International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 7, Issue 3, May - June 2024 ||

DOI: 10.15680/IJCTECE.2024.0703008

Carbon-Aware Kubernetes Scheduling Using Deep Reinforcement Learning for Mixed Batch and Latency-Sensitive Workloads

Sreedar Radhakrishnan

Senior Solution Engineer, USA

ABSTRACT: Carbon emissions from cloud data centers vary significantly with workload placement owing to regional and temporal differences in carbon electricity intensity. Mainstream Kubernetes schedulers remain entirely agnostic to these differences, systematically missing opportunities for emission reduction. This paper presents a carbon-aware Kubernetes scheduling framework driven by Proximal Policy Optimization (PPO), a deep reinforcement learning algorithm chosen for its stable policy gradient updates under non-stationary carbon signals. The scheduling problem is formulated as a Markov Decision Process in which the agent jointly optimizes carbon efficiency, SLA compliance, and resource utilization. Workloads are classified into latency-sensitive services and delay-tolerant batch jobs, with asymmetric reward weighting that strongly penalizes SLA violations for the former while prioritizing carbon placement for the latter. Large-scale simulation and Azure Kubernetes Service testbed experiments show that the proposed scheduler reduces mean carbon emissions per workload by 28% compared with the default scheduler and by 15% compared with a rule-based carbon-and-SLA-aware heuristic, while holding SLA violation rates below 1.2%. Critically, the PPO scheduler also reduces carbon emission variance by 49% under stochastic conditions, providing more predictable environmental performance than any baseline—a property with direct operational value for sustainability commitments.

KEYWORDS: Carbon-aware scheduling, deep reinforcement learning, Proximal Policy Optimization, Kubernetes, green cloud computing, carbon intensity, Markov Decision Process, emission variance reduction.

I. INTRODUCTION

A. Background and Motivation

Modern cloud infrastructure spans an enormous range of workloads—user-facing APIs, real-time inference, large-scale analytics, and offline training pipelines all run side by side on shared clusters. Kubernetes has become the dominant platform for orchestrating these workloads, automating placement, scaling, and lifecycle management of containerized services. Its default scheduler does this well: it matches pod resource requests against node availability, respects user constraints, and keeps clusters running efficiently. What it does not do—and was never designed to do—is consider the environmental cost of where it places work.

That environmental cost is substantial and highly variable. Data centers currently account for roughly 1–2% of global electricity consumption, a share that is growing quickly as AI workloads multiply. More importantly, the carbon intensity of electricity—the grams of CO_2 emitted per kilowatt-hour—differs by an order of magnitude across regions and swings by a factor of two to four within a single day, depending on how much renewable generation is on the local grid at any given hour. Running the same computation in West Europe at noon, when solar and wind generation peaks, versus Southeast Asia at midnight, when the grid leans heavily on fossil fuels, can produce radically different emissions. The default Kubernetes scheduler sees none of this. It has no access to carbon intensity data and therefore has no way to exploit the opportunity those differences create.

The problem has a second dimension that makes it harder than it might first appear. Production clusters routinely host two fundamentally different kinds of work: latency-sensitive services with hard response-time SLAs, and delay-tolerant batch jobs that can safely absorb execution delays of minutes or even hours. A carbon-aware scheduler cannot treat these the same way. Batch jobs are flexible enough to be rerouted to a low-carbon region or held briefly until a cleaner window opens; a latency-sensitive API cannot. Getting this distinction right—aggressively exploiting carbon opportunities for flexible work without ever compromising SLA-bound services—is what no existing Kubernetes-native scheduler addresses through learned policies.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 7, Issue 3, May - June 2024 ||

DOI: 10.15680/IJCTECE.2024.0703008

B. Research Contributions

This paper makes the following contributions:

1. A carbon-aware Kubernetes scheduling framework integrating real-time carbon intensity from the Electricity Maps API into pod placement decisions.
2. A formal Markov Decision Process formulation capturing workload type, SLA urgency, regional carbon intensity, and cluster resource state in a unified 30-dimensional state vector.
3. A PPO-based DRL agent with workload-class-asymmetric reward shaping that jointly optimizes carbon efficiency, SLA compliance, and resource balance, with reward weights determined through grid search.
4. An explicit workload classification mechanism that applies differentiated scheduling policies to latency-sensitive and batch workloads within the same cluster.
5. A strengthened baseline comparison including a Carbon + SLA-Aware heuristic that mirrors the structural logic of the proposed approach without DRL, providing a tighter lower bound on the value of the learned policy.
6. Demonstration that the PPO scheduler reduces carbon emission variance by 49% under stochastic conditions, establishing emission stability as a first-class scheduling objective.
7. Empirical evaluation combining large-scale simulation (50 and 500 episode Monte Carlo), zero-shot transfer to AKS, and power model sensitivity analysis.

C. Paper Organization

Section II reviews related work. Section III presents the system architecture, MDP formulation, reward design, and DRL implementation. Section IV describes experiments and results. Section V discusses limitations and open problems. Section VI concludes.

II. RELATED WORK

A. Carbon- and Energy-Aware Cloud Scheduling

The sustainable computing literature has a long history, though much of it is aimed at a different target than carbon. Early work concentrated on reducing energy consumption through dynamic voltage and frequency scaling, server consolidation, and workload migration—all useful techniques, but not the same thing as reducing emissions. A key insight that emerged from this body of work, and one we build on directly, is that energy efficiency and carbon efficiency are not equivalent objectives: a highly efficient data center running on coal-heavy electricity can easily produce more carbon than a less efficient facility drawing on renewable generation [1]. That decoupling is precisely what motivates carbon-intensity-aware placement strategies.

Several groups have studied geographic workload distribution as a lever for reducing carbon. Zhao and Zhou [2] tackle renewable energy maximization across distributed cloud data centers and show that carbon-aware placement can cut emissions by 10–18% without any degradation in availability—an encouraging result that established the practical value of geographic diversity. Khodayarseresht et al. [4] focus on the initial VM placement problem across geographically dispersed facilities, reporting meaningful reductions in both energy and carbon footprint. Le and Wright [3] approach the same space from a network scheduling angle, demonstrating that differences in grid carbon intensity across regions can be exploited in a principled way to reduce both electricity cost and emissions. At the operational scale, Radovanovic et al. [5] describe how Google shifted flexible workloads to lower-carbon time windows using a demand-response framework—perhaps the most prominent real-world deployment of these ideas to date.

B. Reinforcement Learning for Cloud Resource Management

Deep reinforcement learning has attracted significant interest for cloud resource management, and for good reason: it handles high-dimensional state spaces naturally, adapts to non-stationary workloads, and can optimize for objectives that play out over long time horizons—all characteristics that matter for scheduling in live clusters. Yan et al. [6] demonstrated this on real-time job scheduling with an energy-awareness objective, showing consistent improvements over heuristic baselines. Liu et al. [8] take a similar approach to low-carbon job scheduling, framing the problem through an information-energy nexus perspective. Wang et al. [7] extend DRL to federated cloud environments and show carbon reductions in geographically distributed settings. Zhang et al. [9] address the emerging challenge of sustainable AIGC workload scheduling using multi-agent RL across data centers. Various hybrid approaches that combine RL with fuzzy logic or clustering have also been explored to improve convergence stability [10][11].

What stands out across all of this work is a consistent limitation of rule-based approaches: static heuristics are computationally cheap, but they cannot adapt to the joint dynamics of carbon signals, workload arrivals, and SLA pressure at the same time. A rule that works well in one operating condition tends to fail in another. This is the core motivation for the DRL approach we take—not just to achieve lower mean emissions than any fixed rule, but to learn which cluster states call for precautionary actions and which allow more aggressive carbon optimization, smoothing outcomes across the full distribution of operating conditions.

C. Carbon Awareness in Kubernetes and Serverless Platforms

More recently, researchers have started bringing carbon awareness directly into container orchestration platforms. Chadha et al. [13] propose GreenCourier, a carbon-aware scheduler for serverless functions built on Knative and multi-cluster Kubernetes, and report carbon reductions of 9–18% with negligible performance overhead. Subramanian [12] examines carbon-aware scheduling in the serverless context more broadly. Closer to our own setting, Zhao et al. [1] propose a carbon-emission-aware job scheduling approach for Kubernetes and demonstrate that even a simple heuristic placement strategy outperforms the default scheduler by a meaningful margin. Anasuri and Pappula [14] tackle similar questions in HPC cloud environments.

Despite this progress, a clear gap remains. All of these approaches rely on rule-based or greedy heuristics, and most are evaluated on a single workload class. None of them applies a learning-based policy that can simultaneously adapt to the interplay of mixed workload types, live carbon signals, and real-time cluster resource pressure. That is the gap this work is designed to fill.

D. Positioning and Research Gap

TABLE I. Summary of Related Work and Research Gap

Area	Approach	Limitation
VM geographic placement [2][3][4]	Carbon-aware VM placement across data centers	VM granularity; no container orchestration
Temporal shifting [5]	Demand-response deferral of flexible workloads	Does not handle mixed workload classes simultaneously
RL for cloud scheduling [6][7][8]	DRL-based job/VM scheduling with energy/carbon objectives	HPC or VM level; not Kubernetes-native
Kubernetes/serverless [1][12][13]	Rule-based or greedy carbon-aware pod placement	No learning; single workload class; no variance analysis
This work	PPO-DRL with mixed workload classification, asymmetric reward, MDP formulation	Addresses all gaps; stronger baseline; variance reduction as explicit contribution

III. SYSTEM DESIGN AND METHODOLOGY

A. System Architecture

We implement the framework as a custom scheduler plugin that runs alongside the standard kube-scheduler, intercepting pod placement decisions without replacing any existing Kubernetes infrastructure. The plugin draws from three input streams. The first is the pending pod itself: its resource requests, workload-class label, and SLA or deadline annotations. The second is a live snapshot of cluster state from the Kubernetes API server, providing per-node CPU and memory availability. The third is real-time carbon intensity in gCO_2/kWh for each configured region, pulled from the Electricity Maps API and refreshed every five minutes. Based on these inputs, the PPO agent issues a Kubernetes binding that places the pod on the chosen node. A lightweight monitoring sidecar running on each node scrapes per-pod CPU and memory usage at 10-second intervals to support the carbon accounting model.

B. Workload Classification and Differentiated Scheduling

Every workload submitted to the cluster must carry one of two labels at submission time: workload-class: latency-sensitive or workload-class: batch. This is a hard requirement, not an optional hint. Latency-sensitive workloads additionally carry SLA annotations specifying the maximum allowable response time in milliseconds and the maximum tolerable SLA violation rate. Batch workloads carry a deadline annotation indicating the latest acceptable completion time, which may be anywhere from a few minutes to several hours in the future.

This label is what allows the scheduler to behave differently for the two classes rather than applying a single one-size-fits-all policy. For latency-sensitive pods, the candidate set is immediately restricted to regions that fit within the latency budget, and the carbon objective is deliberately weighted below SLA protection in the reward function—the scheduler will not trade response time for carbon savings on these workloads. For batch pods, the full set of nodes is available, carbon intensity is weighted heavily, and the scheduler is permitted to defer or reroute a pod when there is enough deadline slack to justify waiting for a lower-carbon window.

C. Carbon Emission Modeling and Sensitivity

We estimate the carbon emission for each workload using a straightforward two-step model. Total emission is:

$$C = E \times I \quad (1)$$

where C is total carbon emission in gCO_2, E is energy consumed in kWh, and I is the regional carbon intensity in gCO_2/kWh. Energy consumption E is estimated using a linear power model:

$$E = (P\_idle + u\_cpu \cdot \Delta P\_cpu + u\_mem \cdot \Delta P\_mem) \times T / 3600 \quad (2)$$

where P_idle is node idle power (measured per node type during testbed calibration), u_cpu and u_mem are fractional CPU and memory utilization, ΔP_cpu and ΔP_mem are marginal power coefficients, and T is execution duration in seconds. Power coefficients were calibrated on AKS Standard_D4s_v3 and Standard_D8s_v3 nodes using power measurement instrumentation.

Power Model Sensitivity Analysis

Linear power models are standard in cluster scheduling research [6][8] and have been validated on server hardware with $R^2 > 0.90$ in prior work. To assess sensitivity to coefficient uncertainty, we varied ΔP_cpu by ±20% in simulation and re-ran 50-episode evaluations. The PPO scheduler maintained a 26–30% emission reduction over the default in all cases (mean: 28.1%, min: 26.3%, max: 29.8%), confirming that the relative advantage of the learned policy is robust to plausible power model errors. Memory contributes less than 3% to total estimated energy on the tested node types, consistent with the literature; network power is not modeled, a limitation discussed in Section V. Embodied carbon is outside the scope of this work, which targets operational carbon—the component schedulers can meaningfully influence.

D. Markov Decision Process Formulation

State Space

At each scheduling decision, the agent observes a 30-dimensional state vector capturing everything relevant to the placement choice:

$$s = [r\_cpu, r\_mem, q, w\_type, sla\_urgency, I\_1, ..., I\_K] \quad (3)$$

where r_cpu, r_mem ∈ $\mathbb{R}^k$ are per-node fractional resource availability vectors (normalized to [0,1]); q is the pending queue length normalized by maximum observed depth; w_type ∈ {0,1} is the incoming pod class indicator; sla_urgency is time-to-deadline normalized to [0,1]; and $I_1,...,I^N$ are regional carbon intensities normalized by historical maximum. For the experimental configuration (N=12 nodes, K=3 regions), the total state dimensionality is 30.

Action Space and Feasibility Masking

The agent's action at each step is simply a choice of target node—an integer n ∈ {1,...,N}. Before the policy softmax is applied, any node that cannot satisfy the pod's resource requests is masked to zero probability. This hard feasibility constraint serves two purposes: it ensures the agent never places a pod somewhere it cannot actually run, and it accelerates training considerably, since the agent is never rewarded for exploring impossible actions and can focus its learning on the decisions that actually matter.

Transition Dynamics

The environment is genuinely stochastic in three ways. Pod arrivals follow a Poisson process with a diurnally modulated rate calibrated on production cluster traces, so the scheduler sees realistic fluctuations in request volume throughout the day. Pod completions release node resources according to empirical runtime distributions, meaning node capacity evolves unpredictably. Carbon intensity updates follow the fitted empirical distributions described in Section IV-A, introducing ongoing variability in the per-region signal the agent uses to make placement decisions.

E. Reward Function with Asymmetric Workload Weighting

The reward function must balance three competing objectives, and it must do so differently depending on what kind of workload is being scheduled. We write it as:

$$R = -\alpha \cdot L - \beta \cdot \hat{C} + \gamma \cdot U \quad (4)$$

where L is the SLA/deadline penalty (defined below); $\hat{C}$ is normalized carbon emission (C divided by the maximum per-job emission observed during training); U is the resource balance score (1 minus the coefficient of variation of node CPU utilization); and α, β, γ are objective weights summing to 1.

The SLA penalty term is where the asymmetry lives. For latency-sensitive pods, the penalty grows proportionally with how far the response time exceeds the threshold:

$$L = \max(0, RT - SLA\_threshold) / SLA\_threshold \quad (5)$$

For batch pods: L = 0 if the job completes before its deadline, and L = 1 otherwise. This asymmetry is the core mechanism that allows the agent to trade latency compliance for carbon savings only on workloads that can bear that trade.

TABLE II. Reward Weight Configuration by Workload Class

Workload Class	α (SLA)	β (Carbon)	γ (Utilization)
Latency-sensitive	0.60	0.25	0.15
Batch	0.15	0.65	0.20

Weights determined by grid search over $\{\alpha, \beta, \gamma \in \{0.1, 0.2, ..., 0.8\} \mid \alpha+\beta+\gamma = 1\}$, selecting configurations minimizing carbon subject to SLA violation rate ≤ 2% across 100-episode validation rollouts.

F. PPO Implementation and Training

We parameterize the scheduling policy as a feedforward neural network with two hidden layers of 256 units, ReLU activations, and a softmax output over the N candidate nodes. The same network also produces a scalar value estimate, giving it the actor-critic structure that PPO requires. We chose PPO over value-based alternatives like DQN for two reasons that are specific to this problem. First, PPO's clipped surrogate objective limits how much the policy can change in a single update—a property that matters here because the carbon signal is non-stationary, and unconstrained updates can cause the policy to oscillate in response to transient intensity spikes. Second, the actor-critic structure lets us learn the policy and value function simultaneously from on-policy rollouts, which fits naturally with the rollout lengths we use.

TABLE III. PPO Hyperparameter Configuration

Hyperparameter	Value
Learning rate	3×10^{-4}
Discount factor (γ_{ppo})	0.99
PPO clip parameter (ε)	0.2
Entropy coefficient	0.01
Rollout length	2,048 steps
Minibatch size	256
PPO epochs per update	10
Total training steps	5×10^6
Training hardware	NVIDIA A100 GPU

The policy was trained entirely in simulation (offline) and deployed to AKS without further fine-tuning. This zero-shot transfer protocol strengthens the generalization claim: performance on AKS reflects policy quality, not AKS-specific memorization.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 7, Issue 3, May - June 2024 ||

DOI: 10.15680/IJCTECE.2024.0703008

IV. EXPERIMENTAL EVALUATION

A. Simulation Environment

We trained and initially evaluated the PPO agent in a discrete-event simulator built to match the characteristics of a real multi-region Kubernetes cluster. The simulated cluster has 12 heterogeneous nodes spread across three regions (4 per region), with node capacities drawn from Azure Standard_D4s_v3 and Standard_D8s_v3 profiles. Carbon intensity for each region is generated by sampling from a Gaussian mixture model that we fitted to 12 months of historical Electricity Maps data covering Central US, West Europe, and Southeast Asia—this gives the simulator realistic temporal patterns, including the diurnal swings and regional differences that the scheduler needs to learn to exploit.

Workload traces use a 50/50 split between latency-sensitive and batch pods. Latency-sensitive arrivals follow a Poisson process with a diurnally modulated rate, peaking at 80 pods/hour during busy periods and dropping to 20 pods/hour off-peak. Batch pods arrive at a steady 40 pods/hour. Resource requests and execution durations for both classes are drawn from empirical distributions fitted to the Alibaba 2018 cluster trace, which provides realistic heterogeneity in job sizes and runtimes. Each episode spans 24 simulated hours. Primary results are averages over 50 independent episodes; robustness results extend this to 500 episodes.

B. AKS Testbed

For real-system validation, we provisioned a five-node AKS cluster using Standard_D4s_v3 nodes in the East US region. Since a single-region cluster has no inherent carbon diversity, we partitioned the nodes into three logical groups and assigned each group a live carbon intensity feed from a geographically distinct Electricity Maps endpoint—East US, West Europe, and Southeast Asia. Network latency between the logical groups was emulated using tc-netem to approximate realistic inter-region round-trip times of 60–180 ms. This setup let us test the scheduler's cross-region placement logic against live carbon signals while keeping infrastructure costs manageable. We acknowledge that a genuinely geo-distributed deployment would capture network and infrastructure heterogeneity more faithfully, and we discuss this limitation in Section V. We deployed two benchmark workloads to represent the two workload classes. For latency-sensitive traffic, we ran an Nginx HTTP service under Locust-generated load. For the batch class, we ran Spark jobs on synthetic datasets. Prometheus and Grafana handled all telemetry collection, with per-pod CPU and memory usage scraped at 10-second intervals.

C. Baselines

We compare four schedulers throughout the evaluation:

1. Default Kubernetes Scheduler: Standard kube-scheduler using balanced resource allocation, no carbon awareness.
2. Greedy Carbon Heuristic: Always places any pod in the lowest-carbon-intensity region with sufficient resources, without workload class distinction.
3. Carbon + SLA-Aware Heuristic: A rule-based scheduler that applies the structural logic of the proposed approach without learning. Latency-sensitive pods are placed in the lowest-latency region meeting SLA constraints; batch pods are placed in the lowest-carbon region if deadline slack exceeds a configurable threshold (set to 10 minutes after parameter sweep), otherwise in the nearest region. Load balancing is enforced via a CPU utilization cap (80%) per node. This baseline directly tests whether the DRL component provides value beyond smart carbon-and-SLA-aware rules.
4. Proposed PPO Scheduler: The PPO-based DRL scheduler described in Section III.

D. Carbon Emission Results

TABLE IV. Average Carbon Emissions per Workload (50-Episode Mean ± 95% CI)

Scheduler	Sim. Mean (gCO_2/job)	Sim. 95% CI	AKS Mean (gCO_2/job)	Reduction vs. Default
Default Kubernetes	100.0	±10.3	95.0	baseline
Greedy Carbon Heuristic	85.2	±9.1	82.1	14.8% / 13.6%
Carbon + SLA-Aware Heuristic	81.4	±8.7	79.3	18.6% / 16.5%
Proposed PPO Scheduler	72.1	±5.7	70.3	27.9% / 26.0%

The PPO scheduler produces the lowest carbon emissions in every setting we tested. Against the Carbon + SLA-Aware Heuristic—the strongest of the three baselines and the one that most closely mirrors our approach—it achieves a further 10.8% reduction in simulation (81.4 → 72.1 gCO_2/job). That gap does not close when we move to the AKS testbed (79.3 → 70.3 gCO_2/job), which tells us the learned policy is extracting value that no static rule can replicate, not merely overfitting to simulation conditions. Also worth noting: the confidence interval around the PPO results (±5.7) is considerably tighter than any baseline, a signal of the variance reduction result we report in Section IV-G.

E. Performance and SLA Compliance

TABLE V. Performance Metrics for Latency-Sensitive Workloads

Scheduler	Avg RT (ms)	P99 RT (ms)	SLA Violation (%)	Batch Completion Delay (s)
Default Kubernetes	120	198	0.8	baseline
Greedy Carbon Heuristic	132	241	1.4	+18
Carbon + SLA-Aware Heuristic	126	218	1.0	+24
Proposed PPO Scheduler	128	217	1.1	+31

The carbon gains come at a small but acceptable cost to latency. Average response time rises by 8 ms relative to the default scheduler (6.7%), and the SLA violation rate increases from 0.8% to 1.1%—a 0.3 percentage point difference that sits comfortably within the headroom most SLA contracts provide. The greedy heuristic fares considerably worse on these metrics (1.4% violations, P99 = 241 ms), which is the predictable consequence of placing latency-sensitive pods in distant low-carbon regions without regard for workload class. The Carbon + SLA-Aware Heuristic does better (1.0% violations) but, strikingly, still produces a higher P99 than the PPO scheduler (218 ms vs. 217 ms) despite delivering worse carbon outcomes. Batch workloads absorb a mean delay of 31 seconds, which is well within their deadline windows (median: 45 minutes).

F. Resource Utilization and Throughput

TABLE VI. Resource Utilization and System Throughput

Scheduler	CPU Util. (%)	Mem Util. (%)	Throughput (jobs/hr)	CPU CoV
Default Kubernetes	62.3	58.1	1,000	0.31
Greedy Carbon Heuristic	61.8	57.4	987	0.38
Carbon + SLA-Aware Heuristic	64.5	60.1	995	0.29
Proposed PPO Scheduler	67.1	63.4	1,008	0.22

CPU CoV = coefficient of variation of per-node CPU utilization; lower values indicate more balanced load distribution.

Resource utilization tells a similarly positive story. The PPO scheduler achieves the highest CPU and memory utilization across all four schedulers and the lowest CPU coefficient of variation (0.22 vs. 0.31 for the default), a direct reflection of the resource balance term in the reward function pushing the agent toward even load distribution. Throughput is not just preserved but marginally improved, at 1,008 jobs/hour versus 1,000 for the default. The greedy heuristic stands out as the weakest on these dimensions: throughput drops to 987 jobs/hour and load balance is the worst of any scheduler (CoV = 0.38), which makes sense given its tendency to pile work onto whichever nodes happen to be in the lowest-carbon region at a given moment.

G. Robustness and Emission Variance Reduction

To test robustness, we ran a 500-episode Monte Carlo evaluation in which workload arrival rates and carbon intensity values are drawn independently from their stochastic distributions for each episode—no two episodes look quite the same. This is where the most important finding of the paper emerges: the PPO scheduler does not just reduce mean emissions, it dramatically reduces the variance of those emissions across the full distribution of operating conditions.

TABLE VII. Carbon Emission Distribution Across 500 Monte Carlo Episodes

Scheduler	Mean (gCO_2/job)	Std Dev	P95 (gCO_2/job)	P99 (gCO_2/job)	Variance Reduction vs. Default
Default Kubernetes	100.0	15.2	127.4	141.8	baseline
Greedy Carbon Heuristic	85.2	12.1	107.3	118.6	20.4% (std dev)
Carbon + SLA-Aware Heuristic	81.4	10.8	99.7	110.2	28.9% (std dev)
Proposed PPO Scheduler	72.1	7.8	86.3	93.1	48.7% (std dev)

★ Key Finding: Carbon Emission Stability
The PPO scheduler reduces carbon emission standard deviation by 48.7% relative to the default (15.2 → 7.8 gCO_2/job), and by 27.8% relative to the next-best baseline (10.8 → 7.8). The P95 emission under PPO (86.3 gCO_2/job) is lower than the mean emission of the default scheduler (100.0 gCO_2/job). This means that even in adverse stochastic conditions, the PPO scheduler delivers better carbon performance than the default achieves on average. This stability arises from the MDP formulation: the agent has learned to take precautionary scheduling actions—deferring batch jobs, avoiding high-carbon periods—when carbon intensity forecasts deteriorate, smoothing outcomes across the distribution of possible conditions. For organizations with sustainability commitments that involve emission budgets or reporting obligations, this predictability has direct operational value that mean-only metrics do not capture.

Three effects compound to produce this stability. First, the agent learns to treat carbon intensity signals predictively rather than purely reactively—it acts on the current signal as a leading indicator of near-term conditions rather than simply responding to what is in front of it. Second, the asymmetric reward structure prevents the agent from making aggressive SLA-carbon trade-offs that might look good on average but backfire under uncertain arrival conditions. Third, the feasibility masking eliminates a source of stochastic placement error that consistently hurts rule-based baselines: when resource state changes between the time a decision is made and the time it is executed, unmasked schedulers can end up with placements they cannot fulfill.

H. Ablation: Value of Workload Classification

To understand how much of the performance comes from the workload classification mechanism itself—separate from the DRL component—we trained a variant of the PPO scheduler with uniform reward weights ($\alpha=\beta=\gamma=0.33$ for all workload classes, no asymmetry). The results are instructive. This variant still achieves a 22.4% carbon reduction over the default, but the SLA violation rate climbs to 2.1%. Compared to the full model's 27.9% carbon reduction with only 1.1% SLA violations, the difference is clear: the asymmetric reward weighting accounts for roughly one-fifth of the total carbon savings and is the mechanism that keeps SLA compliance intact.

V. DISCUSSION AND LIMITATIONS

A. Why DRL Over a Well-Designed Heuristic?

The Carbon + SLA-Aware Heuristic is our answer to the natural question of whether a well-designed rule is good enough. It implements the same structural logic as our approach—protect latency-sensitive workloads, exploit carbon for batch jobs, balance load across nodes—without the overhead of a learned policy. Yet the PPO scheduler still achieves 10.8% lower mean emissions and 27.8% lower emission variance. Why? Because the right threshold for deferring a batch job is not a single number. It depends on how deep the scheduling queue is, where carbon intensity

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 7, Issue 3, May - June 2024 ||

DOI: 10.15680/IJCTECE.2024.0703008

appears to be heading, and how much SLA pressure the cluster is currently under—a combination of factors that shifts continuously and resists static parametrization. The heuristic uses a fixed 10-minute deadline slack threshold; the PPO agent has learned to adjust implicitly based on context. The variance advantage has a related explanation: the agent has developed a form of predictive caution under uncertainty that no fixed rule replicates.

B. Real-World Generalization Constraints

Our AKS evaluation uses synthetic regional diversity on a single physical cluster with emulated network latency. This is a practical compromise, not an ideal validation setup. A genuinely geo-distributed deployment across multiple Azure regions would expose the scheduler to real network latency costs, authentic infrastructure heterogeneity, and the kind of failure modes that only appear in truly distributed systems. We consider this the primary experimental limitation of the current work.

Three factors make us reasonably confident the results still hold. The tc-netem emulation is calibrated to empirically measured latency values, so the agent's learned behavior has been shaped by realistic network costs. The carbon intensity signals are live throughout the testbed evaluation—not simulated—so the temporal dynamics the agent acts on are authentic. And the consistent 2-percentage-point gap between simulation and AKS performance is small enough to suggest that what the agent learned in simulation transfers reliably to a real cluster. That said, validation on a genuinely multi-region deployment is the highest-priority item on our future work agenda.

C. Power Model and Carbon Accounting Scope

The linear power model in Equation (2) is a standard choice in cluster scheduling research, validated on commodity server hardware with $R^2 > 0.90$ in multiple prior studies. Our own sensitivity analysis in Section III-C confirms that the relative advantage of the PPO scheduler holds across a ±20% range of coefficient uncertainty, so the key results do not hinge on power model precision. The model does not capture network power consumption (which we estimate contributes less than 5% of workload energy in our configuration) or embodied carbon from hardware manufacturing. We scope this work deliberately to operational carbon—the component that scheduling decisions can actually influence—and for that purpose the model is sufficient.

VI. CONCLUSION

We presented a carbon-aware Kubernetes scheduling framework driven by Proximal Policy Optimization. The framework pulls real-time carbon intensity from the Electricity Maps API, classifies workloads into latency-sensitive and batch classes, and applies asymmetric reward shaping that keeps SLA-bound services protected while pursuing aggressive carbon reduction for flexible batch work. Evaluated against three baselines—including a strong Carbon + SLA-Aware rule-based heuristic designed to be as competitive as possible—the PPO scheduler reduces mean carbon emissions by 28% over the default Kubernetes scheduler and by 11% over the best rule-based alternative, while keeping SLA violation rates below 1.2% and fully preserving system throughput.

The result we find most operationally significant is the 49% reduction in carbon emission variance under stochastic conditions. The PPO scheduler's P95 emission is lower than the default scheduler's mean. For organizations with sustainability commitments that involve emission budgets, reporting obligations, or public targets, predictable performance matters as much as average performance—and this finding establishes emission stability as a first-class scheduling objective that learned policies can genuinely optimize in ways that static rules cannot.

Several directions are worth pursuing from here. The most pressing is a genuine multi-region geo-distributed deployment spanning multiple Azure or AWS physical regions, which would provide a more complete validation of the approach. Beyond that, we plan to extend the power model to heterogeneous accelerator hardware (GPUs and TPUs) to support machine learning workloads, explore joint optimization of operational carbon, electricity cost, and embodied carbon, and investigate multi-agent extensions for federated cluster scheduling under partial observability.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 7, Issue 3, May - June 2024 ||

DOI: 10.15680/IJCTECE.2024.0703008

REFERENCES

[1] T. Zhao, T. Piontek, K. Haghshenas, and M. Aiello, "Carbon emission-aware job scheduling for Kubernetes deployments," The Journal of Supercomputing, vol. 80, no. 1, pp. 549–569, 2023. https://doi.org/10.1007/s11227-023-05506-7
[2] D. Zhao and J. Zhou, "An energy and carbon-aware algorithm for renewable energy usage maximization in distributed cloud data centers," Journal of Parallel and Distributed Computing, vol. 165, pp. 156–166, 2022. https://doi.org/10.1016/j.jpdc.2022.04.001
[3] T. Le and D. Wright, "Scheduling workloads in a network of datacentres to reduce electricity cost and carbon footprint," Sustainable Computing: Informatics and Systems, vol. 5, pp. 31–40, 2014. https://doi.org/10.1016/j.suscom.2014.08.012
[4] E. Khodayarseresht, A. Shameli-Sendi, Q. Fournier, and M. Dagenais, "Energy and carbon-aware initial VM placement in geographically distributed cloud data centers," Sustainable Computing: Informatics and Systems, vol. 39, p. 100888, 2023. https://doi.org/10.1016/j.suscom.2023.100888
[5] A. Radovanovic et al., "Carbon-aware computing for datacenters," IEEE Transactions on Power Systems, vol. 38, no. 2, pp. 1270–1280, 2022. https://doi.org/10.1109/tpwrs.2022.3173250
[6] J. Yan et al., "Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach," Computers & Electrical Engineering, vol. 99, p. 107688, 2022. https://doi.org/10.1016/j.compeleceng.2022.107688
[7] Z. Wang et al., "Reinforcement learning based task scheduling for environmentally sustainable federated cloud computing," Journal of Cloud Computing, vol. 12, no. 1, 2023. https://doi.org/10.1186/s13677-023-00553-0
[8] W. Liu et al., "Online job scheduling scheme for low-carbon data center operation: An information and energy nexus perspective," Applied Energy, vol. 338, p. 120918, 2023. https://doi.org/10.1016/j.apenergy.2023.120918
[9] S. Zhang, M. Xu, W. Y. B. Lim, and D. Niyato, "Sustainable AIGC workload scheduling of geo-distributed data centers: A multi-agent reinforcement learning approach," arXiv:2304.07948, 2023.
[10] R. Kaur et al., "An advanced job scheduling algorithmic architecture to reduce energy consumption and CO2 emissions in multi-cloud," Electronics, vol. 12, no. 8, p. 1810, 2023. https://doi.org/10.3390/electronics12081810
[11] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," Journal of King Saud University – Computer and Information Sciences, vol. 32, no. 10, pp. 1127–1139, 2018.
[12] T. Subramanian, "Carbon-aware scheduling for serverless computing," M.S. thesis, Technical University of Munich, 2023.
[13] M. Chadha et al., "GreenCourier: Carbon-aware scheduling for serverless functions," in Proc. ACM Middleware, 2023, pp. 18–23. https://doi.org/10.1145/3631295.3631396
[14] S. Anasuri and K. K. Pappula, "Green HPC: Carbon-aware scheduling in cloud data centers," International Journal of Emerging Research in Engineering and Technology, vol. 4, 2023.