

Carbon-Aware Autonomous AI Systems: Reinforcement Learning for Sustainable Cloud and Edge Computing

Anjali Rout¹, Yashika Chandra^{2,*}

Abstract

The field of communication and information technology is expanding quickly. Because of this, a significant amount of carbon emissions are produced by cloud data centres and edge computing nodes. In fact they are now responsible for 3 to 4 percent of the worlds total greenhouse gas emissions. Most of the time people who manage these resources focus on how they are working and how quickly they can get things done. They do not think about how the carbon intensity of the electricity grid changes over time. is where CarbonRL comes in. CarbonRL is an intelligence system that is aware of carbon emissions and can make its own decisions to reduce the negative impact on the environment. The system uses real-time carbon intensity signals to manage workloads buy energy and reduce the carbon footprint across cloud and edge infrastructures. CarbonRL uses a kind of learning called reinforcement learning. This is based on something called Proximal Policy Optimization and a Long Short-Term Memory state encoder. This helps the system understand things that happen over time like when carbon will be high or low in the grid when energy will be available and when work will arrive. The system also has a way of figuring out what is important. It has to balance things like meeting the promises it makes to users keeping energy costs low and reducing carbon emissions. Many experiments were done using carbon intensity data from the Electricity Maps API and workload data from Google Cloud and Azure. The results show that CarbonRL reduces carbon emissions by 38.7 percent and lowers energy costs by 24.3 percent compared to systems. Additionally, it ensures that more than 99.1% of the time, it fulfils its commitments to consumers. The system can also move work to data centers that use cleaner energy. This reduces the carbon cost of sending data. CarbonRL was compared to seven systems and was tested in many ways to make sure it works well. Overall CarbonRL is a step, towards making cloud and edge computing more sustainable. By using real-time carbon data to manage resources the system is more efficient reduces energy costs and lowers the carbon footprint of cloud data centers and edge computing nodes.

Keywords: Carbon-aware computing, deep reinforcement learning, cloud resource management, edge computing, sustainable AI, Proximal Policy Optimization, marginal carbon intensity, green data centers, multi-objective optimization, SLA-constrained scheduling

*Author for Correspondence

Yashika Chandra

E-mail: yashikachandra06@gmail.com

¹UG Student, Department of Computer Science Engineering, Greater Noida Institute of Technology, Greater Noida, Uttar Pradesh, India

²UG Student, Department of Computer Science Engineering, Greater Noida Institute of Technology, Greater Noida, Uttar Pradesh, India

Received Date: July 05, 2026

Accepted Date: July 07, 2026

Published Date: July 13, 2026

Citation: Anjali Rout, Yashika Chandra. Carbon-Aware Autonomous AI Systems: Reinforcement Learning for Sustainable Cloud and Edge Computing. Journal of Energy Environment and Carbon Credits. 2026; 16(2): 25–35p.

INTRODUCTION

Data centers and communication networks consume a significant amount of energy worldwide. In 2023, energy use exceeded 1,000 TWh. It is expected to double by 2030 because of the growth of artificial intelligence workloads and the use of the Internet of Things and real-time streaming services. Carbon emissions are a problem in the energy sector. The energy use of data centers and communication networks is getting worse because

of the growth of data centers and communication networks. Carbon emissions from data centers and communication networks are issues that need to be addressed. The carbon intensity of power grids differs significantly depending on the location, time of day, and available renewable energy [1, 2]. A data center using power during peak wind or solar times in the Pacific Northwest might emit as little as 15 g of CO₂ equivalent per kilowatt-hour. However, the same computation performed during peak demand in areas that rely heavily on coal can emit over 900 g of CO₂ equivalent per kilowatt-hour.

Most cloud scheduling algorithms focus on cost and speed, not carbon emissions. This is a problem because organizations must now account for their carbon footprint from the purchased electricity. New rules, such as the EU Carbon Border Adjustment Mechanism and the Science Based Targets initiative, are pushing companies to do this. Recent studies have shown that it is possible to reduce carbon emissions without compromising service quality [3, 4]. They have examined moving workloads to times or places. These approaches use simple rules and do not adapt to changing grid conditions. Reinforcement learning is a method of learning strategies that work well in changing environments. This study introduces CarbonRL, a framework that uses reinforcement learning to manage carbon emissions. It forecasts carbon intensity and schedules workloads across cloud and edge computing [5, 6].

The main contributions are:

- A formal method for making decisions about carbon resource scheduling that considers service-level agreements, energy costs, and real-time carbon intensity.
- A reinforcement learning agent that can capture patterns in the carbon grid signals and workload arrivals.
- A way to balance competing objectives like carbon emissions, cost and speed.
- A module that routes workloads to data centers with energy while respecting network constraints.
- Tests on real-world data show that CarbonRL performs well compared to other methods.

BACKGROUND AND RELATED WORK

Carbon Intensity in Electrical Grids

The carbon intensity of power grids varies depending on the location, time of day, and amount of renewable energy available. If a data center uses power when the wind or sun is strong in the Pacific Northwest, it might only release as little as 15 g of CO₂ equivalent per kilowatt-hour. However, if the same work is done during peak hours in places that use a lot of coal, it can release over 900 g of CO₂ equivalent per kilowatt-hour. Most cloud scheduling algorithms consider cost and speed, not carbon emissions from the carbon intensity of power grids [7, 8]. This is a problem because companies must consider the carbon footprint from the carbon intensity of the power grids of the electricity they buy now. New rules, such as the EU Carbon Border Adjustment Mechanism and the Science Based Targets initiative, are forcing companies to do this for the carbon intensity of power grids. Some studies have shown that we can reduce carbon emissions from the carbon intensity of power grids without compromising services. They have examined moving work to times or places for the carbon intensity of power grids. These methods use rules and do not adapt to changes in the carbon intensity of power grids [9, 10].

Reinforcement learning is a method for learning strategies that work well in changing situations for the carbon intensity of power grids. This study focuses on CarbonRL, a framework that uses reinforcement learning to manage carbon emissions based on the carbon intensity of power grids. It predicts carbon intensity and schedules work across cloud and edge computing for the carbon intensity of the power grids.

The main things we did are:

- We developed a method to determine the scheduling of the carbon intensity of power grids that considers service-level agreements, energy costs, and real-time carbon intensity from the carbon intensity of power grids.

- We developed a reinforcement learning agent that can identify patterns in carbon grid signals and when work arrives for the carbon intensity of power grids.
- We found a way to balance things that compete, such as carbon emissions from the carbon intensity of power grids, cost, and speed for the carbon intensity of power grids.
- We developed a module that sends work to data centers with energy for the carbon intensity of power grids while respecting network rules for the carbon intensity of power grids.
- We tested it on data, and it showed that CarbonRL works well compared with other methods for the carbon intensity of power grids.

Sustainable Cloud and Edge Computing

Some studies have focused on green data centers and how to manage power at the hardware level. This includes voltage and frequency scaling, which saves power and ensures that servers are used efficiently and cooled properly. At the system level, there was a project called GreenCloud that attempted to schedule jobs in a way that was aware of carbon emissions. It assumed that the amount of carbon emissions stayed the same all the time. Then there was Carbon-Kubernetes, which was an extension of container orchestration that used real-time carbon signals but did not have the ability to learn and adapt. In the area of edge computing, some researchers, such as Mao et al., proposed a way to make decisions about offloading tasks in mobile edge computing using a type of reinforcement learning called DQN. They attempted to optimize delay and energy use [11, 12]. They did not consider carbon emissions. Xu et al. introduced a framework for scheduling tasks in cloud-edge deployments using a type of reinforcement learning called actor-critic RL; however, they only focused on how long it took to deliver services, not on the emissions.

Multi-Objective Reinforcement Learning

Multi-objective reinforcement learning is used to solve problems where agents have to work on different goals simultaneously. To do this, people use different methods, such as linear scalarization, looking at the Pareto front, and constrained reinforcement learning with something called Lagrangian relaxation. When it comes to scheduling that is aware of carbon, there are three main things that are important. It is necessary to comply with service-level agreements to minimize the cost of energy and reduce the amount of carbon emitted. These three things can work together. Also be in conflict with each other. The challenge of creating reward functions that do not allow one goal to become too important is a challenge. Therefore, a scalarization mechanism is required for multi-objective reinforcement learning [13, 14].

SYSTEM MODEL AND PROBLEM FORMULATION

Infrastructure Model

We represent the underlying computing environment as a directed graph $G = (V, E)$ that maps the physical and logical topologies of the system. The set of nodes V represents a hybrid ecosystem comprising high-capacity cloud data centers (V_{cloud}) and distributed edge computing nodes (V_{edge}). These nodes are interconnected by network links E , each defined by its available bandwidth b_{ij} and propagation delay d_{ij} , which governs the communication latency.

Each node $v \in V$ is characterized by its hardware constraints and environmental impact. Specifically, we tracked its computational capacity C_v (in CPU cores), memory M_v (in GB), and real-time utilization $u_v(t)$. To integrate sustainability into the model, each node also draws a real-time marginal carbon intensity signal $\gamma_v(t)$ (measured in $\text{gCO}_2\text{eq/kWh}$) from its local power grid, allowing the system to respond to fluctuations in clean energy availability [15, 16].

The system workload is modeled as a dynamic sequence of job arrivals $\omega = \{J_1, J_2, \dots, J_N\}$. Each individual job J_k is defined by a tuple $(r_k, d_k, \tau_k, \delta_k)$, where r_k denotes the requested resources, d_k denotes the completion deadline, and τ_k denotes the estimated execution time. A critical feature of this model is the binary deferability flag δ_k , which indicates whether a

job allows temporal shifting within a flexibility window Φ_k . This distinction is vital for operational efficiency: whereas flexible tasks can be scheduled to coincide with low-carbon periods, latency-sensitive jobs (such as real-time AI inference or interactive services) are marked with $\delta_k = 0$ and must be dispatched immediately to meet their strict performance requirements [17, 18].

Markov Decision Process Formulation

To optimize scheduling under fluctuating environmental conditions, we frame the carbon-aware dispatching problem as a finite-horizon Markov Decision Process (MDP), defined by the quintuple $M = (S, A, P, R, \gamma)$. This framework allows the system to learn optimal policies by interacting with stochastic environments.

State Space (S): The system state at any given time step t provides the agent with a comprehensive snapshot of both the infrastructure and its environmental context. The state vector s_t is defined as: $s_t = [\gamma_v(t), u_v(t), Q_t, F_t, \tau_{remaining,t}]$. Here, the agent monitors the current carbon intensity $\gamma_v(t)$ and utilization $u_v(t)$ across all nodes V . To enable proactive decision-making, we include a forecast vector F_t , which provides a carbon intensity outlook for a horizon H (ranging from 1 to 24 hours). The operational status is captured by the current job queue Q_t and the remaining time until deadlines remaining, t , ensuring that the agent remains aware of time-critical constraints.

Action Space (A): At each decision epoch, the agent intervenes by selecting a dispatch action $a_t = (v^*, t_{start})$. This action specifies a target node $v^* \in V$ and an optimal execution time t_{start} . For deferrable jobs, the agent can strategically delay the execution within the flexibility window $[t, t + \Phi_k]$ to coincide with cleaner energy availability. In contrast, for non-deferrable or latency-critical jobs, the agent must immediately assign a node by setting $t_{start} = t$.

Transition Dynamics (P): The evolution of the system is governed by two primary stochastic processes. First, workload arrivals follow a Poisson process with a time-varying rate of $\lambda(t)$, reflecting natural fluctuations in user demand. Second, carbon intensity trajectories were modeled using historical data augmented by Gaussian process forecasting errors. This addition of "noise" ensures that the model accounts for the inherent uncertainty in weather-dependent energy grids, making the resulting policy more robust to real-world forecast inaccuracies.

Reward Function (R): The multi-objective reward function $R(s_t, a_t)$ quantifies the success of an action by balancing three competing priorities: $R(s_t, a_t) = -\alpha \cdot C_{carbon}(a_t) - \beta \cdot C_{energy}(a_t) + \gamma_{sla} \cdot SLA_{score}(a_t)$. In this formulation, C_{carbon} and C_{energy} represent the environmental and financial costs incurred, respectively, while the SLA_{score} measures how effectively the agent satisfies job deadlines. The relative importance of these objectives is managed by adaptive weights ($\alpha, \beta, \gamma_{sla}$), which are dynamically adjusted by the Pareto Weight Controller (detailed in Section 4.3) to navigate the trade-offs between sustainability and performance.

Carbon Emission Model

To quantify the environmental impact of our scheduling decisions, we model the carbon footprint of executing job J_k on a specific node v at a chosen start time t_{start} . This cost, C_{carbon} , is determined by integrating the local marginal carbon intensity $\gamma_v(t)$ over the job execution window $[t_{start}, t_{start} + \tau_k]$, scaled by the active power consumption P_{active} of the allocated resources. By evaluating this integral, the model accounts for the dynamic nature of grid emissions, penalizing execution during "dirty" energy peaks and rewarding shifts to periods of high renewable availability.

Beyond localized computation, we incorporated the carbon overhead incurred by data movement across the network ($C_{network}$). This additive cost is proportional to the volume of data transmitted, physical distance between nodes, and energy intensity of the routing infrastructure at each hop. The primary optimization goal of the system is to minimize the expected aggregate carbon cost—combining both compute and network emissions—while operating within the strict boundaries of Service Level

Agreement (SLA) deadlines and physical hardware capacities. This dual focus ensures that sustainability goals are pursued without compromising the performance guarantees that are essential for production environments [19, 20].

THE CARBONRL FRAMEWORK

Architecture Overview

CarbonRL is made up of four parts that work closely together: (i) a Temporal Carbon State Encoder, (ii) a policy network that uses PPO and has two actor-critic heads (iii) a Pareto Weight Controller, and (iv) a Carbon Arbitrage Router. These parts of CarbonRL work together in a system that controls the loop. This system makes decisions every five minutes, which is the same as the time between updates from the Electricity Maps carbon-forecast API. CarbonRL uses this system to make decisions at five-minute intervals.

Temporal Carbon State Encoder (TCSE)

Raw state observations, such as carbon intensity over time, workload queues, and node utilization, are quite varied. Show patterns at different time scales. The TCSE addresses this by handling each type of input through an encoder before combining them.

- The Carbon Encoder uses a type of recurrent neural network called a bidirectional LSTM. It has 256 hidden units. Takes the carbon forecast for the next H steps along with 72 h of past data. This produces a representation, z_{carbon} , with 512 dimensions.
- The Workload Encoder uses a transformer-based attention mechanism. It has four attention heads and a model dimension of 128. It encodes the job queue, Q_t , into z_{workload} , which has 256 dimensions.
- The Infrastructure Encoder uses a graph attention network (GAT) to process the utilization and topology features of the infrastructure graph G . This results in z_{infra} , which also has 256 dimensions. The combined embedding, z_t , is created by concatenating z_{carbon} , z_{workload} , and z_{infra} . It has 1024 dimensions. It serves as an input to the policy network.

By doing this, the agent has access to both the carbon intensity over time and the current state of the infrastructure. The carbon intensity and infrastructure state are used together to help the agent make decisions. The agent can understand how carbon intensity changes over time and how infrastructure is used.

PPO Policy Network with Adaptive Reward Shaping

The policy and value networks are two multilayer perceptrons. Each network has three layers, with 512 units using GELU activations and LayerNorm. Policy optimization uses the Proximal Policy Optimization objective, which is $L_{\text{CLIP}} = E_t [\min(\text{ratio} * \text{advantage}, \text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) * \text{advantage})]$. Here*, the ratio = policy network / old policy network is the probability ratio, and * advantage is the advantage estimate.

The Pareto Weight Controller maintains the weight vector. This vector has three elements: alpha, beta, and gamma. The vector was updated every 50 policy rollout steps.

The update uses the -gradient descent algorithm: new weight vector = old weight vector. Learning rate * gradient of Pareto loss

The Pareto loss penalizes scalarizations that produce policies dominated by the Pareto front. This helps the policy to collapse to the objective optima. It maintains a set of near-Pareto-optimal behaviors. These behaviors can be selected based on operator-specified SLA budget values.

Carbon Arbitrage Router (CAR)

The Carbon Arbitrage Router helps with workloads. It chooses the target node and time. The Carbon Arbitrage Router thinks of it like this: it tries to find a way to move things around over time. A Carbon

Arbitrage Router was used for this purpose. The Carbon Arbitrage Router uses a type of graph called a time-expanded graph. This graph is called G' . It has nodes and edges. Each node in the graph is a point in time and space.

For example, node v at time step t is one point. The edges between these points have a limit.

These limits are like road rules. The cost of moving something from one point to another is similar to the road price. The Carbon Arbitrage Router aims to find the roads. It considers the costs of edges between points.

The cost is calculated like this:

$$\text{cost}(v, t) = \alpha_t * \gamma_v(t) * P_{\text{active}} * \tau_k. \quad \beta_t * p_{\text{energy}}(v, t) * \tau_k$$

The Carbon Arbitrage Router also considers moving things across different areas. It considers the carbon used to move items through networks. It does not like routes that use a lot of carbon dioxide. The Carbon Arbitrage Router uses a plan to send jobs. For queues, it uses a mixed-integer program with the Hungarian algorithm. For operations, it uses a greedy auction mechanism in real time. A Carbon Arbitrage Router was used for this purpose.

EXPERIMENTAL SETUP

Datasets

We used three sets of data to determine the following: The carbon intensity data were obtained from the Electricity Maps API. This data covers 12 areas across North America and Europe over two years, from 2022 to 2023. It provides carbon intensity values for each hour of the day. The difference between the carbon intensity in one day is significant, and it can be six times different. We obtained data on cloud workloads from two sources: Google Cloud Cluster Usage Traces and the Azure VM Workload Dataset. This cloud workload data contains information about many jobs, approximately 5.2 million cloud workload jobs. These cloud workload jobs require a large amount of resources and take different amounts of time to finish. Data on edge workloads were generated using the OpenFog Consortium reference architecture. The time by which edge workload jobs need to be completed follows a pattern. Most edge workload jobs must be completed within 4.2 s. There was a slight variation with a standard deviation of 1.1 s. The edge workload jobs and cloud workload jobs are different, and the edge workload jobs are unique. Edge workload data are important for evaluating edge workloads, cloud workloads, and carbon intensity because we need edge workload data to evaluate edge workloads, carbon intensity, and cloud workloads.

Simulation Environment

We performed the simulation using Python 3.10 and the CloudSimPy framework. The CarbonGrid module was also added to improve the simulation. The CarbonGrid module was used to simulate the environment. The simulation has 8 data centers. These data centers are located in different parts of the world. For example, we have US-West, US-East, EU-North, and EU-South. Each of these locations has 24 nodes that help complete the work. The CarbonGrid module and simulation used these nodes. Each node is different. They all have the ability to perform work, and the nodes are connected to each other like a tree with branches. This is called a fat-tree structure. We ensured that the time it took for the nodes to communicate with each other was similar to that in real life. We used some measurements from PlanetLab to achieve this. The CarbonGrid module helps with this simulation and uses information about how much carbon is used to generate electricity. The CarbonGrid module uses this information. We added noise to this information. This noise is similar to that which occurs when we try to guess what the carbon usage will be. The CarbonGrid module and simulation environment use this information to simulate what might happen. We added Gaussian Process noise with a sigma noise of 12 gCO₂eq/kWh. This helps us understand what might happen if our guesses are not completely correct. The simulation and CarbonGrid module assisted us with this.

Baselines

We performed the simulation using Python 3.10 and the CloudSimPy framework. The CarbonGrid module was also added to improve the simulation. The CarbonGrid module was used to simulate the environment. The simulation has 8 data centers. These data centers are located in various parts of the world. For example, we have US-West, US-East, EU-North, and EU-South. Each of these locations has 24 nodes that help complete the work. The CarbonGrid module and simulation use these nodes. Each node is different. They all have the ability to do the work. The nodes are connected to each other, similar to a tree with branches. This is called a fat-tree structure. We ensured that the time it takes for the nodes to communicate with each other is similar to that in real life. We used some measurements from PlanetLab to achieve this. The CarbonGrid module helps with this simulation and uses information about how much carbon is used to generate electricity. The CarbonGrid module uses this information. We added noise to this information. This noise is similar to what happens when we try to guess what the carbon usage will be. The CarbonGrid module and simulation environment use this information to simulate what might happen. We added Gaussian Process noise with a sigma noise of 12 gCO₂eq/kWh. This helps us understand what might happen if our guesses are not completely correct. The simulation and CarbonGrid module assisted us with this.

Training Configuration

CarbonRL was trained for a long time, over 200 million steps, using a method called PPO. The learning rate was set to a number η equal to 0.0003. This changes over time using cosine annealing. The mini-batch size for CarbonRL was 4096, which is a large number. There is also something called a clip ratio for CarbonRL, which is set to 0.2 and an entropy bonus of 0.01. The discount factor for CarbonRL is 0.99, which is very close to 1. CarbonRL was trained on a very powerful computer, a single NVIDIA A100 80GB GPU, and it took approximately 38 h to finish training CarbonRL. The people who developed CarbonRL tried different combinations of settings 120 times using a tool called Optuna to find the best ones for training CarbonRL. They did this to obtain the results for CarbonRL.

RESULTS AND DISCUSSION

Primary Performance Comparison

Table 1 shows the performance of all baseline methods over 24 months. CarbonRL reduces carbon emissions. It reduced carbon emissions by 38.7 percent compared to Round Robin and 26.4 percent compared to the baseline method, which is DQN-MORL. CarbonRL also used 24.3% more energy than the cost-optimized method. The good news is that CarbonRL met the service level agreement 99.14 percent of the time. This is better than all the methods, including the one that was optimized for latency. Therefore, reducing carbon emissions does not mean that service quality must suffer if we use the time we have in a way. CarbonRL shows that we can reduce carbon emissions while maintaining service quality.

Carbon Reduction Decomposition

The analysis shows that carbon reductions come from parts of the system. 28.4 Percent of the total carbon reduction is because of temporal shifting. This means that the system postpones 34.7 percent of the work that can be delayed until the time when carbon use is low. Spatial arbitrage is another method by which the system reduces carbon use. It does this by moving some work from the EU-South to the EU-North. EU-South uses a lot of carbon, on average 420 g of carbon dioxide for each kilowatt hour, and EU-North uses a lot of carbon, on average, approximately 82 g of carbon dioxide for each kilowatt hour. This moving of work reduces carbon use by 18.1 percent. The remaining 9.2% reduction occurs because the system can tell when carbon use will be low and schedules the delayed work for those times. The system uses an LSTM encoder to determine this. It can predict low carbon use 3–6 h in advance. The Carbon Arbitrage Router is also a part of reducing carbon use. It helps move work to the times and places where carbon use is low. This is how the system reduces carbon use.

Ablation Study

Table 2 presents the results of the ablation study. This study examined four versions of the CarbonRL system. We have version A1, which does not use the LSTM encoder. Instead, it uses an MLP on the features from the current step. Version A2 does not use the Pareto Weight Controller. In this version, the weights are fixed at $\alpha = \beta = \gamma = \frac{1}{3}$. The next version is A3, which does not use the Carbon Arbitrage Router. This only sends jobs to nodes. Finally, we have version A4, which does not use temporal shifting. This means that it sends all jobs away. The CarbonRL system was compared with these four versions.

Scalability and Convergence Analysis

CarbonRLs training involves over 200 million environment steps. The reward that CarbonRL receives starts to remain the same at approximately 120 million steps. This occurs with a difference of plus or minus 0.031 when we try it five times with different random starts, which means that CarbonRL learns a good policy without forgetting what it learned before. When we use CarbonRL to make decisions, it takes 2.3 milliseconds on a regular computer, which is fast enough for our needs. The model we used takes up 48 megabytes of memory, which is small enough to work on special computers with ARM Cortex-A72 processors.

We tested CarbonRL on computer systems with 4 to 128 large computers in a data center. We found that it takes a little longer to make decisions when we add computers, but not much longer. This is because the way CarbonRL looks at the problem, by paying attention to parts of the system, helps it work well even when the system gets bigger. We also found that CarbonRL can still significantly reduce carbon emissions even when the system becomes large. It reduces emissions by 52.1 percent when four computers are used and by 41.8 percent when 128 computers are used. This indicates that CarbonRL works well and helps reduce carbon emissions, even when the system becomes more complicated.

Table 1. Performance comparison across all methods (24-month evaluation).

Method	Carbon (tCO ₂)	Energy (\$)	SLA (%)	Latency (ms)	C-Reduction (%)
B1: Round Robin	1,842.3	38,412	97.23	142.1	0.0 (ref.)
B2: Least Loaded	1,791.6	36,218	98.41	118.7	2.7
B3: Cost-Opt.	1,644.9	29,840	97.89	155.3	10.7
B4: Time Shift	1,421.2	31,577	97.14	198.6	22.9
B5: Spatial Mig.	1,312.4	34,002	98.03	231.8	28.8
B6: CAK [10]	1,198.7	32,180	98.62	176.4	34.9
B7: DQN-MORL [12]	1,109.8	27,914	98.76	161.2	39.8
CarbonRL (Ours)	816.4	22,569	99.14	163.7	55.7

All carbon emission values computed over the full 24-month simulation. Carbon reduction percentages are relative to B1 (Round Robin).

Table 2. Ablation study results.

Variant	Carbon (tCO ₂)	SLA (%)	Energy (\$)	Norm. Reward
A1: No LSTM Encoder	1,041.2	98.61	25,830	0.741
A2: Fixed Weights	974.6	98.23	24,112	0.768
A3: No CAR Module	1,148.3	99.08	26,444	0.729
A4: No Time Shifting	1,243.7	99.21	28,001	0.712
CarbonRL (Full)	816.4	99.14	22,569	0.847

DISCUSSION AND LIMITATIONS

The success of CarbonRL demonstrates that autonomous reinforcement learning agents can significantly reduce the carbon footprint of distributed cloud and edge workloads without sacrificing the performance guarantees required by modern service-level agreements (SLAs). However, several critical implications and technical trade-offs emerged during our evaluation that warrant further discussion in this regard.

Forecast Sensitivity and Grid Volatility

A central pillar of the framework is the TCSE's reliance on 24-hour forecast windows. While this temporal lookahead is essential for scheduling deferrable jobs, it introduces sensitivity to the inherent uncertainty of carbon intensity predictions. In real-world scenarios, APIs such as Electricity Maps exhibit Mean Absolute Percentage Errors (MAPE) ranging from 8% to 15% for long-range lookaheads. Although we mitigated this during training using Gaussian Process noise augmentation, we observed a 6.2% degradation in carbon efficiency under high-volatility grid conditions, such as sudden cloud cover impacting solar generation. This highlights a fundamental tension: the agent's effectiveness is ultimately bounded by the reliability of the external telemetry data it consumes.

Evolution of the Network Carbon Model

In its current formulation, the system treats network transmission carbon as a static cost proportional to the data volume and physical distance. Although this provides a robust baseline for optimization, it oversimplifies the complexities of dynamic backbone routing and modern traffic engineering. A more sophisticated approach—and a key direction for future work—would involve integrating Software-Defined Networking (SDN) telemetry directly into the RL state space. This would allow CarbonRL to perform routing-level carbon optimization, selecting paths not only for their latency but also for the real-time energy intensity of the intermediate infrastructure.

Algorithmic Stability and Deployment Realities

The Pareto Weight Controller is vital for balancing competing objectives; however, it introduces a degree of non-stationarity into the optimization landscape. We found that during extreme workload bursts, this shifting reward signal can occasionally destabilize the training process. Future iterations of CarbonRL could benefit from distributional value functions, which provide a more nuanced understanding of reward uncertainty and may offer greater stability in edge cases.

Finally, transitioning CarbonRL from a research prototype to a production environment requires a robust operational ecosystem. Successful deployment hinges on deep integration with cloud orchestration platforms (such as Kubernetes via custom scheduler plug-ins), reliable real-time carbon data pipelines, and standardized energy metering APIs. By addressing these practical integration hurdles, CarbonRL can serve as a foundational blueprint for the next generation of sustainable and carbon-aware computing.

CONCLUSION

This paper presents CarbonRL, a novel deep reinforcement learning framework for carbon-aware autonomous resource management across cloud and edge computing infrastructures. By integrating real-time marginal carbon intensity signals into a multi-objective PPO agent augmented with LSTM temporal encoding, Pareto-adaptive reward shaping, and a Carbon Arbitrage Router, CarbonRL achieved state-of-the-art performance on carbon reduction (55.7% vs. Round Robin; 26.4% vs. best prior RL baseline), energy cost savings (41.3%), and SLA compliance (99.14%).

The key insight underlying CarbonRL is that the temporal and spatial heterogeneity in electrical grid carbon intensity represents a largely untapped optimization dimension in cloud computing. By treating carbon as a first-class scheduling objective and equipping AI agents with rich temporal state

representations, it is possible to significantly decouple computational throughput from carbon emissions, contributing meaningfully to the decarbonization trajectory of the ICT sector.

Future directions include integration with renewable energy procurement markets, extension to multi-tenant scenarios with privacy-preserving federated learning of carbon policies, real-time deployment in production data center environments, and exploration of carbon-aware neural architecture search to minimize the emission footprint of AI model training itself.

Acknowledgment

The author acknowledges the academic resources provided by the Department of Computer Science and Engineering, Great Noida Institute of Technology, and the open-source contributions of the Electricity Maps, CloudSimPy, and Stable-Baselines3 communities that facilitated this research.

REFERENCES

1. Valavanidis A. Global electricity generation from renewable sources. *Journal*. 2016;2(5):99-110.
2. Li B, Samsi S, Gadepally V, Tiwari D. Clover: toward sustainable AI with carbon-aware machine learning inference service. In: SC '23: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis [Internet]. New York: Association for Computing Machinery; 2023Nov 11. p. 1-15. doi:10.1145/3581784.3607034.
3. Radovanović A, Koningstein R, Schneider I, Chen B, Duarte A, Roy B, et al. Carbon-aware computing for datacenters. *IEEE Trans Power Syst*. 2023;38(2):1270-80. doi:10.1109/TPWRS.2022.3173250.
4. Souza A, Jasoria S, Chakrabarty B, Bridgwater A, Lundberg A, Skogh F, et al. CASPER: carbon-aware scheduling and provisioning for distributed web services. In: Proceedings of the 14th International Green and Sustainable Computing Conference (IGSC '23). New York: Association for Computing Machinery; 2023. p. 67-73. doi:10.1145/3634769.3634812.
5. Yang J, Saad Z, Wu J, Niu X, Leung H, Drew S. A survey on task scheduling in carbon-aware container orchestration [preprint]. *arXiv*. 2025. doi:10.48550/arXiv.2508.05949.
6. Brander M, Sood A, Wylie C, Haughton A, Lovell J. Electricity-specific emission factors for grid electricity. *Ecometrica Technical Paper*. 2011 Aug. p. 1-22.
7. Wilson EL, DiGregorio AJ, Villanueva G, Grunberg CE, Souders Z, Milette KM, et al. A portable miniaturized laser heterodyne radiometer (mini-LHR) for remote measurements of column CH₄ and CO₂. *Appl Phys B*. 2019;125(11):211. doi:10.1007/s00340-019-7315-8.
8. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener Comput Syst*. 2012;28(5):755-68. doi:10.1016/j.future.2011.04.017.
9. Kliazovich D, Bouvry P, Khan SU. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *J Supercomput*. 2012;62(3):1263-83. doi:10.1007/s11227-010-0504-1.
10. Dhawan R, Patel MS. Carbon-aware AI control plane for DevOps automation: a reference architecture and next-generation sustainability framework. *IEEE Access*. 2026;14:11163-84. doi:10.1109/ACCESS.2026.3656467.
11. Mao Y, Zhang J, Letaief KB. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J Sel Areas Commun*. 2016;34(12):3590-605. doi:10.1109/JSAC.2016.2611964.
12. Xu J, Chen L, Zhou P. Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. IEEE; 2018. p. 207-15. doi:10.1109/INFOCOM.2018.8485977.
13. Roijers DM, Vamplew P, Whiteson S, Dazeley R. A survey of multi-objective sequential decision-making. *J Artif Intell Res*. 2013;48:67-113. doi:10.1613/jair.3987.
14. Van Moffaert K, Nowé A. Multi-objective reinforcement learning using sets of Pareto dominating policies. *J Mach Learn Res*. 2014;15(1):3483-512.

15. Tessler C, Mankowitz DJ, Mannor S. Reward constrained policy optimization [preprint]. arXiv. 2018 May 28. doi:10.48550/arXiv.1805.11074.
16. Cortez E, Bonde A, Muzio A, Russinovich M, Fontoura M, Bianchini R. Resource Central: understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17). New York: Association for Computing Machinery; 2017. p. 153-67. doi:10.1145/3132747.3132772.
17. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms [preprint]. arXiv. 2017. arXiv:1707.06347. doi:10.48550/arXiv.1707.06347.
18. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735-80. doi:10.1162/neco.1997.9.8.1735.
19. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: International Conference on Learning Representations (ICLR); 2018.
20. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks [Preprint]. 2017. arXiv:1710.10903.