International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 09 | Oct 2026 www.irjet.net p-ISSN: 2395-0072

LLM-Based Multi-Cloud Resource Optimization

Ms. Niyukta Patil[1], Mr. Aryan Chaudhary[2], Ass. Prof. Mrs. Swati Kadu[3], Mr. Aniket Kole[4]

[1234]*Department of Computer Applications, D. Y. Patil Deemed to be University, Navi Mumbai, Maharashtra, India*

Abstract - *Cloud infrastructure across multiple providers and regions has made resource management more complex due to the growth of AI, containers, and microservices. Manual methods and simple rules are no longer sufficient, while cloud costs and carbon emissions have become major concerns. This paper proposes a system that uses a Large Language Model (LLM) as a decision-making layer within AIOps. It analyses cloud usage, costs, and carbon emissions to suggest actions such as scaling resources and moving workloads across cloud environments. The proposed system has three layers: monitoring using Prometheus and Kubernetes metrics, LLM-based decision-making with rule checking, and execution of selected actions. A simulated environment with two cloud regions demonstrates decision-making and budget control. The paper also proposes an evaluation method based on cost, carbon emissions, service disruptions, decision stability, and LLM overhead. Finally, it discusses the challenges, trade-offs, and future improvements required for safe and reliable resource management across multiple cloud environments.*

Key Words: AIOps, large language models, multi-cloud resource management, cloud sustainability, carbon-aware computing, Kubernetes, cloud cost optimization, and autonomous agents.

1. INTRODUCTION

Enterprises increasingly use multiple cloud providers to avoid vendor lock-in, meet data residency requirements, and benefit from regional pricing and hardware. However, resource management still relies heavily on fixed rules, threshold-based autoscaling, and reinforcement learning policies focused on individual clusters or providers.

These approaches face three key challenges: unpredictable AI workloads with high GPU demand, rising cloud costs requiring continuous optimization, and the need to reduce carbon emissions based on regional electricity intensity and time of use.

Large Language Models (LLMs) can interpret diverse inputs, including metrics, logs, pricing data, policies, and historical events. This makes them promising decision-makers for coordinating cloud resources across providers, helping determine where workloads should run or move while considering cost, performance, and sustainability.

1.1 Problem Statement

Existing automated cloud resource management methods are not good enough in three ways when used for -cloud and sustainability-focused situations:

- Single- objective focus: Most cloud autoscalers and research using reinforcement learning mainly focus on cost or service level agreements; carbon impact is not often considered as a goal except in some special cases for AI models.
- Cluster-local reasoning: Decisions made in each cluster on its own can go against goals across clouds causing problems like back and forth changes, slow progress or unstable performance when clusters fight over the same work.
- No standardized evaluation methodology: A recent review of AI-based cloud resource allocation found that sustainability and proper testing methods are still missing in studies.

This paper solves these issues by introducing a system where an AI model acts as a decision maker, across clouds with a safety check part and a testing method that makes it easy to compare cost, carbon and reliability in approaches. A working version of this system is shown in Section 9.

1.2 Contributions

- A layered reference architecture for LLM-augmented multi-cloud AIOps. This architecture splits the process into observation, reasoning, verification and actuation. The LLM never triggers an infrastructure action directly. The LLM must first pass a safety check.
- A cost–carbon–performance decision formulation for multi-cloud resource orchestration. This extends work that used only one objective or one cluster.
- A working, open-source proof-of-concept. It contains a multi-cloud environment, an LLM/threshold baseline decision loop, a safety verifier and a budget enforcer. It does not need a real production multi-cloud budget. Sample output and results are shown in Section 9.
- An evaluation protocol that lists metrics. The metrics cover cost, emissions, SLA compliance, decision stability and the LLM's own computational and financial overhead. This overhead dimension is often missing from studies.

1.3 Paper Organization

Section 2 reviews related work. Sections 3 and 4 present the problem formulation and proposed architecture. Sections 5 and 6 describe the methodology and evaluation metrics. Sections 7 and 8 discuss expected results and limitations. Section 9 presents the prototype and results, followed by conclusions and future work in Section 10.

2. RELATED WORK

2.1 Autoscaling and Resource Management in the Cloud

Cloud resource management has evolved from rule-based provisioning to machine-learning and AI-driven autoscaling. Reinforcement-learning surveys report efficiency gains over heuristics but note limited evaluation across clusters and workloads [3, 8]. Container-orchestration studies also identify limited joint optimization of scheduling, scaling, and migration across multiple clouds [16].

2.2 AI-Driven Multi-Cluster and Multi-Cloud Optimization

Recent research highlights that isolated cluster decisions can conflict with global objectives, causing instability and unpredictable behavior [10]. This motivates a centralized LLM-based reasoning layer that evaluates cross-cloud state before recommending actions, rather than relying on fully decentralized agents.

2.3 Sustainability-Aware and Carbon-Aware Scheduling

Recent studies aim to reduce cloud workloads' carbon and water footprints. Research considers latency, emissions, water use, and electricity costs in LLM inference scheduling [6], while other work examines sustainable LLM deployment [4]. Google demonstrates shifting flexible workloads to lower-carbon periods using forecasts [8]. However, workload placement and temporal shifting are rarely optimized jointly [2]. This study addresses this gap by combining workload placement and scaling for broader LLM-based cloud resource orchestration.

2.4 AIOps and LLM Agents in Cloud Platforms

Cloud platforms increasingly adopt autonomous AI agents, while GreenOps combines cost and energy efficiency. A survey of LLMs in AIOps highlights this trend but notes immature evaluation methods [9]. Research also shows that schema-valid LLM outputs may still contain errors or unsafe actions [5]. These risks motivate the safety-verification layer described in Sections 4 and 9.

3. PROBLEM FORMULATION

We consider a multi-cloud environment with a set of regions/providers R = {r_1, …, r}, each exposing compute capacity, a time-varying unit price p(r,t), and a time-varying grid carbon intensity c(r,t) measured in gCO_2eq/kWh. A set of workloads W = {w_1, …, w} must be placed and periodically re-evaluated across R subject to per-workload latency and availability constraints.

3.1 Decision Variables

- Placement - which region(s) host each workload's replicas at time t.
- Scale - the number of replicas or instance size allocated to each workload.
- Migration - whether to move a running workload to a different region, weighed against migration cost and disruption risk

3.2 Objective

The system aims to minimize total cost, carbon emissions, and placement instability while meeting SLA requirements.

The objective is to minimize:

[\alpha Cost(t)+\beta Carbon(t)+\gamma Instability(t)]

subject to latency requirements for all workloads, regional budget limits, and capacity constraints.

Cost(t) includes compute, storage, and data-transfer charges. Carbon(t) estimates emissions using compute usage and regional carbon intensity. Instability(t) penalizes frequent placement changes to reduce unnecessary workload movement.

The weights (\alpha), (\beta), and (\gamma) are adjustable to reflect operational priorities. A higher (\beta) prioritizes sustainability, while a higher (\alpha) emphasizes cost reduction. Section 9 presents a prototype implementation with safety and budget checks.

3.3 Role of the LLM

Unlike reinforcement learning, which learns policies through trial and error, the LLM receives current state data, including regional utilization, prices, carbon intensity, and SLA headroom, at each decision interval. It proposes actions with brief justifications. This approach improves interpretability and adaptability to new constraints without retraining, while simplifying auditing, but lacks RL's sample efficiency and formal convergence guarantees.

4. PROPOSED FRAMEWORK ARCHITECTURE

The framework is organized into five layers, each with a narrow responsibility, so that the LLM's output is always mediated by a deterministic check before it can affect live infrastructure (Table 1). Section 9 shows a direct implementation of this architecture in a runnable prototype.

Table 1: Layered architecture of the proposed framework.

Layer	Responsibility	Representative components
Observability	Collect cloud usage, cost, and carbon data.	Prometheus, Kubernetes metrics, billing APIs
State aggregation	Summarize collected data for analysis.	Aggregation service, feature store
LLM Decision	Suggest actions with justification.	Claude/GPT-class models, JSON output
Safety Verification	Check budget, capacity, and SLA rules.	Rules engine, policy-as-code
Actuation	Execute approved actions.	Kubernetes API, autoscalers, CloudSim Plus

4.1 Observability Layer

This layer collects workload utilization and latency, cost, and carbon-intensity data at fixed intervals (e.g., every 1–5 minutes). Signals come from Kubernetes metrics and Prometheus, cloud billing APIs or simulated prices, and live carbon APIs or static datasets. All data is timestamped and tagged by region/provider to create a consistent system snapshot.

4.2 State Aggregation Layer

Raw telemetry is windowed (e.g., 5- and 15-minute rolling averages) and compressed into a compact, structured summary one row per region per workload to keep the LLM prompt within a manageable size and to reduce token cost. This layer also computes derived features such as headroom relative to SLA thresholds and the marginal cost/carbon impact of adding one more replica in each region.

4.3 LLM Decision Layer

At each decision point, the system prompts the model with the current state, objective weights ((\alpha,\beta,\gamma)), hard constraints, and a fixed JSON format. The response specifies a scale, migrate, or no-op action for each workload, with a brief reason. This structured output enables reliable parsing and downstream validation instead of free-form text.

5. PROOF-OF-CONCEPT METHODOLOGY

The proposed evaluation is designed to be achievable with open-source tooling and public datasets, without requiring production multi-cloud spend, following a simulation-first, train-then-deploy testbed strategy. The prototype presented in Section 9 implements the simulation-based portion of this methodology directly.

5.1 Experimental Testbed

- Container environment: Use two or three lightweight Kubernetes clusters (kind or k3d) to represent cloud regions. Deploy Prometheus and metrics-server to collect performance metrics.
- Simulation alternative: Use CloudSim Plus to model multiple providers or regions, enabling faster, large-scale experiments.
- Workload traces: Replay public Google or Azure cluster traces as job arrivals to simulate realistic demand.
- Carbon intensity: Use regional carbon-intensity data from public APIs or historical datasets to estimate environmental impact.
- Pricing data: Use publicly available cloud pricing to create a per-region cost table and compare resource expenses.

5.2 Experimental Procedure

- Warm-up: Replay a fixed trace segment to initialize the system; exclude these results.
- Evaluation: Test each baseline and the proposed framework on held-out traces under the same pricing and carbon conditions, using at least three independent trials per condition.
- Sensitivity analysis: Vary weights ((\alpha,\beta,\gamma)) to assess the cost–carbon trade-off.
- Ablation: Disable safety verification and measure potential SLA violations and budget-limit breaches from unconstrained LLM outputs.

6. EVALUATION PROTOCOL

Responding to the evaluation gap identified in Section 2.5, all approaches are compared on the same five metrics, computed over identical replayed trace segments.

Table 2: Proposed evaluation metrics.

Metric	Definition	Purpose
Cost	Total cloud cost.	Measure expenses.
Carbon Emissions	Compute usage × carbon intensity.	Measure environmental

		impact.
SLA Violations	Requests exceeding service targets.	Ensure service quality.
Decision Stability	Frequency of workload changes.	Detect unnecessary changes.
Decision Overhead	LLM latency, token cost, and rejected actions.	Measure system overhead.

Reporting decision overhead alongside cost and carbon is a deliberate methodological choice: any framework that reduces infrastructure cost or emissions by adding a nontrivial LLM inference cost of its own should have that trade-off made explicit rather than left implicit, particularly given that LLM inference has its own energy footprint. Section 9.4 reports these metrics computed directly from the prototype's demonstration run.

7. EXPECTED RESULTS AND DISCUSSION

The framework aims to improve cost–carbon optimization over static heuristics by adapting to changing carbon intensity. It may outperform single-objective RL on combined cost and emissions, but could increase decision latency and LLM costs. Safety checks should help prevent unsafe actions, while sensitivity analysis can reveal cost–carbon trade-offs. These expectations require further validation, as current prototype results remain preliminary.

8. LIMITATIONS AND THREATS TO VALIDITY

- Simulation fidelity: Simplified simulations cannot fully represent real cloud costs, networking, or egress charges.
- Carbon data: Public datasets may lack precise regional and time-based measurements.
- LLM footprint: The energy and carbon costs of LLM decisions are difficult to measure accurately.
- Prototype scale: The two-region simulation tests workflow functionality, not production performance.
- Utilization model: Utilization is independent of replica count, limiting SLA evaluation.
- Small models: Local models may produce invalid outputs; safety checks and fallback rules reduce these risks.
- Production validation: Real multi-cloud testing is required before deployment.

9. PROTOTYPE IMPLEMENTATION AND DEMONSTRATION

A lightweight Python prototype validates the complete workflow: observation, decision-making, safety and budget checks, actuation, and logging. It uses a simulator, decision loop, and experiment harness. A locally hosted open-source model via Ollama enables offline testing without cloud credentials, paid APIs, Kubernetes, or a GPU.

9.1 Component 1: Multi-Cloud Simulator

The simulator.py module emulates Kubernetes clusters and cloud billing/carbon APIs across two regions: one cheaper but more carbon-intensive, the other costlier but cleaner. Seeded conditions ensure fair comparisons across methods. Each simulated tick records:

- Utilization (%) - a daily traffic wave with random noise, offset in time between regions to mimic time-zone effects.
- Price per replica-hour - fluctuates randomly around a fixed regional base price.
- Carbon intensity (gCO_2/kWh) - varies by time of day, generally lower at night, plus noise.
- Current replica count - the state the decision layer is being asked to adjust.

9.2 Component 2: Decision Loop

- Observe: Collect simulator metrics.
- Decide: The LLM returns JSON actions balancing utilization, cost, and carbon intensity (temperature = 0). The baseline scales up above 70% utilization and down below 30%.
- Recover: Use the baseline if the LLM fails or returns invalid JSON; keep unmentioned regions unchanged.
- Safety verification: Limit changes to three replicas per tick, prevent scale-down above 80% utilization, and maintain at least one replica.
- Budget verification: Reject actions exceeding the hourly budget.
- Actuate: Apply approved actions to the simulator; real deployments would use the Kubernetes API.
- Log: Record states, decisions, approved actions, safety notes, and costs in CSV files.

9.3 Component 3: Experiment Harness and Reproducibility

The run_experiments.py script compares the threshold baseline and LLM approach across three seeds (42, 43, 44), with 24 ticks per run.

The harness records cost, emissions, replica changes, clean-region placement, safety overrides, and fallback decisions. LLM overhead metrics include API calls, invalid responses, latency, and token usage. The make_chart.py script generates four-panel charts from run logs. Carbon estimates assume 0.1 kW per replica and one hour per tick.

9.4 Sample Demonstration Run

The baseline run (seed 42) shows the first six ticks. Both regions scale down from two replicas to one at tick 0 due to low utilization. At tick 5, Region B scales up to two replicas as utilization reaches 82.3%, exceeding the 70% threshold. All actions pass safety checks without modification.

Table -3: Sample output log from the prototype baseline run (seed 42, first 6 of 24 ticks). Rerunning the shipped code with the same seed reproduces these values.

Tick	Region	Util (%)	Price ($/hr)	Carbon (gCO2/kWh)	Replicas (before →final)	Total cost ($/hr)
0	A	21.9	455.7	2 → 1	0.1061	0
0	B	28.1	102.9	2 → 1	0.1061	0
1	A	22.5	447.7	1 → 1	0.1021	1
1	B	33.4	88.2	1 → 1	0.1021	1
2	A	7.4	440.0	1 → 1	0.1026	2
2	B	50.7	84.7	1 → 1	0.1026	2
3	A	21.1	429.2	1 → 1	0.1026	3
3	B	62.2	69.7	1 → 1	0.1026	3
4	A	27.0	392.3	1 → 1	0.1087	4
4	B	61.0	85.2	1 → 1	0.1087	4
5	A	30.2	391.1	1 → 1	0.1716	5
5	B	82.3	89.6	1 → 2	0.1716	5

9.5 Result Charts and Metric Readout

Figure 1 shows the 24-tick baseline run. Region B reaches 8 replicas by tick 11 and maintains this level until tick 16, while Region A peaks at 9 replicas at tick 17. Hourly cost peaks at approximately $0.82 at tick 16, when the combined replica count is highest (16). Region A consistently has higher carbon intensity, highlighting the cost–carbon trade-off the LLM aims to optimize.

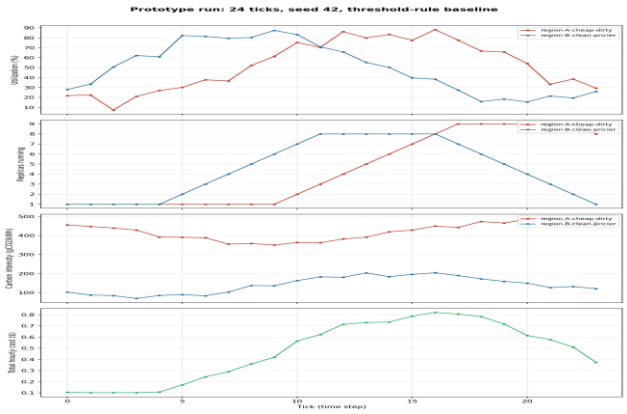


Chart -1: Utilization, replica count, carbon intensity, and total hourly cost over a 24-tick (simulated one-day) run, seed 42, threshold-rule baseline.

Table 4: Evaluation metrics computed from the baseline demonstration run.

Metric	Value in this run (baseline, seed 42)	Notes
Total cost	≈ $11.37	Average $0.47/hour; peak $0.82/hour at tick 16.
Avg. carbon intensity	Region A: 421; Region B: 140 gCO_2/kWh	Region B is cleaner, but baseline ignores carbon intensity.
Estimated emissions	≈ 6.53 kgCO_2e	Assumes 0.1 kW per replica and one hour per tick.
Replica-hours in clean region	50.2%	Placement follows utilization, not carbon intensity.
Utilization above 80%	8 of 48 region-ticks	Not a valid SLA metric because utilization is independent of replica count.
Replica changes	23	Threshold-based scaling adjusts replica counts gradually.
Safety overrides	0	All baseline decisions met configured limits.
Decision overhead	N/A	No LLM was used in the baseline.
Total cost	≈ $11.37	Average $0.47/hour; peak $0.82/hour at tick 16.

9.6 First Result: Local LLM Condition (Single Seed)

The LLM condition used seed 42, matching the baseline settings, with safety and budget checks. Since the run log was unavailable, replica counts were reconstructed from Figure 2 and combined with simulated prices and carbon intensities.

Hourly costs matched the plotted curve within $0.001 per tick. Latency, token usage, invalid replies, and safety overrides remain unreported due to missing logs.

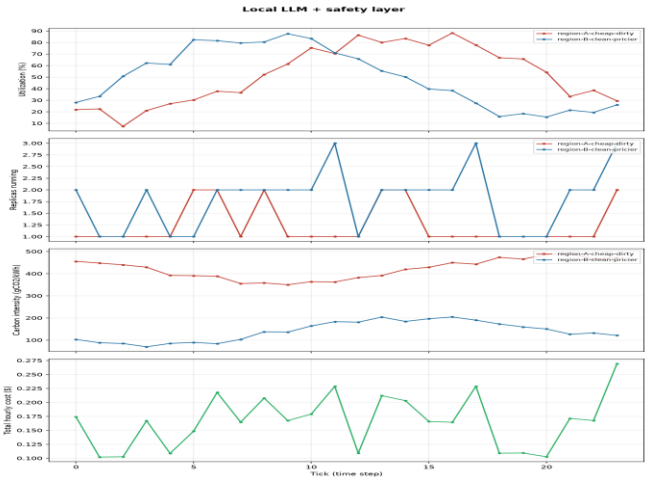


Figure 2. Utilization, replica count, carbon intensity, and total hourly cost for the local LLM condition with the safety layer (seed 42, 24 ticks).

Table 5. Baseline versus local LLM condition (seed 42). LLM-condition values are reconstructed from the run's chart; overhead metrics are not available.

Metric	Threshold baseline	Local LLM + safety layer	Notes
Total cost (24 ticks)	$11.37	$3.98	≈65% lower with LLM.
Average hourly cost	$0.47	$0.17	—
Estimated emissions ($kgCO_2e$)	6.53	1.87	≈71% lower; assumes 0.1 kW per replica and one hour per tick.
Replica-hours in clean region	50.2%	58.9%	Higher share, but overall capacity is lower.
Mean replicas (both regions)	9.0	3.0	LLM used about one-third of baseline capacity.
Peak replicas in one region	9	3	—
Mean replicas above 80% utilization	5.1	1.6	Indicates potential under-provisioning at peaks.
Total replica changes	23	20	Similar counts.
Direction reversals	2	13	More oscillation in the LLM condition.
Safety overrides, invalid replies, latency, tokens	N/A	Not available	Requires overhead logs.

The LLM condition averaged 3 replicas versus 9 for the baseline, reducing estimated cost by 65% and emissions by 71%. However, utilization exceeded 80% at times, indicating under-provisioning.

Replica changes were similar (20 versus 23), but the LLM reversed scaling direction 13 times versus 2 for the baseline, indicating greater instability. These preliminary results do not establish superior optimization.

Future work should enforce minimum capacity, link utilization to replica counts, and evaluate multiple seeds and larger models.

9.7 What the Prototype Demonstrates - and What It Does Not

The prototype demonstrates the complete workflow, from observation and decision-making to safety checks, budget enforcement, execution, and logging, using free local tools. Initial tests with a local LLM showed useful decisions but unstable scaling and under-provisioning. However, results from one seed and a small model cannot establish whether the framework outperforms existing baselines. Future work includes improved simulation, stronger safety rules, multi-seed testing, overhead measurements, and Kubernetes experiments using public workload data. Code, logs, and charts are included with the paper.

10. CONCLUSION AND FUTURE WORK

This paper proposed an LLM-based framework for multi-cloud resource management that balances cost, carbon emissions, and SLA requirements. A safety layer verifies actions before execution. The open-source prototype demonstrated the complete workflow in simulation, showing potential savings but also revealing scaling instability and under-provisioning.

Future work includes multi-seed testing, improved simulation and capacity controls, learned safety policies,

multi-agent designs, carbon-intensity forecasting, and validation on real Kubernetes clusters.

REFERENCES

Bracketed numbers in the text, e.g. [1], correspond to the numbered entries below, in alphabetical order by first author.

1. Alam, S., Ramzan, M. A., Zubair, M., et al. (2026). AI-driven resource allocation in cloud computing: A systematic review. Computing, 108, 67. https://doi.org/10.1007/s00607-026-01655-8

   Use: Highlights sustainability and evaluation gaps in AI-based resource allocation.

2. Asadov, N., et al. (2025). Carbon-aware spatio-temporal workload shifting in edge-cloud environments. Sustainability, 17(14), 6433. https://doi.org/10.3390/su17146433

   Use: Supports combined multi-region and time-based carbon-aware scheduling.

3. Garí, Y., et al. (2021). Reinforcement learning-based application autoscaling in the cloud: A survey. Engineering Applications of Artificial Intelligence, 102, 104288. https://doi.org/10.1016/j.engappai.2021.104288 Use: Supports the need for multi-cloud, multi-objective optimization.

4. Hoxha, J., Thanasi-Boçe, M., & Khalifa, T. (2025). A deployment-aware framework for carbon- and water-efficient LLM serving. Sustainability, 17(23), 10473. https://doi.org/10.3390/su172310473 Use: Provides an example of carbon- and water-aware scheduling.

5. Li, Y. (2026). When JSON is not enough: Semantic reliability of schema-constrained LLM ordering agents. arXiv:2607.18261. https://arxiv.org/abs/2607.18261

   Use: Demonstrates the need for safety verification beyond valid JSON output.

6. Moore, H., et al. (2025). Sustainable carbon-aware and water-efficient LLM scheduling in geo-distributed cloud datacenters. GLSVLSI '25, 929–934. https://doi.org/10.1145/3716368.3735301 Use: Supports multi-objective optimization of cost, latency, carbon, and water.

7. Punniyamoorthy, V., et al. (2025). AI-driven cloud resource optimization for multi-cluster environments. arXiv:2512.24914. https://arxiv.org/abs/2512.24914 Use: Motivates coordinated decision-making across multiple clusters.