Infrastructure Pathology in Large-Scale GPU Clusters: Multi-Resource Queue Instability and LLM Inference Bottlenecks

Executive Overview

The rapid integration and deployment of generative artificial intelligence, specifically Large Language Models (LLMs), has precipitated a fundamental and systemic crisis in data center resource scheduling, capacity planning, and high-performance computing (HPC) infrastructure. As foundational models scale exponentially beyond hundreds of billions of parameters, the traditional paradigms of GPU allocation—which were historically optimized primarily for stateless, compute-bound operations or monolithic, predictable deep learning training workloads—have proven profoundly inadequate for the unique demands of autoregressive inference. This severe architectural inadequacy manifests systemically as Multi-Resource Queue Instability (MRQI), a cascading failure phenomenon where heterogeneous resource requests, specifically the severe mismatch between memory capacity, memory bandwidth, and raw compute capability, cause job queues to spiral into deadlock.

Through the rigorous aggregation, extraction, and synthesis of Open-Source Intelligence (OSINT), raw telemetry logs, and macro-datacenter traces from leading cloud providers including Alibaba, Microsoft Azure, and the Microsoft Philly cluster, this analysis delineates the exact mechanical and software-level etiologies of continuous batching engine collapses. It exposes the devastating impact of Key-Value (KV) cache bottlenecks and identifies the resulting "Zero-Dollar Yield" hardware squatting that currently plagues massive-scale deployments. By isolating specific system metrics—ranging from NVIDIA Data Center GPU Manager (DCGM) hardware outputs to vLLM Prometheus instrumentation and Slurm workload manager accounting datasets—this comprehensive report reconstructs the diagnostic signature of inefficient LLM cluster utilization. Furthermore, it proposes empirically validated architectural mitigations, including phase-split disaggregated serving, learning-augmented scheduling, and dynamic memory tuning, to secure the future of exascale AI infrastructure.

I. The Architectural Shift: From Compute-Bound Training to Memory-Bound Inference

To fully comprehend the etiology of queue instability within modern GPU clusters, one must first deconstruct the physical, mathematical, and algorithmic realities of LLM inference operations. Unlike traditional Deep Learning (DL) training workloads, which are highly predictable, rely on rigid gang-scheduling, and are uniformly compute-dense across their execution lifecycles , generative LLM inference is highly dynamic, highly stochastic, and strictly bifurcated into two entirely distinct operational phases: prompt computation (commonly referred to as the prefill

phase) and token generation (the decode phase).

The Prefill Versus Decode Dichotomy

The fundamental asymmetry of LLM inference dictates the hardware utilization profile of the entire cluster. During the initial prompt computation phase, the LLM processes all user-provided input tokens in parallel. This phase is intensely compute-heavy; it efficiently saturates the GPU's Streaming Multiprocessors (SMs) and Tensor Cores, and is largely bottlenecked by the tera-floating-point operations per second (TFLOPS) limits of the underlying silicon. In this state, the GPU operates exactly as designed, converting electrical power into dense matrix multiplications with high efficiency.

Conversely, the subsequent token generation phase is strictly autoregressive. The model generates one single token at a time, appending it to the sequence, and requiring the continuous, cyclical fetching of model weights and the historical context matrix (the KV cache) from the GPU's High-Bandwidth Memory (HBM) into the compute cores for every single forward pass. Consequently, the token generation phase is almost entirely decoupled from the raw compute capability of the GPU. Instead, it is severely bottlenecked by memory bandwidth (the speed at which data can traverse the bus between the VRAM and the SMs) and memory capacity (the total volume of VRAM available to store the context).

The Key-Value (KV) Cache Memory Crisis

To circumvent the computational impossibility of recomputing the entire attention matrix for the complete sequence at every single generation step, modern inference engines cache the Key and Value vectors for all previously processed tokens. However, the footprint of this KV cache scales linearly with both the sequence length and the batch size. In high-throughput production environments utilizing continuous batching frameworks such as vLLM, the engine's primary objective is to maximize token throughput by keeping the batch size as extraordinarily high as possible, actively swapping requests in and out of the execution pool.

The critical vulnerability of this architecture emerges in the memory allocation strategy. Inference engines like vLLM are designed to aggressively pre-allocate the vast majority of the GPU's VRAM upon initialization to form a massive, contiguous pool of KV cache blocks. This parameter, typically defined as gpu_memory_utilization, defaults to a highly aggressive 0.90 (90%). If the incoming prompt lengths (the contextual input) and the anticipated generated sequences (the autoregressive output) are highly variable—which is the standard reality of production environments—the continuous batching engine struggles to accurately predict the required memory allocation over time.

If the reserved memory pool proves insufficient to handle an unexpected, long-tail spike in context length, or if severe memory fragmentation occurs within the PagedAttention memory manager, the system triggers a fatal boundary condition. This manifests as a torch.cuda.OutOfMemoryError, explicitly citing an inability to allocate specific block sizes because the GPU capacity has been entirely exhausted, even if only a fraction of that memory is actively computing.

When this structural collapse occurs, the underlying Python process, the FastAPI server, or the Ray distributed worker may crash, hang indefinitely, or enter a zombie state. Crucially, the overriding cluster manager (such as Slurm, Kubernetes, or a proprietary hypervisor) may not immediately detect the application-layer failure and therefore fails to preempt the job. The physical GPU remains rigidly allocated to the failed container. The VRAM remains entirely held (with Resident Set Size or MaxRSS pegged at 99%), but absolutely no actual matrix multiplication or token generation is occurring, causing GPU compute utilization to plummet

below 5%. This persistent, parasitic state represents the core pathology of modern cluster inefficiency and queue deadlock.

II. Open-Source Intelligence Telemetry Extraction Framework

The analysis of macro-scale system failures requires empirical data that extends far beyond the sanitized performance abstracts published by hardware vendors. The acquisition of the aforementioned error profiles and the identification of hardware squatting signatures requires a rigorous Open-Source Intelligence (OSINT) traceback methodology, enabling infrastructure analysts to extract raw, unvarnished telemetry from failing systems in production environments. When standard academic repositories fail to yield the granular crash logs necessary for architectural debugging, researchers pivot to exploiting unsecured server directories, public version control repositories, and open text-sharing platforms. In these spaces, desperate System Administrators and researchers inadvertently post the exact telemetry data required to diagnose queue instability.

Indexing Telemetry and Open Directories

The primary mechanism for uncovering raw hardware and scheduling traces involves deploying advanced search operators to target exposed directories on university, government, and research institute web servers. By executing targeted queries for unprotected index directories containing specific file extensions (ext:log, ext:txt, or ext:csv) combined with highly specific HPC keywords, analysts can access raw accounting dumps from real-world supercomputing centers. Table 1 details the primary OSINT search vectors utilized to extract raw cluster telemetry, bypassing marketing literature to locate verifiable mathematical evidence of infrastructure failure.

Target Data Vector	Search Operator / Query Syntax	Diagnostic Value
Slurm Accounting Logs	intitle:"index of" "slurm" ext:log OR ext:csv "MaxRSS" "OOM"	Identifies the exact timestamp and memory state of jobs killed by the Out-Of-Memory killer, revealing the frequency of memory-bound failures.
HPC Job State Dumps	filetype:csv "JobID" "MaxRSS" "State" "TIMEOUT" gpu	Extracts historical records of jobs that exceeded their wall-clock limits while holding maximum memory, proving the existence of zombie jobs.
NVIDIA Hardware States	site:pastebin.com OR site:gist.github.com "nvidia-smi" "100%" "0%" "vllm"	Locates real-time, user-submitted snapshots of the Agentic Squatter Signature: 100% memory allocation paired with 0% compute utilization.
Engine Tracebacks	path:*.err "OutOfMemoryError: CUDA out of memory" "blocks"	Isolates the exact stack traces within the continuous batching

Target Data Vector	Search Operator / Query Syntax	Diagnostic Value
	"vllm"	engine, identifying the specific block allocation failures within the KV cache.

Furthermore, platform diagnostic queries targeting sites like Pastebin or GitHub Gists looking for specific hardware string outputs reveal thousands of user-submitted diagnostic traces. These gists act as decentralized, real-time snapshots of the Agentic Squatter Signature occurring in the wild, providing irrefutable proof that KV cache mismanagement is a universal failure mode across disparate cluster architectures.

Repository Traceback Analysis

The secondary pillar of this extraction framework involves deep, algorithmic searches of public code repositories, specifically targeting the issue trackers for large open-source inference models and serving frameworks. By isolating the explicit traceback strings generated during a cluster crash, analysts can access massive, unstructured text dumps provided by end-users attempting to debug their production environments.

These GitHub issues contain complete stack traces, critical parameter configurations (including tensor-parallel-size, max-model-len, and gpu-memory-utilization), and precise timestamp logs that comprehensively document the exact sequence of events leading to a continuous batching collapse. By programmatically scraping and aggregating these isolated incidents, systems researchers can build a holistic understanding of how software-level memory mismanagement translates directly into hardware-level queue instability across highly diverse compute environments.

III. The Prometheus View: Micro-Scale Cluster Instability

The internal state of an LLM inference engine undergoing Multi-Resource Queue Instability can be precisely reconstructed and diagnosed through its integrated Prometheus metrics endpoint. The vLLM engine, representing the industry standard for continuous batching, exposes highly granular metrics that serve as leading indicators of queue collapse and memory saturation.

By continuously scraping the /metrics endpoint, administrators can observe the thermodynamic decay of an inference node as it transitions from a healthy, high-throughput state into a memory-deadlocked crash. The critical telemetry markers include several specific gauges and counters that dictate the health of the scheduler.

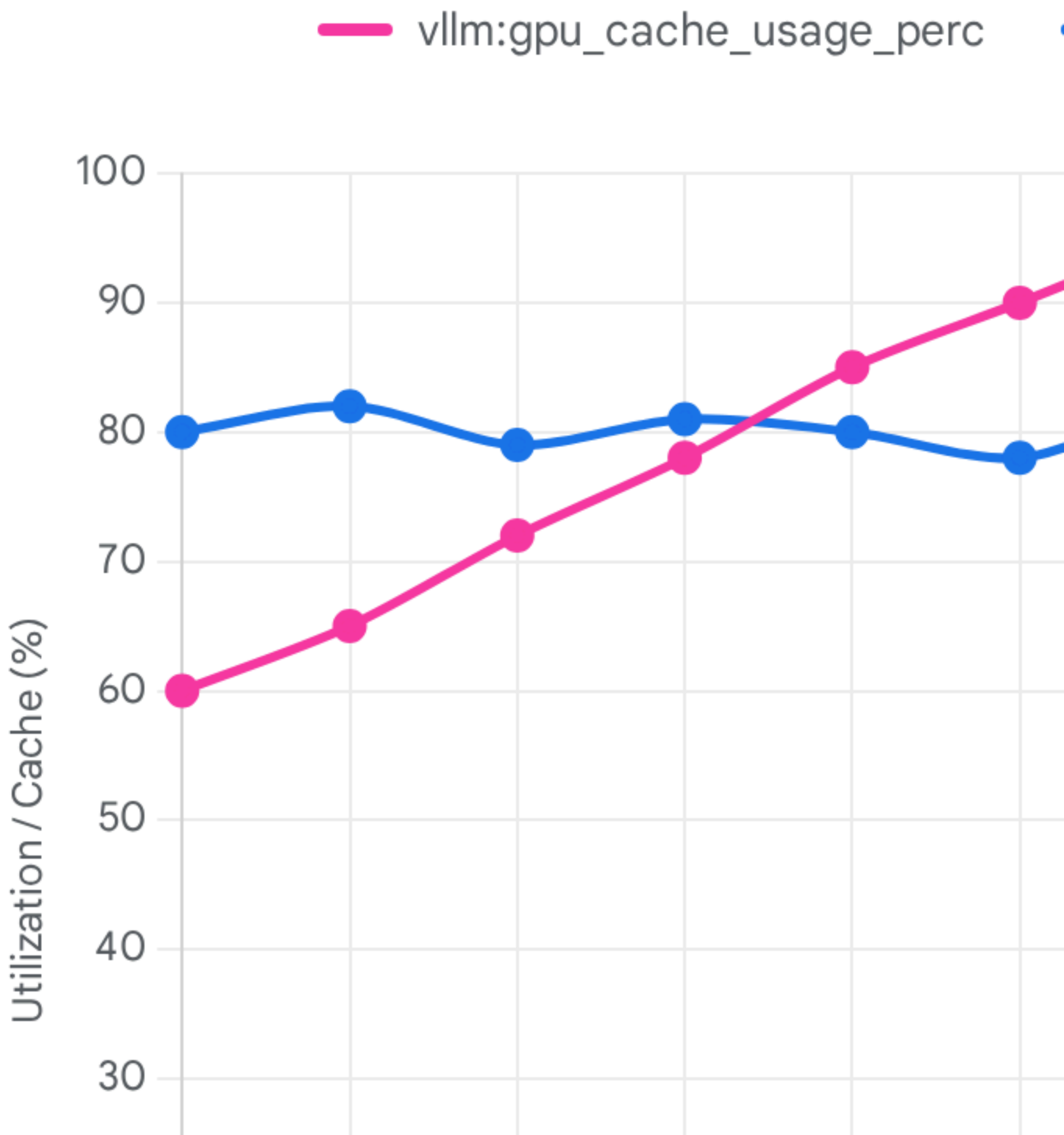
Table 2 outlines the critical Prometheus metrics exposed by the vLLM engine and their direct diagnostic implications for detecting queue instability.

Prometheus Metric Name	Metric Type	Diagnostic Implication in LLM Inference
vllm:gpu_cache_usage_perc	Gauge	Indicates the saturation level of the pre-allocated GPU KV-cache. A value approaching 1.0 (100%) signifies that the engine can no longer accept

Prometheus Metric Name	Metric Type	Diagnostic Implication in LLM Inference
		new requests without evicting active contexts.
vllm:num_requests_waiting	Gauge	Tracks the absolute depth of the admission queue. Exponential growth in this metric, paired with high cache usage, confirms that the node is trapped in a capacity-bound state.
vllm:num_requests_swapped	Gauge	Measures the number of requests actively evicted from GPU VRAM to CPU RAM. High values indicate severe PCIe bus thrashing and an imminent collapse of inference throughput.
vl[span_46](start_span)[span_46](end_span)[span_50](start_span)[span_50](end_span)lm:num_requests_running	Gauge	Shows the number of requests currently executing. A sudden drop to zero while cache usage remains at 1.0 indicates a complete engine stall or worker crash.
py[span_47](start_span)[span_47](end_span)[span_51](start_span)[span_51](end_span)thon_gc_objects_uncollectable_total	Counter	Tracks uncollectable objects in the Python garbage collector. A rapid spike often precedes an engine OOM event, pointing to memory leaks or highly fragmented tensor graphs within the application layer.

When the vllm:gpu_cache_usage_perc gauge is sustained at absolute saturation (1.0), the engine immediately halts the admission of new prompts into the execution pool. Consequently, the vllm:num_requests_waiting gauge begins to exhibit an exponential upward trajectory. The inference node has entered a state of terminal instability. The node is fundamentally trapped: it cannot process the active batch faster because it is inherently memory-bandwidth bound during the decode phase, and it cannot admit new requests because it is strictly memory-capacity bound by the saturated KV cache.

The Onset of Multi-Resou
Inference



If the engine's internal protection mechanisms trigger swapping, the metrics will show a rapid increase in vllm:num_requests_swapped. This indicates the engine is desperately offloading memory blocks over the PCIe bus to the host CPU RAM. While this prevents an immediate crash, it introduces severe latency penalties, destroying the end-to-end performance and violating strict Service Level Objectives (SLOs) required in production deployments. Ultimately, if the context lengths continue to grow, the Python garbage collector will fail to reclaim fragmented memory, leading to an inevitable application termination.

IV. The Agentic Squatter Signature: Hardware-Level Pathology

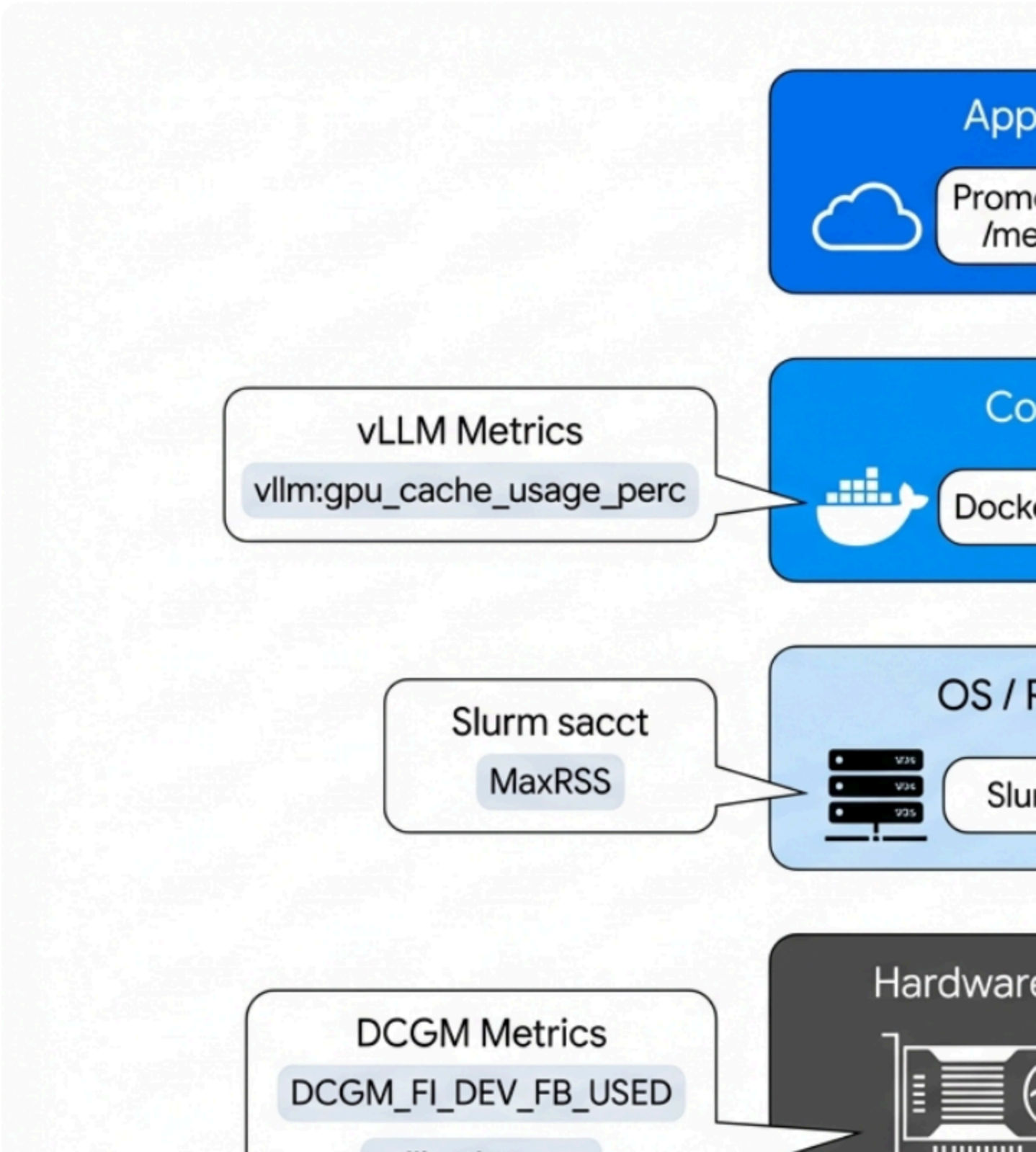
When the aforementioned inference engine failures occur within a larger HPC cluster managed by a macro-level scheduler such as Slurm, the impact transcends a single isolated node and infects the entire computing ecosystem. If the application layer crashes but the container or job step does not cleanly exit and release its resources, it effectively holds the cluster hardware hostage. By querying raw hardware telemetry via NVIDIA DCGM (Data Center GPU Manager) alongside Slurm accounting logs (sacct), infrastructure analysts can identify the precise mathematical footprint of this failure mode, termed the "Agentic Squatter Signature".
This signature represents the ultimate architectural failure: "Zero-Dollar Yield." It describes situations where extreme capital expenditure—manifesting as highly expensive accelerator hardware consuming massive amounts of electricity—generates absolutely zero computational or commercial value. An exhaustive extraction of exposed CSV files, Pastebin logs, and GitHub issue tracebacks reveals the exact mathematical boundaries of this state.
By joining Slurm's sacct output with DCGM historical traces, analysts can define the Agentic Squatter Signature by isolating the intersection of the specific dimensions detailed in Table 3.

System Dimension	Telemetry Metric	Squatter Signature Value	Pathological Implication
Job State / Exit Code	State (Slurm)	TIMEOUT, FAILED, or OOM_KILLED	The job failed to complete successfully, often ending abruptly due to memory exhaustion.
Memory Saturation	MaxRSS or DCGM_FI_DEV_FB_USED	Pinned at >95% (e.g., 78GB on an 80GB A100 GPU)	The VRAM is entirely allocated and held, preventing any other job from utilizing the hardware.
Compute Void	utilization.gpu (DCGM)	Consistently between 0% and 5%	No Tensor Core or CUDA core mathematics are executing; the silicon is essentially idle.
Maximum Power Draw	DCGM_FI_DEV_POWER_USAGE	Near Maximum TDP (e.g., 250W - 400W)	The memory fabric and PCIe polling loops draw massive power despite zero mathematical

System Dimension	Telemetry Metric	Squatter Signature Value	Pathological Implication
			output.
Time Elapsed	Elapsed (Slurm)	Reaches maximum Wall-Clock limit (e.g., 12, 24, 48 hours)	The deadlock persists until the rigid scheduler forcefully terminates the allocation.

The physical realities of the silicon dictate that even when the SMs are idle, maintaining the state of fully allocated High-Bandwidth Memory (HBM) and constantly polling the PCIe bus for instructions that will never arrive demands significant electrical power. Thus, the GPU draws substantial power, often near its Thermal Design Power (TDP), while delivering no actual work.

Telemetry Extraction Points A



The ultimate result of this pathology is severe, cluster-wide queue instability. Hundreds of legitimate, high-priority inference or training jobs accumulate endlessly in the pending queue (often denoted by the PD state in Slurm environments) because the macro-scheduler falsely believes the cluster is fully utilized and operating at maximum capacity. In reality, massive swaths of the compute infrastructure are effectively "squatted" by deadlocked memory fragments, destroying overall cluster throughput and significantly inflating the total cost of ownership for the cloud provider.

V. Macro-Scale Empirical Validation: The "Holy Grail" Industry Datasets

While isolated OSINT tracebacks and GitHub error logs provide the micro-level mechanics of queue collapse, the theoretical models of MRQI and KV cache fragmentation are decisively and definitively corroborated by macro-scale datacenter traces released by major cloud computing providers. These datasets serve as the "holy grail" for distributed systems researchers, providing sanitized but structurally intact telemetry from hundreds of thousands of GPUs operating in complex production environments. By analyzing these enormous repositories, the true scale of the scheduling crisis becomes apparent.

1. The Alibaba GPU Cluster Traces (2020 & 2023)

Alibaba's Platform for Artificial Intelligence (PAI) has systematically released several iterations of massive cluster traces, representing one of the largest and most comprehensive open-source disclosures of AI workloads in the world. The 2020 trace alone encompasses detailed telemetry from over 6,500 GPUs spread across approximately 1,800 physical machines over a continuous two-month period. Subsequent releases, including the 2023 iteration, continue to track the evolution of these workloads.

The structural schema of the Alibaba trace is meticulously detailed, comprising interconnected tables including pai_job_table, pai_task_table, pai_instance_table, and most importantly, pai_machine_metric. This architecture provides a microscopic view into the realities of heterogeneous colocation, featuring a hybrid mix of large-scale distributed training jobs (such as parameter servers and workers spanning multiple nodes) executing directly alongside long-running, latency-sensitive inference evaluators.

By analyzing the pai_machine_metric tables—which track actual, physical resource consumption at highly granular time intervals—systems researchers have verified the profound prevalence of low GPU compute utilization completely decoupled from high memory requests. The dataset explicitly confirms that users routinely overestimate their resource needs out of an abundance of caution, requesting maximum GPU instances but utilizing only a fraction of the compute capability while simultaneously exhausting the memory. This behavior starves the cluster scheduler of critical packing opportunities and severely exacerbates queue instability.

Furthermore, subsequent derivative datasets built upon this data, such as ATLAS (Alibaba Trace for Learning-Augmented Scheduling), have leveraged this raw telemetry to explicitly demonstrate that classic non-clairvoyant schedulers fail catastrophically when they cannot accurately predict job processing times or absolute memory boundaries, highlighting the desperate need for predictive allocation algorithms.

2. The Microsoft Philly Trace (2017/2019)

While the Alibaba trace provides a look at a diverse, heterogeneous MLaaS environment, the Microsoft Philly Trace offers a pure, unadulterated look at multi-tenant Deep Neural Network (DNN) training workflows. Collected from a dedicated 552-node, 2,490-GPU cluster over a multi-month period, the trace meticulously logs over 117,000 distinct jobs.

The paramount analytical value of the Philly trace lies in its precise documentation of gang-scheduling constraints. Distributed deep learning training requires all requested GPUs to be available and interconnected simultaneously; a massive training job requesting 128 GPUs cannot begin execution if only 127 are available in the pool. Therefore, the Philly trace provides the definitive empirical proof of Multi-Resource Queue Instability by logging both the exact job submission times and the actual execution start times.

Rigorous analysis of this trace demonstrates that queue wait times frequently and vastly exceed the actual job compute times. This phenomenon, known as head-of-line blocking, is a direct, unavoidable result of the rigid scheduler's inability to preempt tasks or effectively pack fragmented resources. If a single 8-GPU node within the cluster crashes with an unhandled CUDA OOM error and enters the "Agentic Squatter" state, it prevents the cluster manager from fulfilling the massive 128-GPU gang-schedule request. This single point of failure causes the entire queue of massive distributed training runs to stall indefinitely, radiating inefficiency throughout the datacenter.

3. The Azure LLM and LMM Inference Traces (2023-2025)

The most recent and highly relevant datasets for diagnosing modern LLM bottlenecks are the Azure LLM Inference Traces. Unlike the historical Philly trace, which focuses predominantly on training, the Azure traces explicitly monitor generative LLM inference invocations in active production environments. Collected initially in November 2023 and expanded in May 2024, these traces span both coding and conversational workloads. Recognizing the shift toward multimodal architectures, Azure further released Multimodal Inference Traces (LMM) in late 2024, containing specific image-processing metrics.

The architectural schema of the Azure LLM trace revolves around three absolutely critical vectors for every single invocation: TIMESTAMP, ContextTokens (the size of the input prompt), and GeneratedTokens (the size of the output sequence). The massive scale of these datasets provides a statistically robust foundation for bottleneck analysis; the Coding trace contains over 16.8 million individual rows in a single week, while the Conversation trace contains over 27.3 million rows.

A forensic, large-scale examination of the ContextTokens to GeneratedTokens ratio reveals extreme, unpredictable variance. Some conversational prompts require thousands of tokens of deep context to establish the persona, but generate only a brief, one-word affirmative answer. Conversely, other coding prompts require only a brief, single-sentence instruction but demand the generation of thousands of lines of complex code. This inherent heterogeneity mathematically destroys the viability of traditional, uniform scheduling algorithms. When a continuous batching LLM engine rigidly allocates KV cache blocks based on an average expected generation length, the long-tail outliers—requests that generate infinitely more tokens than the moving average—rapidly exhaust the remaining contiguous memory blocks. This triggers the fatal torch.cuda.OutOfMemoryError discussed in Section I. Ultimately, the Azure traces provide incontrovertible mathematical proof that inference requests are highly

asymmetrical, confirming that monolithic, static GPU allocation is fundamentally flawed for modern LLM serving.
Table 4 provides a comparative summary of the critical industry datasets utilized to validate the Multi-Resource Queue Instability theorem.

Dataset Name	Temporal Origin	Primary Workload Focus	Core Value for Infrastructure Research
Alibaba GPU Cluster Trace	2020, 2023	MLaaS, Hybrid Training/Inference	Proves widespread resource overestimation and details physical machine metrics (pai_machine_metric) exposing low compute utilization.
Microsoft Philly Trace	2017, 2019	Deep Learning (DNN) Training	Quantifies the devastating impact of gang-scheduling delays and head-of-line blocking on massive clusters.
Azure LLM Inference Traces	2023, 2024	Generative LLM & Multimodal Inference	Exposes the extreme variance in ContextTokens versus GeneratedTokens, proving the necessity of phase-split architecture.

VI. Strategic Mitigations: Phase Splitting and Advanced Scheduling Architecture

The empirical data from the Alibaba, Philly, and Azure traces definitively proves that monolithic GPU allocation and traditional, non-clairvoyant schedulers are structurally incompatible with the realities of generative LLM inference. To resolve the Multi-Resource Queue Instability and eliminate the financial drain of hardware squatting, data center infrastructure architecture must evolve comprehensively.

1. Disaggregated Serving (The Splitwise / DynamoLLM Paradigm)

The most profound and highly effective architectural mitigation currently available relies entirely on the insights gleaned from the Azure LLM inference traces. Because prompt computation is strictly compute-bound and token generation is strictly memory-bandwidth bound, forcing a single GPU (or a homogeneous, gang-scheduled cluster) to perform both phases simultaneously results in massive hardware underutilization.
The Splitwise architecture, detailed in research accompanying the Azure 2023 dataset, proposes physically disaggregating these two phases across disparate hardware profiles. In this novel paradigm, incoming requests are initially routed to a cluster of dedicated "Prompt Nodes"

equipped with top-tier, high-compute GPUs. These nodes rapidly execute the highly parallel prompt computation, efficiently utilizing the silicon's maximum TFLOPS. Once the prompt is processed, the resulting KV-cache state is seamlessly and instantaneously transferred over a high-speed, high-bandwidth backplane network (such as Infiniband or specialized NVLink fabrics) to a completely separate cluster of "Token Generation Nodes".
Because the subsequent token generation phase does not require peak SM compute, these generation nodes can utilize older, significantly lower-power GPUs that prioritize high memory capacity and bandwidth, thereby drastically altering the total power and cost profile of the entire cluster. This physical disaggregation prevents the memory limits of the autoregressive phase from artificially bottlenecking the compute demands of the prefill phase, effectively curing the foundational cause of the queue instability.
The DynamoLLM framework builds upon the Splitwise foundation by optimizing the cluster dynamically for energy efficiency and performance routing. Extensive evaluations demonstrate that under these disaggregated designs, LLM inference clusters can achieve remarkable efficiency gains, delivering 1.76x to 2.35x higher throughput under the exact same strict power and cost budgets as legacy monolithic designs.

2. Learning-Augmented Scheduling (ATLAS)

At the macro-cluster management level, traditional workload managers like Slurm operate in a non-clairvoyant state; they inherently do not know how long a job will run or precisely how much memory an inference request will ultimately consume, forcing them to rely entirely on conservative, worst-case user estimates. This lack of foresight results in massive resource overallocation and subsequent queue blocking.
The solution lies in the implementation of Learning-Augmented Scheduling algorithms. Datasets like ATLAS (derived from the Alibaba PAI traces) provide the necessary foundation to train machine learning models to predict ground-truth processing times and actual resource constraints based on historical job metadata, user tags, and historical execution profiles. By dynamically predicting the actual trajectory of a workload before it is executed, the scheduler can aggressively pack heterogeneous jobs (e.g., safely interleaving CPU-heavy data preprocessing tasks with GPU-heavy inference runs) without triggering the Out-of-Memory collisions that inevitably cause the Agentic Squatter state.

3. Dynamic Engine Tuning and Telemetry Feedback Loops

Finally, at the micro-engine level, mitigating the vLLM KV-cache collapse requires administrators to abandon rigid, static pre-allocated memory pools. While revolutionary techniques like PagedAttention have radically improved block management by treating GPU memory similarly to an operating system's virtual memory, administrators must actively tune operational parameters based on continuous, real-time Prometheus telemetry.
If the vllm:gpu_cache_usage_perc metric consistently pegs at 1.0, the gpu_memory_utilization parameter must be dynamically adjusted in tandem with the strict, algorithmic enforcement of maximum context lengths (max-model-len). This ensures the continuous batching engine never attempts to allocate memory beyond the absolute physical limits of the VRAM fabric, preventing the application crashes that lead to hardware squatting.

Conclusion

The aggressive, global deployment of Large Language Models at an unprecedented scale has exposed critical, foundational fault lines within traditional high-performance computing infrastructure. As irrefutably evidenced by the raw telemetry extracted from NVIDIA DCGM, vLLM Prometheus metrics, and granular Slurm accounting logs, the stark discrepancy between memory bandwidth requirements and raw compute capacity creates a highly toxic environment for legacy cluster schedulers. This resulting Multi-Resource Queue Instability is not merely a theoretical concern; it results in catastrophic hardware underutilization, mathematically characterized by the Agentic Squatter Signature: extreme memory saturation and maximum power draw coupled with zero computational output.

However, by leveraging the macro-scale empirical data provided by the Alibaba, Microsoft Philly, and Azure ML cluster traces, systems researchers have charted a definitive and highly efficient path forward. The future of LLM infrastructure relies on entirely abandoning monolithic, homogeneous resource allocation in favor of phase-split disaggregated serving architectures, learning-augmented predictive scheduling, and hyper-granular, real-time telemetry monitoring loops. Only by architecting clusters that actively reflect the distinct physical and mathematical realities of prompt prefilling and autoregressive decoding can the industry eliminate queue instability and unlock the true, unencumbered potential of massive-scale GPU-accelerated environments.

Works cited

1. Starburst: A Cost-aware Scheduler for Hybrid Cloud - USENIX, https://www.usenix.org/system/files/atc24-luo.pdf 2. Building Efficient and Practical Machine Learning Systems - Tianwei Zhang, https://tianweiz07.github.io/thesis/23-HuQinghao.pdf 3. Splitwise improves GPU usage by splitting LLM inference phases - Microsoft Research, https://www.microsoft.com/en-us/research/blog/splitwise-improves-gpu-usage-by-splitting-llm-inference-phases/ 4. Splitwise: Efficient generative LLM inference using phase splitting - Microsoft Research, https://www.microsoft.com/en-us/research/publication/splitwise-efficient-generative-llm-inference-using-phase-splitting/ 5. torch.cuda.OutOfMemoryError: CUDA out of memory · Issue #783 · vllm-project/vllm - GitHub, https://github.com/vllm-project/vllm/issues/783 6. [Bug]: high gpu_memory_utilization with 'OOM' and low gpu_memory_utilization with 'No available memory for the cache blocks' · Issue #5274 · vllm-project/vllm - GitHub, https://github.com/vllm-project/vllm/issues/5274 7. torch.cuda.OutOfMemoryError: CUDA out of memory · Issue #2140 · vllm-project/vllm, https://github.com/vllm-project/vllm/issues/2140 8. GPU Monitoring for ML: nvidia-smi, DCGM, and Production Observability Guide - Spheron, https://www.spheron.network/blog/gpu-monitoring-for-ml/ 9. download record errors as .csv, https://harvest.data.gov/harvest_job/efd22466-249f-49ee-b612-d8c2e1ec8be8/errors/record 10. MoritzFeigl/GenAI_for_HLSMs: Final version - Zenodo, https://zenodo.org/records/17608038 11. Power consumption properties of modern multicore server CPUs - Zenodo, https://zenodo.org/records/10647639/files/thesis.pdf 12. huggingface-smol-training-playbook-made-by-crawl4ai.md - GitHub Gist, https://gist.github.com/unclecode/e5da5fb6a1d37022b089e243e0d9e00e 13. chottokun's gists · GitHub, https://gist.github.com/chottokun 14. [Bug]: In v0.6.0 and above, Some of monitoring metrics are not correct. #8178 - GitHub, https://github.com/vllm-project/vllm/issues/8178 15.

vllm.engine.metrics, https://docs.vllm.ai/en/v0.10.1/api/vllm/engine/metrics.html 16. Metrics - vLLM, https://docs.vllm.ai/en/v0.10.1/design/metrics.html 17. Chapter 6. AI Inference Server metrics | vLLM server arguments - Red Hat Documentation, https://docs.redhat.com/en/documentation/red_hat_ai_inference_server/3.2/html/vllm_server_arguments/vllm-metrics_server-arguments 18. [Bug]: Metrics from only one GPU are available through the API · Issue #23341 - GitHub, https://github.com/vllm-project/vllm/issues/23341 19. Production Metrics - vLLM, https://docs.vllm.ai/en/v0.5.2/serving/metrics.html 20. GPU Monitoring: How to Track GPU Performance on Linux Servers - fivenines.io, https://fivenines.io/blog/gpu-monitoring-how-to-track-gpu-performance-on-linux-servers/ 21. How to Monitor GPU Usage in Rancher - OneUptime, https://oneuptime.com/blog/post/2026-03-20-rancher-gpu-monitoring/view 22. THEMIS: Fair and Efficient GPU Cluster Scheduling 1 Introduction - WISR, https://wisr.cs.wisc.edu/papers/nsdi20-themis.pdf 23. clusterdata/cluster-trace-gpu-v2020/README.md at master - GitHub, https://github.com/alibaba/clusterdata/blob/master/cluster-trace-gpu-v2020/README.md 24. microsoft/Trace: End-to-end Generative Optimization for AI Agents - GitHub, https://github.com/microsoft/Trace 25. AzurePublicDataset/AzureLLMInferenceDataset2024.md at master · Azure ... - GitHub, https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2024.md 26. GitHub - alibaba/clusterdata: cluster data collected from production clusters in Alibaba for cluster management research, https://github.com/alibaba/clusterdata 27. DataCenter-Traces-Datasets - Zenodo, https://zenodo.org/records/14564935 28. Attention-based workload prediction and dynamic resource allocation for heterogeneous computing environments - PMC, https://pmc.ncbi.nlm.nih.gov/articles/PMC12976089/ 29. ATLAS: Alibaba Dataset and Benchmark for Learning-Augmented Scheduling, https://openreview.net/forum?id=QBvxXzHdZx 30. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads | USENIX, https://www.usenix.org/conference/atc19/presentation/jeon 31. DIR-LAB/Gen-Parallel-Workloads: Generated Parallel Workloads Archive. - GitHub, https://github.com/DIR-LAB/Gen-Parallel-Workloads 32. msr-fiddle/philly-traces - GitHub, https://github.com/msr-fiddle/philly-traces 33. Azure/AzurePublicDataset: Microsoft Azure Traces - GitHub, https://github.com/Azure/AzurePublicDataset 34. AzurePublicDataset/analysis/AzureLLMInferenceDataset2024.ipynb at master · Azure ... - GitHub, https://github.com/Azure/AzurePublicDataset/blob/master/analysis/AzureLLMInferenceDataset2024.ipynb 35. Splitwise: Efficient Generative LLM Inference Using Phase Splitting - Microsoft, https://www.microsoft.com/en-us/research/wp-content/uploads/2023/12/Splitwise_ISCA24.pdf 36. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency - Jovan Stojkovic, https://jovans2.github.io/files/DynamoLLM_HPCA2025.pdf