MARLIN: Multi-Agent Game-Theoretic Reinforcement Learning for Sustainable LLM Inference in Cloud Datacenters

Hayden Moore
Colorado State University
Fort Collins, CO, USA
hmoore01@colostate.edu

Sirui Qi
Colorado State University
Fort Collins, CO, USA
alex.qi@colostate.edu

Dejan Milojicic
Hewlett Packard Labs
Milpitas, CA, USA
dejan.milojicic@hpe.com

Cullen Bash
Hewlett Packard Labs
Milpitas, CA, USA
cullen.bash@hpe.com

Sudeep Pasricha
ECE
Colorado State University
Fort Collins, CO, USA
sudeep@colostate.edu

Abstract

Large Language Models (LLMs) have become increasingly prevalent in cloud-based platforms, propelled by the introduction of AI-based consumer and enterprise services. LLM inference requests in particular account for up to 90% of total LLM lifecycle energy use, dwarfing training energy costs. The rising volume of LLM inference requests is increasing environmental footprints, particularly carbon emissions and water consumption. To improve sustainability for LLM inference serving in cloud datacenter environments, we propose a novel multi-agent game-theoretic reinforcement learning framework called MARLIN to co-optimize time-to-first token (TTFT), carbon emissions, water usage, and energy costs associated with LLM inference. MARLIN demonstrates a reduction of at least 18% in TTFT, 33% in carbon emissions, 43% in water usage, and 11% in energy costs compared to state-of-the-art LLM inference management frameworks.

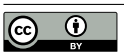
CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Information systems**; • **Computer systems organization** → **Real-time systems**;

Keywords

Large language models, Sustainability, Carbon emissions, Water usage, Energy costs, Cloud datacenters, Reinforcement learning

ACM Reference Format:
Hayden Moore, Sirui Qi, Dejan Milojicic, Cullen Bash, and Sudeep Pasricha. 2026. MARLIN: Multi-Agent Game-Theoretic Reinforcement Learning for Sustainable LLM Inference in Cloud Datacenters. In *International Green and Sustainable Computing Conference (IGSC 2026), June 22–24, 2026, Canandaigua, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3797248.3815404



IGSC 2026, Canandaigua, NY, USA

ACM ISBN 979-8-4007-2520-3/26/06
https://doi.org/10.1145/3797248.3815404

1 Introduction

Large Language Models (LLMs) have experienced explosive growth in recent years, with enterprise adoption expected to exceed 80% in 2026 and global generative AI spending projected to reach $644 billion [20]. This growth is reflected in usage data from large AI companies, with OpenAI reporting over 2.5 billion prompts per day from users across its ChatGPT deployments in 2025 [28]. With this rapid growth in LLM adoption, the associated energy usage in cloud datacenters hosting these LLMs is also rapidly increasing, with an estimated 25-33% compound annual growth rate [16].

The rapid increase in LLM-associated energy consumption in cloud datacenters is accelerating the already high datacenter energy footprint. In 2024, U.S. datacenters consumed 183 terawatt-hours (TWh) of electricity, which is 4% of the total electricity generated in the country. By 2030, this figure is projected to grow by 133% to 426 TWh [15]. This increase is expected to be driven by demand for both LLM training and serving user inference requests. In the past, the training phase was the focus for improving the sustainability of LLMs. However, recent data suggest that inference dominates LLM lifecycle energy usage, accounting for 90% of the overall energy used in this stage [12]. Hence, it is increasingly important to focus on reducing energy costs and improving the sustainability of LLMs by reducing the resources required to serve LLM inference requests.

We target two important sustainability metrics: operational carbon emissions and water usage. Carbon emissions have been a long-term concern associated with high energy usage. LLM inference is currently scaling quickly and is already becoming a concern for carbon emissions. As an example, the yearly carbon emissions from the inference serving of just the GPT-4o model already surpass those of 30,000 cars in a year [12]. As more competing LLMs launch and infrastructure continues to scale, emissions will only climb.

The high water usage associated with cooling LLM-serving datacenters and during electricity generation is also becoming a concern. Cooling accounts for an estimated 29% of datacenter water usage, with electricity generation contributing 71%, so both factors must be considered [33]. For GPT-4o inference only, its annual water consumption is up to 1,579,680 kL, enough to fill more than 500 Olympic-sized pools [12]. Importantly, this consumption refers to evaporated freshwater permanently removed from local ecosystems, straining the infrastructure of large cities, especially since

an estimated 30% of cities with over 1 million people worldwide already are facing water scarcity [9].

Clearly, the current path to scaling LLM inference is unsustainable. There is a critical need for serving LLM inference sustainably without sacrificing performance. To achieve this goal, we propose a novel framework called MARLIN, to optimize carbon emissions, water usage, and energy costs associated with LLM inference while ensuring that performance (time-to-first-token (TTFT)) remains acceptable for each incoming request. The novel contributions of our work are:

- We develop a novel multi-agent reinforcement learning framework called MARLIN that utilizes a game-theoretic approach to balance among competing agents.
- We formulate a multi-objective LLM inference scheduling problem that accurately models the TTFT, carbon emissions, water usage, and energy costs of LLM serving across geo-distributed cloud datacenters.
- We perform comprehensive comparison studies with several state-of-the-art LLM inference scheduling frameworks to demonstrate the flexibility and promise of the MARLIN framework.

2 Related Works

Many prior efforts have addressed workload scheduling in cloud datacenters. The primary focus of optimizing workload scheduling in these works has been to improve performance and reduce overall energy consumption. For instance, Hogade et al. proposed a game-theoretic framework that incorporates network transfer costs to minimize datacenter energy costs [11]. Chen et al. used ant colony optimization to consolidate VMs, reducing both power consumption and thermal hotspots [5].

Recent efforts have shifted towards optimizing sustainability metrics, particularly carbon emissions and water usage. To address the rising datacenter carbon emissions, Qi et al. developed a dual-objective evolutionary optimizer that improved the carbon emissions and throughput of serverless function execution across geo-distributed datacenters [23]. Beena et al. proposed a greedy scheduling heuristic that worked across many distributed datacenters and used an LSTM model for carbon emission prediction [4]. Qi et al. minimized carbon emissions, wastewater, and costs using a hybrid evolutionary-boosting algorithm [24]. These prior approaches do not address sustainable LLM inference scheduling.

Most existing LLM inference schedulers prioritize datacenter performance metrics. For example, Mei et al. used mixed-integer linear programming to maximize data flow across distributed GPUs to maximize LLM inference throughput [17]. Patel et al. divided LLM inference into distinct phases, enabling pipeline parallelism to improve throughput [21]. Yang et al. developed a framework that leveraged edge and cloud servers using constraint satisfaction and upper confidence bounds, thereby improving throughput and costs [34]. Recent work is starting to focus on sustainability in LLM inference scheduling. Moore et al. combined a genetic algorithm and machine learning to generate Pareto-optimal solutions for LLM inference serving [19]. However, the approach lacks scalability and has a slow convergence speed.

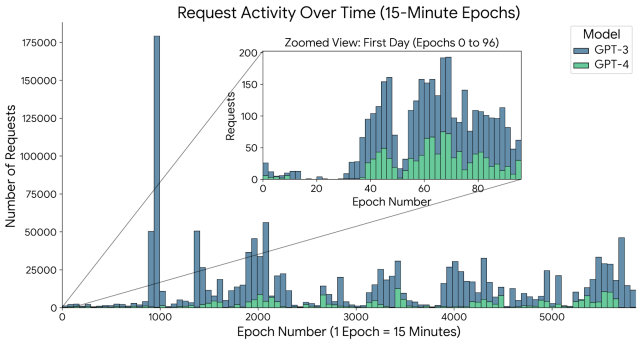


Figure 1: The number of individual LLM requests in each epoch (15 minutes) over two weeks [32]

MARLIN addresses these limitations with a multi-agent game-theoretic RL approach that improves convergence speed, preserves optimization quality, and balances sustainability with TTFT performance.

3 Modeling Overview

Our framework is designed to map incoming LLM inference requests to server nodes across geo-distributed datacenters to leverage natural geographic and temporal variations. We comprehensively model the LLM inference workload and cloud datacenter characteristics across geographic locations, capturing energy costs, carbon emissions, water usage, and data transmission latency for each request.

3.1 LLM Workload Model

Our workload model uses a real-world trace [32] from a ChatGPT service on Azure cloud datacenters. The dataset covers two weeks' worth of requests for GPT-3 and GPT-4. We aggregate requests by epoch (15-minute intervals) and plot their frequencies in Fig. 1. The distribution of LLM inference requests across epochs is quite diverse, highlighting the challenge of scheduling them across cloud datacenters. We created a custom workload model that used the observed arrival patterns from the real cloud trace and paired them with execution models for two contemporary LLMs, Llama-7B and Llama-70B. Execution profiles for these two LLMs were available across multiple GPUs [30].

Real-time LLM serving requires meeting strict memory constraints. Memory constraints require that each request i's memory footprint MF_i not exceed a server node n's maximum available memory. Requests share the underlying model's weights when possible, to reduce memory overhead. We account for shared weights for a model v's memory footprint MF_v, and each request's growing KV cache $KV_{(v,i,\tau)}$ as a function of the number of tokens τ. KV cache growth is proportional to the model size. The total server node memory usage $M_{(n,e)}$ computed for all tokens T at epoch e is:

$$M_{n,e} = (\sum_{v}^{\Upsilon} MF_v) + (\sum_{i}^{I_e} \sum_{\tau}^{T} KV_{v,i,\tau}) \quad (1)$$

where Υ is the total number of all active LLMs and I_e is the total number of requests at epoch e. If this memory usage exceeds the

node n's maximum available memory $M_{tot,n}$, it results in a queuing of requests until prior requests release their KV cache. Real-time LLM serving also requires meeting strict latency constraints. As many LLM applications (e.g., chatbots) require real-time interactivity, minimizing latency LA_i for each request is critical. Latency in this case has three components: LLM weight loading time, network latency, and computation time.

Weight loading time $LA_{load,i}$ depends on whether the model is present on the GPU during execution. We assume each model is present on the node locally in storage. However, the model cannot be used until it is loaded onto the onboard memory. The worst-case loading time depends on the slowest memory bandwidth BW_n in moving the model to the onboard memory. Therefore, the model weight loading time can be expressed as: $LA_{load,i} = MF_v/BW_n$. To reduce loading time for additional requests, a node keeps the model present in onboard memory.

The network latency $LA_{(net,i)}$ consists of moving the user request to a datacenter. The transfer latency of the user request to the datacenter is dependent on the distance $dist$ and transmission media characterized by λ_{media} [6], which is the propagation latency per unit of distance (e.g., milliseconds per kilometer). Requests may also need to be transmitted between datacenters via an inter-datacenter network, which depends on hop count $R_{source,dest}$ as well as the latency of processing each hop σ_{hop}. The network latency $LA_{net,i}$ can be expressed as:

$$LA_{net,i} = (dist \times \lambda_{media}) + (R_{source,dest} \times \sigma_{hop}) \tag{2}$$

At the scheduled server node n, computation proceeds until the first output token τ is generated. We assume profiled execution time $LA_{(tot,exec,i)}$ scales linearly with output token count T_i. We can safely make this assumption due to batched LLM inference being memory bound on GPUs [25]. This bounds the throughput giving the execution time a linear relationship.

To calculate the overall TTFT, all latency components must be combined, with network latency counted twice in the final calculation to account for both sending and receiving the request. Therefore, the total request TTFT across all requests I during an epoch e can be estimated as:

$$LA_{tot,e} = \sum_{i}^{I_e} (LA_{load,i} + 2 \times LA_{net,i} + LA_{tot,exec,i}/T_i) \tag{3}$$

3.2 Datacenter Model

Each geo-distributed datacenter d comprises N_d nodes at a location. All datacenters that are present in the geo-distributed network are defined by D, where each datacenter is present at a unique location. Nodes are organized in standard hot/cold aisle layouts with conventional air cooling [29]. Nodes vary in GPU types and configurations. Each node n contains identical GPUs whose memory is pooled to serve incoming requests. We consider the NVIDIA A100 and H100 as the GPUs present in the nodes. The number of GPUs per node can be 2, 4, or 8, allowing each node to host models of varying complexity.

3.3 Energy Cost Model

Each server node n can operate in preset performance states, each of which dissipates a fraction PC_{pstate} of the node's total thermal design power TDP_n. The information technology (IT) node energy over an epoch e of duration t_e is computed as:

$$E_{IT,n,e} = PC_{pstate} \times TDP_n \times t_e \tag{4}$$

The total datacenter IT energy over an epoch is the sum of contributions from all its server nodes:

$$E_{IT,d,e} = \sum_{n}^{N_d} E_{IT,n,e} \tag{5}$$

Datacenter energy also includes cooling energy $E_{cool,d,e}$ and infrastructure energy $E_{infra,d,e}$. Cooling energy consists of the energy required to run the computer room air conditioning (CRAC) units. CRAC efficiency (COP) varies by datacenter, COP_d. Across an epoch e, the CRAC energy is $E_{CRAC,d,e} = E_{IT,d,e}/COP_d$. Since chiller energy usage and auxiliary equipment energy usage are approximately equivalent to $E_{CRAC,d,e}$ [35], the total datacenter cooling energy is: $E_{cool,d,e} = 3 \times E_{CRAC,d,e}$.

The supporting infrastructure consists of power distribution units and power supply units in the datacenter. Infrastructure energy is equal to about 13% of the IT energy [1]: $E_{infra,d,e} = 0.13 \times E_{IT,d,e}$.

The total datacenter energy usage at a datacenter d during an epoch e consists of the sum of components:

$$E_{tot,d,e} = E_{IT,d,e} + E_{cool,d,e} + E_{infra,d,e} \tag{6}$$

However, the cost of using that energy will vary depending on the location of the datacenter d and the time-of-day when the epoch e occurs, which determines the time-of-use pricing $TOU_{d,e}$ set by the utility provider. Therefore, the total cost of energy usage across all the datacenters for epoch e is:

$$Cost_{tot,e} = \sum_{d}^{D} (E_{tot,d,e} \times TOU_{d,e}) \tag{7}$$

3.4 Water Usage Model

Water is used in two ways in a datacenter: for direct cooling and for energy generation. The two ways can be further divided into three sub-components, which are location-specific: evaporative $G_{E,d,e}$, blowdown $G_{blow,d,e}$, and grid-based $G_{grid,d,e}$. Evaporative water loss is water that exits through the cooling towers, which is proportional to the cooling load $H_{cool,d,e}$ and the latent heat of vaporization J_{water}: $G_{E,d,e} = H_{cool,d,e}/J_{water}$. CRACs require a coolant (typically water) for cooling, which must be treated when pollutant concentration reaches a threshold ϕ [27]. The treatment requires the polluted water to be sent to a blowdown-to-water treatment facility: $G_{blow,d,e} = G_{E,d,e}/(1-\phi)$. Lastly, energy generation consumes water at location-dependent intensity GI_d, which is proportional to the underlying grid makeup. The variation between sources can be dramatic, with wind using only 0.2 L/kWh, whereas hydropower uses 67 L/kWh, most of which is lost to evaporation [13]. This gives a water usage of $G_{grid,d,e} = E_{tot,d,e} \times GI_d$.

The total water usage in an epoch across all datacenters is:

$$G_{tot,e} = \sum_{d}^{D} (G_{E,d,e} + G_{blow,d,e} + G_{grid,d,e}) \tag{8}$$

3.5 Carbon Emissions Model

Carbon emissions arise from electricity generation. This has two components: grid-based emissions $Z_{grid,d,e}$ and water-treatment-based emissions $Z_{G,d,e}$. Grid-based emissions depend on carbon intensity $CI_{d,e}$ (kg of carbon per kWh produced), which depends on the energy sources in the underlying grid. This gives an emissions of: $Z_{grid,d,e} = CI_{d,e} \times E_{tot,d,e}$

Wastewater processing consumes electricity from the local grid. However, potable water creation $EI_{pot,e}$ and wastewater treatment $EI_{waste,e}$ have different energy intensities [24]. Potable water production arises from the need to replace evaporated water and process blowdown water. The potable carbon emissions are: $Z_{pot,d,e} = (G_{blow,d,e} + G_{E,d,e}) \times EIpot,e$

The operation of the wastewater treatment plants is considered in terms of how they process grid-based water usage. The wastewater carbon emissions are: $Z_{waste,d,e} = G_{grid,d,e} \times EI_{waste,e}$

Water-associated carbon emissions at a datacenter is scaled by the carbon intensity:

$$Z_{G,d,e} = (Z_{pot,d,e} + Z_{waste,d,e}) \times CI_{d,e} \tag{9}$$

The total carbon emissions in a single epoch are the sum of all sources across all datacenters D:

$$Z_{tot,e} = \sum_{d}^{D} (Z_{grid,d,e} + Z_{G,d,e}) \tag{10}$$

4 Problem Formulation

We assume a single cloud service provider (e.g., Meta) that manages its geo-distributed datacenters for serving incoming LLM inference requests. Requests originate remotely and thus incur associated transfer times that must also be considered. Our problem optimization objective is to minimize the weighted sum of TTFT, carbon emissions, water usage, and energy costs over E epochs, given resource constraints $M_{tot,n}$ and SLA constraints (SLA):

$$\begin{aligned} Min \sum_{e}^{E} (w_1 LA_{tot,e} + w_2 Z_{tot,e} + w_3 G_{tot,e} + w_4 Cost_{tot,e}) \\ s.t. M_{n,e} \leq M_{tot,n} \forall n \in N \\ LA_{tot,e} \leq LA, \forall i \in I_e \\ \sum_{j=1}^{4} w_j = 1, w_j \geq 0 \end{aligned} \tag{11}$$

Our goal is to develop a geo-distributed scheduling framework that assigns each incoming LLM inference request to a datacenter while optimizing the above objective and satisfying the constraints mentioned. Once a request is assigned to a datacenter, a local load balancer algorithm distributes requests to nodes within the datacenter. All datacenters use a modified round-robin load balancer based on [14], which ensures that nodes within a datacenter are not asymmetrically overwhelmed with requests.

5 MARLIN Framework

Fig. 2 provides an overview of the MARLIN framework, which comprises agents that participate in a two-phase competitive game. To effectively service LLM requests, a scheduling plan a is needed to pair requests with a datacenter to process them. In phase 1, each

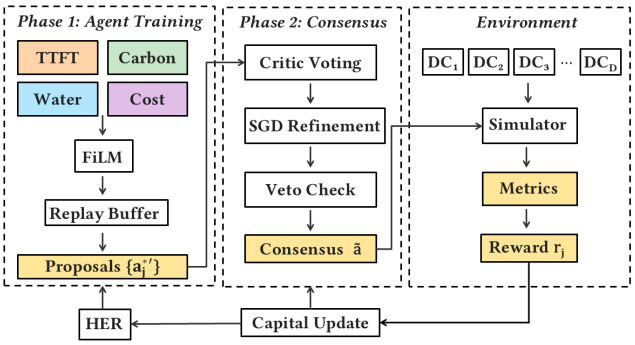


Figure 2: Overview of MARLIN framework and its two phases.

agent j independently proposes a single scheduling plan $a_j^{*'}$ that best optimizes its single objective (e.g., TTFT, carbon, water, cost). The quality of all plans $a_j^{*'}$ in phase 1 is estimated through their respective reward value r_j. Each agent maintains its own reward function to score the plans. In phase 2, all J agents negotiate via weighted voting to determine the final blended scheduling plan $\tilde{a}$ and agent-wise capital $[C_j]_{j=1}^{J}$. The final blended plan $\tilde{a}$ is determined by the agent-wise plan $a_j^{*'}$, capital threshold δ_j, and utility function Q_j. The capital threshold δ_j determines when the agent j can update the final plan according to its interest. Q_j estimates the long-term importance of each proposal (vs. the short term reward in phase 1). On top of the blended plan $\tilde{a}$, each agent j maintains a reserve of capital C_j that is updated after each epoch in the game. Capital C_j represents the influence of each agent when constituting the blended plan. We formulate phase 2 as a weighted resource-allocation game Γ which is defined as:

$$\tilde{a}, [C_j]_{j=1}^{J} = \Gamma([a_j, \delta_j, C_j, Q_j]_{j=1}^{J}) \tag{12}$$

MARLIN combines both phases with workload predictions and environmental state information to output a final blended plan $\tilde{a}$ for sustainable geo-distributed LLM inference scheduling. The following subsections describe the workload predictor, phases 1 and 2, and associated complexity analysis.

5.1 Workload Predictor

LLM request volume varies over time and across locations, as shown in Fig. 1, necessitating accurate prediction to optimize resource allocations. By forecasting workloads, MARLIN can prewarm containers and avoid startup delays. We adopt and pretrain the predictor (line 1 in Algorithm 1) from [18], which utilizes a regression-based approach. The predictor forecasts the number of LLM requests in the next epoch based on information from past epochs within a time window of tw, using an exponentially weighted moving average to estimate the incoming workload. This results in an extremely fast prediction time of roughly 100 microseconds per prediction. To evaluate the accuracy of the predictor it is compared to a baseline neural network predictor [18]. The regression-based predictor achieves over 90% accuracy in empirical validation experiments across various workload intensities and volatilities, demonstrating an average accuracy improvement of 19.01% over the baseline.

Algorithm 1 MARLIN – Phase 1 Agent Training

Input: epoch data $State_e$, $predictor$, agents J, weights $[w_j]_{j=1}^{J}$, past epochs' request volume $[I_{e-1}, \cdots, I_{e-tw}]$, and initial policy parameter θ_j
Output: scheduling plans $a_j^{*'}$ per agent j

1: Forecast request volume: $I_e = Predict(predictor, [I_{e-1}, \cdots, I_{e-tw}])$
2: Complete state information: $State_e \leftarrow I_e$
3: **for each** agent j ∈ J **in parallel do**
4: **for** $k = 1$ **to** K_{opt} iterations **do**
5: Sample action: $a_j \sim \pi_{\theta_j}(\cdot \mid State_e)$
6: Apply feature-wise linear modulation: $a_j' = FiLM(a_j, w_j)$
7: Simulation: $metric_j = Simulate(State_e, a_j')$
8: Compute rewards: $r_j = EMA + ECO + metric_j - penalty$
9: Store in replay buffer: $B_{replay,j} \leftarrow (State_e, a_j, r_j)$
10: **end for**
11: Exploit the best plan: $a_j^* = Exploit(\pi_{\theta_j}(\cdot \mid State_e)$
12: Apply feature-wise linear modulation: $a_j^{*'} = FiLM(a_j^*, w_j)$
13: Evaluate deterministic proposal: $metric_j \leftarrow Simulate(State_e, a_j^{*'})$
14: **end for**
15: Cross-label buffer with HER: $B_{cross,j} \leftarrow HER(B_{replay,j})$
16: **return** $a_j^{*'}$

5.2 Phase 1: Agent Training

In phase 1, we use a multi-agent soft actor-critic (SAC) algorithm with feature-wise linear modulation (FiLM) layer [22] and hindsight experience replay (HER) [3], as shown in Fig. 2. SAC is an actor-critic variant that uses stochastic off-policy learning to improve exploration. We add a FiLM layer to the actor's MLP for improved feature modulation. HER is a technique that modifies the replay buffer labels and enables training in such a complex environment. The details of phase 1 are presented in Algorithm 1.

The algorithm takes initial state information for the current epoch $State_e$, pretrained predictor $predictor$, number of agents J, agent-wise weights $[w_j]_{j=1}^{J}$, the past epochs' request volume $[I_{e-1}, ..., I_{e-tw}]$, and the initial policy parameter θ_j as input for agent training. After SAC training with the help of FiLM layer and HER, the algorithm outputs agent-wise plan $a_j^{*'}$.

The $predictor$ forecasts the current epoch's LLM workload (lines 1-2). This becomes part of the state information $State_e$ of the current epoch, which also includes information about each datacenter's node availability, carbon intensity, water intensity, and energy pricing. All agents are trained in parallel and iterate over K_{opt} iterations to update their policy parameter and Q function (lines 3-4). In each iteration, the agent j will first take an action a_j as a scheduling plan and then apply feature-wise linear modulation on a_j for a modulated plan a_j' (lines 5-6). The FiLM layer applies affine transformations across the neural network input. This integrates workload data and datacenter attributes into a single input. Based on the modulated plan a_j' the optimization metrics are estimated (line 7) to be $metric_j = [LA_{tot}, Z_{tot}, G_{tot}, Cost_{tot}]_{a_j}$

We calculate the reward value r_j for modulated plan a_j' (line 8) by considering an exponential moving average (EMA), $metric_j$, an ecological bonus that rewards consolidation ECO, and an SLA $penalty$ that punishes TTFT beyond a set point in the reward value estimation equation which is $r_j = EMA + ECO + metric_j - penalty$.

The SAC training experience is stored in the replay buffer $B_{replay,j}$ (line 9). After K_{opt} iterations, agent j generates the best plan a_j^* by exploiting the policy parameter θ_j (line 11). We apply FiLM to the best plan a_j^* to obtain the best modulated plan $a_j^{*'}$ (line 12). Based on the modulated plan $a_j^{*'}$, we estimate $metric_j$ through simulation (line 13). To balance current-epoch learning with historical insights, we maintain a cross-epoch buffer $B_{cross,j}$ for each agent j. The cross-epoch buffer is updated per epoch by sampling from the replay buffer $B_{replay,j}$ with HER (line 14).

Algorithm 2 MARLIN – Phase 2 Competitive Proposals and Consensus

Input: epoch data $State_e$, Agents J, agent-wise plan $[a_j^{*'}]_{j=1}^{J}$, agent-wise objective values $[metric_j]_{j=1}^{J}$, capital $[C_j]_{j=1}^{J}$, and utility function $[Q_j]_{j=1}^{J}$
Output: the final plan ã, the final objective values $\widetilde{metric}$, updated capital $[C_j]_{j=1}^{J}$

1: **for each agent** $j \in J$ **do**
2: Score every proposal: $q_j = Q_j(a_j^{*'})$
3: **end for**
4: Calculate the aggregate utility value: $q_{tot} = \sum_{j=1}^{J} q_j$
5: Blend proposals: $\tilde{a} = \sum_{j=1}^{J}(q_j/q_{tot}) \times a_j^{*'}$
6: Set critic weights: $\omega_{j,1} = C_j \,/\, \sum_{j=1}^{J} C_j$
7: Set aggregate Q function: $Q_{tot} = \sum_{j=1}^{J}(Q_j/J)$
8: **for** $k = 1$ **to** K_{opt} gradient steps **do**
9: Update critic weights with gradient ascent: $\omega_{j,k+1} = \omega_{j,k} + \frac{1}{K_{opt}} \nabla Q_{tot}(\omega_{j,k})$
10: **end for**
11: Project onto feasible set: $\tilde{a} = \sum_j^J a_j^{*'} \times \omega_{j,K_{opt}}$
12: **for each** agent j with capital $C_j \geq C_{thresh}$ ***do***
13: Calculate expected utility loss: $\delta_j = \left|Q_j(a_j^{*'}) - Q_j(a_j^{*'})\right| / Q_j(a_j^{*'})$
14: **if** $\delta_j > \delta_{thresh}$ **then**
15: Calculate veto strength: $Veto_j = \min(Veto_{max}, \delta_j \times C_j)$
16: Pull consensus toward own proposal: $\tilde{a} = (1 - Veto_j)\tilde{a} + Veto_j \times a_j^{*'}$
17: **end if**
18: **end for**
19: Execute consensus: $\widetilde{metric} \leftarrow Simulate(State_e, \tilde{a})$
20: **for each** $agent\ j \in J$ **do**
21: $Perf_j = \left|\min[metric_j]_{j=1}^{J} - \widetilde{metric}\right| / \left|\min[metric_j]_{j=1}^{J} - metric_j\right|$
22: $Bonus_j = 1 - \left|metric_j - \widetilde{metric}\right| / \left|metric_j\right|$
23: Update capital with decay ρ and growth rate η, bonus scaling factor β:
$C_j = \eta \times C_j + (1 - \eta) \times (Perf_j + \beta \times Bonus_j)$
24: **end for**
25: **return** ã, $\widetilde{metric}$, $[C_j]_{j=1}^{J}$

5.3 Phase 2: Competitive Proposals

Phase 2 focuses on combining the separate scheduling plans $a_j^{*'}$ generated from phase 1 into a single blended plan $\tilde{a}$. We employ game theory to optimize this blending process. The agent's associated capital C_j is essential in advancing its own plan in the final blended solution. Algorithm 2 presents the details of phase 2.

First, we calculate the utility value q_j for each agent j (lines 1-2). After that, we calculate the aggregated utility value q_{tot} for all agents J (line 4 in Algorithm 2). The weight of each agent's plan in the blended plan is determined by the corresponding utility value q_j scaled by the aggregated utility value q_{tot} (line 5). To help formulate the blended plan, we calculate the initial critic weight ω_j based on each agent's capital C_j (line 6). Moreover, an aggregate Q function Q_{tot} is formulated to estimate the long-term quality of the blended plan $\tilde{a}$ (line 7). The blended plan is refined over K_{opt} iterations via stochastic gradient descent (SGD) [26] to maximize the aggregate Q function (lines 8-10). After each SGD step, a new blended plan is created via the critic weights (line 11).

To maintain a balanced trade-off among all agents, a veto mechanism is designed based on the individual rationality (IR) theory [31]. IR is a game theory approach in which a participant will engage

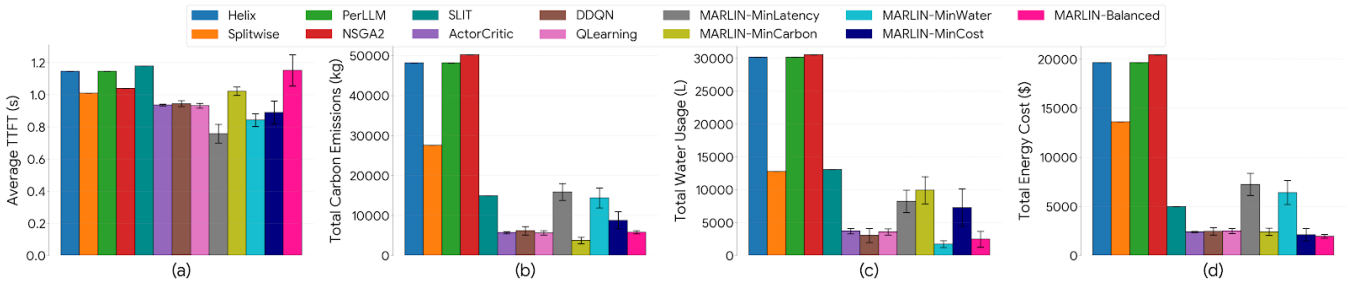


Figure 3: Comparison of the (a) TTFT, (b) carbon emissions, (c) water usage, and (d) energy costs, across frameworks.

only if their expected payoff exceeds that of not participating. This holds for the veto mechanic, as agents are not required to veto in every single epoch and will only consider using their veto under the following circumstances: if 1) their capital exceeds a threshold C_{thresh} and 2) their critic network Q_j estimates that the consensus provides a utility loss θ_j more than θ_{thresh} compared to its own proposal (lines 12-14). If a veto scenario is triggered, we calculate the veto strength $Veto_j$ based on utility value degradation θ_j and maximal veto strength $Veto_j$ (line 15). Based on the veto strength, we update the blended plan $\tilde{a}$ by moving the blended plan toward the agent's plan $a_j^{*'}$ (line 16).

Once all agents are satisfied with the blended plan (no threshold C_{thresh} or θ_{thresh} is violated), the optimization metric values $\widetilde{metric}$ are estimated accordingly via simulation (line 19). To pass down the knowledge we learn from the veto process, we need to update the capital C_j used in the veto mechanism for the next epoch. We first calculate the performance score $Perf_j$ for each agent's capital C_j (line 21). The performance score estimates how well the current blended plan $\tilde{a}$ serves the agent's $metric_j$. After that, the bonus score $Bonus_j$ is calculated to estimate each agent's contribution to the final blended plan (line 22). Given the performance score $Perf_j$, bonus score $Bonus_j$, predefined growth rate η, and predefined scaling factor β, we update all the capitals using a bounded EMA approach (line 23).

5.4 Complexity Analysis

The runtime and memory complexity of the framework depend on the number of datacenters D, policy network parameters θ_j, number of agents J, replay buffer size $B_{replay,j}$, and exploration steps K_{opt}.

In phase 1, the runtime complexity is driven by exploration steps K_{opt} and the policy network parameters θ_j. Agents J are run in parallel. Therefore, the final runtime complexity is $O(K_{opt} \times \theta_j)$. The memory complexity of phase 1 relies on the replay buffers $B_{replay,j}$ and policy network parameters θ_j. The buffer and the policy network sizes with respect to the agents J. The memory complexity of phase 1 is $O(J \times \theta_j + J \times B_{replay,j} \times D)$.

As the iterations of the blended plan are performed, the runtime complexity of phase 2 is identical to phase 1 as $O(K_{opt} \times \theta_j)$. The memory complexity of phase 2 is much lower than that of phase 1. The only two memory requirements are the holding of the proposals $a_j^{*'}$ and the agents J themselves. The proposal footprint from phase 1 is proportional to the number of datacenters D. Therefore, the memory complexity of phase 2 is $O(J \times \theta_j + J \times D)$.

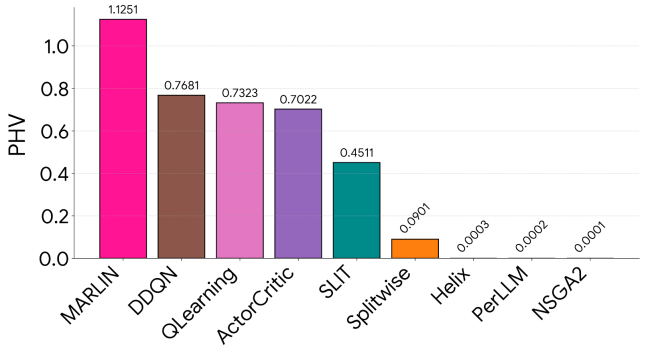


Figure 4: Comparison of the PHV values across LLM inference scheduling frameworks.

6 Experimental Results

To empirically validate our framework, MARLIN, we compare it against eight state-of-the-art workload scheduling algorithms. These can be divided into five heuristic-based approaches (Helix [17], Splitwise [21], NSGA-II [7], PerLLM [34], and SLIT [19]), and three RL methods (QLearning [8], DDQN [2], and ActorCritic [10]). Datacenters are distributed globally with requests originating uniformly across all regions. Each datacenter has 1000 compute nodes across 6 uniformly distributed node types, each containing 2, 4, or 8 NVIDIA A100 or H100 GPUs for LLM inference on Xeon-based server nodes. We developed and validated a Python simulator implementing the models from Section 3, as well as the workload predictor and MARLIN algorithmic framework. Data for computational energy and LLM runtime is obtained from profiling on real server nodes and integrated into the simulator.

We consider five schemes to be run through our framework, which consist of four plans dominated by one agent (MARLIN-MinCarbon, -MinWater, -MinCost, -MinLatency) and one with equal weights (MARLIN-Balanced). Our baseline experimental setup spans 24 hours with the maximum datacenter utilization rate in an epoch at 95% across 8 datacenters. The upcoming epoch's workload is forecast (a 15-minute window) in the previous epoch.

The learning rate hyperparameter in the MARLIN framework was 0.0003 for the actor and 0.001 for the critic networks. We used a

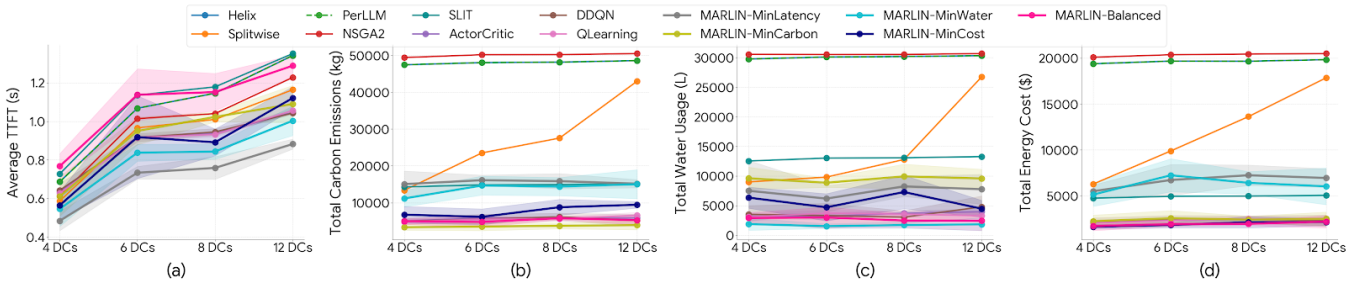


Figure 5: Comparison of (a) TTFT, (b) carbon emissions, (c) water usage, and (d) energy costs across LLM inference scheduling frameworks as the number of datacenters changes between 4 and 12; shaded regions represent the confidence intervals.

gamma of 0.95 and a tau of 0.005 for the agents. The actor network has 128 hidden dimensions, while the critic was increased to 256 to capture the complexity of the design space. The replay buffer size was 20,000 samples for the current epoch and 5,000 samples for previous epochs, with a 30/70 sampling split. For the SGD step, we used a learning rate of 0.05 with 5 steps. To determine the strength and use of the veto, the required threshold was 150 capital, with a 0.5 pull towards the agent's own plan when used.

6.1 State-of-the-art Comparison

Fig. 3 shows MARLIN's performance compared with state-of-the-art frameworks across the TTFT, Carbon, Water, and Energy Cost metrics, with confidence bounds for all RL frameworks. The MARLIN variants outperform all comparison works across these metrics, even considering confidence bounds. For the single-metric schemes, MARLIN-MinLatency reduces TTFT by 18.67% over QLearning, MARLIN-MinCarbon reduces carbon emissions 33.61% more than QLearning, MARLIN-MinWater reduces water 43.88% more than DDQN, and MARLIN-MinCost reduces costs 11.72% more than ActorCritic. MARLIN-Balanced outperforms the heuristic approaches across all metrics except TTFT and outperforms the RL baselines on 2 of 4 metrics; it is at most 23.61% slower in TTFT and 2.77% higher in carbon emissions than QLearning, while achieving at least an 18.66% reduction in water (vs. DDQN) and 18.94% cost reduction (vs. ActorCritic). Compared to the only other sustainable framework, SLIT, MARLIN-Balanced achieves a 2.29% reduction in TTFT, 61.07% in carbon, 81.11% in water, and 60.90% in costs demonstrating that MARLIN succeeds as a sustainable framework where SLIT failed.

In Fig. 4, we summarize the Pareto hypervolume (PHV) values across the different frameworks. The PHV of a Pareto solution front (the front formed by the non-dominated solutions) is a metric that measures the hypervolume of the solution space dominated by the set of solutions. This is an effective measure of the quality of diverse solution sets generated by a multi-objective optimization framework. A higher PHV indicates a more diverse and higher quality solution set.

For comparison works that produce only a single point, the PHV is calculated as the geometric volume of the four-dimensional hyperrectangle defined by that point. We set a reference point for the worst observed metrics. The population of points along the Pareto front was 1 for heuristic frameworks. For the RL frameworks, we archived the best points generated during the search along the confidence interval providing between 10 and 15 points. MARLIN's Pareto front contained 40 points, well exceeding the value of the other frameworks. We observe a significant difference between the heuristic-based and the RL-based approaches. The best-performing heuristic framework is SLIT, which covers only 40.09% of the hypervolume covered by MARLIN. The RL frameworks perform much better, but still only cover at most 68.27% of the volume that MARLIN covers. These results show that MARLIN is able to search the complex solution space much more effectively than the other frameworks, which allows it to provide higher quality solutions.

6.2 Scalability Analysis

To observe how the frameworks behave as the problem size scales, we vary the number of datacenters from 4 to 12. The results of this experiment are presented in Fig. 5.

It can be observed that the heuristic approaches were unable to utilize the new datacenters effectively and to navigate the larger search space, as datacenter counts increased. The RL-based approaches have more success in utilizing the new datacenters and their different carbon and water intensities.

MARLIN variants dominate, with MARLIN-MinCarbon being an outlier as it outperformed MARLIN-MinCost on cost with 12 datacenters. This performance by the carbon agent highlights a limitation of the single-metric variants. Low carbon sources can be low cost, but the signal in the carbon case is stronger than the cost case. MARLIN-MinCarbon finds this strong signal early and takes advantage of low carbon and cheap electricity for longer than MARLIN-MinCost, leading to the discrepancy.

Crucially, the MARLIN-Balanced approach can achieve Pareto-optimal solutions as the search space grows. This is significant, MARLIN-Balanced can reduce water usage by an average of 16.4% compared to smaller search spaces. MARLIN-Balanced leverages the unique sustainability fingerprint of each datacenter it has access to. In this experiment, we observe the same trend between MARLIN-Balanced and SLIT. In 12 datacenters, MARLIN-Balanced reduces latency by 4.66%, carbon emissions by 65.22%, water usage by 81.7%, and cost by 56.59%. MARLIN-Balanced provides a more balanced solution instead of the best across all metrics.

7 Conclusion

In this work, we presented a novel framework for serving LLM inference requests across geo-distributed cloud datacenters, called MARLIN. We empirically demonstrated that MARLIN can optimize both the TTFT of LLM inference requests and sustainability metrics such as carbon emissions and water usage. Solutions generated by the MARLIN framework outperformed other state-of-the-art frameworks and produced a balanced scheduling plan when configured with equal metric weights. This was observed, where MARLIN, decreased TTFT by 18%, carbon emissions by 33%, water usage by 43%, and costs by 11%. MARLIN is designed to operate as a meta scheduler above containers that host LLMs, such as Kubernetes or vLLM. This allows MARLIN to be used by datacenter managers to provide a range of LLM inference services and enable sustainable, multi-objective decision-making. Our future work will explore integrating embodied carbon to optimize life-cycle carbon emissions for LLM serving.

Acknowledgments

This research was made possible with support from HPE and grants from the National Science Foundation (CCF-2324514, CNS-2132385).

References

[1] Kazi Main Uddin Ahmed et al. 2021. A Review of Data Centers Energy Consumption and Reliability Modeling. *IEEE Access* 9 (2021), 152536–152563.

[2] Latifah Alsalem and Karim Djemame. 2026. Task Scheduling in Edge Computing Environments: a Hierarchical Cluster-based Federated Deep Reinforcement Learning Approach. In *Proceedings of the 18th IEEE/ACM UCC (UCC '25)*. Association for Computing Machinery, New York, NY, USA, Article 47, 8 pages.

[3] Marcin Andrychowicz et al. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[4] B. M. Beena et al. 2025. A Green Cloud-Based Framework for Energy-Efficient Task Scheduling Using Carbon Intensity Data for Heterogeneous Cloud Servers. *IEEE Access* 13 (2025), 73916–73938.

[5] Rui Chen et al. 2023. Power and thermal-aware virtual machine scheduling optimization in cloud data center. *Future Generation Computer Systems* 145 (2023), 578–589.

[6] Tracy Yingying Cheng and Xiaohua Jia. 2020. Delay-Sensitive Multicast in Inter-Datacenter WAN Using Compressive Latency Monitoring. *IEEE Transactions on Cloud Computing* 8, 1 (2020), 86–96.

[7] K. Deb et al. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[8] Chao Guo et al. 2025. Q-Learning-Based Workload Consolidation for Data Centers With Composable Architecture. *IEEE Transactions on Industrial Informatics* 21, 3 (2025), 2324–2333.

[9] Chunyang He, Zhifeng Liu, Jianguo Wu, et al. 2021. Future global urban water scarcity and potential solutions. *Nature communications* 12, 1 (2021), 4667.

[10] Taufik Hidayat et al. 2025. Reinforcement Learning-Driven Hybrid Precopy/Postcopy VM Migration for Energy-Efficient Data Centers. *IEEE Access* 13 (2025), 169521–169533.

[11] Ninad Hogade, Sudeep Pasricha, and Howard Jay Siegel. 2022. Energy and Network Aware Workload Management for Geographically Distributed Data Centers. *IEEE Transactions on Sustainable Computing* 7, 2 (2022), 400–413.

[12] Nidhal Jegham et al. 2025. How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference. arXiv:2505.09598 [cs.CY]

[13] Yi Jin et al. 2019. Water use of electricity technologies: A global meta-analysis. *Renewable and Sustainable Energy Reviews* 115 (2019), 109391.

[14] Monalisa Kushwaha et al. 2021. Advanced Weighted Round Robin Procedure for Load Balancing in Cloud Computing Environment. In *2021 11th Confluence*. 215–219.

[15] Rebecca Leppert. 2026. What We Know about Energy Use at U.S. Data Centers Amid the AI Boom. www.pewresearch.org/?p=277682 Accessed on Mar.31.2026.

[16] Yuzhuo Li et al. 2024. The Unseen AI Disruptions for Power Grids: LLM-Induced Transients. arXiv:2409.11416 [cs.AR]

[17] Yixuan Mei et al. 2025. Helix: Serving Large Language Models over Heterogeneous GPUs and Network via Max-Flow. In *Proceedings of the 30th ACM ASPLOS* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 586–602.

[18] Viyom Mittal et al. 2021. Mu: An Efficient, Fair and Responsive Serverless Framework for Resource-Constrained Edge Clouds. In *Proceedings of the ACM SoCC* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 168–181.

[19] Hayden Moore, Sirui Qi, Ninad Hogade, Dejan Milojicic, Cullen Bash, and Sudeep Pasricha. 2025. Sustainable Carbon-Aware and Water-Efficient LLM Scheduling in Geo-Distributed Cloud Datacenters. In *Proceedings of the GLSVLSI 2025 (GLSVLSI '25)*. Association for Computing Machinery, New York, NY, USA, 929–934.

[20] Ariffud Muhammad. 2026. LLM Statistics 2026: Comprehensive Insights into Market Trends and Integration. www.hostinger.com/uk/tutorials/llm-statistics Accessed on Mar.31.2026.

[21] Pratyush Patel et al. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st ISCA*. 118–132.

[22] Ethan Perez et al. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[23] Sirui Qi, Hayden Moore, Ninad Hogade, Dejan Milojicic, Cullen Bash, and Sudeep Pasricha. 2024. CASA: A Framework for SLO- and Carbon-Aware Autoscaling and Scheduling in Serverless Cloud Computing. In *2024 IEEE 15th IGSC*. 1–6.

[24] Sirui Qi, Hayden Moore, Dejan Milojicic, Cullen Bash, and Sudeep Pasricha. 2026. SHIELD-EB: Sustainable Hybrid Evolutionary-Boosting Framework for Carbon, Wastewater, and Cost-Aware Datacenter Management. *IEEE Access* 14 (2026), 40878–40898.

[25] Pol G. Recasens et al. 2025. Mind the Memory Gap: Unveiling GPU Bottlenecks in Large-Batch LLM Inference. In *2025 IEEE 18th CLOUD*. 277–287.

[26] Sebastian Ruder. 2017. An overview of gradient descent optimization algorithms. arXiv:1609.04747 [cs.LG]

[27] Md Abu Bakar Siddik et al. 2021. The environmental footprint of data centers in the United States. *Environmental Research Letters* 16, 6 (may 2021), 064017. doi:10.1088/1748-9326/abfba1

[28] Shubham Singh. 2026. ChatGPT Users Statistics (2026) – Active Users & Global Growth Data. www.demandsage.com/chatgpt-statistics Accessed on Mar.31.2026.

[29] Robert F. Sullivan. 2000. Alternating cold and hot aisles provides more reliable cooling for server farms. *White Paper, Uptime Institute* (2000).

[30] Hugo Touvron et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[31] John Von Neumann and Oskar Morgenstern. 1947. Theory of games and economic behavior, 2nd rev. (1947).

[32] Yuxin Wang et al. 2025. BurstGPT: A Real-World Workload Dataset to Optimize LLM Serving Systems. In *Proceedings of the 31st ACM KDD* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 5831–5841.

[33] Tianqi Xiao et al. 2025. Environmental impact and net-zero pathways for sustainable artificial intelligence servers in the USA. *Nature Sustainability* (2025), 1–13.

[34] Zheming Yang et al. 2024. PerLLM: Personalized Inference Scheduling with Edge-Cloud Collaboration for Diverse LLM Services. arXiv:2405.14636 [cs.DC]

[35] Qingxia Zhang et al. 2021. A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization. *Journal of Systems Architecture* 119 (2021), 102253.