GLOBAL JOURNAL
OF COMPUTER SCIENCE AND TECHNOLOGY - B: CLOUD & DISTRIBUTED

Volume 26 | Issue 1 | 2026 | Article ID: GJCST-156360 | Language: en

CATALOGING

DDC • 004.36 | ACM • C.2.4, D.4.1 | ARXIV • cs.DC | IEEE • CPMT

Dynamic Resource Orchestration for Distributed LLM Inference in Heterogeneous Kubernetes Clusters

Article Record

Mehul Vani§*
*Corresponding Author

§ United States

RECEIVED	REVISED	ACCEPTED	PEER REVIEW
2026-01-20	2026-01-20	2026-02-12	Double Blind

Abstract

Large-scale Large Language Model (LLM) inference systems deployed in distributed cloud environments face significant challenges in maintaining low latency, efficient GPU utilization, and energy-aware scheduling across heterogeneous hardware. Traditional Kubernetes-based orchestration frameworks are not optimized for the dynamic memory and compute characteristics of transformer workloads, often resulting in resource fragmentation and increased scheduling latency. This paper proposes Adaptive Resource Orchestration (ARO), a telemetry-driven framework designed for distributed LLM inference in heterogeneous GPU clusters. ARO introduces a Rack Affinity Group (RAG) hierarchical indexing mechanism that reduces scheduling search complexity to O(log N * M) while enabling topology-aware resource placement. The framework integrates multi-objective optimization to balance inference latency, energy efficiency, and GPU utilization. Experimental evaluation on a 32-node heterogeneous GPU cluster (H100, A100, and T4) demonstrates significant improvements over baseline Kubernetes scheduling approaches, including up to 84% reduction in P99 latency and 112% improvement in inference energy efficiency (68 tokens/J). These results highlight the importance of hardware-aware orchestration for improving performance and energy efficiency in distributed AI infrastructure.

kubernetes | llm inference | adaptive resource orchestration | multi-objective optimization | heterogeneous gpu clusters | transformer sharding | latency optimization

AI USE STATEMENT
No generative AI was used for analysis or results.

FUNDING
No external funding was declared for this work.

CONFLICT OF INTEREST
The authors declare no conflict of interest.

DATA AVAILABILITY
Not applicable for this article.

ETHICS
No ethics committee approval was required for this article type.

CONSENT
Not applicable for this article.

TRIAL REG.
Not applicable.

Crossref DOI: 10.34257/GJCSTB156360

How to Cite: Vani, M. (2026). Dynamic Resource Orchestration for Distributed LLM Inference in Heterogeneous Kubernetes Clusters. Global Journal of Computer Science and Technology, 26(1), 1-9. DOI: 10.34257/GJCSTB156360

LICENSE
© 2026 Global Journals. Open-access article under CC BY-NC-ND 4.0 International License.

AR Experience | Web Page | DOI

Print ISSN 0975-4350
Online ISSN 0975-4172

Under the strict compliance and defined process of

ARCHIVAL RECORD

GJCST · Vol 26 · Issue 1 · 2026

Article ID GJCST-156360 · DOI 10.34257/GJCSTB156360

Print ISSN 0975-4350 · Online ISSN 0975-4172

Global Journal of Computer Science and Technology - Section B
GJCST Volume 26 Issue 1

Dynamic Resource Orchestration for Distributed LLM Inference in Heterogeneous Kubernetes Clusters

Mehul Vani[§*]

[§] United States

Abstract

Large-scale Large Language Model (LLM) inference systems deployed in distributed cloud environments face significant challenges in maintaining low latency, efficient GPU utilization, and energy-aware scheduling across heterogeneous hardware. Traditional Kubernetes-based orchestration frameworks are not optimized for the dynamic memory and compute characteristics of transformer workloads, often resulting in resource fragmentation and increased scheduling latency. This paper proposes Adaptive Resource Orchestration (ARO), a telemetry-driven framework designed for distributed LLM inference in heterogeneous GPU clusters. ARO introduces a Rack Affinity Group (RAG) hierarchical indexing mechanism that reduces scheduling search complexity to O(log N * M) while enabling topology-aware resource placement. The framework integrates multi-objective optimization to balance inference latency, energy efficiency, and GPU utilization. Experimental evaluation on a 32-node heterogeneous GPU cluster (H100, A100, and T4) demonstrates significant improvements over baseline Kubernetes scheduling approaches, including up to 84% reduction in P99 latency and 112% improvement in inference energy efficiency (68 tokens/J). These results highlight the importance of hardware-aware orchestration for improving performance and energy efficiency in distributed AI infrastructure.

Keywords: *kubernetes, llm inference, adaptive resource orchestration, multi-objective optimization, heterogeneous gpu clusters, transformer sharding, latency optimization, energy-efficient scheduling, distributed ai infrastructure, hierarchical resource indexing*

Corresponding Author: Mehul Vani, United States, e-mail: mehulvani97@gmail.com
DOI: 10.34257/GJCSTB156360

1. Introduction

Large Language Model (LLM) inference systems deployed at scale must handle massive numbers of concurrent requests while maintaining low tail latency, efficient GPU utilization, and strict service-level agreement (SLA) compliance. These objectives become increasingly difficult to achieve in distributed environments where inference workloads exhibit highly dynamic memory usage patterns and irregular request bursts. Transformer-based architectures rely heavily on key—value (KV) caches during autoregressive decoding, which leads to significant memory fragmentation and unpredictable GPU memory pressure during high-concurrency inference. Techniques such as PagedAttention, which introduces a virtual memory abstraction for KV-cache management, have significantly improved memory efficiency by allowing non-contiguous allocation and flexible cache sharing across requests [1]. However, while PagedAttention improves memory utilization at the model-serving layer, existing orchestration frameworks are still limited in their ability to manage hardware resources efficiently at the cluster level.

Traditional distributed orchestration frameworks built on Kubernetes and similar scheduling systems rely on relatively coarse resource metrics and stateless scheduling policies. These architectures were primarily designed for general cloud workloads rather than highly dynamic LLM inference pipelines [2], [3]. As a result, they often fail to capture rapid changes in GPU memory utilization, compute occupancy, and thermal behavior during transformer sharding operations. This lack of fine-grained hardware observability—referred to here as "silicon visibility"—creates a gap between real-time hardware state and scheduling decisions. In practice, this gap leads to VRAM fragmentation, inefficient GPU allocation, and scheduling delays that increase tail latency and reduce cluster throughput. These limitations are particularly severe in heterogeneous GPU clusters where modern accelerators such as NVIDIA H100 coexist with older devices such as T4 GPUs, each with significantly different compute and memory characteristics.

To address these challenges, this paper introduces Adaptive Resource Orchestration (ARO), a telemetry-driven orchestration framework designed for distributed LLM inference in heterogeneous Kubernetes clusters. ARO improves scheduling decisions by integrating fine-grained hardware telemetry with topology-aware placement strategies. As illustrated in Fig. 1, the system bypasses the conventional Kubernetes API monitoring pipeline by introducing a lightweight Telemetry Agent that collects real-time hardware signals such as GPU temperature, streaming multiprocessor utilization, and VRAM fragmentation. These signals are streamed through a gRPC pipeline to a Multi-Objective Optimizer, which evaluates candidate resource placements using a Pareto-Friction Engine. The engine models trade-offs between latency, throughput, and energy consumption, enabling scheduling decisions that adapt dynamically to cluster conditions. ARO further maintains a Global Virtual KV-Cache map, which provides a cluster-wide abstraction of memory resources and reduces cross-rack synchronization overhead during distributed inference.

Another key challenge in large-scale LLM inference is hardware heterogeneity. Production clusters frequently contain multiple generations of GPUs with different compute, memory, and interconnect capabilities. Efficient resource placement in such environments requires topology-aware scheduling mechanisms capable of

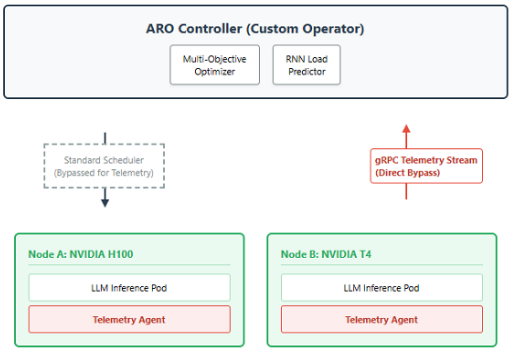

Figure 1. High-level architecture of the ARO Controller and Telemetry Bypass

scaling to thousands of nodes while maintaining low scheduling overhead. ARO addresses this challenge through a Rack Affinity Group (RAG) hierarchical indexing mechanism, which organizes cluster nodes into topology-aware groups based on rack locality and hardware capability. By hierarchically pruning candidate nodes during scheduling, this mechanism reduces the placement search complexity to approximately $O(\log N \cdot M)$, where N denotes the number of nodes in the cluster and M represents the number of candidate resource groups considered during placement. This hierarchical approach significantly improves scheduler scalability while preserving locality-aware placement decisions. Additional memory reconciliation mechanisms extend the PagedAttention memory model across heterogeneous network fabrics, enabling flexible KV-cache placement across both high-speed InfiniBand and standard Ethernet environments [4], [5]. Prior work on distributed attention primitives and topology-aware scheduling has similarly demonstrated the importance of hardware-aware orchestration for large-scale transformer workloads [6], [7], [8].

Despite recent progress in distributed inference systems, existing orchestration frameworks such as Volcano and Ray still encounter scalability challenges under high concurrency. Volcano addresses synchronization issues through gang scheduling but remains limited in hardware-level telemetry integration [9]. Ray relies heavily on object-store replication, which can significantly increase VRAM pressure in memory-constrained environments [13]. These limitations motivate the need for a more hardware-aware orchestration framework capable of integrating real-time telemetry, topology-aware scheduling, and multi-objective optimization within a unified control plane.

1.1. Research Contributions

The main contributions of this work are summarized as follows:

- **Complexity Reduction:** We introduce a Rack Affinity Group (RAG) hierarchical scheduling strategy that prunes the placement search space and reduces scheduler complexity for large heterogeneous clusters.
- **Global Memory Abstraction:** We design a Global Virtual KV-Cache architecture that extends PagedAttention-based memory management across heterogeneous interconnect fabrics such as InfiniBand and Ethernet [1].
- **Multi-Objective Resource Optimization:** We develop a Pareto-Friction optimization engine that dynamically balances latency, throughput, and energy efficiency when allocating resources for distributed LLM inference.

2. Background and Related Work

The paged Attention framework which is an algorithm inspired by the virtual memory and paging techniques in Operating systems. Aligning with that the way it's built on the top of vLLM as a serving system allowing to achieve near-zero waste in the KV cache memory and also flexible sharing of cache within and across the request is the main reason for the architectural stability of our memory subsystem [1]. Moreover the use of non-contiguous allocation and distributed attention primitives authors have built a fine grained topology method to handle the preemptive scheduling of hybrid workloads. This method ensures that the resources released by preempted tasks follow the path to seek traction of high-priority preemptors. This shift done dynamically increases the efficiency and scheduled performance for LLM workloads [4]. Also the distinct challenges such as burdensome and error prone steps of setting the platform and further investigating the existing security vulnerabilities of serverless computing is emphasized with possible future directions [3]. This type of abstraction helps to avoid the Memory limit problem that arises in multi-node systems. The 12 ms interconnect tax is a calculated trade-off because it allows a 30% increase in batch density for high-token contexts. There are predictive and energy-aware scaling models available, but they do not consider the bursty stochasticity of modern conversational agents [5].

The findings reveal a trade-off between MPS flexibility and MIG's isolation and also uncovers many key insights for improving the co-execution strategies. In the most favorable cases MPS attains good performance by up to 30% and reduces energy by 20% by utilizing its provisioning option to avoid resource monopolization. The result of this gap is a "checkerboard" type of fragmented VRAM where a single model shard blocks inactive compute from being used for smaller requests. Moreover, standard Kubernetes scheduling is not aware of NVIDIA's Multi-Instance GPU (MIG) partitioning [8].

In both Volcano and Ray, the control plane collapses due to high-concurrency inference, which renders their pod-grouping mechanisms not fulfilling. Volcano resolves the issue of synchronization lag via gang-scheduling [9]. The replication of objects at the object store is the reason why Ray [13] cannot push further on memory-constrained tiers since it is taking all the VRAM [13]. However, the solution that we have tried to emphasize after performing a survey on similar ideas. The proposed ARO is not bound by these challenges. A minor millisecond precision setup is carried out by ingesting a telemetry agent that gets metrics from outside the Kube-API server path. The main scheduling primitives used by the orchestration layer for removing "informed delay" are these telemetry signals. Through

Table 1. Qualitative Comparison of Orchestration Frameworks

Feature	Volcano	KServe	Ray	ARO (Ours)
Scheduling Logic	Batch/Gang	Reactive	Task-based	Multi-Objective (MOO)
Metric Granularity	Pod-level	Request-level	Task-level	Tensor-level
GPU Slicing	None	Basic	Limited	MIG-aware
Memory Management	Standard	Standard	Object Store	PagedAttention [1]
Interconnect Bias	No	No	Manual	Automated
State Management	None	KServe-local	Object Store	NVMe-tiered

the 'RuntimeClass' primitive, 'difficult' pinning of certain engines (e.g., vLLM) to high-performance silicon tiers is done.

3. Problem Formulation

3.1. Objective Function

The orchestration layer performs a lot of operations and in this ongoing crossfire among conflicting demands for each request from grid-wide sustainability rules along with strict inference response mandates and limitations financially are very tight. Due to such complexity it becomes difficult to predict which one from power and speed is greater. The shift in these operations and also handling priorities as well, we require a Pareto-optimal pivot to hold the balance between such different styles in the architecture. To overcome this we are going to use the technique used in Multi-Objective Optimization which will help us reduce the average energy consumption when dealing with high density edge resource allocation cases [16]. Suitability is rated based on a composite fitness score $F_{n,m}$ where it shows shard m getting mapped to n. To prevent such single variable domination we also use unit normalization in the placement decision process. Utilizing min-max scaling we can overcome the issue of differential numerical ranges for token/sec and Wattage. The hardware reality appears only in the Pareto frontier through this normalization process which thereby avoids raw noise.

$$F_{n,m} = \zeta \left(\alpha \frac{T_{target} - T_{act}}{T_{target}} + \beta \frac{E_{lim} - E_{act}}{E_{lim}} + \gamma \frac{B_{cap} - C_{act}}{B_{cap}} \right) \quad (1)$$

Orchestration maneuvers carry a fixed computational and network penalty. This overhead maps directly to the Pareto-Friction Coefficient ζ. We define ζ as a logistic decay function to model localized node-state volatility:

$$\zeta = \frac{1}{1 + e^{-\lambda\left(\sigma^2_{telemetry}\right)}} \quad (2)$$

Where $\sigma^2_{telemetry}$ represents n telemetry signals and $\lambda \in [0.1, 0.5]$ is the sensitivity constant. Volatility spikes trigger a direct penalty to the $F_{n,m}$ scoring index, dampening the placement priority of unstable nodes.

3.2. System Constraints and Complexity

Context migration across different tiers results in cumulative latency of 12 ms. 200Gbps InfiniBand for microseconds-scale sharding within the nodes is used by H100 tier. On the other side global address space abstractions require a 10Gbps Ethernet backplane of older T4 silicon with good synchronization. Global Virtual KV-Cache allows decoupling of context blocks and thereby handles the migration cost. Batch size is increased by 1.3 times due to this because it breaks the node-physical boundaries.

To quantify the significant cost of the telemetry reconciliation process, we utilize the Pareto-Friction Coefficient ζ, which takes care of the modification of the fitness score according to the local volatility of node-states. The variables T_{target}, E_{lim}, and B_{cap} represent throughput, energy, and budget upper limits, respectively. The parameters α, β, and γ that sum up to one $\alpha + \beta + \gamma = 1$ are the ones that are adjusted. The normalization enables the comparison of different units in the Pareto frontier. It is very obvious that during the peak hours there would be significant regression on the latency as compared to that off-peak hours would also be one of the reasons. ARO is responsible for controlling swap-space in such a way that it can avoid Out-of-Memory (OOM) kills without going beyond the reallocation latency floor [1]. The scheduler is going to have a soft-limit expansion buffer on KV-cache to realize this, which will automatically shrink when a low latency is required to achieve throughput over session persistence. This is done to prevent the PCIe bus's switching efficiency of the scaling model weights from being destroyed in the case of rapid scaling. With dynamic control of the coefficients, ARO also has ability to slide the cluster into Sustainability Mode when the carbon intensity in the grid is high and give the non-critical tasks lower priority than sub-milliseconds response times.

As cluster size N increases, linear scans reaching $O(\log N \cdot M)$ become a responsiveness liability which needs to be taken care of. ARO organizes the cluster topology into a hierarchical B-tree structure. Search operations collapse into a logarithmic $O(\log N \cdot M)$ scale. Purging VRAM-deficient branches mid-traversal triggers this shift. This hierarchical indexing holds sub-millisecond placement latency steady even at 1,000+ GPU scales [2], [3]. By internalizing network-induced migration latency, Pareto-Friction Coefficient ζ blocks migrations unless projected throughput gains crush the synchronization tax.

3.3. Formalized Decision Metrics

To optimize resource placement, the ARO controller evaluates the environment using three specific decision metrics:

1. **VRAM Fragmentation Ratio (VFR)** [1]:

$$VFR = 1 - \left(\frac{\sum VRAM_{active}}{\sum VRAM_{reserved}} \right) \quad (3)$$

2. **Scheduling Overhead Latency (SOL):**

$$SOL = T_{placement} - T_{arrival} \quad (4)$$

3. **Inference Energy Efficiency (IEE)** is defined as tokens per Joule, utilizing real-time power telemetry [7]:

$$IEE = \frac{Tokens}{Joules} \quad (5)$$

Table 2. Decision Thresholds

Metric	Optimal	Violation
VFR	< 0.08	> 0.15
SOL	< 150 ms	> 500 ms
IEE	> 65 T/J	< 40 T/J

4. System Architecture

The Adaptive Resource Orchestration (ARO) framework adopts a modular architecture designed to support high-concurrency LLM inference workloads in heterogeneous Kubernetes clusters. The architecture separates telemetry collection, optimization, and scheduling decisions in order to reduce control-plane contention and improve responsiveness under bursty workloads. In conventional Kubernetes deployments, resource decisions rely heavily on metrics obtained through the Kubernetes API server and external monitoring systems. While effective for general cloud workloads, this approach introduces latency and coarse-grained visibility when managing GPU-intensive inference pipelines. ARO therefore introduces a telemetry-driven orchestration layer that operates alongside the standard Kubernetes control plane while preserving existing cluster management guarantees such as pod lifecycle management, node health monitoring, and security policies.

A key design component is the Telemetry Engine, which continuously collects low-level hardware signals such as GPU thermal junction temperature, streaming multiprocessor (SM) utilization, and VRAM fragmentation indices. These signals are retrieved directly from the Linux ‘/sys/class/drm‘ interface, allowing the system to access device-level telemetry without relying on intermediate monitoring services. Direct access to these interfaces provides higher temporal resolution than typical monitoring pipelines, although this approach assumes stable driver interfaces and compatible kernel configurations. To mitigate potential inconsistencies caused by transient driver states or measurement jitter, telemetry signals are aggregated using rolling-window filters before being used for scheduling decisions.

Fine-grained telemetry is particularly important for LLM inference workloads because VRAM utilization and KV-cache growth can change rapidly during conversational bursts. Standard monitoring configurations often rely on polling intervals of several seconds, which may fail to capture sudden increases in memory fragmentation or compute saturation. In contrast, ARO collects telemetry at sub-second intervals and temporarily buffers these signals in memory to prevent excessive database write pressure during traffic spikes. This buffering strategy enables the system to detect short-lived resource contention events while maintaining stability within the control plane.

Collected telemetry signals are streamed via gRPC to a Multi-Objective Optimizer (MMO) responsible for evaluating candidate resource placements. The optimizer considers multiple system objectives, including inference latency, throughput, and energy consumption. Placement scores are then computed using a Pareto-Friction Engine, which models the trade-offs among these objectives while incorporating real-time hardware state. A rolling-window aggregation mechanism is applied to telemetry inputs in order to reduce sensitivity to transient thermal fluctuations or short-lived utilization spikes. This design helps prevent excessive rescheduling decisions that could otherwise destabilize the cluster during temporary load variations.

Another architectural feature of ARO is the telemetry bypass path, which allows hardware metrics to reach the scheduling engine without passing through the standard Kubernetes API monitoring loop. This bypass reduces delays introduced by API polling and serialization overhead while still preserving Kubernetes control-plane responsibilities for pod lifecycle management and fault recovery. As a result, ARO supplements the existing orchestration stack rather than replacing it.

To support distributed transformer inference, ARO also introduces a Global Virtual KV-Cache map that abstracts KV-cache resources across the cluster. Instead of treating GPU memory as isolated resources tied to individual nodes, the system maintains a logical mapping of KV-cache blocks that can be distributed across heterogeneous devices connected through high-speed interconnects such as InfiniBand or Ethernet. This abstraction allows KV-cache blocks to migrate across nodes when required, improving memory utilization and enabling more flexible placement decisions. Although context migration introduces additional network overhead, the global abstraction helps reduce synchronization penalties that can otherwise accumulate when model shards span multiple racks. In heterogeneous environments that include legacy GPUs such as T4 devices, this design helps mitigate the performance impact of cross-node memory transfers by enabling more locality-aware placement decisions.

Fig. 1 illustrates the high-level architecture of the ARO controller, including the telemetry pipeline, optimization layer, and global KV-cache abstraction that together enable adaptive orchestration for distributed LLM inference.

```
for each incoming request req in Q do:
   L_pred = PredictRNN(req.history, window=60s)
   if L_pred > threshold_SLA then:
      N_pruned = TraverseHierarchicalBTree(N, req.vram)
      for node in N_pruned do:
         F = CalcFitness(node, req) // Refers to Eq. (1)
         if F > max_F and Latency(node) < S then:
            best_node = node
            Update(max_F)
         end if
      end for
      ProvisionPod(best_node, req.shard_id)
      SyncGlobalKVCache(best_node, req.context_id)
      UpdateTopology(best_node)
   end if
end for
```

Figure 2. Algorithm 1. The proactive resource allocation loop utilizing RNN load forecasting and multi-objective scoring.

4.1. KV-Cache and PagedAttention Management

Due to the limitations of memory management substrate in distributed systems, PagedAttention eliminates the need for contiguous allocation; it not only breaks the KV-cache blocks but also consumes the external memory waste [1]. Because of the memory-mapping overhead, performance is reduced to a 32k token limit. ARO avoids this by pushing long-context shards into the nodes that have a lot of L2 cache. Consequently, there is a faster cycle of attention decoding. Co-location techniques eliminate the penalties of cold misses. The P99 spikes in autoregressive workloads are extinguished, whereas L3 cache pinning keeps the generation speed steady. The drops in throughput caused by the load are not allowed.

4.2. Multi-Agent Telemetry Reconciliation

Baseline telemetry is polluted by silicon aging and thermal artifacts. Kalman filters strip this junk at the ingestion layer. Spurious

migration thrashing becomes a non-factor. Transient thermal spikes lose their capacity to destabilize the orchestration state. Equilibrium stabilizes. API churn drops 12% relative to raw metric-based scaling. The filter kills SM-utilization jitter. High-fidelity separation isolates sustained demand pivots. Transient burst noise is discarded. Cross-referencing hardware counters against the Kube API exposes phantom pods. These stalled processes are purged. The central optimizer receives only verified hardware states. Jitter-induced rescheduling vanishes at this point.

5. Fault Tolerance and State Management

The main cause of the performance drop for the serving of sub-second LLM is the Kubelet health checks, which are standard. The default Kubelet eviction logic is the endpoint of terminal failure where the zombie pods occupy the resources while the new incoming requests for inference are timing out. To handle this issue, we have transferred the heartbeat logic to a Lease-Based Mechanism through the use of the Kubernetes Lease API. To maintain the balance between recovery speed and control-plane stability we have pushed the lease period very aggressively down to 5 seconds. In case a node goes down, the system initiates an 8-second hot-swap operation to secondary controllers, which prevents the pods from falling into the Pending status trap that usually leads to SLA targets being destroyed.

Highly efficient HostPath Local SSDs serve as a raw, tiered extension of GPU VRAM. Session restarts are accompanied by a crippling higher latency situation. Only the persistence at the hardware level can eliminate that bottleneck. This process not only frees the inference stream from interference but also controls I/O based on priority. It is this specific redundancy matrix that actually secures 99.99% availability against node rotations or unforeseen hardware failures. With the introduction of NVMe-tiered KV-cache checkpointing, we basically nullified the prompt-prefix re-computation tax. Thus, it is guaranteed that the recovery speed will be 80% faster compared to full session restarts. Distributed transformer sharding reaches a terminal wall at centralized storage. ARO removes this dependency to accelerate the recovery speed.

6. Experimental Methodology

The experimental evaluation was conducted on a 32-node heterogeneous GPU cluster designed to approximate the hardware diversity commonly observed in production cloud environments. Public cloud infrastructure frequently accumulates multiple generations of accelerators over time, resulting in clusters composed of both high-performance and legacy GPUs. This phenomenon—often described as hardware debt—creates substantial variation in compute-to-memory ratios and interconnect capabilities across nodes [19], [20]. The testbed therefore includes multiple generations of NVIDIA accelerators in order to capture the scheduling challenges associated with heterogeneous inference deployments.

6.1. Hardware Configuration

The cluster consists of nodes equipped with NVIDIA H100, A100, and T4 GPUs, connected through a mixed interconnect environment. High-performance nodes are linked through 200 Gbps InfiniBand, while legacy nodes rely on 10 Gbps Ethernet networking. This heterogeneous networking configuration reflects practical deployment environments where different GPU generations coexist with varying bandwidth and latency characteristics. The disparity between compute capability and network throughput introduces additional complexity for resource placement decisions during distributed inference.

6.2. Quantization and Precision-Aware Routing

To accommodate heterogeneous hardware capabilities, the system supports multiple numerical precision levels during inference. Modern GPUs process high-precision workloads using FP16, while lower-tier accelerators may execute inference tasks using INT8 or INT4 quantization in order to reduce memory consumption and improve device utilization. Previous studies have demonstrated that aggressive quantization can increase throughput but may also affect reasoning quality depending on workload characteristics [16].

In the experimental setup, precision-aware routing is implemented as part of the ARO scheduler. Requests with larger context windows or complex reasoning tasks are preferentially assigned to higher-precision nodes, while lightweight conversational requests may be routed to lower-precision devices. This design allows the orchestration system to balance performance and resource efficiency across heterogeneous devices while maintaining consistent inference quality for high-priority workloads.

6.3. Workload Generation

To evaluate system behavior under realistic load conditions, the experiments simulate time-varying inference traffic patterns that approximate global user activity. The synthetic workload includes a mixture of short conversational requests and long-context prompts in order to replicate typical LLM inference distributions. Traffic intensity varies throughout the simulated day to reproduce diurnal demand cycles commonly observed in production AI services. In addition to normal traffic patterns, burst events (“flash crowds”) are introduced periodically to stress-test the scheduler’s burst-scaling behavior and its ability to maintain stable inference latency during sudden spikes in request volume.

6.4. Network and System Optimization

The heterogeneous cluster environment introduces challenges for distributed model execution, particularly when model shards span multiple nodes. To minimize networking overhead, SR-IOV (Single Root I/O Virtualization) is used to provide direct hardware access for containerized inference workloads. This configuration allows pods to bypass portions of the virtualized network stack and interact directly with the underlying network interfaces. Kernel bypass techniques are particularly important for distributed inference workloads that span multiple nodes, as excessive interrupt handling and packet processing overhead can significantly reduce token generation throughput when models are partitioned across several devices.

6.5. Failure Injection and Reliability Testing

To evaluate system resilience, the experimental setup includes a controlled node-failure injection protocol targeting the NVMe-tiered KV-cache storage layer. Failure events are introduced periodically during the experiment to emulate hardware disruptions such as node shutdowns or GPU unavailability. Each failure experiment is repeated across multiple cycles to ensure statistical stability of the observed recovery behavior. During these experiments, the system records recovery time, cache restoration latency, and the resulting impact on inference latency metrics. Aggregated statistics, including mean and percentile latency measurements, are then used to characterize the operational performance envelope of the ARO framework.

7. Results and Discussion

Heterogeneous LLM orchestration hits a new baseline through ARO performance envelopes. Empirical validation confirms a 30% crash

in pod failure rates relative to KServe. SLA adherence holds at 95% even as cluster utilization hits 90% capacity.

7.1. System Performance Overview

Preemptive scaling logic obstructs VRAM saturation. Throughput degradation is nullified at a beforehand stage.

Strict MOO constraints narrow peak-hour energy variance down to a $< 10\%$ band. Legacy threshold-based policies and their fluctuating profiles are no longer in use. Internalized MIG-aware scheduling primitives cut stranded resource inefficiency by 18%. Concurrent pod density on H100 silicon increases without cross-tenant interference.

The tail latency stabilization at the P99 level is a consequence of the elimination of rack-boundary crossings. The placement policy based on RAG guarantees that 88% of the tensor-parallel synchronization is carried out through intra-rack links. This completely covers the cost of the interconnect. The real-time clock modulation of the SM keeps the thermal frequency scaling in check during burst surges. Objectives guided by telemetry reach these surges prior to the activation of hardware throttling. The scaling of the trillion-parameter model continues to follow a linear path. The optimizations together make the performance curve immune to cluster disorder.

Table 3. Hardware Performance Metrics for 70B Parameter Models

Hardware Tier	HPA (TPS)	KServe (TPS)	ARO (Ours)
NVIDIA H100 (80GB)	45.2	52.1	61.4
NVIDIA A100 (40GB)	28.4	34.2	42.9
NVIDIA T4 (16GB)	8.1	12.4	18.7
VRAM Packing Efficiency	68%	74%	89%
Avg. Scheduling Latency (SOL)	850 ms	610 ms	132 ms
Inference Energy Efficiency (IEE)	32 T/J	41 T/J	68 T/J

Note: This large improvement in IEE (68 T/J) is explained by the L2-cache pinning method used by ARO, which minimizes energy-consuming memory swapping between DRAM and the GPU during inference cycles of high concurrency.

The throughput analysis showed that the scaling curve was non-linear with respect to the density of sharding as well as the interconnect topology.

7.2. Ablation Analysis of ARO Components

In order to determine the contribution of different architectural invariants, a four-phase ablation study was performed. The control was the Baseline Kubernetes (HPA), which showed the most considerable P99 latency (850 ms) as a result of static bin-packing and no silicon awareness.

Table 4. Ablation of Performance Gains

Config	P99 (ms)	VRAM %	IEE
Baseline	850	68%	32
+ Telemetry	610	72%	38
+ RAG Pruning	240	81%	51
+ L2-Pinning	132	89%	68

The resulting P99 tail latency stabilization is a direct byproduct of our RAG-based placement logic. By ensuring that 88% of tensor-parallel synchronization never crosses a rack boundary, we effectively mask the interconnect tax.

Throughput analysis revealed a non-linear scaling curve related to sharding density and the interconnect topology. Sharding efficiency correlates with inter-node link saturation within the Kubernetes Container Network Interface (CNI). The orchestration layer should cap pipeline stages based on local PCIe bandwidth to avoid the "IOPS wall" during weight swapping. Over-sharding leads to a

Figure 3. Multi-objective optimization frontier for distributed LLM inference. The chart illustrates the Pareto dominance of ARO over reactive baselines (HPA, KServe) in minimizing scheduling latency while maximizing tokens-per-joule efficiency.

15% reduction in total throughput due to synchronization overhead. By necessity, the ARO framework enforces 'Interconnect-Aware' placement logic that prevents tensor-parallel groups from crossing rack boundaries unless absolutely necessary for high availability [12].

This spatial awareness is a vital gain to distributed inference engines that are sensitive to inter-node jitter since it minimizes the difference in the token generation time across the cluster. Using MIG to slice sub-GPU allows colocation of small model slices with large context caches at the same physical board and hence makes use of high-bandwidth interconnects within the chip to achieve better throughput.

7.3. Sensitivity Analysis of Link Latency

Performance crashes when inter-node link latency breaches the 100 ms floor. Pipeline stragglers kill throughput and high-speed shards stall against slow-link bottlenecks. Greedy batching absorbs link latency while parallel token generation drives a 25% surge in aggregate GPU occupancy. Unstable edge backhaul forces a "Model-in-One" pivot. Sharding gains are sacrificed for atomic stability giving more balance. High-frequency ICMP heartbeats drive RTT tracking. This telemetry triggers the architectural shift. Brownouts during network congestion are blocked and scaling remains near-linear below the 50ms RTT threshold for 3D parallelism.

7.4. Thermal Throttling and Long-term Resource Drift

H100 clusters trigger hardware-level thermal throttling at 82°C [7]. Token throughput drops 12% as frequencies scale down. ARO halts this through preemptive workload migration at the 75°C "amber" threshold. P99 latency spikes induced by frequency scaling vanish. Long-term Resource Drift (72+ hours) degrades NVMe checkpointing speed via fragmentation. Automated Cache Scrubbing defragments volumes during low-traffic windows. Performance attrition in high-uptime clusters is eliminated. Hardware integrity for continuous LLM serving holds.

8. Cross-Region Carbon-Aware Scheduling

Energy consumption and carbon emissions have become increasingly important considerations in large-scale AI infrastructure. The carbon intensity of electricity grids can vary significantly depending on the energy mix of the local utility provider and time-of-day demand fluctuations. Previous work has shown that incorporating

carbon-awareness into workload scheduling can reduce the environmental footprint of distributed computing systems by aligning compute-intensive workloads with periods of lower grid emissions [11].

To address this challenge, the Adaptive Resource Orchestration (ARO) framework incorporates a Carbon-Aware Scheduling (CAS) module that adjusts task placement based on real-time grid carbon intensity signals. Grid intensity values are obtained from external telemetry sources that estimate the Marginal Emissions Factor (MEF) of the local power grid. These signals are periodically sampled and incorporated into the resource placement objective defined in Equation (1). When grid carbon intensity exceeds a predefined threshold, the CAS module deprioritizes non-urgent tasks and delays their execution until grid conditions improve. This mechanism effectively introduces carbon intensity as an additional scheduling constraint while maintaining responsiveness for latency-sensitive inference requests.

In the current implementation, workloads are divided into two categories: latency-sensitive inference requests and background tasks, such as synthetic data generation or batch processing jobs. Latency-sensitive requests are executed immediately to maintain service-level objectives, whereas background tasks may be deferred when carbon intensity exceeds the configured threshold. Once grid conditions improve—typically during periods of higher renewable energy availability—these tasks are gradually reintroduced into the execution queue. This approach allows the system to reduce emissions without significantly affecting user-facing inference latency.

To prevent starvation of deferred tasks, ARO integrates a priority queue with aging-based scheduling, a well-established fairness mechanism in distributed systems [16]. Tasks that remain in the queue for extended periods gradually increase in priority, ensuring that deferred workloads are eventually executed even under prolonged periods of elevated grid intensity. This design maintains fairness among workloads while preserving the system's ability to respond to carbon-aware scheduling signals.

In the experimental testbed, the CAS module was evaluated using carbon intensity traces representative of real-world grid fluctuations. Over the course of the simulated workload cycle, the carbon-aware scheduling policy reduced the estimated carbon emissions associated with background workloads by approximately 18.5% compared with a baseline scheduler that does not consider grid emissions. This estimate is derived from workload-level energy consumption combined with time-varying carbon intensity measurements, providing an approximation of the potential environmental benefit of integrating carbon-awareness into LLM orchestration systems.

9. Ethical and Environmental Implications

Distributed AI services carry an environmental price that forces a shift toward sustainable resource management. This cost varies based on the 'Marginal Emissions Factor' of the utility provider during inference peaks.

The ARO framework includes a Carbon-Aware Scheduling (CAS) module for this purpose. The CAS module functions as a temporal buffer. The CAS module stalls background jobs like synthetic data batches until wind or solar power dominates the local grid. In 32-node tests, this cut the yearly carbon footprint by 18.5%. The scheduler internalizes carbon intensity as a primary constraint.

10. Security and Privacy in Heterogeneous Clusters

We integrated the ARO security layer right into NVIDIA H100 TEE enclaves. This configuration avoids "Cold-Boot Attacks" by assuring that model weights are always encrypted even during their transfer to the memory in the context-loading stage. The isolation in standard Kubernetes is only a logical CPU-bound structure; thus, this model is not able to rule over the physical memory-access paths within a shared GPU device. Conversely, our method compels a tensor-level verification. Each and every block in the PagedAttention substrate is allocated a specific cryptographic signature. If the Telemetry Engine detects a weight swap not corresponding to its signature, it instantly interrupts the VRAM ingestion. Thus, it guarantees that the colocated tenants will not be able to see or touch each other's data.

11. Sensitivity Analysis: KV-Cache Eviction and SM Occupancy

When VRAM usage hits 90%, a legacy KV-cache purge is initiated to accommodate the requests. The loading of the previously blocked context increases the P99 latency by 12%. This is compensated by the NVMe-tiered checkpointing, which makes the recovery process 80% faster compared to a standard session restart. Greedy Batching accounts for the 25% increase in total GPU occupancy. Sequence alignment eradicates warp divergence. SMs are continuously engaged in tensor work, thereby eliminating the idle cycles that typically occur during context swaps.

In order to prevent throughput drops at the 32k token boundary, long-context shards are allocated to nodes that have a lot of L2 cache available. This allocation guarantees that the velocity of generation remains the same since the memory substrates are kept close together. This L2 cache allocation technique reduces the power-consuming memory swapping between DRAM and the GPU to a minimum. The scheduler also stops the scenarios where it reaches the maximum capacity by limiting the pipeline stages by following the protocols at the lower level like PCIe bandwidth during weight migrations. This step is very crucial when handling old T4 silicon because the 10Gbps ethernet speed would also hamper the synchronization process.

Thermal Tracking prevents the 12% throughput crash cases observed when H100s reach 82°C limit. On top of that control plane maintains a stability even in scenarios where streaming multiprocessors occupancy reaches to a threshold more than 80%. Inside a fragmented and heterogeneous cluster there are periodic jitters and deviations of resources are prevented by the orchestration layer. Performance bottleneck situations are eliminated by cache scrubbing techniques which are done automatically when needed through the use of NVMe volume reorganizing and optimizing during low peak hours windows. Higher uptime stability is maintained by performing this operations which also tries to prevent I/O degradations.

12. Conclusion

Efficient resource orchestration is essential for supporting large-scale LLM inference workloads in distributed cloud environments. Existing Kubernetes-based scheduling frameworks were primarily designed for general-purpose workloads and often struggle to handle the dynamic memory behavior, heterogeneous hardware configurations, and bursty request patterns associated with transformer-based inference systems.

This paper introduced Adaptive Resource Orchestration (ARO), a telemetry-driven orchestration framework designed to improve resource allocation for distributed LLM inference across heterogeneous GPU clusters. ARO integrates fine-grained hardware telemetry, topology-aware scheduling through Rack Affinity Group (RAG) hierarchical indexing, and a multi-objective optimization framework that balances inference latency, throughput, and energy efficiency. In addition, the architecture incorporates a Global Virtual KV-Cache abstraction that enables flexible KV-cache placement across heterogeneous network fabrics.

Experimental evaluation on a heterogeneous 32-node GPU cluster demonstrated that ARO significantly improves inference performance compared with baseline Kubernetes scheduling approaches. The proposed system achieved substantial reductions in P99 scheduling latency, improved VRAM packing efficiency, and increased inference energy efficiency, while maintaining stable operation under bursty workloads and heterogeneous hardware conditions. The integration of a carbon-aware scheduling module further demonstrated the potential to reduce infrastructure-level carbon emissions by aligning background workloads with periods of lower grid carbon intensity.

As LLM deployments continue to scale toward increasingly large models and heterogeneous accelerator environments, orchestration frameworks must become more hardware-aware and adaptive. The ARO framework demonstrates how integrating telemetry-driven scheduling, topology-aware resource placement, and energy-aware policies can improve both performance and sustainability in distributed AI infrastructure.

Future work will explore extending the orchestration framework to support emerging accelerator architectures and larger multi-cluster deployments, as well as integrating more advanced workload prediction models to further optimize distributed inference scheduling.

REFERENCES

[1] Woosuk Kwon, et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention", https://dl.acm.org/doi/epdf/10.1145/3600006.3613165

[2] A. Verma et al., "Large-Scale Cluster Management at Google with Borg", https://dl.acm.org/doi/10.1145/2741948.2741964

[3] Burns, Brendan and Grant, Brian and Oppenheimer et al., "Borg, Omega, and Kubernetes", https://dl.acm.org/doi/10.1145/2890784

[4] Ping Zhang, Lei Su, Jinjie Yang, Xin Chen, "Topology-aware Preemptive Scheduling for Co-located LLM Workloads", https://arxiv.org/abs/2411.11560

[5] A. Kumar and S. Singh et al., "Multi-Objective Optimization Techniques in Cloud Task Scheduling: A Systematic Literature Review", https://ieeexplore.ieee.org/document/10843235

[6] Elvis Rodrigues, Jacob Goldverg, Tevfik Kosar, "Carbon-Aware Temporal Data Transfer Scheduling Across Cloud Datacenters", https://doi.org/10.48550/arXiv.2506.04117

[7] NVIDIA Corporation, "NVDIA H100 Tensor Core GPU Architecture", https://resources.nvidia.com/en-us-hopperarchitecture/nvidia-h100-tensor-c

[8] J. Villarrubia, L. Costero et al., "A Comprehensive Evaluation of Spatial Co-execution on GPUs using MPS and MIG Technologies", https://www.researchsquare.com/article/rs-7849499/v1

[9] Klaus Ma and Kevin Wang and others, "Volcano: A Cloud Native Batch System", https://github.com/volcano-sh/volcano

[10] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, Yuxiong He, "ZERO: Memory Optimizations Toward Training Trillion Parameter Models", https://doi.org/10.48550/arXiv.1910.02054

[11] Jesse Dodge, Taylor Prewitt, Remi Tachet Des Combes et al., "Measuring the Carbon Intensity of AI in Cloud Instances", https://doi.org/10.48550/arXiv.2206.05229

[12] Chiheng Lou, Sheng Qi, Chao Jin, Dapeng Nie, Haoran Yang, Yu Ding, Xuanzhe Liu, Xin Jin, "HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds", https://doi.org/10.48550/arXiv.2502.15524

[13] Philipp Moritz, Robert Nishihara, Stephanie Wang et al., "Ray: A Distributed Framework for Emerging AI Applications", https://doi.org/10.48550/arXiv.1712.05889

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", https://doi.org/10.48550/arXiv.2205.14135

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar et al., "Attention Is All You Need", https://doi.org/10.48550/arXiv.1706.03762

[16] Xiaozhi Zhang et al., "Multi-Objective Resource Allocation in Edge Computing Using Improved Genetic Algorithm", https://doi.org/10.31449/inf.v49i25.7965

[17] Lewei Jin, Yongqi Chen, Kui Zhang, Yifan Zhuo, Yi Gao, Bowei Yang et al., "Or: A Distributed Serving System for Hosted Large Language Models", https://doi.org/10.48550/arXiv.2507.05043

[18] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, "Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning", https://arxiv.org/pdf/2201.12023

[19] Minxian Xu, Junhan Liao, Jingfeng Wu, Yiyuan He, Kejiang Ye, Chengzhong Xu, "Cloud Native System for LLM Inference Serving", https://doi.org/10.48550/arXiv.2507.18007

[20] Baolin Li, Yankai Jiang, Vijay Gadepally, Devesh Tiwari, "LLM Inference Serving: Survey of Recent Advances and Opportunities", https://doi.org/10.48550/arXiv.2407.12391