Intelligent Job Scheduling for HPC Systems: A Statistical Evaluation of Deep Reinforcement Learning Approaches

Justin M. Cheney
University of the Western Cape
Robert Sobukwe Rd, Bellville,
Cape Town, 7535
South Africa
4323819@myuwc.ac.za

ABSTRACT

Over the past three decades, supercomputers and their workloads have become increasingly complex. Scheduling systems have evolved from traditional heuristics to Deep Reinforcement Learning (DRL) approaches that adapt policies to specific workloads. Though there are several studies that develop various DRL models, no clear consensus exists on the optimal algorithm. This project trains and evaluates representative algorithms from three prominent DRL families: DQN, PPO, and A2C. Statistical testing (Friedman, Nemenyi and Wilcoxon-based confidence intervals) determines whether significant performance differences exist across five industry-standard metrics.

KEYWORDS

High-Performance Computing, Deep Reinforcement Learning, Statistical Analysis, Job Scheduling

1 INTRODUCTION

High-Performance Computing (HPC) systems consist of several software components including package managers, job schedulers, and monitoring tools [7]. Job schedulers manage job queue ordering and handle resource allocation, determining when jobs execute and which computational resources they have access to [21]. As HPC workloads have diversified (from traditional, purely arithmetic-based tasks to machine learning and data-centric tasks) scheduling algorithms have had to evolve to address the increased complexity [28].

Job scheduling has evolved from the simple heuristics (First-Come-First-Served, Shortest Job First) to meta-heuristic optimisation techniques (genetic algorithms, particle swarm optimisation) [21]. As heterogeneous clusters became more prevalent, a new set of challenges were introduced [28].

Traditional heuristics struggle to address multiple objectives across diverse hardware [21]. Deep Reinforcement Learning (DRL) models address this by enabling schedulers to learn adaptive policies from previous behaviour [5, 31]. However, the rapid adoption of DRL raises a question: which algorithm family (policy gradient, value-based, actor-critic) has the best performance for modern HPC workloads? An algorithm that reduces the makespan may fall short in fully utilising the system's resources, therefore justifying a multi-objective evaluation.

Previous studies have compared DRL schedulers against traditional methods [22, 24]. However, systematic statistical evaluation across multiple DRL algorithm families has not been explored. A systematic review of 59 recent DRL scheduling papers (2024-2025) found that 46% used PPO-based approaches, yet fewer than half (12 of 27 papers) of those justified this choice empirically.

This paper addresses this gap through a rigorous comparison of six DRL algorithms: MaskablePPO, MaskableDQN, MaskableA2C, and their vanilla counterparts. These represent three major DRL algorithm families, policy gradient, value-based, and actor-critic, respectively. These algorithms are evaluated on real heterogeneous HPC workloads, reflecting the shift toward accelerator-based systems. Evaluation on real Slurm traces (approximately 84,000 jobs) employs Friedman tests, Nemenyi post-hoc analysis, and Wilcoxon-based confidence analysis [1] to identify statistically significant performance differences across metrics (waiting time, slowdown, turnaround time, and resource utilisation). Section 2 reviews previous work on HPC scheduling and DRL approaches. Section 3 outlines the focus of the research, followed by the proposed methodology and statistical framework in Section 4.

2 LITERATURE REVIEW

HPC job scheduling has evolved from rule-based heuristics to learning-based adaptive techniques. This section reviews the evolution of scheduling algorithms, from traditional approaches (FCFS, SJF, backfilling) to meta-heuristics (genetic algorithms, particle swarm optimisation) to modern DRL methods, and identifies methodological gaps motivating the current study.

2.1 HPC Job Scheduling

Job schedulers fulfil two primary functions within a cluster: resource allocation and queue management [31]. Queue management determines the job execution order based on priority policies (e.g., FIFO, priority queues, fair-share). Resource allocation assigns computational resources (CPU cores, memory, GPUs, storage) to individual jobs while preventing resource contention. Effective scheduling maximises system utilisation, minimises job waiting times, and ensures fairness across users [21].

Schedulers' performance is evaluated using several standard metrics [21]. Makespan measures total time to complete all jobs. Throughput measures number of jobs completed per unit time. Waiting time is the amount of time a job sits in the queue. Slowdown (ratio of turnaround time to execution time) assesses fairness to short jobs. Resource utilisation indicates how efficiently the cluster is being used. Early HPC systems prioritised throughput and makespan [21], while modern deployments aim to balance multiple objectives: performance, cost, energy consumption, and fairness [17].

2.2 Traditional Job Scheduling Techniques

All traditional job scheduling techniques utilise some form of static heuristics to make decisions when it comes to both job queue and

resource allocation. These techniques act as the foundation to the modern counterparts.

2.2.1 Heuristic-Based Scheduling Algorithms

First-Come-First-Served (FCFS) executes jobs in arrival order [25]. Though simple and fair, FCFS suffers from poor resource utilisation due to large jobs blocking the queue. This leaves resources idle even when smaller jobs could execute, a phenomenon known as head-of-line blocking [25]. Additionally, high-priority jobs may experience extended waiting times behind low-priority workloads.

Shortest Job First (SJF) executes jobs based on expected execution time [25]. While SJF provides lower average waiting times compared to FCFS, it suffers from potential starvation for longer jobs. SJF performance depends on accurate job duration estimates; inaccurate estimates result in poor resource utilisation similar to FCFS.

Backfilling improves resource utilisation by scheduling smaller jobs to fill idle resources between larger scheduled jobs [12]. Conservative backfilling preserves the reserved times for high-priority jobs, while aggressive backfilling (e.g., EASY) allows flexible scheduling, causing delays for lower-priority jobs if higher-priority jobs arrive [18].

Priority-based scheduling is employed by production HPC schedulers such as Simple Linux Utility for Resource Management (Slurm) [33] and Portable Batch System (PBS) [9]. Jobs execute based on assigned priority levels, determined by factors including user quotas, job ageing (to prevent starvation), and fair-share policies [10].

Traditional approaches have been shown to be simple and predictable, making them solid options. Backfilling and priority-based scheduling improve resource utilisation and reduce waiting times. However, these methods struggle to handle the complex resource requirements or dynamic workloads, characteristics of modern heterogeneous HPC systems [31].

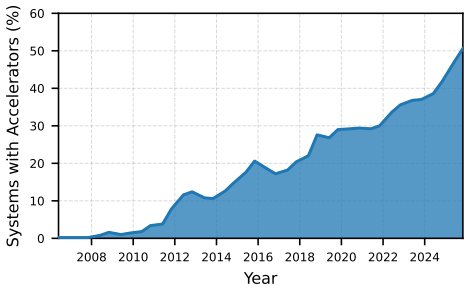

Figure 1: Growth of heterogeneous systems in the TOP500 supercomputers (2006–2025) [28]

2.2.2 Meta-Heuristic and Optimisation Approaches

Meta-heuristic algorithms extend simple heuristics by systematically exploring large solution spaces through bio-inspired search strategies. Genetic algorithms (GA) apply evolutionary principles (selection, crossover, mutation) to optimise resource allocation and job sequencing while avoiding local optima. Particle swarm optimisation (PSO) mimics decentralised swarm behaviour to explore multi-objective solution spaces efficiently. Simulated annealing (SA) achieves faster convergence than GA through probabilistic local search with controlled randomness [21].

Though meta-heuristics outperform the simple heuristics, they remain computationally expensive and sensitive to parameter tuning. More critically, these approaches optimise each workload instance independently; they cannot learn to generalise policies that adapt to changing workload patterns without re-optimisation [21]. This limitation motivates the shift toward learning-based methods that leverage previous experiences, realised through DRL.

2.3 Modern Techniques: Deep Reinforcement Learning for Job Scheduling

Reinforcement learning (RL) addresses the generalisation limitation through trial-and-error policy optimisation. Unlike supervised learning, which requires labelled optimal scheduling decisions, RL learns from environmental feedback while interacting with the cluster, avoiding the need for pre-labelled training data unavailable in practice. DRL extends this to high-dimensional state spaces (100+ job queues and heterogeneous resources) through neural network function approximation.

The sequential decision-making in HPC scheduling is naturally formalised as Markov Decision Processes (MDP) [8], enabling end-to-end learning from cluster state observations to job selection policies.

DRL has been widely explored for cloud and edge scheduling, employing value-based methods (DQN variants), policy gradient approaches (PPO, TRPO), and actor-critic techniques (A3C, A2C). However, in recent works, there tends to be notable concentration around specific algorithms.

PPO-based approaches appear in 46% of recent work (27 of 59 surveyed papers, 2024-2025). Schulman *et al.* [23] attribute this popularity to superior stability and reliability. However, there is a lack of systematic empirical validation of these claims within the HPC job scheduling space. Understanding comparative performance of these algorithms requires examining their foundational principles.

2.3.1 Fundamentals of Deep Reinforcement Learning

Job scheduling is a fully observable environment and therefore can be modelled as MDPs. MDPs are formally defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ [27]. $\mathcal{S}$ represents the state space, both the current job queue and the cluster resource state. $\mathcal{A}$ denotes the action space, job selection decisions. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the likelihood of transitioning from one state to another, typically expressed as $P(s_{t+1} \mid s_t, a_t)$. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns scalar feedback based on scheduling outcomes. The discount function $\gamma \in [0, 1]$, governs the trade-off between immediate and future rewards, with γ closer to 1 emphasising long-term gains and γ closer to 0 focusing on short-term outcomes.

At each discrete time step t, the agent observes state s_t, selects action a_t according to policy $\pi : \mathcal{S} \to \mathcal{A}$, and receives a reward $r_t = R(s_t, a_t)$, then transitions to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The goal is to identify an optimal policy π^* that maximises the expected cumulative reward over time:

$$\sum_{t=0}^{\infty} \gamma^t r_t$$

The three major DRL algorithm families use different approaches to approximate π^*: value-based methods learn $Q(s, a)$, policy-based methods optimise π directly, and actor-critic methods combine both strategies.

Value-based methods approximate the optimal action-value function $Q^*(s, a)$ via the Bellman optimality equation [27]:

$$Q^*(s,a) = \mathbb{E}\Big[R(s,a) + \gamma \max_{a'} Q^*(s',a')\Big]$$

Deep Q-Networks (DQN) use these principles with neural networks to approximate Q, utilising experience replay and target networks for stable training [16]. MaskableDQN extends this by excluding invalid actions from the max operation. This paper focuses on masked variants, while recognising that additional variants (Duelling DQN, Double DQN) address other challenges.

Policy-based methods optimise policy π_θ directly via gradient ascent on expected return over trajectories τ sampled from π_θ [27]. In practice, gradients are computed using advantage estimates:

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) A(s,a)$$

Proximal Policy Optimisation (PPO) stabilises training through a clipped surrogate objective that prevents destructively large policy updates [23]. MaskablePPO applies action masking by zeroing invalid action probabilities before sampling.

Actor-critic methods combine policy optimisation (actor) with value estimation (critic), where the critic estimates either $V(s)$ or $Q(s,a)$ to reduce variance in policy gradient updates [27]. Advantage Actor-Critic (A2C), the synchronous variant of A3C, employs synchronous parallel actors for stable training [15].

Action masking simplifies the agent's decision process by excluding infeasible actions [26]. For value-based methods, invalid actions are masked before arg max selection; for policy-based methods, invalid probabilities are zeroed before normalisation. In HPC scheduling, this prevents selecting jobs whose resource demands exceed current capacity. This paper compares masked variants (MaskableDQN, MaskablePPO, MaskableA2C) to their unmasked variants to evaluate the inherent benefits of masking.

2.3.2 DRL Applications in Job Scheduling

DRL has been widely adopted for scheduling across cloud computing [6, 13], edge systems [29], and HPC clusters. Applications target a range of objectives including makespan, resource utilisation, and fairness. These implementations primarily employ value-based, policy-based, and actor-critic methods, each with distinct reward formulations.

Value-based schedulers employ DQN and variants to learn action-value functions over discrete job selection decisions. Wu *et al.* [32] employed Double DQN ("IDDQN-II") for dual-objective optimisation, but only compared against three meta-heuristics, omitting evaluation against other DRL approaches. Wang *et al.* implemented a three-phase approach "DTPDQN", which used DQN-based algorithms. They achieved approximately 45.6%, 8.9%, and 19.9% improvement on tardiness, utilisation, and makespan, respectively, compared to an unspecified "best current algorithm" [29].

Policy gradient schedulers, particularly PPO, dominate recent work (46% of surveyed papers). Gao *et al.* [6] applied Gated GtrXL with PPO to address challenges posed by high-dimensional decision spaces and complex temporal dependencies found in heterogeneous workloads. Wang *et al.* employed hierarchical PPO for two-level scheduling (job selection & node allocation) [31]. These works cite PPO's stability and sample efficiency as primary motivations, though few provide direct algorithmic comparisons.

Actor-critic methods, particularly A3C, appear in distributed scheduling contexts. Mangalampalli *et al.* employed "MOPTSA3C", a multi-objective A3C-based scheduler within a cloud environment. They claimed that asynchronous parallel actors across multiple cloud environments are able to outperform DQN, A2C, and MOABCQ in makespan, resource utilisation, resource cost, and reliability [13].

Specialised techniques extend base algorithms to complex scenarios. Multi-agent formulations model cooperative agents [14], while transfer learning allows for accelerated adaptation under hardware changes [30].

These studies demonstrate DRL's potential for adaptive scheduling, learning policies that balance objectives without intervention. However, evaluation of algorithms in these studies differ dramatically. Studies provide varying levels of statistical information, employ different performance metrics, workload characteristics, and use a variety of other algorithms as a baseline. Systematic statistical comparison of DRL algorithm families under controlled conditions remains to be extensively explored.

2.3.3 Challenges and Limitations of DRL in HPC Scheduling

DRL's benefits in HPC Scheduling (self-learning and handling of large state spaces) come with challenges and limitations that can affect implementation of these algorithms within production environments. The extensive training required to create a production-grade DRL model requires large-scale datasets and specialised hardware to produce complex states, which results in high computational costs. As HPC workloads grow in complexity, the state and action spaces that represent them expand exponentially, which points to a key issue of scalability. In long-running processes, sparse rewards become a limitation, such as in cases of total makespan or system utilisation, where the true outcome may only be seen after all jobs are complete. The "black box" nature of deep reinforcement learning limits the interpretability of the space, and in mission-critical environments, the inability to understand why specific decisions are made can be a factor that limits wider spread adoption.

3 RESEARCH FOCUS

Recent work rarely evaluates PPO, A2C, and DQN together (only one paper found in all review recent literature), and comparisons are more often outside HPC scheduling, based on synthetic benchmarks, or lacking non-parametric, head-to-head analysis—gaps that motivate the methodological choices that follow. This paper aims to perform a comprehensive statistical comparison to systematically determine the best DRL algorithm family for HPC job scheduling. It investigates whether PPO's dominance is statistically justified, quantifies masking's performance benefits, and measures training costs for production deployment. The primary focus is on the application of DRL algorithms on physical HPC clusters over cloud-based systems. Existing algorithms from the stable-baselines3 and sb3-contrib packages [20] will be used to standardise implementation, and additional code will only be added where applicable.

4 PROPOSED APPROACH

To achieve the research goals outlined in the research focus, the HPCSim outlined in [31] will be used to train models and provide a test environment for multiple objectives. This process includes training the six algorithms on existing HPC hardware and testing them within a simulated environment. CPU- and GPU-bound Slurm traces of approximately 84k jobs will be used in the testing. Once raw data has been collected, statistical testing will be performed, namely, Shapiro-Wilk diagnostics, Wilcoxon

signed-rank (pairwise), Friedman (omnibus), Kendall's W effect size, Nemenyi with CD plots, Page trend test, and Wilcoxon-based non-parametric confidence analysis [1]. After testing is concluded, the results will be used to determine Pareto optimality across waiting time, turnaround time, slowdown, and resource utilisation. To manage the time allocated to each task a Gantt Chart (seen in Figure 2) was used.

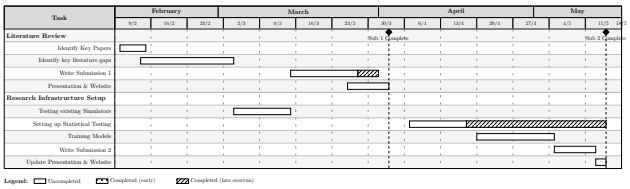

Figure 2: Project Plan for Semester 1

5 METHODOLOGY

This chapter presents the workflow and statistics analysis for an empirical comparative study of six DRL algorithms, and three traditional algorithms across two Slurm workload traces. To ensure rigorous comparison the allocator policy is fixed to "best_fit". A set of five seeds fix randomness at a point across each run ensuring reproducibility. In total this provides an experimental scope of (9 algorithms x 5 seeds x 2 traces), with final assessment on the best model using held-out splits from both the "physical_jobs" and "deeplearn_jobs" traces.

5.1 HPCSim Environment Configuration

HPCSim [31] provides the Gymnasium-based HPC scheduling environment where the observation space comprises the cluster state and a job queue window, and the action space represents job selection decisions. Table 1 summarises the various environment parameters applied uniformly across all experiments.

The "best_fit" allocator is fixed as a controlled variable. Though there are inherent benefits to a topology-aware allocator it's benefits are only found to be truly beneficial for the deep learning workload. Fixing to "best_fit" provides consistent allocation across both workload types, ensuring comparisons reflect scheduling policy differences rather than allocation strategy differences.

5.2 Dataset Preparation

The dataset is taken from Wang *et al.* [31], it is comprised of two Slurm traces of real HPC workloads: a physical job trace (84k jobs) and a deep learning job trace (68k jobs), these are summarised in Table 2. Pandas was used to apply a 70/30 train-test split to both traces, ordered by submission time. Ordering by time prevents data leakage and reflects realistic scheduling conditions where future arrivals are unknown. The test splits are not accessed throughout the run, and are reserved for the final generalisation assessment.

Table 1: Environment Configuration

Parameter	Value	Justification
Allocator	best_fit	Fixed as controlled factor
Window Size	512	Observable queue depth
Tail Size	64	Queue tail buffer
Topology	physical_topology.txt, deeplearn_topology.txt	Cluster layout
Node file	nodes.csv	Node resource definitions
Random jobs	False	Deterministic trace replay

Table 2: Dataset Characteristics

Characteristic	Physical	Deeplearn
Trace	physical_jobs.csv	deeplearn_jobs.csv
Total Jobs	84, 135	68, 720
Train Jobs (70%)	58, 895	48, 104
Test Jobs (30%)	25, 240	20, 616
Temporal Range	37d 11h 42m 8s	382d 5h 21m 49s

5.3 Algorithms and Treatments

The statistical analysis covers six treatments: PPO, A2C, and DQN from stable-baselines3, MaskablePPO from sb3-contrib [20], alongside custom maskable variants, MaskableA2C and MaskableDQN. As defined in Table 3, each base algorithm appears both with and without masking, allowing action masking to be treated as a regularised experimental factor. Three traditional algorithms from HPCSim [31] are included as a baseline.

5.4 Training and Evaluation

Table 4 shows the shared architecture that all agents are trained with, hidden layers, activation function, discount factor, policy being applied to all and replay buffer size being specific to the DQN variants. Five static seeds are used to fix random, numpy, and torch for reproducibility. For all five seeds each algorithm is run on a single train split yielding 30 runs, with two splits, 60 total.

All 60 trained models are evaluated deterministically on their respective dev splits, and using their respective seed. Collecting the primary, secondary and operational metrics, used by [31] and outlined in Table 5. All metric values are stored in a csv file titled "run_log.csv" with a per-run json file that stores all necessary metadata. To standardise the data, the aggregates of all metrics across the five seeds per algorithm are again stored in a set of corresponding csv and an accompanying json metadata file.

Table 3: Algorithm Treatments

Algorithm	Family	Masking	Library
MaskablePPO	Policy Gradient	True	sb3-contrib
MaskableDQN	Value-Based	True	Custom
MaskableA2C	Actor-Critic	True	Custom
PPO	Policy Gradient	False	stable-baselines3
DQN	Value-Based	False	stable-baselines3
A2C	Actor-Critic	False	stable-baselines3
SJF	Traditional	—	HPCSim
LCFS	Traditional	—	HPCSim
FCFS	Traditional	—	HPCSim

Table 4: Hyperparameters

Hyperparameter	Value	Applies To
Hidden Layers	[4096, 2048, 1024]	All
Activation function	tanh	All
Discount factor (γ)	0.99	All
Policy	MultiInputPolicy	All
Replay Buffer Size	1, 000, 000	DQN variants only

Table 5: Performance Metrics

Metric	Definition	Direction	Role
avg_waiting	Mean job wait time	Lower	Primary
avg_slowdown	Mean bounded slowdown	Lower	Primary
max_waiting	Maximum wait time	Lower	Secondary
max_slowdown	Maximum bounded slowdown	Lower	Secondary
avg_turnaround	Mean turnaround time	Lower	Secondary
cpu_utilization	CPU usage ratio	Higher	Secondary
gpu_utilization	GPU usage ratio	Higher	Secondary
episode_reward	Cumulative reward	Higher	Operational
decision_latency_mean_ms	Mean decision time (ms)	Lower	Operational
eval_wall_s	Total evaluation wall time (s)	Lower	Operational
decision_count	Scheduling decisions made	—	Operational

5.5 Statistical Framework

Adapting a framework from Carrasco *et al.* [1], a non-parametric repeated-measures framework is used. Seeds serve as blocks and treatments as algorithm conditions. Given n=5 seeds, normality cannot be assumed therefore rank-based methods are most appropriate.

Shapiro-Wilk diagnostics are run per treatment to determine whether the normality of the distribution can be rejected. The Friedman test is used to detect differences in treatments across multiple related groups, and remains the standard omnibus test in [1]; aligned-ranks is noted as a higher-power alternative for low-k settings. Friedman serves as the omnibus test per metric per trace ($\alpha = 0.05$). Kendall's W is computed as the effect size for every Friedman result ($W = \frac{\chi^2_w}{N(k-1)}$); interpreted as slight agreement (≥0.0), fair (≥0.2), moderate (≥0.4), substantial (≥0.6), and almost perfect agreement (≥0.8) following [11]. Effect size is reported alongside p-values to avoid sample-size-driven conclusions.

Where Friedman is significant, Nemenyi post-hoc pairwise comparisons are performed, with Critical Difference (CD) plots used to visualise non-significant rank groupings [2].

Wilcoxon-based non-parametric confidence intervals (CI) are reported alongside confidence curves, providing Uncertainty and effect-size context for pairwise differences [1]. Convergence comparisons use the Page trend test for paired algorithms [3].

A single best algorithm is Pareto-selected (primary & secondary), ties by avg_waiting → avg_slowdown → cpu_utilization, then evaluated on holdout/test splits across the same seeds; results are descriptive only.

Table 6: Statistical Analysis Summary

Stage	Method	Condition	Purpose
Normality	Shapiro-Wilk	Always	Diagnostics only
Pairwise	Wilcoxon signed-rank	Two algorithms	Median difference
Omnibus	Friedman	Always	Group differences
Effect Size	Kendall's W	Always	Magnitude
Post-hoc	Nemenyi	Friedman significant	Pairwise comparison
Uncertainty	Wilcoxon CI + confidence curves	Always	Median intervals
Convergence	Page trend test	Paired algorithms	Trend comparison
Visualisation	CD Diagram	Friedman significant	Rank comparison

5.6 Reproducibility and Workflow

To ensure a reproducible environment and workflow, a triple-layer stack is employed. Git tracks source history with commit hashes and manifest sha256 fingerprints in all output metadata. Nix flake with a nixos-25.05 channel pin [4] creates a fully declarative environment where every dependency, including the Python interpreter, is content-addressed (not simply version-pinned). Apptainer containerises the environment for cluster execution. The end-to-end pipeline, seen in Figure 3, is orchestrated by Snakemake 9.4.3 [19], with config-driven separation of smoke test and production runs, interacting with Slurm and Apptainer containers.

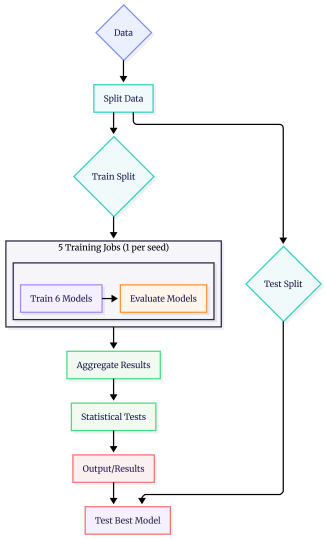

Figure 3: Snakemake Workflow

6 CONCLUSION

This paper addresses the lack of systematic validation in DRL-based HPC job scheduling research. Through rigorous statistical comparison across DRL algorithm families on real heterogeneous workloads, this research aims to determine whether PPO's dominance reflects genuine performance or publication bias. The findings will guide future practitioners in algorithm selection and inform future research directions in resource-constrained DRL scheduler deployment.

Using HPCSim with a reproducible and scalable Snakemake and Nix pipeline, enables systematic comparison of DRL algorithm families for HPC job selection. A smoke-test confirms a functioning pipeline, full cluster evaluation will provide definitive results.

7 ACKNOWLEDGEMENT

The author acknowledges the use of OpenCode to assist with grammatical editing, sentence restructuring, and minor code refinement. All AI-assisted suggestions were reviewed, revised, and verified by the author, who takes full responsibility for the final content and code.

REFERENCES

[1] Jacínto Carrasco, Salvador García, M Mar Rueda, Swagatam Das, and Francisco Herrera. 2020. Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm and Evolutionary Computation* 54, (2020), 100665.

[2] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7, (2006), 1–30.

[3] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation* 1, 1 (2011), 3–18. https://doi.org/10.1016/j.swevo.2011.02.002

[4] Eelco Dolstra, Merijn de Jonge, and Eelco Visser. 2004. Nix: A Safe and Policy-Free System for Software Deployment. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA)*, 2004. 79–92.

[5] Valerio Firmano. 2024. Deep Reinforcement Learning for Dynamic Job-Shop Scheduling in High-Utilization Systems. Master's Thesis.

[6] Xu Gao, Hang Dong, Lianji Zhang, Yibo Wang, Xianliang Yang, and Zhenyu Li. 2025. Self-Attention Mechanisms in HPC Job Scheduling: A Novel Framework Combining Gated Transformers and Enhanced PPO. *Applied Sciences (Switzerland)* 15, 16 (2025). https://doi.org/10.3390/app15168928

[7] I. Gitler, C. J. Barrios Hernández, and E. Meneses. 2022. *High Performance Computing: 8th Latin American Conference, CARLA 2021, Guadalajara, Mexico, October 6–8, 2021, Revised Selected Papers*. Springer International Publishing.

[8] Yan Gu, Zhaoze Liu, Shuhong Dai, Cong Liu, Ying Wang, Shen Wang, Georgios Theodoropoulos, and Long Cheng. 2025. Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review. Retrieved from http://arxiv.org/abs/2501.01007

[9] Robert L. Henderson. 1995. Job Scheduling Under the Portable Batch System. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'95) (Lecture Notes in Computer Science)*, 1995. Springer, 279–294. https://doi.org/10.1007/3-540-60153-8_34

[10] G. J. Henry. 1984. The UNIX System: The Fair Share Scheduler. *AT&T Bell Laboratories Technical Journal* 63, 8 (October 1984), 1845–1857. https://doi.org/10.1002/j.1538-7305.1984.tb00068.x

[11] J. R. Landis and G. G. Koch. 1977. "An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers". *Biometrics* 33, 2 (June 1977), 363. https://doi.org/10.2307/2529786

[12] David A. Lifka. 1995. The ANL/IBM SP Scheduling System. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'95) (Lecture Notes in Computer Science)*, 1995. Springer, 295–303. https://doi.org/10.1007/3-540-60153-8_35

[13] S. Sudheer Mangalampalli, Ganesh Reddy Karri, Sachi Nandan Mohanty, Shahid Ali, Muhammad Ijaz Khan, Sherzod Abdullaev, and Salman A. Alqahtani. 2024. Multi-Objective Prioritized Task Scheduler Using Improved Asynchronous Advantage Actor Critic (a3c) Algorithm in Multi Cloud Environment. *IEEE Access* 12, (2024), 11354–11377. https://doi.org/10.1109/ACCESS.2024.3355092

[14] Fady Nashat Manhary, Marghny H. Mohamed, and Mamdouh Farouk. 2025. A scalable machine learning strategy for resource allocation in database. *Scientific Reports* 15, 1 (2025). https://doi.org/10.1038/s41598-025-14962-5

[15] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016) (Proceedings of Machine Learning Research)*, 2016. PMLR, 1928–1937. Retrieved from http://proceedings.mlr.press/v48/mniha16.html

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (February 2015), 529–533. https://doi.org/10.1038/nature14236

[17] Seraj Al Mahmud Mostafa, Aravind Mohan, and Jianwu Wang. 2025. SLA-MORL: SLA-Aware Multi-Objective Reinforcement Learning for HPC Resource Optimization. Retrieved from http://arxiv.org/abs/2508.03509

[18] Ahuva W. Mu'alem and Dror G. Feitelson. 2001. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Transactions on Parallel and Distributed Systems* 12, 6 (June 2001), 529–543. https://doi.org/10.1109/71.932708

[19] F Mölder, KP Jablonski, B Letcher, MB Hall, PC van Dyken, CH Tomkins-Tinch, V Sochat, J Forster, FG Vieira, C Meesters, S Lee, SO Twardziok, A Kanitz, J VanCampen, V Malladi, A Wilm, M Holtgrewe, S Rahmann, S Nahnsen, and J Köster. 2025. Sustainable data analysis with Snakemake [version 3; peer review: 2 approved]. *F1000Research* 10, 33 (2025). https://doi.org/10.12688/f1000research.29032.3

[20] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. Retrieved from http://jmlr.org/papers/v22/20-1364.html

[21] Yousef Sanjalawe, Salam Al-E'mari, Salam Fraihat, and Sharif Makhadmeh. 2025. AI-driven job scheduling in cloud computing: a comprehensive review. *Artificial Intelligence Review* 58, 7 (July 2025). https://doi.org/10.1007/s10462-025-11208-8

[22] Prasanna Sankaran, Shiva Kiran Lingishetty, and Mrinal Kumar. 2025. Leveraging Reinforcement Learning for Efficient Task Scheduling in Multi-Cloud Environments. *International Journal of Innovative Research in Computer Science and Technology* 13, 2 (April 2025), 35–41. https://doi.org/10.55524/ijircst.2025.13.2.6

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. Retrieved from http://arxiv.org/abs/1707.06347

[24] Aasish Kumar Sharma and Julian Kunkel. 2025. GrapheonRL: A Graph Neural Network and Reinforcement Learning Framework for Constraint and Data-Aware Workflow Mapping and Scheduling in Heterogeneous HPC Systems. Retrieved from http://arxiv.org/abs/2506.00260

[25] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2018. *Operating System Concepts* (10th ed.). Wiley, Hoboken, NJ.

[26] Roland Stolz, Hanna Krasowski, Jakob Thumm, Michael Eichelbeck, Philipp Gassert, and Matthias Althoff. 2024. Excluding the Irrelevant: Focusing Reinforcement Learning through Continuous Action Masking. In *Advances in Neural Information Processing Systems*, 2024. Curran Associates, Inc., 95067–95094. https://doi.org/10.52202/079017-3013

[27] R.S. Sutton and A.G. Barto. 2018. *Reinforcement Learning, second edition: An Introduction*. MIT Press. Retrieved from https://books.google.co.za/books?id=uWV0DwAAQBAJ

[28] Top500. 2026. TOP500 List Statistics.

[29] Hao Wang, Xu Chen, Kunpeng Zhang, and Robert H Smith. 2024. *Dynamic Multi-objective Flexible Job Shop Scheduling: A Three-phase Deep Reinforcement Learning Approach*. Retrieved from https://ssrn.com/abstract=5312558

[30] Lingfei Wang, Maria A. Rodriguez, and Nir Lipovetzky. 2025. MetaPilot: A DRL-based controller for dynamic adaptation to shifting scheduling objectives in HPC systems. *Future Generation Computer Systems* 178, (2025), 108269. https://doi.org/10.1016/j.future.2025.108269

[31] Lingfei Wang, Maria A. Rodriguez, and Nir Lipovetzky. 2025. Optimizing HPC scheduling: a hierarchical reinforcement learning approach for intelligent job selection and allocation. *Journal of Supercomputing* 81, 8 (June 2025). https://doi.org/10.1007/s11227-025-07396-3

[32] Rui Wu, Jianxin Zheng, and Xiyan Yin. 2025. Dynamic Scheduling for Multi-Objective Flexible Job Shops with Machine Breakdown by Deep Reinforcement Learning. *Processes* 13, 4 (2025). https://doi.org/10.3390/pr13041246

[33] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing (JSSPP 2003) (Lecture Notes in Computer Science)*, 2003. Springer, 44–60. https://doi.org/10.1007/10968987_3