Scheduling jobs on Multi Instance GPUs (MIG) using Reinforcement Learning

Tanvi Hisaria, Devyani Vij, Aayush Srivastava

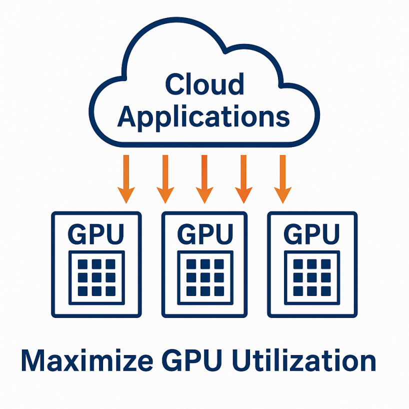
ORCSE4529: Reinforcement Learning
Columbia University

December 9, 2025

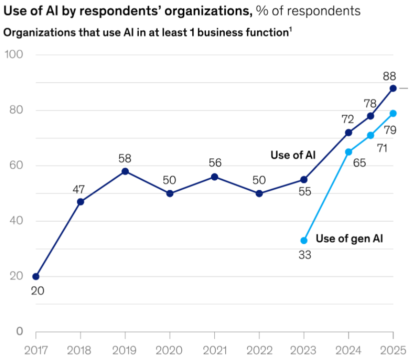
Multi-Instance GPU (MIG)

MiG hardware partitioning technology is great for isolated workloads on GPUs.

Relevance of MIG Scheduling Algorithms



Cloud computing utilizes GPU scheduling



Rapid AI growth demands scalable GPU capacity

Relevance of MIG Scheduling Algorithms

Energy is expensive and high in demand

Real-time AI inference workloads

- Autonomous driving
- Voice assistants
- Fraud detection
- Real-time recommendations

Real-time inference requires minimal delays.

Existing Work

Two Components of MIG Resource Management

- **Repartitioning:** Deciding how to split a GPU into MIG slices
- **Scheduling:** Assigning incoming jobs to existing slices

Prior Research

- Mao et al. (2019): RL for cluster scheduling, showing that queue-based features improve scheduling quality.
- SMART-MIG (IPDPS 2026): Mean-field MARL for *repartitioning* MIG GPUs to reduce energy and tardiness.

Scheduling Goal: Minimize Energy + Tardiness (ET)

$$ET = \frac{1}{N}\sum_{k=1}^{N}\frac{ae_k + t_k}{a+1},$$

where e_k is total energy, t_k is average tardiness, and a balances their importance.

MDP Formulation: SARP

State (S)

- MIG slice availability, queue features, job durations deadlines
- Added features: queue_len_norm, free_slice_fraction

Action (A)

- Assign next job to a valid MIG slice

Reward (R)

- Negative weighted combination of energy + tardiness
- Derived from ET metric: $ET = \frac{ae_k + t_k}{a+1}$

Transitions (P)

- Job executed on slice, queue updates, new jobs may arrive
- Environment evolves via simulator dynamics

Additional Setup

Environment Type

- Custom NumPy-based simulator of MIG-partitioned GPUs
- Mixed inference/training workloads with stochastic arrivals

Setting

- Simulator-based online interaction (Gym-style episodes)
- Agent learns through repeated rollouts in the simulator, not a real GPU cluster

RL Paradigm

- Policy gradient method suitable for high-dimensional, continuous state spaces
- Maskable PPO is a variant that prevents invalid actions and maintains PPO's stablility, sample-efficiency, and wide use for scheduling-like tasks PPO Clipped Objective

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\big)\Big]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

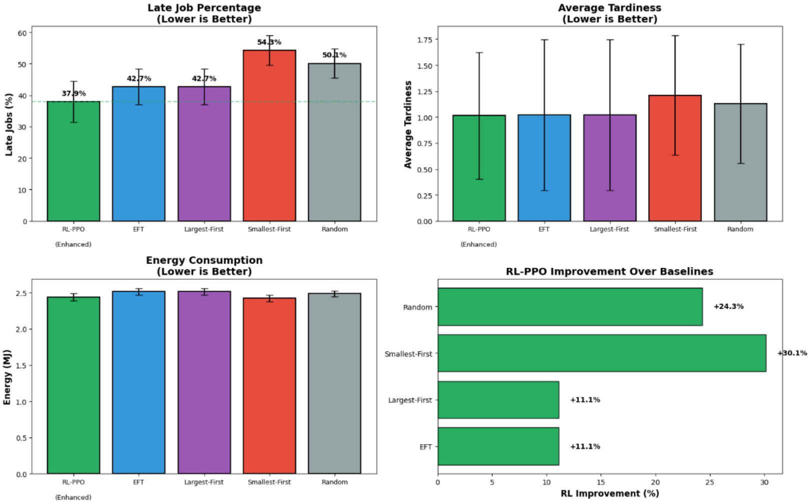
Summary of Enhancements

Aspect	Original	Improved	Impact
Environment	Pandas	NumPy	10–50× faster
Deadlines	1.0–1.5×	2.0–4.0×	Learnable problem
States Features Added	Basic	+slice sizes, +urgency, load	Better learning signal
Rewards	End-only	Immediate	Better credit assignment
Network	[256,256]	[256,256,128]	More capacity
Training steps	200k	500k	More learning
LR	Fixed	Annealing	Stability
Entropy	Fixed	Decaying	Explore → exploit

Training Configuration (Final RL Agent)

Parameter	Value	Improvement over Original
Network	[256,256,128]	Deeper (+1 layer)
Batch size	4096	2× larger
Epochs	10	2× more
Timesteps	500,000	2.5× more
Learning rate	3e-4 → 1e-5	Annealing schedule
Entropy	0.02 → 0.001	Decaying entropy bonus
Clip range	0.15	Tighter updates

Final Results: RL Vs Heuristic Baselines



Final Results: RL vs Heuristic Baselines

Performance Comparison: Enhanced RL vs Heuristics

Method	Late Jobs (%)↓	Avg. Tardiness↓	Energy (MJ)
RL-PPO (Enhanced)	**37.9±6.5**	**1.01±0.61**	2.44±0.05
EFT	42.7±5.7	1.02±0.73	2.51±0.05
Largest-First	42.7±5.7	1.02±0.73	2.51±0.05
Smallest-First	54.3±4.7	1.21±0.58	**2.42±0.05**
Random	50.1±4.7	1.13±0.57	2.49±0.04

- RL reduces late jobs from **42.7%** (best heuristic) to **37.9%**: ∼11.7% relative improvement.

Future Work

Scalability & Complexity

- Scale to larger multi-GPU clusters and more diverse workload patterns.
- Test robustness under heterogeneous GPU types and mixed job profiles.

Alternative RL Models

- Compare PPO with variants (adaptive KL, clipping strategies, GRPO).
- Explore off-policy RL (e.g., SAC) for higher sample efficiency.
- Try hierarchical RL: high-level repartitioning, low-level placement.

Joint Optimization

- Combine RL-based repartitioning with job scheduling for end-to-end optimization.
- Use bi-level control: RL selects MIG configuration; scheduler assigns jobs.

References I

[1] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). *Learning scheduling algorithms for data processing clusters.* ACM SIGCOMM.

[2] Stable-Baselines3 Documentation. "Tips and Tricks." `https://stable-baselines3.readthedocs.io/`

[3] Andrychowicz, M., Raichuk, A., Stańczyk, P., et al. (2020). *What matters in on-policy reinforcement learning?* arXiv:2006.05990.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.* arXiv:1707.06347.

[5] RL Baselines3 Zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`

[6] Ahmed, Z., Le Roux, N., Norouzi, M., & Schuurmans, D. (2019). *Understanding the impact of entropy on policy optimization.* ICML.
[7] https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai/