AReaL-Hex: Accommodating Asynchronous RL Training over Heterogeneous GPUs

Anonymous authors
Paper under double-blind review

Abstract

Heterogeneous deployment offers a promising approach to improving cost-efficiency in RL training. Recent research shows that asynchronous RL training can disaggregate computation stages, i.e., rollout generation, reward computation, and policy/value updates, across disjoint hardware while maintaining training stability, creating great opportunities for heterogeneous deployment. We present AReaL-Hex, a heterogeneity-aware asynchronous RL training system that effectively schedules rollout generation and policy model training across heterogeneous GPUs under data staleness bounds. We design a two-phase scheduler: (**i**) a constrained search with MILP to select per-stage parallel strategies and workload assignments given the available resources, and (**ii**) a graph-partitioning algorithm that allocates heterogeneous GPUs to maximize RL training throughput. Built atop a fully asynchronous RL training system, AReaL-Hex maps HBM-I/O-bound inference and compute-bound training to cost-efficient resources and balances their execution to avoid idleness and stale rollout trajectories. On mathematical reasoning tasks with various model scales (1.5B to 14B), compared to homogeneous deployments of state-of-the-art asynchronous RL systems: (i) with the same total budgets, AReaL-Hex delivers up to 1.50× higher throughput; (ii) achieving the same throughput, AReaL-Hex results in up to 1.46× reduction in cost.

1 Introduction

Reinforcement learning (RL) has emerged as a popular approach for enhancing large language models (LLMs) across various tasks (Zhang et al., 2025), including math (Ren et al., 2025), code generation (Hui et al., 2024), agentic tool use (Yao et al., 2025), etc, which becomes essential to advance AI development. Unlike conventional large-scale pretraining by standard stochastic gradient-based optimization, RL training comprises multiple coupled stages, each with heterogeneous compute intensities, memory footprints, and communication patterns (Sheng et al., 2025); furthermore, recent studies suggest that asynchronous RL can disaggregate these stages over different hardware without compromising training performance (Fu et al., 2025; Zhong et al., 2025a). Such intrinsic computational heterogeneity under disaggregated infrastructure naturally provides opportunities for heterogeneous deployment. We explore *how to effectively schedule asynchronous RL workloads on heterogeneous GPUs to maximize end-to-end RL training throughput and cost-efficiency*.

Deploying asynchronous RL training in heterogeneous environments promises substantial gains in resource utilization and reductions in cost. RL training for LLM includes three stages: (**i**) *rollout generation*, which conducts HBM IO–bound inference on GPUs to produce rollouts; (**ii**) *reward estimation*, which uses strategies that require CPUs (e.g., sandboxed execution for coding tasks or rule-based evaluation for math) or GPUs for LLM-based value models; and (**iii**) *model training*, which conducts compute-bound updates of the policy (and optionally value) LLMs via stochastic gradient-based optimizations, potentially with a reference model for stability. Recently, fully asynchronous RL training paradigms have been proven effective, where RL stages can execute over disjoint GPUs. The rollout GPUs continuously generate, while training GPUs update opportunistically, rather than waiting for the slowest rollout in a batch. This disaggregated paradigm exhibits inherent heterogeneity in terms of compute intensity, memory bandwidth, and communication paradigm in parallel deployment, making it a natural fit for heterogeneous GPUs, which are common in cloud.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

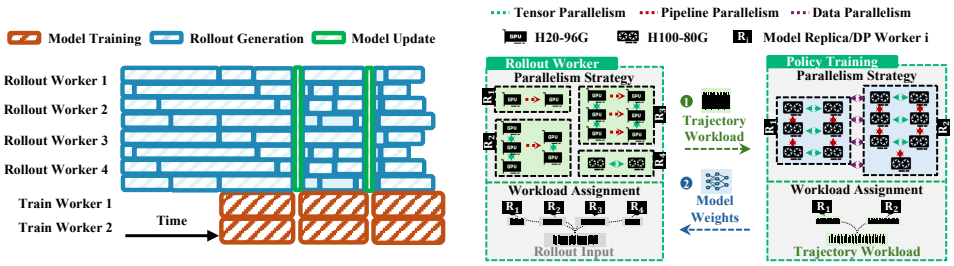


Figure 1: **Left:** Asynchronous RL training workflow. Rollout generation and model training execute concurrently; both are paused when updated parameters become available to synchronize rollout workers' weights. **Right:** An example scheduling result deploying multiple rollout replicas, each potentially configured with distinct parallel strategies and hardware resources. (**i**) Completed rollouts are transmitted to training workers. (**ii**) Training workers broadcast updated parameters to rollout workers via NCCL collective operations.

We believe *an efficient scheduler* that maps distributed executions across stages to more cost-efficient GPUs and balances their asynchronous interactions could lower both time and monetary cost.

However, scheduling asynchronous RL training workflows over heterogeneity is fundamentally challenging due to complex RL training dynamics and data staleness constraints (Noukhovitch et al., 2025). An effective heterogeneity-aware scheduler must regulate the asynchronous interaction between rollout and training workers under data staleness constraints, i.e., training workers require a steady supply of rollouts, which must not be generated by policy LLM too stale. The difficulty is further amplified by the heterogeneity that enlarges the search space: GPUs differ in compute throughput, HBM memory capacity and bandwidth, and interconnect topology, leading to an exponential search space. The scheduled plan must both maintain high GPU utilization and limit rollout data staleness, which demands careful design of the scheduler.

Despite growing interest in heterogeneity-aware scheduling for LLM serving, existing efforts address only isolated slices of the problem, i.e., either inference (Mei et al., 2025; Jiang et al., 2024; 2025b) or training (Yan et al., 2024; Um et al., 2024), thus falling short of the requirements of asynchronous RL training. Simply composing these techniques is inadequate: an asynchronous RL training pipeline is inherently multi-stage and requires explicit control of rollout data staleness, i.e., ensuring the number of rollouts generated by stale parameters is within an upper-bound. Thus, an effective scheduler must jointly consider cross-stage heterogeneity and inter-stage coupling, which makes the scheduling space of asynchronous RL training fundamentally different. This coupling introduces practical inefficiencies if not properly handled. For instance, when training or inference is optimized in isolation, excessively fast inference workers may become idle once the generated rollouts quickly reach the data staleness bound. Conversely, if the execution speeds of rollout and training workers are not well balanced, the non-overlapped cost of weight updates can introduce additional inefficiencies.

In this paper, we introduce AReaL-Hex, a novel asynchronous RL training system designed for efficiently deploying RL training workloads on heterogeneous GPUs. The core of AReaL-Hex is a heterogeneity-aware scheduling algorithm that determines effective allocation of generation, training computation, and corresponding communication under different parallelism over heterogeneous GPUs and network interconnects. Concretely, our contributions are:

- **Contribution 1.** We systematically characterize asynchronous RL workloads and demonstrate that their intrinsic computational heterogeneity naturally aligns with heterogeneous GPU clusters. Through profiling, we show that matching each stage's resource demands to appropriate GPU types substantially improves overall system efficiency compared to homogeneous deployments.
- **Contribution 2.** We formulate the problem of asynchronous RL training scheduling over heterogeneity, encompassing resource allocation, parallel strategies, and workload assignments, as a constrained optimization problem with data staleness bounds. We solve the problem by a two-phase algorithm: (**i**) the first phase employs constrained search and MILP to determine optimal parallel strategies and workload assignments for rollout and training under given resources; and (**ii**) the second phase uses graph partition to optimize resource allocation across heterogeneous GPUs.

- **Contribution 3.** We implement AReaL-Hex and conduct experiments for mathematical reasoning RL tasks across 1.5B, 7B, and 14B models. Compared to state-of-the-art asynchronous RL systems on homogeneous GPUs: (**i**) with the same training budget, AReaL-Hex achieves 1.31× ∼1.50× higher training throughput, demonstrating the substantial benefits of heterogeneity-aware scheduling for asynchronous RL training on heterogeneous GPUs; (**ii**) achieving the same training throughput, AReaL-Hex results in up to 1.46× reduction in training cost.

2 Preliminaries and Related Work

RL training for LLMs. RL has been applied to enhance the reasoning capabilities of LLMs (OpenAI, 2024; 2025), improving performance on mathematical reasoning, code generation, and multi-hop question answering (Chen et al., 2021; Hendrycks et al., 2021; Liu et al., 2023; Wang et al., 2025; Gao et al., 2025). In *synchronous* RL training, rollout generation uses the latest parameters to produce rollouts, and training updates parameters based on these rollouts (Schulman et al., 2017; Sheng et al., 2025). In *asynchronous* RL training, generation continuously produces rollouts using potentially stale parameters, while training updates parameters based on such rollouts (Mnih et al., 2016; Espeholt et al., 2018; 2019; Mei et al., 2023; Fu et al., 2025).

RL training systems. Recent works develop efficient RL training systems (Mei et al., 2024; Zhong et al., 2025b; Wu et al., 2025). Among them, Verl (Sheng et al., 2025) accelerates RL training via a 3D-HybridEngine for zero-redundancy resharding and auto-mapping for GPU allocation. AReaL (Fu et al., 2025) decouples generation from training with staleness-aware asynchronous execution. StreamRL (Zhong et al., 2025a) supports one-step asynchronous streaming, while HetRL (He et al., 2026) jointly schedules RL tasks on heterogeneous GPUs and networks. AReaL-Hex jointly optimizes heterogeneous allocation, parallelism, replicas, and workload assignment under a configurable multi-step staleness bound.

LLM services. Recent works enhance LLM services via scheduling (Um et al., 2024; Mo et al., 2025; Mao et al., 2025; Jiang et al., 2025a; Yuan et al., 2022; Miao et al., 2023; Jiang et al., 2024; 2025b). In LLM training, HexiScale (Yan et al., 2024) supports hybrid and asymmetric parallelism for heterogeneous GPUs. In LLM inference, Helix (Mei et al., 2025) formulates heterogeneous GPU allocation as a max-flow problem solved via MILP. DistServe (Zhong et al., 2024) and BlendServe (Zhao et al., 2024) disaggregate prefill and decoding for independent optimization. Different from these works, we study asynchronous RL training, which requires *joint optimization of training and inference*. Rollout generation and model training are coupled through a data staleness bound, making isolated optimization ineffective. Our algorithm jointly optimizes all components under staleness constraints.

3 Observation and Opportunity

Heterogeneous GPUs align their hardware capabilities with heterogeneous computational demands of asynchronous RL training. We explore this and present our observations through a case study.

Benchmark setups. Maintaining same total costs, we benchmark asynchronous RL training throughputs across three settings on 1.5B, 7B, and 14B models: (**i**) *Homogeneous setting 1*: 32 H800 GPUs. (**ii**) *Homogeneous setting 2*: 88 H20 GPUs. (**iii**) *Heterogeneous setting*: 24 H800 and 24 H20 GPUs. We break down the major training execution costs, i.e., rollout generation and model training latency. Similar to MegaScale-Infer (Zhu et al., 2025): rental price for H800/H20 GPUs is \$5.28/h or \$1.85/h.

Observation 1: Performance of homogeneous setting 1 is suboptimal with inefficient inference. H800 GPUs have limited HBM memory bandwidth but high computational power; under asynchronous execution, we allocate 24 H800 GPUs to rollout generation and 8 H800 GPUs to model training to balance their speed. The performance remains suboptimal due to inefficient rollout generation.

Observation 2: Performance of homogeneous setting 2 is suboptimal with inefficient model training. We allocate 24 H20 GPUs to rollout generation and 64 H20 GPUs to model training. Slow training causes rollout workers to wait at the staleness bound; this explains the effective rollout-latency gap despite using the same 24 H20 rollout GPUs as the heterogeneous setting.

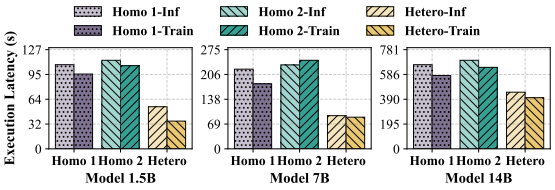


Figure 2: Effective rollout (INF) and training (TRAIN) latency across homogeneous settings 1, 2 and the heterogeneous setting. Rollout latency includes staleness-induced waiting.

Table 1: Rollout and training per-token cost comparison.

Model	GPU	$/Inference Token	$/Training Token
1.5B	H800	$\$5.58 \times 10^{-3}$	$\$1.66 \times 10^{-3}$
	H20	$\$2.06 \times 10^{-3}$	$\$5.16 \times 10^{-3}$
7B	H800	$\$1.14 \times 10^{-2}$	$\$3.05 \times 10^{-3}$
	H20	$\$4.15 \times 10^{-3}$	$\$9.51 \times 10^{-3}$
14B	H800	$\$3.43 \times 10^{-2}$	$\$0.99 \times 10^{-2}$
	H20	$\$1.27 \times 10^{-2}$	$\$3.10 \times 10^{-2}$

Observation 3: The heterogeneous setting delivers superior performance by leveraging the advantages of both GPU types. We assign the HBM memory-bound rollout generation to 24 H20 GPUs, fully utilizing the abundant HBM memory bandwidth. On the other hand, we assign the compute-bound model training to H800 GPUs, fully leveraging the computational power. Under this arrangement, both phases achieved optimal performance. Compared to homogeneous settings 1 and 2, the heterogeneous setting achieved up to 2.67×, and at least 1.49× reduction in training latency.

Cost-efficiency analysis. Asynchronous RL training workloads across heterogeneous GPUs naturally align with the distinct cost-efficiency of different GPU types. In Table 1, we present the per-token cost for H800 and H20 GPUs for both training and inference. For rollout generation, H20 demonstrates 2.72× higher cost efficiency; for training, H800 demonstrates 3.12× higher cost efficiency.

4 SCHEDULING WITH HETEROGENEITY

We formalize the problem in §4.1, the search phase in §4.2, and the repartition phase in §4.3.

4.1 PROBLEM FORMULATION

Formulation. Given a set of heterogeneous GPUs $\mathcal{D}$, a staleness constraint η, and an expected generated rollout length distribution P, the goal of the scheduler is to identify the optimal asynchronous RL execution plan that minimizes $\delta(\eta)$-step averaged RL training time, where $\delta(\eta)$ represents a window size used to calculate the average training time depending on the staleness η. To optimize asynchronous RL training, the scheduling algorithm should determine:

- *Resource allocation*, i.e., the optimal allocation of the GPU set $\mathcal{D}_T$ for policy (and optionally value) model training, and the GPU set $\mathcal{D}_I$ for rollout generation. Note that $\mathcal{D}_T \cup \mathcal{D}_I = \mathcal{D}, \mathcal{D}_T \cap \mathcal{D}_I = \emptyset$, and ideal balances between $\mathcal{D}_T$ and $\mathcal{D}_I$ should maximize RL training throughput.
- *Parallel strategy and workload assignment*, i.e., the allocation of RL workload to heterogeneous GPUs under asynchronous RL training, given a resource allocation. This component needs to assign the parallel inference plan τ over $\mathcal{D}_I$ and the training plan σ over $\mathcal{D}_T$.

We define a solution including these two components as a *scheduled plan*. Concretely, given $\mathcal{D} = \{d_1, \ldots, d_N\}$ as a set of N GPUs with device memory limit M_d for GPU d, we define the scheduling problem to seek optimal parallel execution plans σ^* for model training and τ^* for rollout generation over the optimal resource allocation $\mathcal{D}_T^*$ and $\mathcal{D}_I^*$ to minimize $\delta(\eta)$-step average RL training execution time subject to each GPU's memory constraints:

$$
\begin{aligned}
&\sigma^*, \tau^*, \mathcal{D}_T^*, \mathcal{D}_I^* = \arg\min_{\sigma,\tau,\mathcal{D}_T,\mathcal{D}_I} \max\{C_T, C_I\} \\
&\text{where } C_T = C_{\text{TRAIN}}\left(\sigma, \mathcal{D}_T, \delta(\eta)\right), \\
&\qquad C_I = C_{\text{ROLLOUT}}\left(\tau, \mathcal{D}_I, P, \delta(\eta)\right) + C_{\text{REWARD}}\left(\delta(\eta)\right) \\
&\qquad\qquad + C_{\text{UPDATE}}(\sigma, \mathcal{D}_T, \tau, \mathcal{D}_I, \delta(\eta)), \\
&\text{s.t. } \mathcal{D}_T \cup \mathcal{D}_I = \mathcal{D}, \;\; \mathcal{D}_T \cap \mathcal{D}_I = \emptyset, \\
&\qquad \text{MEM-CUMSUM}\left(d\right) \le M_d \quad \forall d \in \mathcal{D}.
\end{aligned}
\tag{1}
$$

where C_T is the $\delta(\eta)$-step average training cost: $C_{\text{TRAIN}}(\sigma, \mathcal{D}_T, \delta(\eta))$ covers gradient computation and parameter updates for the policy (optionally value) model. C_I is the $\delta(\eta)$-step average rollout cost: $C_{\text{ROLLOUT}}(\tau, \mathcal{D}_I, P, \delta(\eta))$ for rollout generation, $C_{\text{REWARD}}(\delta(\eta))$ for reward and reference model computation, and $C_{\text{UPDATE}}(\sigma, \mathcal{D}_T, \tau, \mathcal{D}_I, \delta(\eta))$ for synchronizing updated weights to rollout workers. MEM-CUMSUM(d) denotes memory consumption on device d. We assume $C_I > C_T$: when $C_T > C_I$, the system approaches its staleness bound, causing rollout workers to stall until training consumes a batch. When $C_I > C_T$, the system leverages asynchronous execution.

Overview of the scheduling algorithm. Finding the optimal scheduled plan is NP-hard due to the exponential space of candidate allocations. Moreover, determining *resource allocation* $\mathcal{D}_T$, $\mathcal{D}_I$ and *parallel strategy/workload assignment* σ, τ introduces a circular dependency: the optimal plan σ, τ depends on the resource allocation $\mathcal{D}_T$, $\mathcal{D}_I$, which in turn depends on the plan. Inspired by the EM algorithm (Moon, 1996), we resolve this via a two-phase iterative approach:

- **Search-Phase**: Given a resource allocation, we employ *constrained and MILP-based search* to determine the optimal parallel strategy and workload assignment (§4.2). This phase optimizes system performance for both model training and rollout generation within allocated resources.
- **Repartition-Phase**: Given the estimated execution costs obtained from the Search-Phase, we apply a *cost-guided graph partition heuristic* to refine the resource allocation (§4.3). It balances resources between stages without a global-optimality guarantee; Appendix H.3 evaluates solution quality against exhaustive search.

In Algorithm 1 (Appendix G), two phases execute iteratively and terminate when the maximum execution cost (i.e., $\max\{C_T, C_I\}$) remains stable for multiple (e.g., 20) consecutive iterations.

4.2 SEARCH PHASE ALGORITHM

Given a resource allocation $\mathcal{D}_T$ and $\mathcal{D}_I$, the search phase determines the optimal execution plans for model training σ and rollout generation τ that maximize their throughput.

4.2.1 SEARCHING MODEL TRAINING PLAN

Given the resources set $\mathcal{D}_T$ allocated for the model training, we determine an optimal training execution plan σ, which involves both parallel strategy and workload assignment noted as CONSTRAINED_SEARCH$(\mathcal{D}_T)$, as follows:

Parallel strategy search. Given resources allocated to model training, we determine an optimal parallel strategy—encompassing data parallelism (DP), tensor model parallelism (TP), and pipeline parallelism (PP)—that minimizes training execution cost. Since different GPU types are typically connected via low-bandwidth networks (Um et al., 2024; Yan et al., 2024), we impose the constraint that both TP and DP must utilize GPUs of the *same* type. We enumerate all feasible strategies under this constraint, which substantially reduces search complexity while maintaining solution quality. Following Metis (Um et al., 2024), we select the strategy yielding maximal training throughput. Across pipeline stages, we allocate transformer layers proportionally to each GPU type's computational capability, with distinct DP and TP degrees per stage.

Workload assignment optimization. We employ a greedy sequence packing strategy (Fu et al., 2025) to balance computation across data parallel workers: For each training batch, we sequentially assign sequences to the DP worker with the minimum workload, measured by token count.

4.2.2 SEARCHING ROLLOUT GENERATION PLAN

Denote $\mathcal{D}_I$ as the GPU set allocated for rollout generation. We search for a rollout generation plan through MILP feasibility queries, denoted as MILP$(\mathcal{D}_I, P, \delta(\eta))$, as follows:

Model replica configurations. Different from model training, rollout inference replicas should independently generate rollouts. Note that such replicas could duplicate the same configuration, i.e., execute the same parallel strategy over the same type of GPUs connected by the same network bandwidth. To enumerate this process, define Z as GPU types: $\mathcal{T}_I = [i_1, i_2, ..., i_Z]$ as the count of each type, e.g., i_z represents the count of type-z GPUs. To determine the execution plan τ, we characterize rollout replica parallel strategies, replica counts, and workload assignment as follows:

Let Ψ denote possible configurations for one rollout inference replica. Each configuration $\psi \in \Psi$ represents the configuration for a rollout inference replica characterized by $(v_\psi, s_\psi, h_\psi, y_\psi)$: (**i**) Vector $v_\psi = [i_1^\psi, i_2^\psi, ..., i_Z^\psi], 0 \le i_z^\psi \le i_z, \forall z \in [1, Z]$ indicates GPU count of each type indexed by z used in configuration ψ. (**ii**) Vector $s_\psi = [\text{tp}_1^\psi, \text{tp}_2^\psi, ..., \text{tp}_{S^\psi}^\psi]$ indicates this inference serving replica is equipped with S^ψ pipeline stages, where tp_s^ψ is the TP degrees of the s-th stage ($s \in [1, S^\psi]$). (**iii**) The inference throughput h_ψ is estimated following the cost model from HexGen (Jiang et al., 2024). (**iv**) Integer variable y_ψ denotes the number of replicas using configuration ψ.

Workload assignment to rollout replicas. During rollout generation, the input prompt length is predefined by the task; in contrast, the output length of each rollout is nondeterministic during RL training. We sample output lengths from the skew distribution P profiled during cold-start to estimate output tokens, noting $\text{len}_{\text{rollout}} \sim P$[1]. The allocation problem determines the number of rollouts each replica should host in its local inference batch. Formally, let $X_{\delta(\eta)}$ denote the maximum processing capacity (in number of rollouts) within $\delta(\eta)$ training steps, and B denote the total rollouts that all replicas should process. Our algorithm determines an assignment $\mathcal{A}_I = \{x_\psi : \psi \in \Psi\}$, where $x_\psi \in [0, B]$ represents the number rollouts assigned to configuration ψ, subject to $\sum_{\psi \in \Psi} x_\psi = B$.

Optimization objective. We minimize the rollout makespan across $\delta(\eta)$ training steps. An active configuration ($y_\psi > 0$) has effective throughput $y_\psi h_\psi$ and completion time $\Theta_\psi = x_\psi \text{len}_{\text{rollout}}/(y_\psi h_\psi)$. Inactive configurations have $y_\psi = x_\psi = 0$ and are excluded from this fractional expression. The makespan is the maximum completion time over active configurations.

MILP-based solution. We binary-search a candidate makespan $\widehat{\Theta}$ and solve the following feasibility problem at each fixed candidate:

$$\begin{aligned} \text{find} \quad & \{x_\psi, y_\psi\}_{\psi \in \Psi}, \\ \text{s.t.} \quad & x_\psi \text{len}_{\text{rollout}} \le \widehat{\Theta} h_\psi y_\psi, \quad \forall \psi \in \Psi, \\ & \sum_{\psi \in \Psi} x_\psi = B, \\ & \sum_{\psi \in \Psi} i_z^\psi y_\psi \le i_z, \quad \forall z \in [1, Z], \\ & 0 \le x_\psi \le B, \quad y_\psi \in \mathbb{Z}_{\ge 0}, \quad \forall \psi \in \Psi. \end{aligned} \tag{2}$$

With fixed $\widehat{\Theta}$ and profiled h_ψ and rollout lengths, all constraints are linear; $y_\psi = 0$ forces $x_\psi = 0$. Monotone feasibility permits binary search to a chosen tolerance (Appendix H.7). The resulting τ specifies replica counts and workloads. We restrict replicas to memory-feasible configurations with intra-machine TP to accelerate search.[2] Reward cost is profiled as a constant for the evaluated tasks.

Step-window adaptation. We adaptively tune $\delta(\eta)$ when optimizing training and rollout plans; Appendix B gives the rationale and procedure.

4.3 Repartition Phase Algorithm

We seek a low-cost resource allocation with a cost-guided graph partition heuristic, denoted by Graph_Partition$(C_T, C_I, \mathcal{D})$.

Cost-guided graph partition. To refine the resource allocation of the GPU set $\mathbf{D}$ for model training ($\mathcal{D}_T$) and for rollout generation ($\mathcal{D}_I$), we design a graph partition approach guided by the costs generated in the first phase. We extend GPU set $\mathcal{D}$ to a graph $G = (\mathcal{D}, \mathcal{E})$, where each GPU $d \in \mathcal{D}$ is a vertex noted with a computation power c_d, HBM memory bandwidth m_d, and HBM limit M_d; $\forall d_1, d_2 \in \mathcal{D}$, bandwidth β_{d_1,d_2} represents an edge between GPUs in set $\mathcal{E}$. We bisect graph G into partition $\mathcal{D} = \{\mathcal{D}_T, \mathcal{D}_I\}$, where $\mathcal{D}_T \cup \mathcal{D}_I = \mathcal{D}$ and $\mathcal{D}_T \cap \mathcal{D}_I = \emptyset$. We partition the graph by:

[1] Appendix E provides more analysis on cases where output length distribution estimation is inaccurate.

[2] Appendix A evaluates allowing inter-machine TP.

- *Allocating high network bandwidth for model training.* We allocate GPUs with high aggregate interconnects to communication-intensive training. We define the metric $\frac{\sum_{d_1,d_2\in\mathcal{D}_T}\beta_{d_1,d_2}}{\sum_{d_1',d_2'\in\mathcal{D}}\beta_{d_1',d_2'}}$, i.e. the aggregated network bandwidth of $\mathcal{D}_T$ over the aggregated bandwidth of $\mathcal{D}$, to guide the partition.
- *Allocating high HBM bandwidth GPUs for rollout generation.* We allocate GPUs with high HBM bandwidth to HBM I/O-bound rollout generation. Formally, we define the metric $\frac{\sum_{d\in\mathcal{D}_I} m_d}{\sum_{d'\in\mathcal{D}} m_{d'}}$, i.e. the aggregated HBM bandwidth of $\mathcal{D}_I$ over the aggregated HBM bandwidth of $\mathcal{D}$.
- *Allocating partial computation power for model training.* To maintain an adaptive balance, we let the fraction of computation capacity occupied by model training, i.e., $\frac{\sum_{d\in\mathcal{D}_T} c_d}{\sum_{d'\in\mathcal{D}} c_{d'}}$, to fall into a tunable range $[\gamma_L, \gamma_H]$, where γ_L and γ_H will be tuned automatically as we will enumerate next.

By integrating the above criteria, we define the final constrained optimization as:

$$\begin{aligned} &\mathcal{D}_T^*, \mathcal{D}_I^* = \arg\max_{\mathcal{D}_T,\mathcal{D}_I} \left(\frac{\sum_{d_1,d_2\in\mathcal{D}_T}\beta_{d_1,d_2}}{\sum_{d_1',d_2'\in\mathcal{D}}\beta_{d_1',d_2'}} + \frac{\sum_{d\in\mathcal{D}_I} m_d}{\sum_{d'\in\mathcal{D}} m_{d'}} \right), \\ &\text{s.t.} \quad \gamma_L \le \frac{\sum_{d\in\mathcal{D}_T} c_d}{\sum_{d'\in\mathcal{D}} c_{d'}} \le \gamma_H. \end{aligned} \tag{3}$$

Iterative refinement. Across graph partition iterations, to balance model training and rollout generation costs, we dynamically adjust the fraction of total computational power allocated to model training (i.e., γ_L and γ_H) based on execution costs generated in the search phase algorithm (i.e., C_T and C_I). Detailed tuning methods are listed in Appendix C.

5 Evaluation

5.1 Experimental Setup

Hardware. We use (**i**) NVIDIA H20 GPUs, which provide 148 TFLOPS computational power, 4 TB/s memory bandwidth, and 450 GB/s intra-machine bandwidth; (**ii**) NVIDIA H800 GPUs, which deliver 756 TFLOPS computational power, 2 TB/s memory bandwidth, and 200 GB/s intra-machine bandwidth; (**iii**) NVIDIA A100 GPUs, which deliver 312 TFLOPS computational power, 2 TB/s memory bandwidth, and 300 GB/s intra-machine bandwidth. The bandwidth connecting H20 and H800 machines is 1.5 GB/s, and the bandwidth connecting A100 and H800 GPUs is 200 GB/s.

Baselines. We compare AReaL-Hex on heterogeneous clusters against AReaL and VeRL on homogeneous clusters under equivalent budgets. For each homogeneous AReaL baseline, we evaluate candidate configurations manually and report the highest measured throughput.

5.2 End-to-end System Performance Comparison

End-to-end experiments: For the GRPO algorithm, we use the AReaL-Boba dataset and benchmark the end-to-end asynchronous RL training throughput of AReaL-Hex on heterogeneous clusters against AReaL on homogeneous clusters under the same total budget across the following settings:

H800 and H20 clusters: On the 1.5B to 14B Qwen models (DeepSeek-AI, 2024a;c;b), we evaluate AReaL-Hex running over heterogeneous clusters (i.e., H800 + H20 GPUs) against AReaL running over either homogeneous H800 GPUs or H20 GPUs. As shown in Figure 3, AReaL-Hex consistently outperforms AReaL. Compared to the H800 homogeneous baseline, AReaL-Hex achieves up to 1.50×, and 1.39× speedup on average. Compared to the H20 homogeneous baseline, AReaL-Hex achieves up to 2.76×, and 2.62× speedup on average.

Discussion: Homogeneous configurations suffer from inherent imbalance: H800 clusters underutilize resources in memory-bound rollout generation, while H20/A100 clusters are less efficient for compute-intensive model training. Our scheduler addresses this by assigning memory-bandwidth-efficient GPUs (e.g., H20/A100) to rollout generation and compute-efficient GPUs (e.g., H800) to

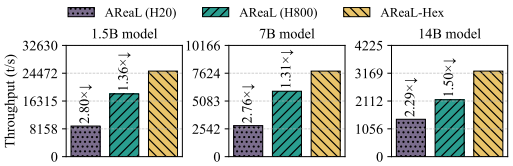


Figure 3: End-to-end comparisons between AReaL-Hex on heterogeneous GPUs (H800+H20 GPUs) and AReaL on homogeneous H800/H20 GPUs. All settings have the same total budgets.

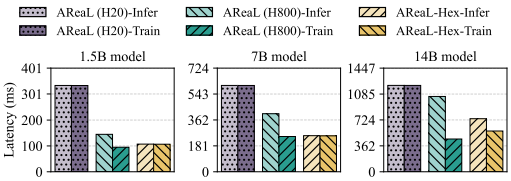


Figure 4: Latency breakdown of AReaL-Hex on a 56-GPU heterogeneous cluster versus AReaL on a 24-GPU H800 homogeneous cluster. Infer and Train denote generation and training latencies.

Table 2: Weight update time comparison.

Model	AReaL (H800)	AReaL (H20)	AReaL-Hex
1.5B	4.75 s	2.74 s	10.06 s
7B	14.79 s	7.46 s	58.34 s
14B	26.00 s	13.05 s	112.93 s

model training. When homogeneous resources are insufficient to balance pipeline stages, heterogeneous GPUs are further allocated to the bottleneck stage, enabling balanced execution.

5.3 System Performance Breakdown

Performance breakdown results. As shown in Figure 4, AReaL-Hex reduces rollout generation latency by up to 1.61×, and 1.46× on average, over homogeneous H800 clusters, and reduces model training latency by up to 3.13×, and 2.46× on average, over homogeneous H20 clusters. As reported in Table 2, weight synchronization accounts for only 1.9% of iteration time under H800 homogeneous settings, but rises to 13.54% under heterogeneous settings.

Discussion: The breakdown confirms that rollout generation is the bottleneck for H800 clusters, while model training is the bottleneck for H20 clusters. AReaL-Hex's scheduler resolves both bottlenecks by assigning the two stages across different GPU types. Weight synchronization is manageable in these settings despite the 1.5 GB/s inter-machine link. Appendix H.4 reports its fraction across four staleness bounds and models up to 235B; the observed range is 6.73%–16.40%.

Table 3: Comparison between AReaL-Hex and AReaL-Hex with uniform resource allocation, i.e., AReaL-Hex (u).

Model	AReaL-Hex (u)	AReaL-Hex
1.5B model	1.17×10^4 t/s	1.84×10^4 t/s (1.57×↑)
7B model	3.57×10^3 t/s	5.98×10^3 t/s (1.68×↑)
14B model	1.29×10^3 t/s	2.16×10^3 t/s (1.67×↑)

5.4 System Ablation & Case Studies

Ablation on scheduling algorithm. In Table 3, we present the performance of AReaL-Hex under different resource allocation strategies. Results demonstrate that our resource allocation optimization (detailed in §4.3) delivers substantial and consistent improvements in system performance for 1.5B, 7B, and 14B models. Specifically, our optimization achieves a maximum speedup of 1.68×, and an average speedup of 1.63× compared to configurations where this scheduling optimization is disabled, highlighting the necessity of careful resource allocation.

Scheduling algorithm comparison. Table 4 compares our scheduling algorithm's execution speed[3] against baselines where each phase is replaced with exhaustive search. Our algorithm achieves 20.0× to 44.2× speedup in search time over the baselines. Although exhaustive baselines can achieve similar throughput as our scheduler, their overhead makes them impractical for complex heterogeneous settings.

Controlled heterogeneous comparisons. On the same pool of 48 H20 and 32 H800 GPUs, AReaL-Hex improves 14B PPO throughput by 1.13× over static H20-rollout/H800-training specialization

[3]We present the parallelized algorithm running overhead in Appendix D.

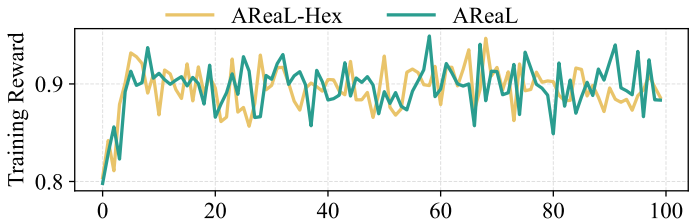


Figure 5: Reward comparison between AReaL-Hex on heterogeneous GPUs and AReaL on homogeneous GPUs for the GRPO algorithm and 1.5B model.

Table 4: Scheduling convergence time across cluster sizes. "Ours (w/o Search)" and "Ours (w/o Repartition)" replace the search phase (§4.2) and repartition phase (§4.3) with exhaustive search, respectively. Convergence criterion: the solution remains stable for 20 consecutive iterations.

Cluster Size	Ours	Ours (w/o Search)	Ours (w/o Repartition)
24 GPUs	14.9s	6.1min (24.6×↑)	5.0min (20.1×↑)
32 GPUs	23.1s	11.3min (29.4×↑)	7.7min (20.0×↑)
40 GPUs	46.7s	34.4min (44.2×↑)	16.4min (21.1×↑)
56 GPUs	2.0min	$\geq$ 40min ($\geq$ 20.0×↑)	$\geq$ 40min ($\geq$ 20.0×↑)

by reallocating 16 H800 GPUs to rollout. Table 5 further isolates scheduling choices on one simulated 56-GPU pool with five GPU types. The naive policy splits each type equally; the greedy policy prioritizes memory bandwidth per dollar for rollout; the hand-tuned policy tests 30 whole-type assignments. AReaL-Hex improves mean throughput by 1.76×, 2.10×, and 1.19×, respectively. Appendix H.1 gives the GPU counts and a stronger count-level baseline.

Table 5: Simulated throughput on the same five-type GPU pool (tokens/s).

Policy	1.5B	7B	14B
Naive	7.99×10^3	2.49×10^3	1.03×10^3
HBM/FLOPS greedy	6.68×10^3	2.08×10^3	0.87×10^3
Type-level hand-tuned	11.82×10^3	3.68×10^3	1.53×10^3
AReaL-Hex	13.88×10^3	4.33×10^3	1.85×10^3

Heterogeneous RL systems. For 1.5B GRPO on the same heterogeneous hardware, AReaL-Hex achieves 1.43× HetRL's and 1.57× StreamRL's throughput. These full-system comparisons include different asynchronous execution policies (Appendix H.2).

Reward and staleness. Figure 5 compares reward curves for 1.5B GRPO. Across three seeds, increasing η from 2 to 4 improves throughput by 10.4% but reduces mean final reward from 0.917 to 0.880; time-to-target results also depend on the target reward (Appendix H.6).

Additional evaluations. Due to space limitations, Appendix F reports additional evaluations covering H800/A100 and three-type GPU clusters (§F.1), a 235B MoE model (§F.2), PPO (§F.3), cost efficiency and cluster scaling (§F.4), synchronous RL (§F.5), and VeRL comparisons (§F.6). Appendix H details the baseline setups, search quality and overhead, synchronization and price sensitivity, and absolute hardware scaling; Appendix E covers response-length drift.

6 Conclusion

We presented AReaL-Hex, a system that accommodates asynchronous RL training on heterogeneous GPUs. We designed a two-phase scheduling algorithm, which is capable of determining the resource allocation and parallel strategy for RL computation stages. Under the same costs, experiments validate that AReaL-Hex on heterogeneous GPUs achieves superior performance than AReaL on homogeneous GPUs across 1.5B to 14B models.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript editing, migration to the ICLR 2027 conference template, formatting checks, and integrating previously reported author-response results and clarifications into the manuscript. This assistance included summarizing experiments, checking arithmetic consistency, and drafting explanatory text.

REFERENCES

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

DeepSeek-AI. Deepseek-r1-distill-qwen-1.5b, 2024a. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B.

DeepSeek-AI. Deepseek-r1-distill-qwen-14b, 2024b. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-14B.

DeepSeek-AI. Deepseek-r1-distill-qwen-7b, 2024c. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

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.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl. *arXiv preprint arXiv:2508.07976*, 2025.

Yongjun He, Shuai Zhang, Jiading Gai, Xiyuan Zhang, Boran Han, Bernie Wang, Huzefa Rangwala, and George Karypis. HetRL: Efficient reinforcement learning for LLMs in heterogeneous environments. In *Proceedings of the 9th Conference on Machine Learning and Systems*, 2026. URL https://arxiv.org/abs/2512.12476.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Youhe Jiang, Ran Yan, Xiaozhe Yao, Yang Zhou, Beidi Chen, and Binhang Yuan. Hexgen: Generative inference of large language model over heterogeneous environment. In *International Conference on Machine Learning*, pp. 21946–21961. PMLR, 2024.

Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Guoliang He, Xupeng Miao, Ana Klimovic, Bin Cui, Binhang Yuan, and Eiko Yoneki. Demystifying cost-efficiency in llm serving over heterogeneous gpus. *arXiv preprint arXiv:2502.00722*, 2025a.

Youhe Jiang, Ran Yan, and Binhang Yuan. Hexgen-2: Disaggregated generative inference of llms in heterogeneous environment. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Ziming Mao, Tian Xia, Zhanghao Wu, Wei-Lin Chiang, Tyler Griggs, Romil Bhardwaj, Zongheng Yang, Scott Shenker, and Ion Stoica. Skyserve: Serving ai models across regions and clouds with spot instances. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 159–175, 2025.

Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 586–602, 2025.

Zhiyu Mei, Wei Fu, Jiaxuan Gao, Guangju Wang, Huanchen Zhang, and Yi Wu. Srl: Scaling distributed reinforcement learning to over ten thousand cores. *arXiv preprint arXiv:2306.16688*, 2023.

Zhiyu Mei, Wei Fu, Kaiwei Li, Guangju Wang, Huanchen Zhang, and Yi Wu. Real: Efficient rlhf training of large language models with parameter reallocation. *arXiv preprint arXiv:2406.14088*, 2024.

Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, and Zhihao Jia. Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. *Proceedings of the VLDB Endowment*, 16(9):2354–2363, 2023.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.

Zizhao Mo, Jianxiong Liao, Huanle Xu, Zhi Zhou, and Chengzhong Xu. Hetis: Serving llms in heterogeneous gpu clusters with fine-grained and dynamic parallelism. *arXiv preprint arXiv:2509.08309*, 2025.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6): 47–60, 1996.

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. In *The Thirteenth International Conference on Learning Representations*, 2025.

OpenAI. Learning to reason with llms. OpenAI Blog, 09 2024. URL `https://openai.com/index/learning-to-reason-with-llms/`.

OpenAI. Introducing openai o3 and o4-mini. OpenAI Blog, 04 2025. URL `https://openai.com/index/introducing-o3-and-o4-mini/`.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. Metis: Fast automatic distributed training on heterogeneous {GPUs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 563–578, 2024.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.

Junyu Wu, Weiming Chang, Xiaotao Liu, Guanyou He, Haoqiang Hong, Boqi Liu, Hongtao Tian, Tao Yang, Yunsheng Shi, Feng Lin, et al. G-core: A simple, scalable and balanced rlhf trainer. *arXiv preprint arXiv:2507.22789*, 2025.

Ran Yan, Youhe Jiang, Xiaonan Nie, Fangcheng Fu, Bin Cui, and Binhang Yuan. Hexiscale: Accommodating large language model training over heterogeneous environment. *arXiv preprint arXiv:2409.01143*, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 36:25464–25477, 2022.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025.

Yilong Zhao, Shuo Yang, Kan Zhu, Lianmin Zheng, Baris Kasikci, Yang Zhou, Jiarong Xing, and Ion Stoica. Blendserve: Optimizing offline inference for auto-regressive large models with resource-aware batching. *arXiv preprint arXiv:2411.16102*, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

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*, 2025a.

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, et al. Optimizing {RLHF} training for large language models with stage fusion. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 489–503, 2025b.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. Megascale-infer: Serving mixture-of-experts at scale with disaggregated expert parallelism. *arXiv preprint arXiv:2504.02263*, 2025.

A Evaluation on Scheduling Algorithm

In §4.3, we restrict inference replicas to use intra-machine TP. However, when inter-machine bandwidth is sufficiently high, inter-machine TP can also be a viable option. To study this, we conduct additional experiments to evaluate the scheduling overhead *with* and *without* inter-machine TP. The results are summarized in Table 6.

Table 6: Scheduling overhead with and without inter-machine tensor model parallelism.

Cluster Size	Without Inter-machine TP	With Inter-machine TP
24 GPUs	14.9 s	15.6 s
32 GPUs	23.1 s	26.3 s
40 GPUs	46.7 s	52.4 s
56 GPUs	120.0 s	144.0 s

Discussion. The results show that scheduling *with* inter-machine TP introduces, on average, a $1.1\times$ increase in scheduling overhead compared to scheduling *without* inter-machine TP, shown in Table 4. Notably, the maximum TP degree of a given LLM is bounded by the number of key and value heads, as modern architectures implement TP by partitioning attention heads across the TP group. Consequently, even when accounting for inter-machine TP, the additional overhead introduced by our scheduler remains manageable.

B Adjust Step Window $\delta(\eta)$

Optimize across different $\delta(\eta)$ values. Across algorithm iterations, we iteratively optimize parallel strategy and workload assignment for model training and rollout generation under varying step window $\delta(\eta)$. The choice of $\delta(\eta)$ involves a fundamental tradeoff: optimizing over the entire training horizon ($\delta = \infty$) is computationally infeasible and requires an unknown total training iteration, while optimizing over too few steps ($\delta \approx 1$) fails to accurately capture asynchronous RL training dynamics. To this end, we adopt the following adaptive strategy to tune $\delta(\eta)$.

In §4.2, we fine-tune the $\delta(\eta)$ as follows: We initialize the step window by the staleness bound η, i.e., $\delta(\eta) = \eta$. We then iteratively increase $\delta(\eta)$ in increments of η, resulting in $\delta(\eta) = 2\eta, \delta(\eta) = 3\eta, \ldots$, to progressively explore larger step windows. The ending point is determined based on a latency stability criterion. Specifically, when the scheduling algorithm outputs an RL training latency under step window $\delta(\eta)$ that varies less than $\alpha\%$ (a hyper-parameter) from the previous scheduling results, we stop increasing the step window $\delta(\eta)$. This approach leverages the key insight that when $\delta(\eta)$ accurately captures the asynchronous training dynamics and data staleness characteristics, scheduling decisions remain stable under larger values of $\delta(\eta)$, indicating diminishing returns from further increases of $\delta(\eta)$.

C Adjust TFLOPS Across RL Stages

In §4.3, we fine-tune the computational power assignment with the following method: We initialize $\gamma_L = \gamma_H = 1$, assigning all FLOPS to model training. In subsequent iterations, the optimal resource allocation is determined by tuning γ_L and γ_H using a binary search procedure. Specifically, we define the search range as $[q, r]$, where $q = 0$ and $r = 1$ initially. At each iteration, we first update the FLOPS allocated to model training by updating $\gamma_L = \gamma_H = (q + r)/2$, and then evaluate the execution costs C_T and C_I: If $C_T < C_I$, we set the upper bound $r = (q + r)/2$; otherwise, if $C_T \geq C_I$, we update the lower bound $q = (q + r)/2$. This process continues until $C_T \approx C_I$, at which point the final resource allocation is selected to minimize the overall execution cost.

D Scheduling Overhead under Parallel Execution

In §5.4, we present the detailed evaluations of the non-parallel execution time of our scheduling algorithm. However, using parallel execution, the scheduling time of our algorithm can be further optimized. Table 7 presents a comparison of non-parallel and parallel versions of our algorithm on large clusters. Experimental results demonstrate that, with parallel execution, our algorithm scheduling overhead can be significantly reduced for large clusters.

Table 7: Scheduling overhead for non-parallel and parallel execution.

Cluster Size	Non-Parallel Execution	Parallel Execution (160 CPU Cores)
56 GPUs	2.0 min	0.3 min
128 GPUs	7.1 min	1.2 min
256 GPUs	16.2 min	2.7 min

E Analysis on Distribution Length Estimation

In §4.2.2, we estimate the number of output tokens for rollout responses via profiling during the cold-start phase. As RL training progresses, workload characteristics may shift, and bias in the estimated output distribution can degrade system performance. To quantify the impact of such shifts, we conduct additional experiments using the 1.5B model on the H20+H800 cluster, consistent with the setup in Figure 3 (left) of our paper. We compare system throughput under two strategies:

- **Initial scheduling decision:** placement based on the sampled distribution.
- **Re-optimized placement:** placement based on the latest output distribution, which deviates from the initial sampled distribution.

Table 8 shows that the initial schedule retains 88.6% of the throughput obtained after re-optimizing for the shifted distribution (2.42×10^4 versus 2.73×10^4 tokens/s). Updating the schedule recovers a $1.13\times$ performance gap in this controlled 1.5B setting. Figure 3 provides the corresponding baseline comparisons.

Table 8: Impact of output distribution shift on system throughput.

Strategy	Throughput
Initial scheduling decision	2.42×10^4 t/s
Re-optimized placement	2.73×10^4 t/s

Observed response lengths. Table 9 reports 105,926 trajectories from GRPO training on the AReaL-Boba dataset, grouped into 2k-token intervals. The counts describe the profiled workload used to characterize response-length variation.

Table 9: Observed response-length distribution. Counts are trajectories; k denotes thousands of tokens.

Length range	Count	Length range	Count
0k–2k	2,187	12k–14k	13,196
2k–4k	8,723	14k–16k	8,819
4k–6k	12,392	16k–18k	4,272
6k–8k	17,519	18k–20k	1,448
8k–10k	19,871	20k–22k	451
10k–12k	16,927	22k–24k	121

Online re-optimization. A background thread monitors trajectories after each RL iteration. Let P be the average output-length distribution over the preceding ω iterations and Q the current iteration's distribution over the same length bins. The monitor evaluates

$$D_{\mathrm{KL}}(Q\|P) = \sum_i Q(i) \log \frac{Q(i)}{P(i)}. \tag{4}$$

When the divergence exceeds a configurable threshold ρ, it triggers re-optimization and relaunches training with the updated schedule. The monitoring window ω and threshold ρ are configurable. Table 8 quantifies the benefit of refreshing the schedule after one controlled distribution shift; it does not separately measure monitoring and relaunch overhead under arbitrary drift.

F Additional Evaluation Results

F.1 Additional Heterogeneous Clusters

H800 and A100 clusters: On 1.5B and 4B Qwen models, we evaluate AReaL-Hex running over heterogeneous clusters (i.e., H800 + A100 GPUs) against AReaL running over either homogeneous H800 GPUs or A100 GPUs. As shown in Table 10, AReaL-Hex achieves up to 1.40× and 1.27× speedup on average over AReaL running on homogeneous H800 or A100 clusters.

H800, H20, and A100 clusters: On 1.5B and 4B Qwen models, we evaluate AReaL-Hex running over heterogeneous clusters (i.e., H800 + A100 + H20 GPUs) against AReaL running over homogeneous H800/H20/A100 GPUs. As shown in Table 10, compared to the H800 homogeneous baseline, AReaL-Hex achieves up to 1.73×, and 1.60× speedup on average. Compared to the H20 homogeneous baseline, AReaL-Hex achieves up to 1.13×, and 1.11× speedup on average.

Table 10: Performance comparison on H800/A100/H20 heterogeneous clusters.

Model	GPU Configuration	System	Throughput
1.5B	16×A100	AReaL	1.84×10^3 t/s
	8×H800	AReaL	1.75×10^3 t/s
	24×H20	AReaL	2.34×10^3 t/s
	8×A100 + 4×H800	AReaL-Hex	2.45×10^3 t/s
	4×A100 + 6×H20 + 4×H800	AReaL-Hex	2.58×10^3 t/s
4B	16×A100	AReaL	7.47×10^2 t/s
	8×H800	AReaL	7.29×10^2 t/s
	24×H20	AReaL	1.12×10^3 t/s
	8×A100 + 4×H800	AReaL-Hex	8.67×10^2 t/s
	4×A100 + 6×H20 + 4×H800	AReaL-Hex	1.26×10^3 t/s

F.2 Large-Model Evaluation

End-to-end experiments on large models. We conduct additional experiments on a 235B MoE model to compare system throughput between AReaL-Hex using 24×H20 + 32×H800 heterogeneous GPUs and AReaL using 40×H800 homogeneous GPUs. The results are summarized in Table 11. Compared to AReaL, AReaL-Hex achieves a 1.24× speedup in RL training throughput.

Table 11: Throughput comparison on a 235B model.

GPU Configuration	System	Throughput
24×H20 + 32×H800	AReaL-Hex	2.13×10^3 t/s
40×H800	AReaL	1.72×10^3 t/s

F.3 PPO Evaluation

Evaluation on the PPO algorithm. We conduct experiments for another RL training algorithm, i.e., PPO. Under the same total budgets, we compare AReaL-Hex on heterogeneous GPUs with AReaL on homogeneous GPUs across 1.5B/7B/14B models. Experimental results are shown in Table 12. Compared to AReaL on homogeneous GPUs, AReaL-Hex on heterogeneous GPUs achieves up to a 3.08× speedup and an average speedup of 2.58×. For 1.5B and 7B models, performance gains of AReaL-Hex stem from the cost efficiency of H800 GPUs. For the 14B model, AReaL-Hex's gains stem from its scheduler assigning both H800 and H20 GPUs to rollout generation.

Table 12: Training throughput comparison between AReaL-Hex and AReaL under PPO. Parallel strategies follow the format dxpytz (x=DP-, y=PP-, z=TP-degree). Data-parallel refers to replication for rollout workers.

Model	AReaL-Hex Strategy	AReaL-Hex Throughput	AReaL Strategy	AReaL Throughput
1.5B	d24p1t1 (H20) + d24p1t1 (H800)	8.97×10^4 t/s	d24p1t1 + d8p1t1	3.26×10^4 t/s
7B	d24p1t1 (H20) + d24p1t1 (H800)	4.07×10^4 t/s	d16p1t1 + d16p1t1	1.32×10^4 t/s
14B	d8p1t8 (H800, H20) + d16p1t1 (H800)	1.57×10^4 t/s	d4p1t8 + d16p1t1	8.20×10^3 t/s

F.4 COST COMPARISON

Case study on the cost efficiency of AREAL-HEX. In Figure 6, we present the per-dollar end-to-end system performance of AREAL-HEX under varied cluster sizes. Results demonstrate that AREAL-HEX maintains stable per-dollar throughput across heterogeneous GPU clusters of 24 to 56 GPUs. Concretely, the per-dollar throughput for the 1.5B, 7B, and 14B models is around 200, 62, and 24 tokens/s, across 24–56 heterogeneous GPU configurations. These results validate that heterogeneous RL training delivers stable performance regardless of cluster size.

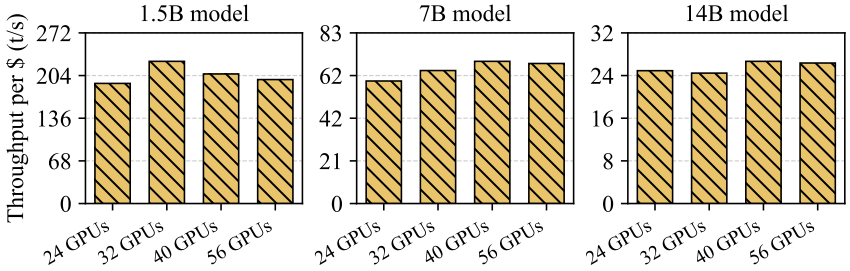


Figure 6: Per-dollar throughput (cost-efficiency) of AREAL-HEX across heterogeneous cluster sizes ranging from 24 to 56 GPUs.

Case study on the cost efficiency comparison. Table 13 compares the costs of AREAL-HEX and AREAL for 1.5B/7B/14B models. The heterogeneous setting demonstrates 1.42×, 1.31×, and 1.50× lower per-hour training cost than the H800 baseline, and 50% cost reduction than the H20 baseline. These results confirm that heterogeneous RL training is an economical choice.

Table 13: Cost comparison between AREAL-HEX on 32 hetero-GPUs and AREAL on 24 H800 GPUs.

Model	System	Throughput	Cost
1.5B model	AREAL (H800)	1.84×10^4 t/s	$126.72/h
	AREAL-HEX	1.79×10^4 t/s	$86.64/h
7B model	AREAL (H800)	5.98×10^3 t/s	$126.72/h
	AREAL-HEX	6.00×10^3 t/s	$86.64/h
14B model	AREAL (H800)	2.16×10^3 t/s	$126.72/h
	AREAL-HEX	2.30×10^3 t/s	$86.64/h

F.5 SYNCHRONOUS RL TRAINING

Evaluation on synchronous RL training scenarios. For the GRPO algorithm, under the same total compute budget, we compare the synchronous RL training throughput of AREAL-HEX using mixed H20/H800 GPUs against AREAL using 24 H800 GPUs. As shown in Table 14, AREAL-HEX achieves up to 1.34× and, on average, 1.18× higher throughput than AREAL.

Table 14: Synchronous RL training throughput comparison under the GRPO algorithm.

Model	Configuration	Throughput
1.5B	H800, AREAL	1.45×10^4 t/s
1.5B	H20 + H800, AREAL-HEX	1.95×10^4 t/s
7B	H800, AREAL	6.00×10^3 t/s
7B	H20 + H800, AREAL-HEX	6.50×10^3 t/s
14B	H800, AREAL	2.50×10^3 t/s
14B	H20 + H800, AREAL-HEX	2.80×10^3 t/s

F.6 COMPARISON WITH VERL

Comparison with VeRL. Using the settings in Figure 3, we compare AREAL-HEX over heterogeneous GPUs against VeRL (Sheng et al., 2025) over homogeneous GPUs. As shown in Table 15, AREAL-HEX achieves up to 1.70× and 1.43× average higher throughput.

Table 15: Throughput comparison between AReaL-Hex and VeRL.

Model	AReaL-Hex	VeRL
1.5B	2.52×10^4 t/s	1.48×10^4 t/s
7B	7.82×10^3 t/s	5.80×10^3 t/s
14B	3.25×10^3 t/s	2.60×10^3 t/s

G Scheduling Pseudocode

Algorithm 1 Scheduling Algorithm

Input: $(\mathcal{D}_T^0, \mathcal{D}_I^0)$: initial allocation, $\mathcal{D}$: GPU device info, K: stable iteration threshold, $\delta(\eta)$: window size
Output: optimized σ^*, τ^*, $\mathcal{D}_T^*$, $\mathcal{D}_I^*$ /* σ: model training plan, τ: rollout generation plan */
Intermediate Variables: C_T: Model training cost, C_I: Rollout generation cost
$\mathcal{D}_T, \mathcal{D}_I \leftarrow \mathcal{D}_T^0, \mathcal{D}_I^0$ /* Initialize training and inference resource allocation. */
while convergence not reached **do**
  $(\sigma, C_T) \leftarrow$ Constrained_Search$(\mathcal{D}_T)$ /* Search-Phase: Construct execution plans (§4.2) */
  $(\tau, C_I) \leftarrow$ MILP$(\mathcal{D}_I, P, \delta(\eta))$
  $(\mathcal{D}_T, \mathcal{D}_I) \leftarrow$ Graph_Partition$(C_T, C_I, \mathcal{D})$ /* Repartition-Phase: Update resource allocation (§4.3) */
  if $\max\{C_T, C_I\}$ is stable for K iterations **then**
    break
  end if
end while

H Additional Validation and Method Details

H.1 Scheduler Comparisons on the Same GPU Pool

Static stage specialization. For 14B PPO, we compare schedules on the same pool of 48 H20 and 32 H800 GPUs. A manually specialized policy assigns all H20 GPUs to rollout and all H800 GPUs to training, leaving rollout as the bottleneck. AReaL-Hex instead assigns 48 H20 and 16 H800 GPUs to rollout and the remaining 16 H800 GPUs to training. This cross-stage reallocation improves throughput by 1.13× over static specialization, showing a benefit beyond assigning one GPU type to each stage.

Five-type simulation. We simulate a 56-GPU pool containing 12 RTX 4090, 8 RTX 3090, 8 H20, 12 H100, and 16 A800 GPUs. Under the pricing assumptions of this comparison, its total cost equals that of 24 H800 GPUs. All policies in Table 5 use this identical pool. The naive policy divides each GPU type equally between rollout and training. The HBM/FLOPS-greedy policy assigns the three types with the highest memory-bandwidth-per-dollar ratios to rollout and the remaining types to training. The type-level hand-tuned policy evaluates the 30 nonempty assignments that place each GPU type entirely in one stage. These are simulation results; the corresponding five-type pool has not been validated on real hardware.

Across the three models, AReaL-Hex achieves mean speedups of 1.76×, 2.10×, and 1.19× over the naive, greedy, and type-level hand-tuned policies, respectively. Table 16 shows the selected allocations. A stronger count-level hand-tuned baseline allows any feasible count of each GPU type in either stage and selects the allocation with the highest estimated throughput. Against this baseline on the same simulated pool, AReaL-Hex achieves 1.14×, 1.15×, and 1.17× throughput for 1.5B, 7B, and 14B, respectively.

Table 16: AReaL-Hex allocations in the five-type simulation. Vector entries follow the order RTX 4090, RTX 3090, H20, H100, A800. The naive policy uses $[6, 4, 4, 6, 8]$ for each stage for all models.

Model	Rollout allocation	Training allocation
1.5B	$[9, 8, 8, 8, 16]$	$[3, 0, 0, 4, 0]$
7B	$[10, 7, 8, 8, 16]$	$[2, 1, 0, 4, 0]$
14B	$[9, 8, 8, 9, 16]$	$[3, 0, 0, 3, 0]$

H.2 Comparison with Heterogeneous RL Systems

StreamRL (Zhong et al., 2025a) overlaps generation and training with one-step asynchronous streaming. HetRL (He et al., 2026) jointly optimizes RL task placement and parallelism over heterogeneous GPUs and networks using hybrid and ILP-based schedulers; its scheduling formulation does not expose the configurable rollout-age bound used here. AReaL-Hex makes the user-specified staleness bound η part of its joint resource-allocation and execution-plan search.

For 1.5B GRPO under the same heterogeneous hardware configuration, AReaL-Hex achieves 1.43× the throughput of HetRL and 1.57× that of StreamRL. These are full-system throughput comparisons, including their different asynchronous execution policies. The comparisons within a fixed GPU pool in Appendix H.1 separately assess allocation policies. On a 24-GPU cluster, AReaL-Hex takes 14.9 s to obtain its schedule, compared with approximately 1.3 min for HetRL's ILP solver, a 5.2× search-time speedup. This timing comparison concerns HetRL's ILP variant.

H.3 Search Quality and Scheduling Overhead

The graph-partition objective in §4.3 is a proxy for the end-to-end objective and does not guarantee a global optimum. Table 17 compares the resulting solution quality with exhaustive search on two 56-GPU configurations. The scheduler matches exhaustive search on the two-type setting and achieves 99.74% normalized quality in the five-type simulation. Search time increases from 2.0 to 4.34 min as the number of GPU types increases from two to five. These results establish empirical solution quality for the evaluated settings.

Table 17: Search cost and normalized solution quality on 56 GPUs; exhaustive search is normalized to 100%. The five-type result is simulated.

GPU pool	Search time (min)	Solution quality
H20 + H800	2.00	100.00%
RTX 4090/3090 + H20 + H100 + A800	4.34	99.74%

Table 18: Scheduling time as a fraction of end-to-end training time.

Cluster size	24 GPUs	32 GPUs	40 GPUs	56 GPUs
Scheduling fraction	0.16%	0.32%	0.77%	2.19%

Table 18 puts search overhead in the context of a training run. The mean fraction across these four settings is 0.86%. On 56 GPUs, the 2 min scheduling cost is included in approximately 91.2 min of total training time. Replacing it with at least 40 min of exhaustive search increases total time to at least 129.2 min and the scheduling fraction to at least 30.96%. Thus, a small fraction for the proposed scheduler does not imply that exhaustive search is negligible.

H.4 Weight Synchronization across Model Scales and Staleness Bounds

Table 19 reports the percentage of total training time spent synchronizing weights for dense models and a 235B MoE model, across $\eta \in \{0, 2, 4, 8\}$. The fractions range from 6.73% to 16.40%; their arithmetic mean across the 16 settings is 12.25%. The fraction rises substantially from 1.5B to 7B for several settings, but does not increase monotonically with model size. These measurements bound the observed overhead in the tested configurations rather than establish a scaling law.

Table 19: Weight-update time as a percentage of total training time.

Model	$\eta = 0$	$\eta = 2$	$\eta = 4$	$\eta = 8$
1.5B dense	7.24%	6.73%	9.40%	10.58%
7B dense	10.42%	12.41%	15.98%	16.01%
14B dense	9.04%	11.08%	15.24%	16.40%
235B MoE	12.18%	13.31%	14.44%	15.52%

H.5 Rental-Price Sensitivity and Absolute Hardware Scaling

Rental-price sensitivity. Table 20 varies the H800/H20 hourly rental-price ratio while maintaining the same total cost as the baseline AReaL deployment on 24 H800 GPUs. Gains persist for all three tested price ratios, but shrink as H20 becomes relatively more expensive. Thus, the magnitude of the cost-efficiency advantage depends on the available prices.

Table 20: Equal-cost throughput speedup over AReaL on 24 H800 GPUs. The first column gives the hourly price ratio $p_{\mathrm{H800}}/p_{\mathrm{H20}}$.

Price ratio	AReaL-Hex GPU pool	1.5B	7B	14B
3	48 H20 + 8 H800	1.36×	1.31×	1.50×
2	32 H20 + 8 H800	1.25×	1.10×	1.14×
1.5	24 H20 + 8 H800	1.03×	1.02×	1.06×

Hardware scaling. Table 21 complements the per-dollar results in Figure 6 with absolute throughput. From 24 to 56 GPUs, throughput increases by 1.77× for 1.5B and 1.60× for 7B. The 14B workload runs out of memory on 24 GPUs and improves by 1.41× from 32 to 56 GPUs. These measurements demonstrate increasing throughput over the evaluated range, without assuming linear scaling or extrapolating beyond 56 GPUs.

Table 21: Absolute throughput as the heterogeneous cluster grows (tokens/s). OOM denotes out of memory.

Total GPUs	1.5B	7B	14B
24	1.42×10^4	4.90×10^3	OOM
32	1.79×10^4	6.00×10^3	2.30×10^3
40	2.30×10^4	6.57×10^3	2.47×10^3
56	2.51×10^4	7.82×10^3	3.25×10^3

H.6 Staleness, Final Reward, and Time to Target Quality

For the 1.5B model, we evaluate three independent random seeds for each of $\eta = 2$ and $\eta = 4$. Table 22 reports every trial. Increasing η raises mean throughput from 1.667×10^4 to 1.840×10^4 tokens/s, a 10.4% improvement, while reducing mean final reward from 0.917 to 0.880. A larger staleness bound therefore trades training quality for systems throughput in these runs.

Table 22: Throughput (in 10^4 tokens/s) and final reward for three independent seeds per staleness setting. Trial indices identify runs within each setting.

	$\eta = 2$		$\eta = 4$	
Trial	Throughput	Reward	Throughput	Reward
1	1.67	0.91	1.84	0.89
2	1.65	0.92	1.82	0.87
3	1.68	0.92	1.86	0.88
Mean	1.667	0.917	1.840	0.880

A separate time-to-target evaluation measures elapsed wall-clock time when reward first reaches each common threshold (Table 23). The larger staleness bound reaches rewards of 0.25 and 0.50 sooner, but takes 10.8 h to reach 0.75, compared with 9.4 h for $\eta = 2$. Hence a throughput improvement alone does not establish faster convergence to a specified quality target.

Table 23: Average elapsed time to first reach a target reward.

Target reward	$\eta = 2$	$\eta = 4$
0.25	1.2 h	1.0 h
0.50	4.1 h	3.8 h
0.75	9.4 h	10.8 h

H.7 MILP Feasibility and Network-Aware Scheduling

Inactive replicas and linearity. Equation 2 is a feasibility query for a fixed candidate $\widehat{\Theta}$, not a joint optimization over Θ and y_ψ. Profiled throughput h_ψ and the sampled rollout workload are constants in a query. Consequently, $x_\psi \text{len}_{\text{rollout}} \leq \widehat{\Theta} h_\psi y_\psi$ is linear in the decision variables x_ψ and y_ψ. For positive rollout length, $y_\psi = 0$ enforces $x_\psi = 0$, so unused configurations receive no workload and no division by zero is evaluated. Any allocation feasible at $\widehat{\Theta}$ remains feasible at a larger candidate, which permits binary search to the desired makespan tolerance. This procedure concerns the rollout subproblem under fixed resources and profiles; it does not establish global optimality for the joint two-phase scheduler.

Interconnect bandwidth. Bandwidth enters all three scheduling components. The training-plan search uses pipeline parallelism across machines to avoid placing communication-intensive collectives on slow inter-node links. The rollout cost model estimates each replica's completion time using its GPU composition and intra- and inter-node bandwidth. The repartition phase represents communication links as bandwidth-weighted edges and considers the placement of training and rollout workers, including the communication needed to synchronize policy weights. These mechanisms incorporate topology, while the final partition remains a heuristic whose quality is evaluated in Appendix H.3.

H.8 Scope and Limitations

Our empirical evaluation focuses on mathematical-reasoning RL with autoregressive language models. The five-type GPU comparisons are simulated, and the controlled response-length-shift experiment in Appendix E covers one 1.5B workload. These results do not establish performance on arbitrary heterogeneous hardware, tasks, or patterns of workload drift. The time-to-target results also show that a larger staleness bound can improve throughput while slowing attainment of a higher reward target.

Adapting the scheduler to another RL application requires profiling its stage latencies, memory demands, communication, and response lengths. Workloads with substantial reward-computation cost require extending the current constant-cost treatment and evaluating the resulting stage interactions. Diffusion-based and non-autoregressive language models may change the arithmetic intensity and memory behavior of generation, requiring revised cost models and hardware-allocation criteria. These extensions and validation beyond mathematical reasoning remain future work.