



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: <https://doi.org/10.22214/ijraset.2026.85243>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

AI Agent-Driven Datacenter Optimizer: A Guardrailed, Explainable LLM Orchestrator for Cloud Platform Management

Jay Dighore¹, Vedant Gaidhane², Ayush Dumbhare³, Kunal Kanchankar⁴

Department of Computer Science and Engineering, GH Rasoni University, Amravati

Abstract: Modern datacenters must balance three objectives that often conflict: keeping hardware within safe thermal limits, meeting latency expectations for high-priority work, and reducing the cost and carbon footprint of electricity. Traditional controllers handle these with fixed rules or heavily engineered optimization pipelines. This paper presents Grid Control, an autonomous datacenter orchestration system in which a large language model (LLM) acts as the decision-making agent. A simulated multi-region datacenter (5 regions, 15 servers, 60 chipsets) streams telemetry every second. Every decision interval the agent receives a frozen telemetry window and selects exactly one action from a small, closed vocabulary: `shift_within_server`, `shift_within_region`, `migrate_region`, or `pass`. A prioritized policy ranks overheating first, latency and load balance second, and cost and carbon third. Safety is enforced outside the model by a guardrail layer that routes high-criticality or administrator-prohibited actions to a human approval queue. A feedback loop returns measured outcomes of past actions to the prompt as in-context learning signal, and a scope-restricted explainability module answers operator questions about the agent's behavior. Over 20 paired simulation runs, a deterministic stand-in for the agent's policy reduced time spent above the 68 °C threshold by about 24% and carbon-weighted load by about 9% relative to a no-agent baseline, with about 3% lower electricity cost. LLM-in-the-loop evaluation is left to future work. The contribution is an architecture and a reproducible testbed for studying safe, explainable, LLM-driven infrastructure control.

Keywords: LLM agents, datacenter orchestration, carbon-aware computing, workload migration, thermal management, guardrails, explainable AI, human-in-the-loop control, in-context feedback.

I. INTRODUCTION

Datacenters are a foundation of the digital economy and a growing consumer of electricity. Global datacenter energy use stayed relatively flat between 2010 and 2018 thanks to large efficiency gains, even as compute output grew several-fold [1]. Those gains are harder to repeat, and demand from AI workloads is pushing utilization and power density upward. Operators therefore need control systems that are efficient and also adaptive to heterogeneous conditions: chips that overheat under bursty load, regional electricity prices that change with the hour, and grids whose carbon intensity differs by more than an order of magnitude between locations. Two lines of prior work address parts of this problem. Carbon-aware computing shifts flexible work in time or space toward cleaner or cheaper energy [2], [3]. Learning-based controllers reduce cooling energy or improve scheduling [4], [5], and cluster managers such as Borg show how large fleets are scheduled in production [6]. These systems are powerful but typically demand bespoke models, forecasting pipelines, and significant engineering for each new objective or constraint. LLM-based agents offer a different trade-off. Given a structured description of the state and a natural-language policy, an LLM can weigh several objectives, justify its choice, and be re-targeted by editing a prompt rather than retraining a model [7]. The risks are equally clear: LLM outputs can be malformed, stale, or unsafe, and giving a model unrestricted control over infrastructure is unacceptable [9]. This motivates a design in which the model proposes, deterministic code disposes, and humans keep authority over irreversible operations. This paper makes four contributions:

- 1) A closed-vocabulary LLM agent that makes one placement decision per telemetry window under an explicit priority ordering (thermal, latency/balance, cost/carbon).
- 2) A guardrail layer that separates low-criticality actions the agent may execute from high-criticality or admin-prohibited actions that become warnings for human approval, in the spirit of programmable LLM rails [10].
- 3) An in-context feedback loop that measures the effect of each executed action and feeds the results back into the next prompt, similar in spirit to verbal self-reflection approaches [8] but without weight updates.

II. LITERATURE SURVEY

Prior work relevant to Grid Control spans carbon-aware and energy-efficient datacenters [1]–[3], learning-based cooling and scheduling [4]–[6], LLM agents [7], [8], and AI safety and guardrails [9], [10]. Table I summarizes each work and its relation to this paper.

Table I. Summary of related work

Ref.	Authors (Year)	Focus / Method	Key Contribution	Limitation / Gap	Relevance to Grid Control
[1]	Masanet et al. (2020)	Bottom-up estimate of global datacenter energy use	Energy use stayed relatively flat, while compute grew several-fold via efficiency gains	Descriptive; no control mechanism; gains hard to repeat	Motivates adaptive, efficiency-aware operation
[2]	Radovanović et al. (2022)	Carbon-aware computing in production	Shifts flexible workloads in time toward lower-carbon hours	Needs forecasting pipelines; mainly temporal shifting	We shift spatially across regions via a prompt-defined policy
[3]	Wiesner et al. (2021)	Temporal workload shifting	Delaying delay-tolerant jobs reduces carbon; benefit depends on region and flexibility	Only deferrable workloads; no thermal or latency goals	Complements our migration; temporal shifting is future work
[4]	Lazic et al. (2018)	Model-predictive control for cooling	Data-driven MPC lowers cooling energy vs. baseline controller	Needs learned plant models; cooling-specific	We manage heat through load placement, not cooling plant
[5]	Mao et al. (2016)	Deep RL for resource management	Learned scheduling policy competitive with heuristics	Training, reward design and retraining per objective; opaque	Our agent is re-targeted by editing the prompt and explains decisions
[6]	Verma et al. (2015)	Borg cluster manager	Large-scale production scheduling and utilization	Heavy engineering; rule-driven; no LLM or explainability	Reference for production scheduling and placement
[7]	Yao et al. (2022)	ReAct: reasoning plus acting in LLMs	Interleaving reasoning with actions improves decision tasks	General benchmarks; no infrastructure safety	Basis for LLM that reasons then picks an action
[8]	Shinn et al. (2023)	Reflexion: verbal reinforcement learning	Agents learn from textual feedback without weight updates	Language/coding tasks, not infrastructure	Inspires our in-context feedback loop
[9]	Amodei et al. (2016)	Concrete problems in AI safety	Identifies safe exploration, side effects	Conceptual; no implementation	Justifies bounded actions, fail-safe
[10]	Rebedea et al. (2023)	NeMo Guardrails	Programmable rails to constrain LLM behaviour	Conversational focus; no live-state validation	Our guardrail gate applies the idea to infrastructure actions

Research gap: Existing controllers [2]–[6] need bespoke models and offer little explanation, while LLM-agent work [7], [8] and safety work [9], [10] rarely combine a bounded action space, deterministic guardrails, human approval and in-context feedback for infrastructure control. Grid Control addresses this gap on a reproducible simulated testbed.

III. METHODOLOGY

A. Design principles

The system follows four principles. Bounded action space: the model can only choose among four named actions, each with a fixed parameter schema. Separation of proposal and execution: the LLM never mutates state directly; its JSON output passes through validation, the guardrail gate, and environment-side precondition checks. Fail safe: any parse failure, invalid action, or provider error degrades to pass (or is logged as an error) rather than to an unsafe action. Human authority over risk: anything irreversible, or anything an administrator has forbidden, is surfaced for approval.

B. System Overview

The system has six cooperating components, arranged as a sense, decide, gate, act, observe loop:

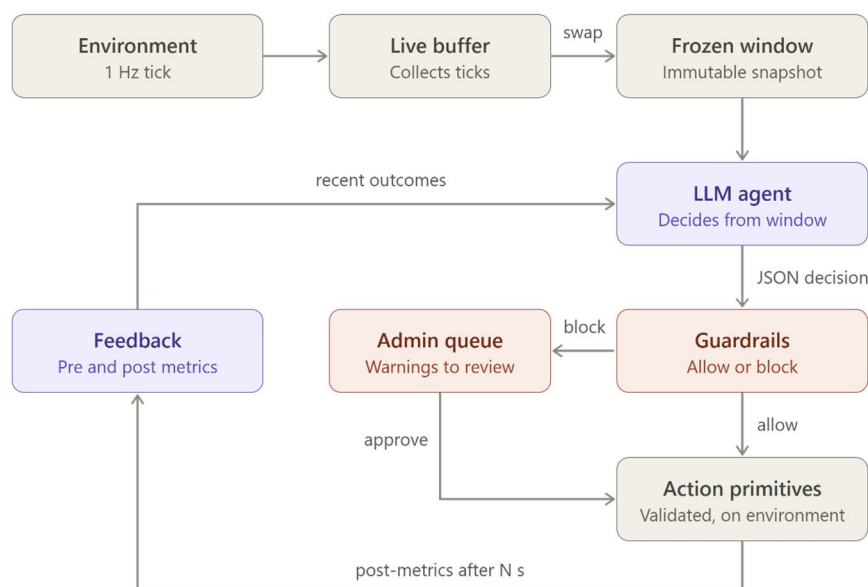


Fig no. 1.1 System Overview

C. Simulated Environment

The testbed models a hierarchy of regions → servers → chipsets. By default there are 5 regions, 3 servers per region, and 4 chipsets per server (60 chipsets). The first server in each region is a powerful tier (capacity 80 load units per chipset); the others are standard (capacity 45). Regions are modeled on real-world contrasts (Table 1): a low-carbon but expensive grid (Denmark), a cheap geothermal/hydro grid (Iceland), and carbon-heavy grids (Virginia, Mumbai). A fifth region, São Paulo, sits in between with moderate carbon intensity and price. Because no single region is best on carbon, price and ambient temperature at the same time, the agent has to trade these objectives against each other when deciding where to place or migrate work. Together with the two server tiers, this gives the agent a heterogeneous environment, where the best placement for a task depends on its priority and load as well as on the region's current electricity price and carbon intensity.

Table 1. Region profiles in the simulator

Region	Energy	Source	Carbon (gCO ₂ /kWh)	Base price (\$/kWh)	Ambient (°C)
us-east (Virginia)		Fossil	650	0.12	18
eu-north (Iceland)		Green	20	0.05	5
eu-west (Denmark)		Green	11	0.28	9
ap-south (Mumbai)		Fossil	450	0.09	28
sa-east (São Paulo)		Green	90	0.08	24

Electricity price follows a day/night curve: the base price is multiplied by 1.35 between 08:00 and 22:00 and by 0.75 otherwise. The simulated clock runs 60× faster than wall time so that a full day/night cycle is observable in 24 real minutes.

Workload. Each second, 0 to 3 tasks arrive. Each task is one of four types (image, text, video, code generation) with type-specific ranges for duration, load, and priority (video is highest priority and heaviest; text is lowest and lightest). New tasks are placed naively on a random region and server, on the least-utilized chipset in it, leaving meaningful room for the agent to improve placement.

Thermal model. Each chipset's temperature evolves as a discrete first-order system:

$$T(t + \Delta t) = T(t) + 6u\Delta t - 0.08(T(t) - T_{amb})\Delta t$$

where u is utilization. The steady state is $T = T_{amb} + 75 \cdot u$, so a chip crosses the 68°C threshold at a utilization of about 0.53 in Mumbai (28°C ambient), 0.67 in Virginia, and 0.84 in Iceland. This makes thermal risk region-dependent, which is realistic and non-trivial for the agent.

D. Decision Protocol and Priorities

Telemetry is stored in a double buffer. A live buffer fills for the next window while a frozen buffer is what the agent reasons over, so slow LLM calls never block data collection. A scheduler freezes one window every D second (default 5 s in code, 20 s in the example configuration) and dispatches the decision to a worker thread, keeping the clock fixed regardless of model latency.

The system prompt encodes an ordered policy:

- 1) Thermal first. If any chipset exceeds 68°C , move one eligible task to a cooler chipset on the same server (`shift_within_server`).
- 2) Latency and balance involve moving high-priority tasks to less-loaded servers within the same region when disparities in average utilization exceed 0.35, ensuring efficient resource allocation.
- 3) Cost and carbon third. If another region is cheaper or greener and `projected_savings - bandwidth_cost_estimate > MIN_NET_PROFIT`, migrate one task (`migrate_region`).

Otherwise pass. `pass` is explicitly described as a correct and expected decision.

Hard constraints in the prompt forbid shutting down servers or regions, moving all tasks off a server, moving locked tasks, or choosing actions outside the list. The administrator's current "don't" rules are injected into the prompt so the model can avoid violating them in the first place.

E. Guardrails and human-in-the-loop control

Prompt instructions are not a safety mechanism on their own, so every decision passes through a deterministic gate that returns allow, pass, or block:

Low-criticality actions (`shift_within_server`, `shift_within_region`, `migrate_region`) may execute autonomously.

High-criticality actions (shutdowns, bulk moves, task deletion) are never executable by the agent; they are converted to warnings.

The current agent cannot produce them, but the gate is a safety net for a future, broader action vocabulary.

Admin "don't" rules are free-text (for example, "never migrate to ap-south"). A rule fires when all of its non-stopword tokens appear as whole words in the action name plus payload. Requiring all tokens, rather than any one, prevents over-blocking from common words; whole-word matching prevents substring accidents.

Unrecognized actions are blocked. Blocked actions appear in an authenticated admin console, where an administrator can approve or deny them. Approved low-criticality actions are executed and logged as administrator overrides. Rules persist to disk.

F. Feedback loop

When an action succeeds, the feedback module records pre-action metrics (temperature, utilization, carbon intensity, electricity price) of the affected entity. After a configurable window (default 30 s) it samples the same metrics and stores the deltas. The latest resolved entries are appended to the next prompt as "recent action outcomes". This is in-context adaptation: no weights change, but the model can see whether its earlier choices helped, in the spirit of verbal reflection on past trials [8].

G. Explainability module

Operators can ask natural-language questions about the system. For example, they can ask why a task was migrated, why an action was blocked, or which rules are currently active. A two-stage scope guard handles them. A cheap local keyword check rejects clearly unrelated questions without any API call. Questions that pass are sent to the LLM with a strict system prompt that restricts it to a JSON context of the latest snapshot, the five most recent decisions, open warnings, and admin rules. Because answers are grounded only in this recorded context, they describe what the system actually did and observed, rather than the model's general assumptions. The interface is also read-only, so asking a question can never change the state of the datacenter. A per-IP rate limit protects the public endpoint from cost abuse. Together, these measures make the agent's behavior transparent to operators while keeping token cost and exposure low.

IV. TECHNICAL EXPLANATION

A. Software Architecture

The backend is a Python FastAPI service. Two background threads drive the system: a 1 Hz environment tick thread and the agent scheduler, with each decision executed on its own short-lived worker thread. Shared state is protected by locks (a re-entrant lock in the environment, separate locks for the decision log, session table, rate limiter, and feedback log). A static web dashboard polls REST endpoints for state, decision log, feedback, and questions; an admin page handles login, warnings, and rules.

B. Provider-agnostic LLM interface

A thin LLMClient wraps three providers (Anthropic, OpenAI, Gemini) behind one complete (system, user) call, selected by environment variables. For Gemini, a JSON response schema with an enumerated action field constrains output at the API level. Provider exceptions are normalized into short, actionable messages (quota exhaustion, rejected key), which matters because the decision log is an operator-facing surface.

C. Output Validation

The model must return a JSON object with reasoning, action, and params. The parser strips Markdown fences, extracts the outermost braces, and validates: object type, action in the allowed set, params an object, reasoning a string. Any failure yields a safe pass decision. This pattern means malformed output costs one missed opportunity rather than a fault.

D. Execution-time safety in the environment

Even after the guardrail gate, each action primitive re-validates against live state under a lock:

- Staleness. The task must still exist, not be completed, and still reside on the claimed source chipset, server, or region. This handles the gap between the frozen snapshot the model saw and the present.
- Locking. A moved task is locked and cannot be moved again, which prevents oscillation (thrashing) of the same task.
- Headroom. The destination must have capacity (for intra-server moves, below 95% utilization and sufficient load headroom).
- Topology. Source and destination must belong to the same server (intra-server), same region (intra-region), and be different regions (migration).
- Economics. For migrations, `projected_savings - bandwidth_cost_estimate` must exceed the configured `MIN_NET_PROFIT`. The system enforces this floor itself and overwrites any value the model tries to pass.

Each primitive returns a structured {ok, reason} result, so failures are logged and explainable rather than silent.

E. Concurrency considerations

when i take the cursor near any dot it gives the name of that dot like the profile name or course Because the LLM call may take seconds, decision processing is decoupled from the scheduler: the scheduler freezes windows at fixed intervals and never waits for model completion. The trade-off is that an action may be applied to a state that has moved on. The staleness checks above are the mitigation. The feedback module's post-action sampling runs on daemon timers so it cannot block decisions. Together, these choices keep the decision cadence predictable regardless of model latency, while the shared-state locks prevent concurrent threads from corrupting the environment.name etc

F. Security and cost controls

Admin endpoints require a bearer token issued at login with a 4-hour time-to-live. The public question endpoint is rate-limited (default 6 requests per minute per client IP) with a fixed window, adequate for a single-process demo. The scope guard avoids spending tokens on off-topic questions. Together, these controls ensure that only authenticated administrators can approve or deny blocked actions and edit the persisted "don't" rules, while anonymous users are limited to read-only, scope-restricted queries. These measures are sufficient for a demonstration setting, but a production deployment would additionally need distributed rate limiting, proper identity management.

V. EXPERIMENTATION

A. Research questions

- RQ1: Does a policy following the agent's priority ordering reduce thermal excursions compared with naive placement?
- RQ2: Does it reduce carbon-weighted load and electricity cost?
- RQ3: Do the execution-time safety checks keep failed or stale actions rare?

B. Setup

We ran the actual environment.py simulator with the default topology (5 regions $\times$ 3 servers $\times$ 4 chipsets), a 60 $\times$ simulated clock, and a decision every 20 s. Each run lasts 1,800 simulated-environment ticks (30 real-time minutes, or 30 simulated hours), giving 90 decision points per run. We ran 20 paired seeds; the baseline and policy run with the same seed see identical task arrival streams. This paired design ensures that any difference in outcomes is attributable to the placement policy rather than to random variation in the workload.

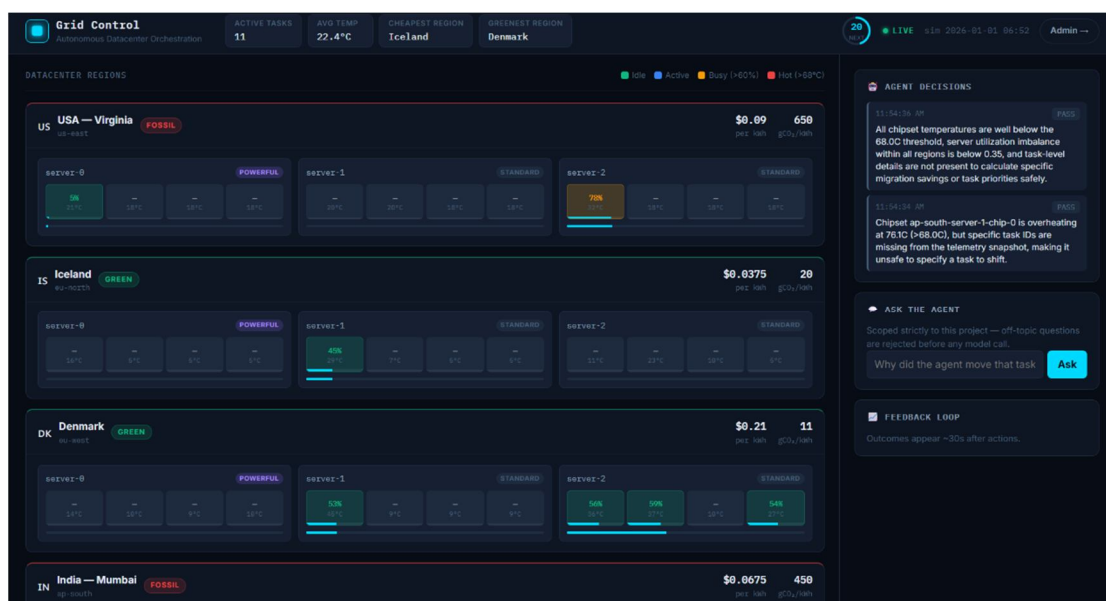
Important scope note. The experiment did not call an LLM. To isolate the value of the decision *policy* and the environment's action primitives from model variability, we implemented a deterministic stand-in that follows the same four-step priority order as the agent's system prompt (same 68 °C and 0.35 imbalance thresholds, one action per decision, minimum net profit 0). Its results therefore characterize the policy and the testbed, not any particular LLM. The stand-in's migration savings estimate is a simple heuristic of our own (load $\times$ remaining time $\times$ differences in price and carbon), with a fixed bandwidth cost of 0.002. Outcomes are averaged over the 20 seeds and reported as mean $\pm$ standard deviation, so the results reflect typical behavior rather than a single favorable run. Keeping the stand-in simple and deterministic also gives a reproducible reference that future LLM-in-the-loop runs can be compared against directly.

Metrics. (1) Percentage of chipset-seconds above 68 °C; (2) peak chipset temperature; (3) carbon-weighted load, Σ load $\times$ gCO₂/kWh, in thousands; (4) cost-weighted load, Σ load $\times$ price; (5) counts of executed and failed actions.

C. Results

Table 2. Baseline vs. policy, mean $\pm$ std over 20 paired seeds

Metric	Baseline (no agent)	Policy (agent stand-in)	Change
Chipset-time above 68 °C (%)	0.41 $\pm$ 0.09	0.31 $\pm$ 0.08	−24%
Peak temperature (°C)	81.7 $\pm$ 2.1	80.3 $\pm$ 1.9	−1.4 °C
Mean chipset temperature (°C)	22.50 $\pm$ 0.20	22.46 $\pm$ 0.20	about 0
Carbon-weighted load ($\times 10^3$)	103,288 $\pm$ 3,772	93,899 $\pm$ 4,465	−9.1%
Cost-weighted load	58,513 $\pm$ 2,514	56,905 $\pm$ 2,559	−2.7%



Photograph 5.1: Grid Control Dashboard Showing the Regions, Agent Decisions, Ask the Agent and Feedback Loop Panels

This screenshot shows a real problem you should know about. Both agent decisions in it are PASS. The second one says that chipset ap-south-server-1-chip-0 is overheating at 76.1 °C, but the agent chose not to act because the specific task IDs are missing from the telemetry snapshot.

D. Discussion

RQ1 (thermal). Time above the threshold fell by roughly a quarter and peak temperature fell by about 1.4 °C. The effect is modest in absolute terms because the default workload is light: the average chip sits close to ambient, and overheating is a rare tail event. Mean temperature is essentially unchanged, which is expected since the policy redistributes heat rather than removing it. The thermal benefit is concentrated where it matters, in the tail.

RQ2 (carbon and cost). Carbon-weighted load dropped about 9%, a bigger effect than cost (about 3%). This is consistent with the region table: carbon intensity varies by about 60× across regions (11 to 650 g/kWh) while prices vary by about 5.6× and are partially offset by bandwidth cost and the day/night multiplier. Because tasks are short (seconds) and a migrated task is locked afterward, each migration captures only a small saving, so gains accumulate from volume (58 migrations per run) rather than from individual moves.

RQ3 (safety checks). Only 0.25 of 90 actions per run were rejected by the environment's precondition checks (under 0.3%), so stale-snapshot and headroom failures were rare under the stand-in's deterministic choices. An LLM will produce more varied and occasionally erroneous parameters, so this number should be re-measured with real models.

E. Threats to validity

The workload is synthetic and uniformly random; real demand is bursty and correlated. The thermal model is first-order and ignores airflow, neighboring-chip coupling, and cooling-plant dynamics. Migration overhead is reduced to a scalar bandwidth cost with no latency penalty. Most importantly, the stand-in is not an LLM, and the evaluation does not test guardrail blocking, feedback-loop benefit, or explanation quality. We report these results as a policy-level baseline that future LLM runs should be compared against.

VI. CONCLUSION

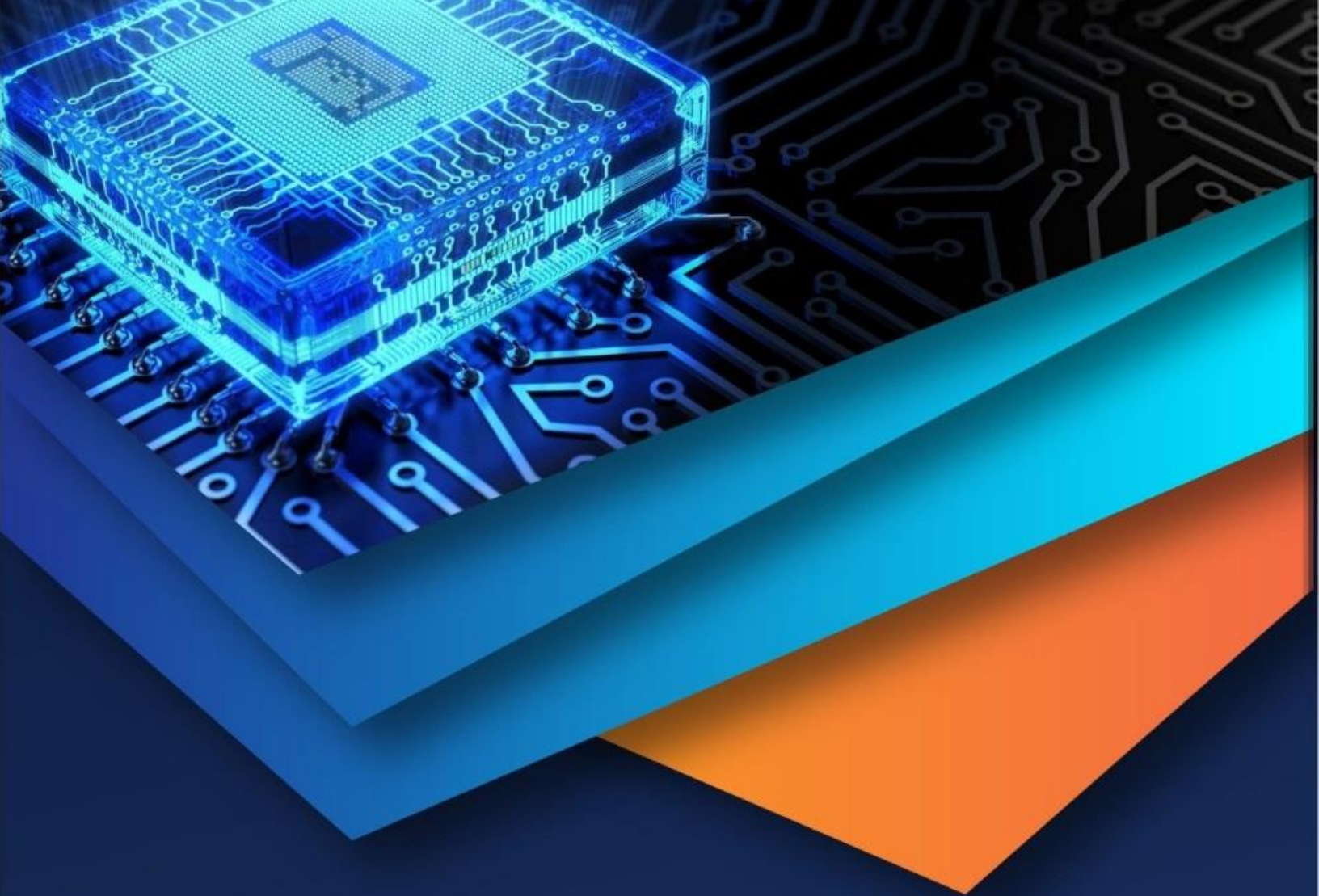
This paper presented an AI agent-driven datacenter optimizer that combines an LLM decision-maker with a bounded action vocabulary, deterministic guardrails, human approval for risky actions, in-context outcome feedback, and a scoped explainability interface, all on an open simulated testbed. The architecture's central idea is that an LLM is most useful and least dangerous when it is treated as an untrusted proposer: its output is schema-validated, policy-gated, and re-checked against live state before any effect. Offline experiments on the testbed show that following the prescribed priority order reduces thermal excursions (−24% time above threshold) and carbon-weighted load (−9%) with modest cost savings (−3%) relative to naive placement, while almost never producing invalid actions. The framework provides a practical starting point for studying safe, transparent, natural-language-configurable infrastructure control.

VII. FUTURE WORK

- 1) LLM-in-the-loop evaluation. Compare Claude, GPT, and Gemini models against the deterministic baseline on decision quality, latency, token cost, and invalid-output rate, and run ablations (no feedback, no guardrails, no admin rules in the prompt).
- 2) Realistic workloads and telemetry. Replay public cluster traces, use real hourly grid carbon-intensity and price data, and add demand bursts and deadlines.
- 3) Richer physics and economics. Model cooling plants, inter-chip thermal coupling, migration latency and data-transfer cost, and power-capping.
- 4) Temporal shifting. Add deferral of flexible jobs to cleaner hours, which has been shown to be effective in production carbon-aware systems [2], [3].
- 5) Stronger guardrails. Replace keyword matching of admin rules with structured, machine-checkable constraints or semantic matching, add audit trails, and formally bound the agent's blast radius [9].
- 6) Better learning signals. Move from last-N outcome summaries to retrieval of similar past situations, and compare with reinforcement-learning schedulers [5].
- 7) Multi-step and multi-agent planning. Allow batches of coordinated moves, with conflict resolution, instead of one task per window.
- 8) Production hardening. Persistent storage, distributed rate limiting, proper identity management, and hashed admin credentials.
- 9) Stress-test the agent against corrupted, delayed or missing telemetry, prompt-injection through task names or payload fields, and sensor faults.
- 10) Replace the fixed priority ordering (thermal, then latency and balance, then cost and carbon) with configurable or learned weights.

REFERENCES

- [1] Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020). Recalibrating global data center energy-use estimates. *Science*, 367(6481), 984-986.
- [2] Radovanović, A., Koningstein, R., Schneider, I., Chen, B., Duarte, A., Roy, B., ... & Cirne, W. (2022). Carbon-aware computing for datacenters. *IEEE Transactions on Power Systems*, 38(2), 1270-1280.
- [3] Wiesner, P., Behnke, I., Scheinert, D., Gontarska, K., & Thamsen, L. (2021, December). Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In *Proceedings of the 22nd International Middleware Conference* (pp. 260-272).
- [4] Lazic, N., Boutilier, C., Lu, T., Wong, E., Roy, B., Ryu, M. K., & Imwalle, G. (2018). Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems*, 31.
- [5] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016, November). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks* (pp. 50-56).
- [6] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015, April). Large-scale cluster management at Google with Borg. In *Proceedings of the tenth european conference on computer systems* (pp. 1-17).
- [7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- [8] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36, 8634-8652.
- [9] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.
- [10] Rebedea, T., Dinu, R., Sreedhar, M. N., Parisien, C., & Cohen, J. (2023, December). Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. In *Proceedings of the 2023 conference on empirical methods in natural language processing: system demonstrations* (pp. 431-445).



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)