a poker table. For each episode $i$, the environment records a raw interaction log $\mathcal{L}_i$:

$$\mathcal{L}_i = \{(s_{i,t}, a_{i,t}, r_{i,t})\}_{t=1}^{T_i}, \tag{14}$$

where:

- $s_{i,t}$ is a textual description of the private hand, community cards, pot size, and derived odds (e.g., `"Hand: [Ah, Kd], Board: [Qs, Th, 2c], Pot: 100"`).
- $a_{i,t}$ is a structured action rendered as text (e.g., `"Action: Raise 50"`).
- $r_{i,t}$ is the *immediate chip delta*, i.e., the change in chip stack relative to the previous turn.

Unlike math tasks where rollouts are generated by the model under training, poker logs may initially come from rule-based baselines or prior model iterations, demonstrating ECHO-2's ability to consume *off-policy* data (and strictly on-policy data if connected to live rollout workers). In ECHO-2, the scalar reward $r$ is produced in the Rollout Plane (co-located with environment interaction), while the Data Plane defines $\mathcal{R}$ and the post-processing rules used to derive $\Omega$ for learning.

### F.4 Phase 2: Standardization and Conversion (Raw Logs → Canonical Messages)

The core responsibility of the Data Plane is to convert heterogeneous logs $\mathcal{L}_i$ into a unified rollout format compatible with the generic Learning Plane. The adapter $\mathcal{C}$ transforms the raw log into a chat-formatted message sequence $\mathcal{M}_i$ by flattening complex game states into a standard prompt-response template:

$$\mathcal{M}_i = \left[m_{i,0}^{\text{sys}},\ m_{i,0}^{\text{usr}},\ m_{i,1}^{\text{asst}},\ m_{i,1}^{\text{usr}},\ \ldots,\ m_{i,T_i}^{\text{asst}}\right], \tag{15}$$

where $m_{i,0}^{\text{sys}} = \texttt{SystemPrompt}$ encodes global poker rules $\mathcal{P}_{\text{rules}}$, and each turn is represented as a user state message followed by an assistant action message:

$$m_{i,t}^{\text{usr}} = \text{``}\texttt{State:}\quad s_{i,t}\text{''}, \qquad m_{i,t}^{\text{asst}} = \text{`'}a_{i,t}\text{''}. \tag{16}$$

Optionally, for bookkeeping we may insert rewards as user messages ``$\texttt{Reward:}\quad r_{i,t}$''; however, training signals are ultimately computed from numeric rewards inside the Data Plane.

We then linearize $\mathcal{M}_i$ using the tokenizer chat template and obtain token IDs:

$$\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,L_i}) = \text{Tokenize}(\mathcal{M}_i), \tag{17}$$

along with an attention mask $\mathbf{a}_i \in \{0,1\}^{L_i}$. ECHO-2 uses left padding to batch variable-length episodes.

The following implementation demonstrates how raw environment outputs are iteratively converted into the user-assistant message structure:

**Constructing Canonical Messages**

```python
# Extract from GemEnvRollout.get_lm_inputs
messages = [
    {"role": "system", "content": self.system_prompt},
    {"role": "user", "content": self.prefix_lookup[env_id]}
]

for idx, content in enumerate(env_output["history"]):
    messages[-1]["content"] += f"\nTurn␣{idx␣+␣1}:\n"

    # Flatten State s_{i,t}
    if "state" in content:
        FORMAT = "<answer>␣...␣</answer>"
        messages[-1]["content"] += f"State:\n{content['state
            ']}\n␣Always␣output:␣{FORMAT}\n"

    # Append Assistant Action a_{i,t}
    if "llm_response" in content:
        messages.append({"role": "assistant", "content":
            content["llm_response"]})

    # Optional: Insert intermediate rewards for bookkeeping
    if "reward" in content:
        messages.append({"role": "user", "content": f"Reward:\n
            {content['reward']}\n"})
```

### F.5 Phase 3: Turn-Aware Masking and Reward-Augmented Rollouts

Poker supervision is sparse and episodic; therefore, the Data Plane additionally computes (i) **turn-aware masks** that restrict learning to assistant tokens, and (ii) **advantages** derived from final chip outcomes.

#### F.5.1 Turn-Aware Masking

We construct turn indicators using a special *turn-start* token ID $\tau_{\text{start}}$ (e.g., <|im_start|> in Qwen-style templates). Define:

$$u_{i,t} = \mathbb{I}[x_{i,t} = \tau_{\text{start}}], \qquad c_{i,t} = \sum_{k=1}^{t} u_{i,k}, \tag{18}$$

where $c_{i,t}$ is the chat-turn index of token $t$. The assistant-response mask is:

$$m_{i,t}^{\text{resp}} = \mathbb{I}[c_{i,t} > 1] \cdot \mathbb{I}[c_{i,t} \bmod 2 = 1], \tag{19}$$

selecting tokens after the system prompt that belong to assistant turns. We set the loss mask as $m_{i,t}^{\text{loss}} = m_{i,t}^{\text{resp}}$, so learning is restricted to the agent's action tokens. Under next-token prediction, masks are aligned with shifted targets $y_{i,t} = x_{i,t+1}$.

The implementation below corresponds to the calculation of $m_{i,t}^{\text{resp}}$ and the logic for aligning rewards to turn boundaries:

Turn-Aware Mask Computation

```python
def get_masks_and_scores(input_ids, tokenizer, all_scores,
    enable_response_mask=False):
    special_token, reward_token = get_special_tokens(tokenizer)

    # Calculate turn indicators c_{i,t}
    turn_starts = torch.where(input_ids == special_token, 1, 0)
    turn_indicators = torch.cumsum(turn_starts, dim=-1)

    # Generate Response Mask m^{resp}_{i,t}
    # Selects tokens where turn count is odd (assistant) and >
        1 (after sys prompt)
    response_mask = (turn_indicators % 2 == 1) & (
        turn_indicators > 1)

    # Assign scores to the last token of the turn
    score_tensor = torch.zeros_like(input_ids, dtype=torch.
        float32)
    scores = [sum(i) for i in all_scores] # Sum of rewards R_i
    score_tensor[:, -1] = torch.tensor(scores, dtype=torch.
        float32)

    # Alignment adjustments for causal masking
    score_tensor = score_tensor[:, 1:]
    loss_mask = response_mask[:, :-1]

    return score_tensor, loss_mask, response_mask
```

### F.5.2 Outcome-Based Returns and Group-wise Normalization

While poker is high-variance, our primary evaluation metric is the *final chip change*. For episode $i$, the trajectory-level return is:

$$R_i = \sum_{t=1}^{T_i} r_{i,t}, \tag{20}$$

equal to the net profit/loss in chips. We use the trajectory-level return as the scalar reward stored in the record, i.e., $r_i := R_i$.

To reduce variance and stabilize policy updates, the Data Plane applies **group-wise normalization**. For a group of episodes $G$ (e.g., sharing similar initial private hands or other coarse state descriptors), we compute the normalized advantage:

$$\hat{A}_i = \frac{R_i - \text{Mean}\big(\{R_j\}_{j \in G}\big)}{\text{Std}\big(\{R_j\}_{j \in G}\big) + \epsilon}. \tag{21}$$

We then broadcast $\hat{A}_i$ to the response tokens:

$$\hat{A}_{i,t} = \hat{A}_i \cdot m_{i,t}^{\text{resp}}, \tag{22}$$

so that only assistant tokens receive non-zero advantage.

This normalization logic supports multiple grouping strategies (e.g., by initial state or batch) to compute the standardized returns used in the GRPO objective:

Group-wise Reward Normalization

```python
def _normalize_score_tensor(self, score_tensor, env_outputs):
    # Grouping logic (G)
    if self.reward_normalization['grouping'] == "state":
        group_tags = [out["group_id"] for out in env_outputs]

    # Calculate Mean and Std for the group
    # \hat{A}_i = (R_i - Mean) / (Std + epsilon)
    norm_func = lambda x: (x - x.mean(dim=-1, keepdim=True)) / \
                          (x.std(dim=-1, keepdim=True) + 1e-6)

    # Map indices to groups
    group2index = {}
    for i, tag in enumerate(group_tags):
        if tag not in group2index: group2index[tag] = []
        group2index[tag].append(i)

    # Apply normalization per group
    acc_scores = score_tensor[:, -1]
    normalized = acc_scores.clone()
    for group, idxs in group2index.items():
        normalized[idxs] = norm_func(normalized[idxs])

    score_tensor[:, -1] = normalized
    return score_tensor
```

### F.5.3 GRPO-Style Policy Gradient Objective

The adapter emits the canonical version-tagged record $\tau_i = (x_i, y_i, r_i, v_i, \Omega_i)$, where $r_i = \sum_{t=1}^{T_i} r_{i,t}$ is the episode return and $\Omega_i$ includes task metadata such as $(m_i^{\text{loss}}, m_i^{\text{resp}})$ and grouping tags used to compute $\hat{A}_i$. After sampling $\tau_i$ from the replay buffer, the learner materializes the training tensors $(\mathbf{x}_i, \mathbf{a}_i, m_i^{\text{loss}}, m_i^{\text{resp}}, \hat{\mathbf{A}}_i)$.

To account for the distribution shift between the sampling policy $\pi_{\text{sampler}}$ and the current learner $\pi_{\text{learner}}$, we define the token-level likelihood ratio for token $t$ in episode $i$ as:

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid \mathbf{x}_i, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid \mathbf{x}_i, y_{i,<t})}. \tag{23}$$

The training objective $\mathcal{J}(\theta)$ incorporates truncated importance sampling to stabilize updates when reusing off-policy data from the buffer:

$$\mathcal{J}(\theta) = \mathbb{E}_{a \sim \pi_{\text{sampler}}(\theta_{\text{old}})} \left[ \underbrace{\min\left( \frac{\pi_{\text{learner}}(a, \theta_{\text{old}})}{\pi_{\text{sampler}}(a, \theta_{\text{old}})}, C \right)}_{\text{truncated importance ratio}} \cdot \bar{\mathcal{J}}(\theta) \right], \tag{24}$$

where $C$ is a hyper-parameter and $\bar{\mathcal{J}}(\theta)$ denotes the GRPO-style clipped surrogate objective with KL regularization:

$$\bar{\mathcal{J}}(\theta) = \frac{1}{\sum_t m_{i,t}^{\text{resp}}} \sum_{t=1}^{L_i - 1} m_{i,t}^{\text{resp}} \cdot \min\Big( \rho_{i,t}(\theta) \hat{A}_{i,t},\ \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon_c, 1+\epsilon_c\big) \hat{A}_{i,t} \Big) - \beta\, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}). \tag{25}$$

Intuitively, the truncated ratio prevents gradient instability when the current policy deviates significantly from the data-collection policy. This allows the Learning Plane to robustly leverage diverse experiences from the Data Plane, demonstrating that poker environment support requires only a specialized Data Plane instantiation.

Table 7: Texas Hold'em evaluation of different player policies against three rule-based opponents and an LLM opponent (highlighted). The reported metric is the *final chip change* (net chips at the end of an episode/match relative to the initial stack). Positive values indicate net profit, while negative values indicate net loss. Only Qwen3-0.6B includes a GRPO-trained variant (second row, marked +GRPO); all other rows are direct LLM policies without GRPO training.

| Model | Rule-based$_1$ | Rule-based$_2$ | Rule-based$_3$ | LLM |
|---|---|---|---|---|
| Qwen3-0.6B [Team, 2025] | 0.571 | 0.514 | 0.592 | -1.677 |
| Qwen3-0.6B [Team, 2025] +GRPO | -0.195 | -0.599 | -0.451 | 1.245 |
| Qwen3-30B-A3B [Team, 2025] | -0.397 | 0.093 | -0.1225 | 0.4265 |
| Qwen3-next-80B-A3B-instruct [Team, 2025] | -0.399 | 0.1005 | -0.0055 | 0.304 |
| GPT-5 [OpenAI, 2025] | -0.216 | -0.1955 | 0.1455 | 0.266 |
| Grok-4 [xAI, 2025] | -0.3915 | -0.2915 | -0.228 | 0.911 |
| Claude-sonnet-4.5 [Anthropic, 2025] | -0.1155 | -0.055 | 0.056 | 0.1145 |
| Gemini-2.5-flash [Comanici et al., 2025] | -0.173 | -0.2165 | 0.0305 | 0.359 |
| Gemini-2.5-pro [Comanici et al., 2025] | -0.199 | -0.2055 | -0.04 | 0.4445 |

### F.6 Texas Hold'em Performance

We evaluate our method on a Texas Hold'em environment using the *final chip change* (net chips at the end of a match relative to the initial stack; higher is better). In Table 7, **columns correspond to different opponents**—three rule-based opponents (Rule-based$_1$–Rule-based$_3$) and one LLM opponent (highlighted). **Rows correspond to the evaluated player/agent**. For **Qwen3-0.6B**, we report both the *base* model (first row) and its *GRPO-trained* variant (second row, marked +GRPO). For all other backbones, we report the performance of their *direct LLM policies* (i.e., without GRPO training) under the same evaluation protocol.

**Main result: GRPO improves Qwen3-0.6B against the LLM opponent.** For Qwen3-0.6B, GRPO flips the outcome against the LLM opponent from a net loss ($-1.677$) to a net profit ($+1.245$), demonstrating that GRPO can substantially improve end-of-game profitability in the most challenging setting. Meanwhile, performance against the three rule-based opponents becomes negative after GRPO, suggesting a trade-off that may be addressed by multi-opponent training or more diverse opponent sampling.