Innovation and Technology Studies
Print ISSN: 3007-6919
Online ISSN: 3007-6927
DOI: https://doi.org/10.61784/its4028

HYBRID DEEP REINFORCEMENT LEARNING AND INTEGER OPTIMIZATION FOR RESOURCE SCHEDULING IN HETEROGENEOUS AI COMPUTING CLUSTERS

PengFei Li[1*], EnMiao Feng[1], GaiKe Wang[2], TianJun Mo[1]
[1]*Duke University, Durham, NC, USA.*
[2]*New York University, New York, NY, USA.*
[*]*Corresponding Author: PengFei Li*

Abstract: AI computing clusters increasingly combine several generations of GPUs with CPU-only servers and host a mix of long-running training jobs and short, latency-sensitive inference or interactive jobs whose demands span GPUs, CPU cores and memory. Scheduling such clusters is a sequential, multi-objective combinatorial problem: heuristic and myopic optimization-based schedulers ignore how current placements constrain future arrivals, whereas pure deep reinforcement learning (DRL) schedulers face an exploding action space and offer no feasibility guarantee. This paper presents HyRIS, a hybrid scheduler that couples DRL with mixed-integer programming (MIP). At each decision epoch, a permutation-equivariant Transformer policy trained by proximal policy optimization emits state-dependent weights of the job-utility function and per-accelerator shadow prices, and a compact MIP converts them into a gang-, multi-resource- and heterogeneity-aware placement that satisfies all capacity constraints by construction. Zero-initialized residual heads anchor the policy to a hand-tuned MIP so that learning starts from a strong and safe operating point, and a reward derived from Little's law jointly accounts for job completion time, monetary cost and service-level-objective (SLO) violations. Trace-driven experiments on the Alibaba PAI GPU cluster trace with V100, P100, T4, legacy GPUs and CPU-only nodes show that, under heavy load, HyRIS reduces the composite scheduling objective by 13.0–23.9% relative to FIFO, fragmentation-aware, multi-resource packing, heterogeneity-aware LP and pure DRL schedulers and remains within 1.9% of the best baseline (Static-MIP), while keeping the SLO-violation rate of latency-sensitive jobs at 11.9% (shortest-job-first: 10.8%). Each MIP is solved in 2.6 ms on average, making HyRIS practical for online use.
Keywords: GPU cluster scheduling; Deep reinforcement learning; Mixed-integer programming; Heterogeneous computing; Multi-resource allocation; AI infrastructure

1 INTRODUCTION

Deep learning (DL) workloads now dominate the capacity of shared AI computing clusters. Production traces from Microsoft [1], SenseTime and Alibaba reveal three properties that make such clusters hard to schedule [2-3]. First, clusters are heterogeneous: several GPU generations coexist with CPU-only servers, and the speedup a job obtains from a newer accelerator varies widely across models [4]. Second, jobs request several resources at once—GPUs (often fractions of a GPU), CPU cores and host memory—in gangs of pods that must start together, so CPU or memory exhaustion can strand otherwise idle GPUs [5-6]. Third, long-running training jobs share the cluster with short inference and interactive jobs whose queueing delay must stay within a service-level objective (SLO) [7]. At every arrival or completion, a scheduler must therefore decide which jobs to start, on which accelerator type and on which nodes, trading off job completion time (JCT), SLO attainment, cost and utilization.

Existing approaches address parts of this problem. Heuristic schedulers such as shortest-job-first with best-fit, multi-resource packing or fragmentation-aware placement are fast but encode a fixed trade-off [6, 8]. Optimization-based schedulers solve a linear or integer program in each round [4, 9]; they handle hard constraints and heterogeneity elegantly, but their objectives are hand-crafted and myopic: a program that greedily fills idle GPUs cannot anticipate that a burst of latency-sensitive jobs is about to arrive, or that a V100 should be kept for a job with a higher speedup. DRL schedulers can learn such foresight from experience [10-12], yet they construct a placement through long sequences of discrete choices whose number grows with jobs × accelerator types × nodes, and they offer no guarantee that an action respects capacity and gang constraints. Learning such long action sequences from scratch is sample-inefficient.

This paper argues that the two paradigms are complementary and should be combined through a division of labor: DRL decides what to optimize at each moment—the state-dependent value of each waiting job and the opportunity cost of each resource—while integer optimization decides how to realize it as a feasible placement. We instantiate this idea in HyRIS (Hybrid Reinforcement-learning and Integer-programming Scheduler). A Transformer encodes waiting jobs, accelerator pools and global statistics as a set of tokens and outputs state-dependent weights of the job-utility function together with shadow prices for every GPU type, a CPU/memory stranding price and a cost weight [13]. These coefficients parameterize a per-epoch mixed-integer program (MIP), solved by HiGHS [14], that selects jobs, accelerator types and per-node pod counts under gang, type-compatibility and three-dimensional capacity constraints. The policy is trained with the solver in the loop by proximal policy optimization (PPO) [15]. The main contributions are as follows.

© By the Author(s) 2024, under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0).

A hybrid DRL–MIP scheduling architecture in which the policy acts in a compact space of MIP objective coefficients, so that every action is feasible by construction and the combinatorial placement is solved exactly at millisecond scale.
A shadow-price action design with residual anchoring: zero-initialized output heads make the untrained policy reproduce a hand-tuned MIP, and PPO learns state-dependent corrections, which yields stable and sample-efficient training.
A multi-objective reward that combines Little's-law-based JCT accounting with monetary cost and SLO penalties for mixed training and inference workloads.
A trace-driven evaluation on the Alibaba PAI GPU trace against seven baselines [3], including heterogeneity-aware LP scheduling [4], fragmentation-aware placement and a pure DRL scheduler [6], complemented by load-sensitivity, ablation and scalability studies.

2 RELATED WORK

2.1 Deep Learning Cluster Scheduling

Early GPU cluster managers targeted JCT and fairness for training jobs. Gandiva exploits the predictability of DL iterations for time-slicing and migration [16]; Tiresias schedules distributed jobs without duration knowledge [17]; Optimus fits online performance models to allocate parameter servers and workers [18]; Themis pursues finish-time fairness [19]; HiveD provides guarantees for virtual clusters [20]; and AntMan co-executes jobs through dynamic scaling [21]. Pollux co-adapts resources and batch sizes to maximize goodput [22]. Heterogeneity was addressed by Gavel [4], which expresses scheduling policies as linear programs over effective throughputs, and by Sia [9], which solves an integer program in every round to assign heterogeneous GPUs to elastic jobs. Synergy shows that CPU and memory allocations strongly influence DL throughput [5], FGD reduces GPU fragmentation for GPU-sharing workloads [6], Lyra loans capacity between inference and training clusters [7], and Lucid relies on interpretable models for non-intrusive scheduling [23]. Workload studies of Philly [1], Helios and PAI form the empirical basis of this line of work [2-3]. HyRIS is complementary to these systems: instead of fixing the objective of the per-round program, it learns state-dependent objective coefficients for an exact placement MIP.

2.2 Learning-Based Schedulers

DeepRM first cast multi-resource job scheduling as a DRL problem [10], Decima combined graph neural networks with policy gradients for data-processing DAGs [11], and DL2 applied DRL to resource allocation in DL clusters [12]. These agents emit placement decisions one element at a time and rely on action masking or penalties to remain feasible. Our pure-DRL baseline follows this design.

2.3 Machine Learning for Combinatorial Optimization

Learning to improve combinatorial solvers is an active field [24]: imitation of branching rules [25], RL-based cut selection and attention-based construction heuristics are representative examples [26-27]. Delarue et al. handle RL with combinatorial actions by optimizing a learned value function inside an MIP [28]. HyRIS adopts the opposite coupling—the network produces the objective and the solver produces the action—which keeps the MIP small and its solution time predictable. The anchoring scheme is related to residual policy learning [29], which improves a non-differentiable base controller with a learned residual.

3 SYSTEM MODEL AND PROBLEM FORMULATION

3.1 Heterogeneous Cluster

The cluster consists of a node set $\mathcal{N}$. Node n has type $\tau(n) \in \mathcal{T} = \{\text{V100, P100, T4, MISC, CPU}\}$, where MISC denotes legacy GPUs, and a capacity vector $C_n = (C^g_n, C^c_n, C^m_n)$ of GPUs, CPU cores and memory (GB); $F_n(t)$ denotes its free capacity at time t. GPU type t has a peak-speed ratio r_t relative to the V100 and an hourly price p_t per GPU, and CPU cores and memory are priced at p^c and p^m.

3.2 Jobs

Job j arrives at time a_j and consists of k_j identical pods that must start simultaneously (gang scheduling) on nodes of a single accelerator type $t \in \mathcal{A}_j$. Each pod requests $d_j = (d^g_j, d^c_j, d^m_j)$; a fractional $d^g_j < 1$ represents GPU sharing. A job pinned by its user has $|\mathcal{A}_j| = 1$, and CPU-only jobs may run on any node. Following the heterogeneity model of [4], the speed of job j on type t is $s_{j,t} = r_t \eta_j$, where η_j ($0 \le \eta_j \le 1$) captures its sensitivity to the accelerator generation, so its runtime on t is $W_j/s_{j,t}$ with W_j its work in V100-seconds. The scheduler does not know W_j but receives the history-based estimate provided with the trace [3]. Jobs are either latency-sensitive (LS)—short single-pod inference or interactive jobs whose queueing delay should not exceed θ—or best-effort (BE) training jobs. Execution is non-preemptive.

3.3 Objective and Decision Process

Let c_j be the completion time, q_j the queueing delay and κ_j the monetary cost of job j. The scheduler minimizes

$$J = \sum_j (c_j - a_j) + \lambda_c \sum_j \kappa_j + \lambda_s \sum_{j \in LS} \mathbb{1}[q_j > \theta]. \quad (1)$$

By Little's law, $\Sigma_j(c_j - a_j) = \int N(t)dt$, where $N(t)$ is the number of jobs in the system, so (1) decomposes over time. Decisions are taken at epochs t_k triggered by arrivals, completions and a 300-s timer while jobs are waiting. At epoch k the scheduler observes the state s_k—queue, free capacities and running jobs—and starts a set of jobs together with their accelerator types and node assignments. The reward is

$$r_k = -\left[\int_{t_k}^{t_{k+1}} N(t)dt \,/\, 3600 + \lambda_c \Delta K_k + \lambda_s \Delta V_k \right], \quad (2)$$

where ΔK_k and ΔV_k are the cost incurred and the new SLO violations in $(t_k, t_{k+1}]$. Maximizing the expected return therefore minimizes (1) up to discounting, and every epoch receives a dense learning signal.

4 THE HYRIS SCHEDULER

4.1 Overview

Figure 1 depicts the decision loop. At each epoch a candidate window $\mathcal{W}_k$ of at most $K = 24$ waiting jobs is formed by feasibility-first selection: jobs that fit somewhere are preferred and then ranked by a base utility, which enables backfilling while keeping blocked high-priority jobs visible when fewer than K jobs fit. The Transformer policy maps the state to MIP coefficients, the MIP returns a feasible placement, and the cluster advances to the next epoch.

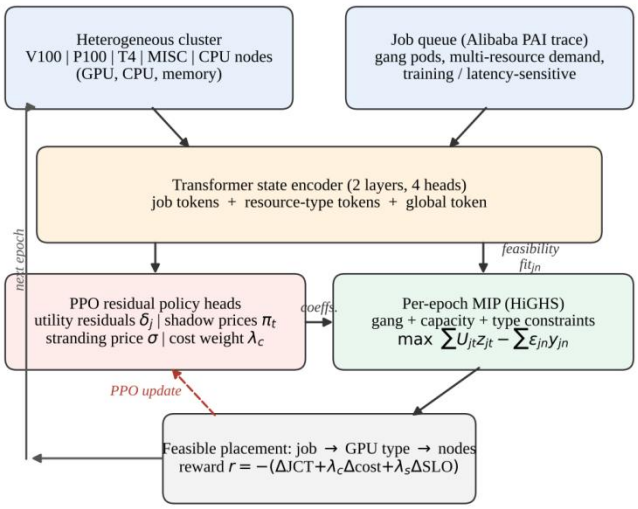


Figure 1 Overview of HyRIS

Note: The learned policy sets the objective coefficients of a per-epoch MIP; the MIP guarantees feasibility of every placement. PPO updates the policy from the multi-objective reward.

4.2 Base Utility and Residual Action Space

A base utility encodes standard scheduling knowledge—aging, class priority and a shortest-job preference:

$$u_j^0 = 1 + \beta_a \ln\left(1 + w_j/\hat{d}_j\right) + \beta_l \mathbb{1}[j \in LS] + \beta_s \log_{10}\left(D/\hat{d}_j\right), \quad (3)$$

where w_j is the current waiting time, $\hat{d}_j$ the estimated runtime on the fastest admissible type, $D = 1$ day and $\beta_a = \beta_l = \beta_s = 1$. The policy outputs a Gaussian action $a_k = (\omega, \alpha, \alpha_\sigma, \alpha_\lambda)$ with $\omega \in \mathbb{R}^4$ and $\alpha \in \mathbb{R}^4$. The weights ω re-weight four job features φ_j—aging $\ln(1 + w_j/\hat{d}_j)$, class $\mathbb{1}[j \in LS]$, shortness $\log_{10}(D/\hat{d}_j)/4$ and size $\ln(1 + k_j d_j^g)/3$—through the residual $\delta_j = \omega^\top \varphi_j$, and the action is mapped to

$$u_j = u_j^0 e^{\delta_j}, \ \pi_t = \mathrm{softplus}(\alpha_t - 2), \ \sigma = \sigma_0 e^{\alpha_\sigma}, \ \tilde{\lambda} = \lambda_0 e^{\alpha_\lambda} \quad (4)$$

with clipped residuals, $\sigma_0 = 0.5$ and $\lambda_0 = 0.1$. Here π_t is the shadow price of type-t GPU capacity, σ penalizes stranding and $\tilde{\lambda}$ weighs monetary cost. The action has only 10 dimensions, independent of the cluster size and the queue length, in contrast to the jobs × types × nodes choices faced by a placement-level DRL agent.

4.3 Per-Epoch Mixed-Integer Program

Let $z_{j,t} \in \{0,1\}$ indicate that job j starts on type t and $y_{j,n} \in \mathbb{Z}_{\geq 0}$ be the number of its pods on node n. The utility of starting j on t is

$$U_{j,t} = u_j \rho_{j,t} - \tilde{\lambda} \ln\left(\hat{c}_{j,t}/\min_{t'} \hat{c}_{j,t'}\right) - \pi_t k_j d_j^g / G_t, \quad (5)$$

where $\rho_{j,t} = s_{j,t}/\max_{t' \in \mathcal{A}_j} s_{j,t'}$ is the relative speed, $\hat{c}_{j,t}$ the estimated cost of running j on t, and G_t the number of GPUs per type-t node. A per-pod placement penalty

$$\varepsilon_{j,n} = \beta_b f_n/k_j + \sigma \cdot \max\left(0, d_j^c/C_n^c - d_j^g/C_n^g, d_j^m/C_n^m - d_j^g/C_n^g\right) \quad (6)$$

favors best-fit packing (f_n is the free fraction of node n and $\beta_b = 0.05$) and charges CPU or memory consumption beyond a GPU node's proportional share, i.e., the resources whose exhaustion would strand its GPUs (the second term applies to GPU nodes only). HyRIS solves

$$\max \sum_{j \in \mathcal{W}} \sum_{t \in \mathcal{A}_j} U_{j,t}\, z_{j,t} \;-\; \sum_{j} \sum_{n} \varepsilon_{j,n}\, y_{j,n} \tag{7}$$

$$\text{s.t.} \sum_{t} z_{j,t} \;\le\; 1, \quad \forall j, \tag{8}$$

$$\sum_{n:\, \tau(n)=t} y_{j,n} \;=\; k_j\, z_{j,t}, \quad \forall j, t, \tag{9}$$

$$\sum_{j} d_j^r\, y_{j,n} \;\le\; F_n^r, \quad \forall n, r \in \{g, c, m\}, \tag{10}$$

$$0 \;\le\; y_{j,n} \;\le\; \min\left(k_j, \varphi_{j,n}\right), \; z_{j,t} \in \{0, 1\}, \tag{11}$$

where $\varphi_{j,n}$ is the number of pods of j that fit in F_n. Constraint (9) enforces gang scheduling on a homogeneous accelerator type and (10) co-allocates GPUs, CPU cores and memory. Only pairs with $U_{j,t} > 0$ enter the model: high shadow prices let the policy withhold low-value jobs from scarce accelerators and keep headroom for future arrivals, while low prices recover work-conserving packing. The model has at most $|\mathcal{W}|(|\mathcal{T}| + |\mathcal{N}|)$ integer variables and is solved by HiGHS with a 1-s limit and a 0.1% relative gap; infeasible actions cannot occur.

4.4 Transformer State Encoder

The state is a set of tokens: one per candidate job (21 features, including log-estimated runtime, waiting time, per-pod demands, gang size, class, pinning, relative speeds, per-type feasibility, base utility and SLO slack), one per resource type (13 features, including free GPU/CPU/memory fractions, idle-node fraction, largest free GPU block, queue pressure, price, speed and a type embedding), and one global token (queue length and demand, allocation ratios, recent arrival rate, time since the last decision and time of day). Tokens are linearly embedded into 64 dimensions and processed by two pre-norm Transformer layers with four attention heads [13]; padded job slots are masked. Self-attention is permutation-equivariant, so the policy does not depend on the order of the queue, and the fixed token count lets one network serve clusters of any size. GPU-type tokens feed the price heads, and the global token, which attends to all job and resource tokens, feeds the priority-weight, stranding and cost heads as well as, with mean pooling, the value head.

4.5 Training with Residual Anchoring

All output heads are zero-initialized. At initialization the deterministic policy therefore outputs $\omega = \alpha = 0$ and HyRIS reproduces the hand-tuned Static-MIP exactly, while Gaussian exploration with standard deviation e^{-1} perturbs the coefficients around it. PPO with generalized advantage estimation then learns state-dependent residuals [15, 30]. The MIP is treated as part of the environment, so no differentiation through the solver is required. Training episodes are 6-h windows drawn from the training portion of the trace with a load factor sampled uniformly from [1.0, 2.5], which exposes the policy to diverse congestion levels. Algorithm 1 summarizes the procedure.

Algorithm 1: Solver-in-the-loop training of HyRIS

```
Initialize encoder θ; zero-initialize output heads; log σ ← −1
for iteration i = 1, …, I do
    for episode e = 1, …, E do
        sample a 6-h training window and load ρ ~ U[1.0, 2.5]
        for each decision epoch k do
            encode tokens; (μ_k, V_k) ← f_θ(s_k); a_k ~ 𝒩(μ_k, σ²)
            map a_k to (u, π, σ, λ̃) by (4); solve MIP (7)–(11)
            start the selected jobs; observe r_k by (2)
        compute GAE advantages; update θ and σ by PPO-clip
    every 5 iterations: validate deterministic policy, keep best
```

5 EXPERIMENTAL SETUP

5.1 Dataset and Workload

We use the Alibaba PAI GPU cluster trace (cluster-trace-gpu-v2020), collected in July and August 2020 from a production cluster with over 6,500 GPUs [3], specifically the publicly released 100,000-job sample that accompanies the trace's simulator and covers 12.1 days of submissions. Each record provides the number of pods, the total (fractional) GPU and CPU requests, the GPU type the job ran on, its duration and history-based duration estimates of recurring job groups. Per-pod demands are obtained by dividing totals by the pod count. Because the sample omits memory, per-pod memory is synthesized as the CPU request multiplied by a log-normal ratio with a median of 4 GB per core. Jobs that request more than 16 GPUs or 384 cores, or that cannot fit the scaled cluster, are removed, leaving 96,971 jobs (97.0%). Of these, 39.4% are LS jobs (single pod, at most one GPU, at most 15 min), 15.1% are CPU-only and 8.4% are pinned to their recorded GPU type (10% of GPU jobs). The GPU sensitivity η_j is drawn uniformly from [0.2, 1.0]. The trace is split chronologically into days 0–7.5 for training, 7.5–8.5 for validation and 8.5–11.0 for testing (five disjoint 12-h windows). Arrivals are thinned to a load factor ρ, defined as the offered GPU-hours at recorded runtimes divided by the cluster's GPU capacity. Because flexible jobs run faster on newer GPUs than on the legacy GPUs on which most of them were recorded, $\rho > 1$

corresponds to a stable but heavily contended cluster. The main load is $\rho = 2.0$, and $\rho \in \{1.0, 1.5, 2.0, 2.5\}$ is used for sensitivity analysis.

5.2 Cluster Configuration and Cost Model

Table 1 lists the 30-node cluster, whose node types mirror the GPU types of the trace. GPU prices follow representative public-cloud on-demand rates, and CPU cores and memory cost 0.0316 USD per core-hour and 0.0042 USD per GB-hour. The SLO threshold is $\theta = 120$ s, $\lambda_c = 0.1$ per USD and $\lambda_s = 2$ job-hours per violation.

Table 1 Heterogeneous Cluster Configuration

Type	Nodes	GPU/node	Cores	Mem (GB)	Speed rt	USD/GPU-h
V100	2	8	96	512	1.00	2.48
P100	8	2	64	512	0.55	1.46
T4	12	2	96	512	0.50	0.35
MISC	4	8	96	384	0.30	0.45
CPU	4	0	96	512	–	–
Total	30	88 GPUs	2,624	14,848		

5.3 Baselines

FIFO-FF serves jobs in arrival order with backfilling and first-fit placement. *SJF-BF* serves the shortest estimated job first, tries the fastest admissible type first and places pods best-fit. *Tetris* combines a multi-resource alignment score with a shortest-remaining-time term [8]. *Gavel-LP* solves a heterogeneity-aware LP that maximizes normalized throughput [4], adapted to non-preemptive placement. *FGD* applies fragmentation gradient descent at the node level [6], using the pod-shape distribution of the training split. *Static-MIP* is the MIP of Section 4 with the hand-tuned coefficients, i.e., HyRIS without learning. *DRL-PPO* is a pure DRL scheduler in the style of [10-11]: it shares the HyRIS encoder and sequentially selects (job, GPU type) pairs or a no-op, with infeasible actions masked and best-fit node placement. All schedulers use the same simulator, duration estimates and constraints.

5.4 Implementation and Metrics

The event-driven simulator, the baselines and HyRIS are implemented in Python; the networks use JAX and the MIPs are solved by HiGHS through SciPy. HyRIS and DRL-PPO receive an identical budget of 30 PPO iterations with four episodes each (Adam with learning rate 3×10^{-4}, clipping 0.2, $\gamma = 0.99$, GAE $\lambda = 0.95$, four epochs, minibatch 256) on a single CPU core, and the checkpoint with the best objective on four validation episodes (two windows × two loads) is evaluated deterministically. For DRL-PPO, whose arg-max policy degenerates to excessive no-op actions, we evaluate the stochastic policy with a fixed random seed and select between its best and final checkpoints on the validation windows, which favors this baseline. We report the average and 95th-percentile JCT, the average queueing delay, the SLO-violation rate of LS jobs, the resource cost, the GPU allocation ratio and the composite objective (1) normalized per job.

6 RESULTS

6.1 Overall Performance

Table 2 reports means over the five test windows at $\rho = 2.0$ (1,230 jobs per window on average). HyRIS attains a composite objective of 2.115 per job, within 1.9% of the strongest baseline (Static-MIP); its average JCT of 1.74 h is within 2.1% of the best JCT-oriented baseline (Static-MIP, 1.70 h); the SLO-violation rate of latency-sensitive jobs is 11.9% (best baseline: SJF-BF, 10.8%); and resource cost is 3641 USD per window, 0.3% below the cheapest baseline (DRL-PPO) and 2.1% below FIFO-FF. Among the baselines, Static-MIP attains the shortest average JCT (1.70 h) and Static-MIP the best composite objective (2.076 per job). SJF-BF benefits from the accurate group-based duration estimates of the trace, but it starts latency-sensitive jobs only when capacity happens to be free and therefore violates the SLO for 10.8% of them. Tetris (2.21 h) weighs packing alignment more heavily than job length and Gavel-LP (2.26 h) maximizes normalized throughput without regard to job length, while FGD (1.96 h) minimizes fragmentation but serves jobs in arrival order. DRL-PPO reaches 2.616 per job, 23.7% above HyRIS, which illustrates how difficult it is to construct gang placements action by action in a large discrete space with the same training budget.

Table 2 Performance at $\rho = 2.0$ (Mean of Five Test Windows)

Method	JCT (h)	P95 JCT (h)	Wait (min)	SLO viol. (%)	Cost (kUSD)	Obj./job
FIFO-FF	1.93	10.82	40.2	27.9	3.72	2.431
SJF-BF	1.73	9.56	29.5	10.8	3.72	2.098
Tetris	2.21	10.98	53.2	31.2	3.66	2.707
Gavel-LP	2.26	10.76	60.7	33.4	3.67	2.779
FGD	1.96	10.81	38.5	25.3	3.65	2.446
Static-MIP	1.70	9.25	28.7	11.6	3.65	2.076
DRL-PPO	2.05	10.78	44.0	38.6	3.65	2.616

Method	JCT (h)	P95 JCT (h)	Wait (min)	SLO viol. (%)	Cost (kUSD)	Obj./job
HyRIS	1.74	9.26	29.5	11.9	3.64	2.115

HyRIS places 37.9% of its GPU jobs on V100s, versus 39.6% for Static-MIP and 39.2% for SJF-BF, and its average GPU allocation over the makespan is 39.0% (SJF-BF: 39.5%). The mean accelerator sensitivity η of the jobs placed on V100s is similar across these schedulers (0.63 under HyRIS, 0.62 under Static-MIP and 0.61 under SJF-BF). Under HyRIS, 15.5% of LS jobs run on T4 nodes (Static-MIP: 14.9%).

6.2 Sensitivity to Load

Figure 2 and Table 3 show how the schedulers behave as the load factor increases from 1.0 to 2.5. HyRIS stays within 1.9% of the strongest baseline at every load level (objective difference: +0.7% vs. SJF-BF at ρ = 1.0, +1.1% vs. SJF-BF at ρ = 1.5, +1.9% vs. Static-MIP at ρ = 2.0, +0.7% vs. Static-MIP at ρ = 2.5). The spread of the objective across schedulers grows from 19.1% at ρ = 1.0 to 78.9% at ρ = 2.5, confirming that scheduling decisions matter most under contention.

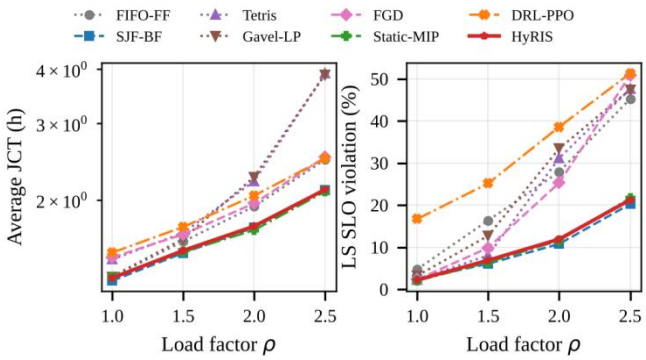


Figure 2 Average JCT (Log Scale) and SLO-Violation Rate of LS Jobs Versus Load Factor ρ

Table 3 Composite Objective per Job under Different Loads

Method	ρ = 1.0	ρ = 1.5	ρ = 2.0	ρ = 2.5
FIFO-FF	1.658	2.003	2.431	3.093
SJF-BF	1.619	1.833	2.098	2.536
Tetris	1.765	2.015	2.707	4.517
Gavel-LP	1.653	1.999	2.779	4.487
FGD	1.786	2.027	2.446	3.163
Static-MIP	1.639	1.840	2.076	2.525
DRL-PPO	1.928	2.201	2.616	3.140
HyRIS	1.630	1.853	2.115	2.542

6.3 Queueing-Delay Distribution

Figure 3 plots the distribution of queueing delays at ρ = 2.0, separately for LS and BE jobs. Under HyRIS, 87.0% of LS jobs start within the 120-s SLO, compared with 88.2% under SJF-BF, 87.2% under Static-MIP and 71.4% under FGD; the 90th-percentile LS queueing delay is 175 s for HyRIS versus 166 s for SJF-BF and 170 s for Static-MIP. For BE training jobs the median queueing delay is 0 s under HyRIS, 0 s under SJF-BF and 0 s under FIFO-FF, and the 90th percentile is 2,262 s, 2,300 s and 6,274 s, respectively.

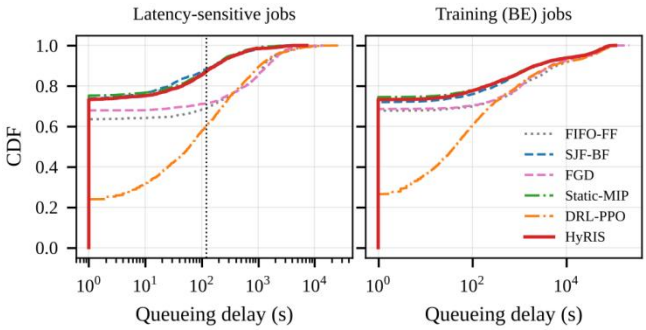


Figure 3 CDF of Queueing Delay at ρ = 2.0 for Latency-Sensitive (Left, Dotted Line: 120-s SLO) and Best-Effort Training Jobs (Right)

6.4 Learning Efficiency

Figure 4 plots the objective of the training episodes, whose sequence of windows and loads is identical for all variants. HyRIS begins at 1.564 per job—the level of the hand-tuned MIP plus exploration noise—and averages 1.678 over its last five iterations; on the validation windows its deterministic policy improves from 2.028 (identical to Static-MIP) to 1.991, a 1.8% reduction obtained in 8.2 min of single-core training. On the unseen test windows, the learned coefficients perform within 1.9% of the static ones (Table 2). DRL-PPO starts from a random policy at 1.758 per job and averages 2.006 over its last five iterations, 19.5% above HyRIS. The comparison confirms that learning in the space of MIP coefficients is markedly more sample-efficient than learning placements directly.

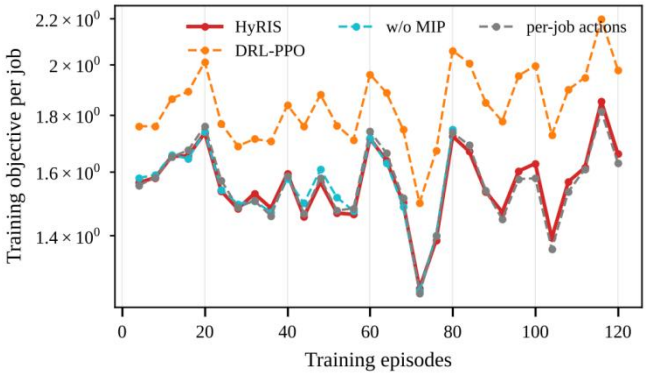


Figure 4 Objective per Job of the Training Episodes (Moving Average over Three Iterations, Log Scale)
Note: All variants are trained on the identical sequence of randomly drawn windows and loads.

6.5 Ablation Study

Table 4 isolates the components of HyRIS at ρ = 2.0. To match budgets, every learned variant is trained for 80 episodes (20 PPO iterations) and selected on the validation episodes. Removing the learned policy (Static-MIP) decreases the objective by 1.8%. Replacing the MIP by greedy priority-ordered decoding with the same learned coefficients decreases the objective by 2.0%. Replacing the 10-dimensional coefficient action by per-job residual utilities ($K + 6 = 30$ dimensions) decreases the objective by 2.1%. All variants that share the utility-and-price formulation lie within 2.2% of each other, whereas DRL-PPO, which lacks this structure, is 23.7% worse than HyRIS: the structured coefficient space solved at every epoch accounts for most of the gain over the baselines, and learning acts as a bounded adjustment around the anchored prior.

Table 4 Ablation Study at ρ = 2.0

Variant	JCT (h)	SLO viol. (%)	Cost (kUSD)	Obj./job	Δ Obj.
HyRIS (full)	1.74	11.9	3.64	2.115	+0.0%
w/o DRL (Static-MIP)	1.70	11.6	3.65	2.076	-1.8%
w/o MIP (greedy decoding)	1.71	10.6	3.67	2.072	-2.0%
per-job residual actions	1.69	11.7	3.66	2.070	-2.1%

6.6 Learned Behavior

Figure 5 traces the shadow prices emitted by HyRIS over the first test window at ρ = 2.0 together with the queue length. The learned prices stay close to their means: their means are V100 0.124, P100 0.124, T4 0.122, MISC 0.122 (the hand-tuned value is 0.127), and the V100 price has the strongest correlation with the queue length (Pearson r = -0.58). The stranding price averages 0.54 (default 0.5) and the cost weight 0.095 (default 0.1). The priority weights adapt as well: their means are 0.02 (aging), -0.10 (class), -0.03 (shortness) and 0.07 (size), and the shortness weight has a correlation of 0.18 with the queue length.

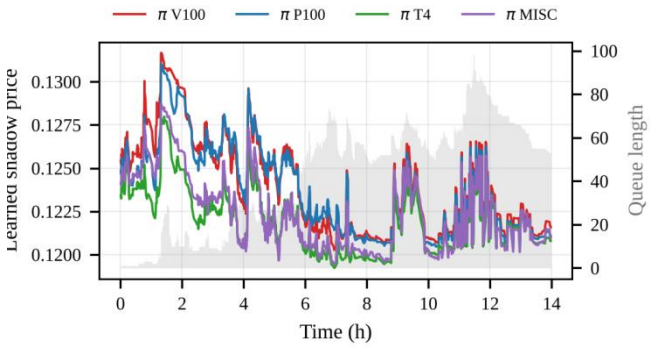


Figure 5 Learned GPU Shadow Prices (5-Epoch Moving Average) and Queue Length during the First Test Window at ρ = 2.0

6.7 Overhead and Scalability

Figure 6 reports MIP solve times as the cluster grows and as the candidate window is enlarged. On the 30-node cluster the mean MIP solve time of HyRIS is 2.6 ms (95th percentile 5.6 ms) with 20 integer variables on average. When the cluster and the workload are scaled to 240 nodes (704 GPUs), the mean solve time grows to 3.5 ms for HyRIS and 3.3 ms for Static-MIP, with 95th percentiles of 7.8 ms and 7.8 ms. Enlarging the window to K = 48 on 120 nodes keeps the mean solve time at 2.3 ms. The policy network has 79,052 parameters; a forward pass takes 0.20 ms and state encoding 0.07 ms, so the end-to-end decision latency of HyRIS on the 30-node cluster is 5.3 ms, negligible relative to the 300-s timer and to job runtimes.

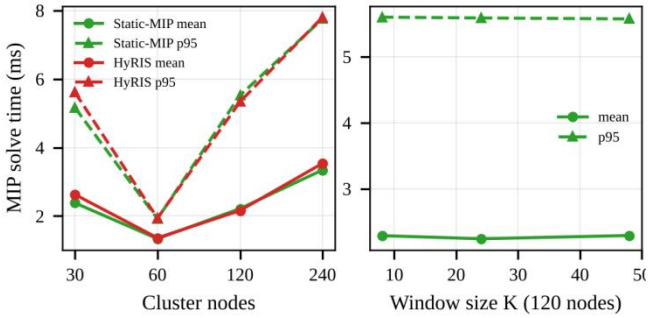


Figure 6 MIP Solve Time Versus Cluster Size (Left; Workload Scaled Proportionally) and Versus Candidate-Window Size K on 120 Nodes (Right)

7 CONCLUSION

This paper presented HyRIS, a scheduler for heterogeneous AI computing clusters that splits online scheduling into a learned, state-dependent objective and an exact per-epoch MIP. The policy acts through job utilities and shadow prices, so every decision is feasible by construction; residual anchoring makes learning safe and sample-efficient; and the Transformer encoder makes the policy independent of queue order and cluster size. On the Alibaba PAI trace, HyRIS reduces the composite scheduling objective by 13.0–23.9% relative to FIFO, fragmentation-aware, multi-resource packing, heterogeneity-aware LP and pure DRL schedulers, stays within 1.9% of the best baseline (Static-MIP) and is 19.2% better than a pure DRL scheduler trained with the same budget, with an end-to-end decision latency of 5.3 ms. Future work will extend the action space to preemption and elastic resizing, incorporate power-aware objectives, and evaluate the scheduler on larger production deployments.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Jeon M, Venkataraman S, Phanishayee A, et al. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In: Proceedings of the USENIX Annual Technical Conference (ATC), 2019.

[2] Hu Q, Sun P, Yan S, et al. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2021.

[3] Weng Q, Xiao W, Yu Y, et al. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2022.

[4] Narayanan D, Santhanam K, Kazhamiaka F, et al. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020.

[5] Mohan J, Phanishayee A, Kulkarni J, et al. Looking beyond GPUs for DNN scheduling on multi-tenant clusters. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022.

[6] Weng Q, Yang L, Yu Y, et al. Beware of fragmentation: Scheduling GPU-sharing workloads with fragmentation gradient descent. In: Proceedings of the USENIX Annual Technical Conference (ATC), 2023.

[7] Li J, Xu H, Zhu Y, et al. Lyra: Elastic scheduling for deep learning clusters. In: Proceedings of the European Conference on Computer Systems (EuroSys), 2023: 835-850. DOI: 10.1145/3552326.3587445.

[8] Grandl R, Ananthanarayanan G, Kandula S, et al. Multi-resource packing for cluster schedulers. In: Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM), 2014.

[9] Jayaram Subramanya S, Arfeen D, Lin S, et al. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. SOSP '23: Proceedings of the 29th Symposium on Operating Systems Principles, 2023: 642-657. DOI: 10.1145/3600006.3613175.

[10] Mao H, Alizadeh M, Menache I, et al. Resource management with deep reinforcement learning. In: Proceedings of ACM Workshop on Hot Topics in Networks (HotNets), 2016.
[11] Mao H, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM), 2019.
[12] Peng Y, Bao Y, Chen Y, et al. DL2: A deep learning-driven scheduler for deep learning clusters. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(8): 1947-1960. DOI: 10.1109/TPDS.2021.3052895.
[13] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.
[14] Huangfu Q, Hall J A J. Parallelizing the dual revised simplex method. Mathematical Programming Computation, 2018, 10(1): 119-142.
[15] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. arXiv:1707.06347, 2017. DOI: 10.48550/arXiv.1707.06347.
[16] Xiao W, Bhardwaj R, Ramjee R, et al. Gandiva: Introspective cluster scheduling for deep learning. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018.
[17] Gu J, Chowdhury M, Shin K G, et al. Tiresias: A GPU cluster manager for distributed deep learning. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2019.
[18] Peng Y, Bao Y, Chen Y, et al. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the European Conference on Computer Systems (EuroSys), 2018.
[19] Mahajan K, Balasubramanian A, Singhvi A, et al. Themis: Fair and efficient GPU cluster scheduling. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2020.
[20] Zhao H, Han Z, Yang Z, et al. HiveD: Sharing a GPU cluster for deep learning with guarantees. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020.
[21] Xiao W, Ren S, Li Y, et al. AntMan: Dynamic scaling on GPU clusters for deep learning. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020.
[22] Qiao A, Choe S K, Jayaram Subramanya S, et al. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021.
[23] Hu Q, Zhang M, Sun P, et al. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023.
[24] Bengio Y, Lodi A, Prouvost A. Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, 2021, 290(2): 405-421.
[25] Gasse M, Chételat D, Ferroni N, et al. Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems (NeurIPS), 2019.
[26] Tang Y, Agrawal S, Faenza Y. Reinforcement learning for integer programming: Learning to cut. In: Proceedings of the International Conference on Machine Learning (ICML), 2020.
[27] Kool W, van Hoof H, Welling M. Attention, learn to solve routing problems! In: Proceedings of the International Conference on Learning Representations (ICLR), 2019.
[28] Delarue A, Anderson R, Tjandraatmadja C. Reinforcement learning with combinatorial actions: An application to vehicle routing. In: Advances in Neural Information Processing Systems (NeurIPS), 2020.
[29] Silver T, Allen K, Tenenbaum J, et al. Residual policy learning. arXiv:1812.06298, 2018.
[30] Schulman J, Moritz P, Levine S, et al. High-dimensional continuous control using generalized advantage estimation. In: Proceedings of the International Conference on Learning Representations (ICLR), 2016.