OASIS: Optimal Allocation Strategy for Inference Services in Cloud Environments

Viyom Mittal
viyom.mittal@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

Mohammed Baydoun
mohammed.baydoun@hpe.com
American University of Beirut
Beirut, Lebanon

Alok Mishra
alok.mishra@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

Pavana Prakash
prakash@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

Gourav Rattihalli
gourav.rattihalli@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

Aditya Dhakal
aditya.dhakal@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

Eitan Frachtenberg
eitan.frachtenberg@hpe.com
Hewlett Packard Labs
Milpitas, CA, USA

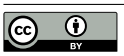
Izzat El Hajj
izzat.elhajj@aub.edu.lb
American University of Beirut
Beirut, Lebanon

Michails Faloutsos
michalis@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

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

Abstract

Provisioning language models as inference services can be significantly challenging to cloud providers because of the need to balance service-level objectives (SLOs) with capital and operational costs. The heterogeneous nature of user queries, combined with temporal variations in workload patterns and the high energy consumption of GPUs warrants an intelligent resource-allocation strategy. We present OASIS (Optimal Allocation Strategy for Inference Services), a comprehensive methodology that combines static and dynamic optimizations to provision inference services efficiently. OASIS employs a two-phase approach: (1) pre-deployment profiling to determine optimal parameter configurations for different query types across various hardware options, and (2) run-time query routing and dynamic provisioning based on observed workload patterns. We evaluate OASIS using real-world BurstGPT traces on NVIDIA A100 and H100 GPUs. Our results show that: (1) GPU frequency scaling from 1320 MHz to 855 MHz reduces power by 39% with only 7% throughput loss; (2) query type classification reveals long-input-short-output workloads achieve 2.1× higher throughput than short-input-long-output; (3) MIG-based multi-tenancy achieves 42-43% power reduction when running two models simultaneously; and (4) runtime adaptation on real campus traces achieves statistically significant 12.4% power reduction while maintaining SLO compliance and identical throughput.



ICPE Companion '26, Florence, Italy

ACM ISBN 979-8-4007-2326-1/2026/05
https://doi.org/10.1145/3777911.3800633

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Hardware** → **Power estimation and optimization**.

Keywords

Language models, Inference serving, Energy efficiency, Resource allocation, Multi-tenancy, Dynamic scheduling

ACM Reference Format:
Viyom Mittal, Mohammed Baydoun, Alok Mishra, Pavana Prakash, Gourav Rattihalli, Aditya Dhakal, Eitan Frachtenberg, Izzat El Hajj, Michails Faloutsos, and Dejan Milojicic. 2026. OASIS: Optimal Allocation Strategy for Inference Services in Cloud Environments. In *Companion of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '26), May 04–08, 2026, Florence, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3777911.3800633

1 Introduction

Motivation

How can a cloud provider optimize resource use in providing language model services? This is the high-level question we address in our work.

The rapid adoption of Large Language Models (LLMs) and Small Language Models (SLMs) in cloud services has created unprecedented demand for efficient inference infrastructure. Cloud providers increasingly offer LLM/SLM inference as a service, typically employing serverless or Function-as-a-Service (FaaS) paradigms, where providers allocate resources dynamically and customers pay per request. We take a cloud-provider perspective in addressing the resource optimization problem for LLM inference services. **Given** a set of language models to be deployed, available hardware options

and customer SLO requirements (maximum P95/P99 latency thresholds), we seek to **minimize total operational cost** (hardware and energy) while **meeting all SLO constraints**. However, this deployment model faces several critical, novel challenges, considering the nature of the LM architecture:

High Energy Consumption. GPU-based LLM inference is computationally intensive and energy-demanding. With millions of daily requests, even small per-query costs create significant energy expenses. Graphic Processing Units (GPUs), the accelerators powering many of these inference services [9], draw high power even when underutilized, making efficient resource allocation both an economic and environmental necessity [23].

Economic cost In addition to the high environmental and economic cost of GPU power consumption (opex) [5], high-end GPUs for use in LM inference are expensive to acquire and suffer from supply-chain volatility (capex) [19]. It is therefore an economic imperative to utilize GPUs efficiently.

Variable Query Traffic Real-world workloads exhibit significant temporal variations. Analysis of the BurstGPT dataset reveals peak request rates of up to 36 queries per second from 3,000 concurrent users, while off-peak periods may see minimal activity [29]. A single LLM instance processing queries sequentially would incur unacceptable latency. For example, with 36 concurrent queries, the last query would wait approximately 108 seconds (assuming 3 seconds per query) if processed sequentially.

Query Heterogeneity User queries vary dramatically in size and complexity, which we classify into four categories based on input and output token counts: Small Input Small Output (SISO), Small Input Large Output (SILO), Large Input Small Output (LISO), and Large Input Large Output (LILO). Each query type has distinct resource requirements and optimal serving configurations. Applying uniform provisioning strategies leads to either resource waste (over-provisioning for simple queries) or SLO violations (under-provisioning for complex queries).

Multi-tenancy Requirements To amortize infrastructure costs, cloud providers must efficiently share GPU resources among multiple tenants or model instances [6]. Traditional approaches like time-slicing provide fairness but suffer from context-switching overhead and do not improve aggregate throughput. Hardware-based solutions like NVIDIA's Multi-Instance GPU (MIG) technology offer true resource isolation and parallelism but require careful configuration to balance performance and utilization [12].

Technical Challenges

Addressing these systemic challenges requires solving four technical challenges:

- **Model-Specific Parameter Optimization:** For a given SLM and hardware platform, what batch size, GPU frequency, and inference variant (e.g., prefix caching, chunked prefill) minimize energy consumption while meeting SLO requirements? Do these optimal parameters generalize across different models of similar size, or does each model require individual profiling?
- **Hardware Selection and Configuration:** Among available GPU options with different costs and capabilities, which provides the most cost-effective deployment for a target workload? Which frequency setting would be optimal for the workload and whether MIG partitioning will help to reduce power consumption?
- **Runtime Adaptation:** Can the system dynamically adjust resource allocation based on observed workload patterns without violating SLOs during transitions?
- **Multi-tenancy Strategy:** Should resources be partitioned using MIG, shared via time-slicing, or managed through software-level batching? Under what conditions is each approach optimal?

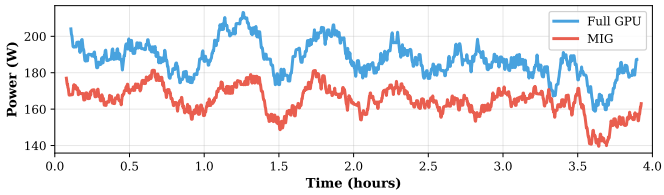


Figure 1: Power consumption over time (60s rolling mean)

Proposed Approach

We present OASIS, a framework that addresses these challenges through a two-phase methodology:

Phase 1: Pre-deployment profiling. Before deploying an SLM, OASIS systematically profiles its performance across different hardware configurations, parameter settings, and query types. For each combination, it measures the maximum serviceable request rate (MSRR)—the highest query rate the configuration can sustain while meeting SLO constraints—and monitors energy consumption. This profiling generates a *hardware cost table* that maps (hardware type, query type) pairs to optimal parameter configurations and their associated MSRR values. In this work, we show the impact of individual parameters (GPU frequency, batch size, inference variants) in isolation; joint optimization across multiple parameters is planned as future work.

Phase 2: Run-time query routing and provisioning. During operation, OASIS analyzes incoming queries and workload patterns to make real-time provisioning decisions. It classifies queries by type, routes them to appropriately configured instances, and dynamically provisions or deallocates resources based on temporal load patterns. This adaptive approach minimizes idle power consumption during low-traffic periods while ensuring sufficient capacity during peaks, (Figure 1), showing how OASIS reduces power consumption by using MIG when the workload is low.

OASIS integrates both system-level optimizations (MIG partitioning) and software-level techniques (continuous batching via vLLM [11]) to maximize resource efficiency. It includes a comprehensive metrics collection system that monitors GPU utilization, memory usage, power consumption, CPU activity, and network I/O, enabling fine-grained performance analysis and correlation studies.

2 Related Work

General and Parameter Optimization

Several recent studies address the problem of optimizing parameters for LLM and SLM runs on GPUs, and a systematic literature

review was conducted by Rostam et al. [22]. Of the 65 papers included in the review, most dealt with LLM training, but several LLM inference frameworks were discussed as well and several optimization techniques were summarized. Unlike this work, a generalized parameter-optimization technique for inference on GPUs was not discussed. However, some specific suggestions for optimizing inference, especially when dealing with limited memory, were proposed, such as selecting the appropriate GPU [34], rewriting the inference engine [24], and optimizing the LLM's KV cache [3, 30].

More broadly, a number of studies focused on optimizing deep-neural-networks in general with techniques such as intermediate representation for GPUs [27], managing data movement to the GPU [7], controlling data layout [13], and even CPU-tailored joint optimization [17]. Even more generally, one can even use LLMs, combined with other metaheuristics, to optimize hyper-parameters of many types of code, not just language models [26]. However, many of these LM-based approaches can be very demanding when it comes to computational resources [4].

Hardware Selection

Zhang et al. have evaluated the suitability of three different consumer- and server-grade GPUs for inference by different LLMs [34]. Li et al. comprehensively evaluated multiple LLMs on multiple hardware platforms (and not only GPUs), with different batch sizes [14]. They found that there is still significant room for software and hardware optimizations. In a related vein, Chitty et al. benchmarked 8 different LLMs on 7 different GPU platforms [2]. Although the paper's main contribution is the benchmarking framework, it does provide specific advice as to which model is most suitable to which GPU among those evaluated.

More generally, LLMCompass is a framework to evaluate LLM inference workloads on different hardware that, like our work, attempts to find the most cost-effective hardware for each given LLM and tune parameters for it [32]. Unlike our work, however, it relies heavily on hardware simulation. Another general approach by Jiang et al. combines low-level performance metrics from benchmarking with a mixed-integer linear programming algorithm to compute the most cost-effective hardware platform for each LLM [8].

Runtime Adaptation

Runtime adaptation for LLM inference has gained significant attention as workloads exhibit temporal variations. A recent study tackled serving disaggregated LLMs with dynamic resource provisioning across multiple nodes, adapting to workload changes in distributed environments [33]. Another study employed statistical multiplexing to dynamically allocate resources across multiple models based on demand patterns, optimizing cluster utilization [15]. The Orca project [31] introduced iteration-level scheduling with selective batching that adapts to varying request arrival patterns and query characteristics.

While these works address runtime adaptation through dynamic scheduling and resource allocation, they focus primarily on model-level scaling without systematically exploring hardware configuration choices (such as MIG partitioning vs. full GPU allocation) or integrating pre-deployment profiling with runtime adaptation for heterogeneous query types, as OASIS does.

Dynamic Voltage Frequency Scaling (DVFS) has been extensively studied as a method for reducing energy consumption. The WAGES [28] project demonstrates how sharing GPUs and dynamically adjusting their frequency can save energy while still meeting SLOs. WAGES uses Nvidia MPS for this purpose, whereas OASIS employs MIG to partition GPUs. Other approaches, such as DynamoLLM [25] and GreenLLM [16], focus solely on exclusive DVFS scaling. These achieve comparable energy savings but do not address GPU sharing or partitioning. Conversely, some studies have explored GPU sharing with power-efficient scheduling to reduce energy consumption [21].

Multi-tenancy Strategy

Multi-tenancy for inference workloads can be achieved through hardware-level partitioning (e.g., NVIDIA MIG) or software-level sharing mechanisms. Software-level approaches enable multi-tenancy through intelligent scheduling and batching. Ma et al. introduce a co-designed compilation and scheduling framework that achieves spatial multiplexing of multiple DNN models on shared GPUs through priority-aware scheduling [18]. The work by Chen et al. addresses multi-tenant low-rank adaptation (LoRA) serving by efficiently batching heterogeneous LoRA adapter requests on a shared base model using specialized CUDA kernels [1]. Kwon et al. pioneered continuous batching with PagedAttention for LLM serving, enabling dynamic request scheduling that improves GPU utilization across varying workloads [10].

These works demonstrate effective multi-tenancy mechanisms, but do not provide systematic guidance on when hardware partitioning (MIG) is optimal compared to full-GPU allocation based on workload characteristics. OASIS addresses this gap by evaluating MIG-based multi-tenancy against full GPU deployment across different query types and load patterns.

3 Methodology

OASIS addresses the SLM provisioning problem through a systematic two-phase methodology that combines offline profiling with online adaptation. This section formalizes the problem, presents our query classification scheme, and details both phases of the OASIS approach.

Problem Definition

We formalize the SLM provisioning problem as an optimization task that minimizes cost while maintaining SLO.

Given:

(1) SLM model M with specific architecture
(2) Set of hardware configurations $H = \{H_1, H_2, \ldots, H_n\}$, where each H_i has cost C_i and specifications (GPU type, memory, compute)
(3) SLO specifying maximum P95 latency L_{max}
(4) Maximum query rate R_{max} that users can submit

Find:

(1) Optimal hardware configuration $H^* \in H$
(2) Optimal parameters for model deployment (batch size which could be continuous as in vLLM, GPU frequency, scheduling policy)

(3) Configuration mode: exclusive GPU or MIG partition
(4) Query routing strategy for different query types
(5) Dynamic provisioning policy based on temporal patterns

Objective: Minimize total operational cost (combining hardware and energy costs):

$$\min_{H^*,\text{config}} \quad C_{total} = C_{hardware} + C_{energy} \tag{1}$$

subject to SLO constraint at maximum query rate:

$$P_{95}(\text{Latency}) \leq L_{max} \quad \text{when serving queries at rate } R_{max} \tag{2}$$

Query Classification

Different query types exhibit vastly different resource requirements. We classify queries based on input and output token counts, using an arbitrary threshold of $\tau = 500$ tokens to distinguish "small" from "large":

SISO (small input, small output): $|T_{in}| \leq \tau \wedge |T_{out}| \leq \tau$. Examples: short Q&A, simple paraphrasing. Characterized by minimal compute and memory requirements.

SILO (small input, large output): $|T_{in}| \leq \tau \wedge |T_{out}| > \tau$. Examples: story generation, essay writing. Compute-intensive during the autoregressive generation phase.

LISO (large input, small output): $|T_{in}| > \tau \wedge |T_{out}| \leq \tau$. Examples: document summarization, classification. Memory-intensive during prefill, requiring substantial KV cache.

LILO (large input, large output): $|T_{in}| > \tau \wedge |T_{out}| > \tau$. Examples: translation, long-form rewriting. Resource-intensive throughout both prefill and generation phases.

Our empirical evaluation (Section 4) focuses on LISO and SILO as representative of the two extremes: LISO dominates prefill cost while SILO dominates generation cost.

Phase 1: Pre-Deployment Profiling

Before deploying a model, OASIS systematically profiles its performance to build a *hardware cost table* that guides runtime decisions.

Profiling procedure: For each model M and hardware configuration H_i, OASIS explores multiple parameter dimensions to identify optimal settings:

(1) **Parameter space definition:** Define the search space:
- GPU clock: $F = \{855, 1035, 1230, 1320, 1410\}$ MHz
- Batch sizes: $B = \{16, 24, 32, 40, 48, 64\}$
- Inference variants: $V = \{\text{standard, prefix-caching, chunked-prefill}\}$

(2) **Query type sweep:** Test each query type (LISO, SILO, etc.) separately
(3) **Single-parameter variation:** For each parameter dimension, vary one parameter while keeping others fixed, to understand individual impact. *Note: Joint optimization across multiple parameters (e.g., co-optimizing frequency and batch size) is computationally expensive and currently out of scope. We demonstrate the sensitivity to individual parameters; full multi-dimensional parameter space exploration is planned as future work.*
(4) **Concurrency ramp:** For each parameter configuration, gradually increase concurrent requests (offered load)from low to high.
(5) **Measure performance:** For each concurrency level c and parameter configuration (f, b, v), measure:
- P95 latency: $L_{95}(H_i, f, b, v, QT, c)$
- Throughput (tokens/sec): $T(H_i, f, b, v, QT, c)$
- Power consumption: $P(H_i, f, b, v, QT, c)$
- Energy per token: $E_{tok} = P/T$

(6) **Determine maximum serviceable request rate:**

$$\text{MSRR}(H_i, f, b, v, QT) = \max\{T(H_i, f, b, v, QT, c)\}$$
$$where\{: L_{95}(H_i, f, b, v, QT, c) \leq L_{max}\} \tag{3}$$

(7) **Record in hardware cost table:** Store tuple $(H_i, f, b, v, QT, \text{MSRR}, P, E_{tok})$

Hardware cost table structure: The resulting table captures the multi-dimensional configuration space. Each entry maps a configuration to its performance characteristics:

$$\text{HCT} : (\text{GPU, Freq, Batch, Variant, QueryType})$$
$$\rightarrow (\text{MSRR}, P, E_{tok}) \tag{4}$$

This enables run-time decisions that balance throughput, latency, and power based on observed workload patterns.

Phase 2: Run-time Provisioning and Adaptation

Armed with the hardware cost table from Phase 1, OASIS makes dynamic provisioning decisions based on observed workload patterns and resource requirements.

Run-time decision making: At run-time, OASIS continuously monitors incoming queries and determines the appropriate resource allocation. The key insight is that GPU requirements are not exact integers but rather continuous—a workload might need 0.3 GPUs, 1.7 GPUs, etc.

GPU requirement calculation: For each model M and query type QT, OASIS computes:

$$GPUs_{required}(M, QT) = \frac{RR(QT)}{\text{MSRR}(H_{full}, f^*, b^*, v^*, M, QT)} \tag{5}$$

where $RR(QT)$ is the observed request rate for query type QT, and (f^*, b^*, v^*) represents the selected parameter configuration.

MIG vs. full GPU decision: The fractional GPU requirement determines the provisioning strategy:

- **Sub-GPU workload** ($GPUs_{required} < 1$): Use MIG partitioning to allocate only the necessary compute resources. This partitioning enables multi-tenancy and reduces idle power consumption.
- **Full GPU workload** ($GPUs_{required} \geq 1$): Allocate $\lceil GPUs_{required} \rceil$ complete GPUs to maximize throughput and ensure SLO compliance.

Multi-Tenancy with MIG: When multiple models require sub-GPU resources, OASIS packs them onto a single physical GPU using MIG partitioning.

Dynamic provisioning algorithm:

Energy-aware provisioning: The provisioning decision balances performance and energy efficiency. Our evaluation (Section 4) demonstrates that MIG-based multi-tenancy achieves 42-43% power reduction (270W vs. 467W) when running two models simultaneously, making it optimal for scenarios where individual model loads are light, but combined tenancy justifies GPU utilization.

Algorithm 1 OASIS runtime provisioning

Require: Hardware cost table HCT, monitoring interval Δt
Initialize: Deploy based on expected peak load
while system running **do**
  Collect queries over interval Δt
  Classify each query q into type $QT(q)$ based on token counts
  Compute request rate $RR(QT)$ for each query type
  for each model M and query type QT **do**
    Retrieve MSRR(H_{full}, M, QT) from HCT
    Calculate required GPU capacity:
    $GPUs_{required}(M, QT) = \frac{RR(QT)}{\text{MSRR}(H_{full}, M, QT)}$
    if $GPUs_{required}(M, QT) < 1$ **then**
      Use MIG partition (resource requirement less than full GPU)
      Calculate MIG partition size based on $GPUs_{required}$
      Deploy model M on MIG partition
    else if $GPUs_{required}(M, QT) \geq 1$ **then**
      Use $\lceil GPUs_{required}(M, QT) \rceil$ **full GPUs**
      Deploy model M across required GPUs
    end if
  end for
  Route queries to appropriate instances based on QT
end while

Query routing: Once resources are provisioned, OASIS routes incoming queries based on their classified type (LISO, SILO, etc.) to instances configured optimally for that query type. This routing ensures that compute-intensive SILO queries and memory-intensive LISO queries receive appropriate resource allocations.

4 Experimental Evaluation

4.1 Experimental Platform

Hardware: **GPU** NVIDIA A100-SXM4 (80GB), H100-SXM; **CPU** AMD EPYC 7443 24-Core Processor; **Memory** 256 GB.

Software: **vLLM** v0.6.3.post1; **CUDA** 12.4; **PyTorch** 2.5.1; **Python** 3.10.

Models evaluated: Llama-3-8B, Llama-3.1-8B; Qwen-7B-Chat; Mistral-7B-v0.3, Baichuan-7B; Yi-6B; starcoder2-15b

SLO Constraint: The P95 latency of responses should be within 20 seconds [20].

Query types: We evaluate two representative patterns:

- **LISO (Large Input, Small Output):** 1000 input tokens, 50 output tokens — representative of document summarization, classification tasks
- **SILO (Small Input, Large Output):** 50 input tokens, 1000 output tokens — representative of content generation, story writing tasks

4.2 Phase 1: Pre-Deployment Profiling Results

Table 1: Llama3-8B pre-deployment profiling on full GPU

Query Type	Concurrency	P95 Latency (s)	Throughput	SLO Met
LISO	50	11.56	4.41	✓
	75	16.85	4.51	✓
	81	18.33	4.50	✓
	84	18.82	4.55	✓
	86	19.39	4.52	✓
	87	19.68	4.54	✓
	88	20.11	4.49	×
SILO	25	15.69	1.78	✓
	38	17.17	2.10	✓
	44	19.20	1.98	✓
	47	19.50	2.00	✓
	49	19.36	2.15	✓
	50	20.09	2.55	×

Table 2: Qwen-7B pre-deployment profiling on full GPU

Query Type	Concurrency	P95 Latency (s)	Throughput	SLO Met
LISO	50	7.45	6.76	✓
	75	11.43	6.75	✓
	87	13.11	6.76	✓
	90	13.05	6.78	✓
	91	15.89	6.77	✓
	92	23.13	6.76	×
SILO	25	17.81	1.52	✓
	37	19.56	1.84	✓
	38	20.10	1.83	×
	40	21.29	2.00	×

4.2.1 Query Type Comparison. We systematically profile two representative models—Llama3-8B and Qwen-7B—by increasing concurrency for both LISO and SILO query types to understand their distinct resource requirements.

Tables 1 and 2 show the profiling results for both models under increasing concurrency levels. Our key findings are:

- **Query Type Dominance:** LISO workloads achieve **1.8-2.5× higher concurrency** and **2.1× higher throughput** compared to SILO across both models, demonstrating that query type classification is critical for resource allocation.
- **MSRR Values:** Llama3-8B achieves MSRR(LISO) = 4.54 req/s and MSRR(SILO) = 2.15 req/s, while Qwen-7B achieves MSRR(LISO) = 6.77 req/s and MSRR(SILO) = 1.84 req/s.
- **Model-Specific Differences:** Qwen-7B achieves **1.5× higher throughput** than Llama3-8B for LISO workloads (6.77 vs 4.54 req/s), but similar performance for SILO (1.84 vs 2.15 req/s), validating the need for per-model profiling.

Table 3: GPU frequency scaling: Llama-3-8B on A100

Frequency (MHz)	Avg. Power (W)	Throughput (tok/s)	P99 Latency (ms)	Energy (mJ/tok)
1320	239.4	914.8	162.4	261.7
1035	153.6	864.6	4060.8	177.7
855	145.2	846.7	869.9	171.5

Takeaway 1. *Query type significantly impacts resource requirements: LISO workloads achieve 2.1× higher throughput than SILO. Different models exhibit different optimal operating points—Qwen-7B achieves 1.5× higher LISO throughput than Llama3-8B, validating the need for per-model profiling.*

4.2.2 GPU Frequency Scaling Analysis. To understand the impact of GPU frequency on power consumption and performance, we profile Meta-Llama-3-8B across different frequency settings on an A100 GPU. Table 3 and Figure 2 illustrate the frequency-power-throughput trade-offs. Our key findings are:

- **Power Reduction:** Decreasing frequency from 1320 MHz to 855 MHz reduces average power consumption by **39%** (239.4W → 145.2W).
- **Throughput Impact:** Throughput decreases by only **7%** (914.8 → 846.7 tok/s), showing minimal performance penalty.
- **Energy Efficiency:** Energy per token improves by **34%** (261.7 → 171.5 mJ/tok) at lower frequencies.
- **Latency Trade-off:** While lower frequencies reduce power, latency can spike (4060ms at 1035 MHz) depending on workload characteristics.
- **Optimal Selection:** For SLO = 20s, 855 MHz provides the best energy efficiency while maintaining acceptable latency.

Takeaway 2. *GPU frequency scaling offers significant energy savings with minimal throughput loss. Reducing frequency from 1320 MHz to 855 MHz saves 39% power (94W) with only 7% throughput reduction, demonstrating frequency tuning as a key optimization parameter.*

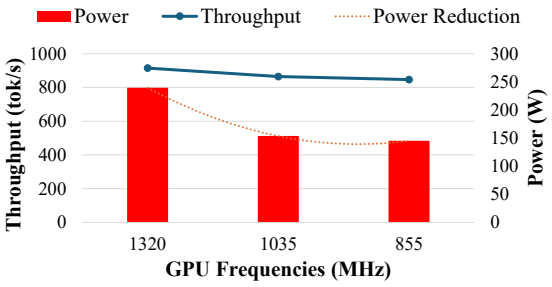


Figure 2: GPU frequency scaling: Llama-3-8B on A100

4.2.3 Cross-Model Energy Efficiency Comparison. We profile four different SLMs (7B-15B parameters) at a fixed GPU frequency of 1335 MHz and SLO of 20 seconds to compare their energy efficiency characteristics under identical hardware conditions. Table 4 and Figure 3 present the comparative results. Our key findings are:

Table 4: Cross-model efficiency at 20s SLO (1335 MHz)

Model	Avg. Power (W)	Throughput (tok/s)	Energy (mJ/tok)
Mistral-7B-v0.3	221.4	2571.5	86.1
Baichuan-7B	235.5	2227.2	105.7
Meta-Llama-3.1-8B	223.9	2043.0	109.6
bigcode/starcoder2-15b	233.4	1971.0	118.4

- **Energy Efficiency Variation:** At the same GPU frequency (1335 MHz), energy consumption ranges from **86.1 mJ/tok (Mistral-7B)** to **118.4 mJ/tok (starcoder2-15b)**—a **1.4× difference** even under identical hardware conditions.
- **Power Consumption Consistency:** Despite different model architectures, the average power consumption remains relatively stable (221-236W), demonstrating that GPU frequency, not model size alone, dominates power draw.
- **Model Architecture Matters:** Even at similar parameter counts, energy efficiency varies by up to **1.3×** (Mistral: 86.1 vs Llama-3.1: 109.6 mJ/tok), highlighting architectural differences in compute efficiency.
- **Throughput Diversity:** Mistral-7B achieves highest throughput (2571 tok/s) while starcoder2-15b achieves lowest (1971 tok/s), a **1.3× variation**, with smaller models generally performing better.
- **Size-Performance Trade-off:** The larger starcoder2-15b (15B parameters) delivers lower throughput and worse energy efficiency compared to smaller 7B models, indicating that model size does not always correlate with efficiency.

Takeaway 3. *Model selection significantly impacts energy costs even at the same GPU frequency. Among 4 SLMs at 1335 MHz, energy efficiency varies by 1.4× (86.1-118.4 mJ/tok) and throughput by 1.3×. Model architecture and parameter count both influence efficiency—per-model profiling remains essential.*

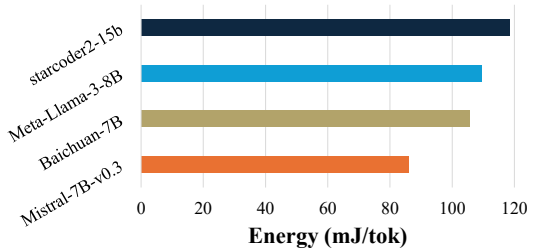


Figure 3: Cross-model efficiency at 20s SLO (1335 MHz)

4.2.4 Hardware Platform Comparison. We compare the same model (Yi-6B) across A100 and H100 GPU platforms to understand hardware selection trade-offs. We evaluate multiple frequencies for each platform to identify optimal operating points. Table 5 and Figure 4 compare the platforms across multiple frequency settings. Our key findings are:

Table 5: Hardware platform comparison: Yi-6B performance

GPU	Freq (MHz)	Avg. Power (W)	Throughput (tok/s)	P99 Latency (ms)
A100	855	142.2	1444.8	2040.9
A100	1035	140.9	1511.3	1208.9
H100	1530	265.9	1698.5	86.3
H100	1650	280.6	1702.3	96.2
H100	1755	291.1	1704.4	2632.1

- **A100 Efficiency:** At 1035 MHz, A100 delivers 1511 tok/s with only **140.9W**—the most energy-efficient configuration across both platforms.
- **H100 Performance:** At 1530 MHz, H100 achieves **12% higher throughput** (1698 vs 1511 tok/s) but consumes **89% more power** (266W vs 141W) compared to A100.
- **Power Scaling:** H100 average power consumption ranges from 266W to 291W across frequencies, consistently higher than A100's 141-142W range.
- **Latency Variability:** H100 at 1530 MHz achieves excellent latency (86ms) but shows high variability at other frequencies (2632ms at 1755 MHz).
- **Cost-Benefit:** For light workloads meeting SLO, A100 at lower frequencies provides better energy efficiency. H100 justifies its power premium only for throughput-critical workloads.

Takeaway 4*. Hardware selection requires workload-specific analysis. H100 provides 12% higher throughput but 89% higher average power consumption compared to A100. For energy-conscious deployments, A100 at optimized frequency offers better efficiency.*

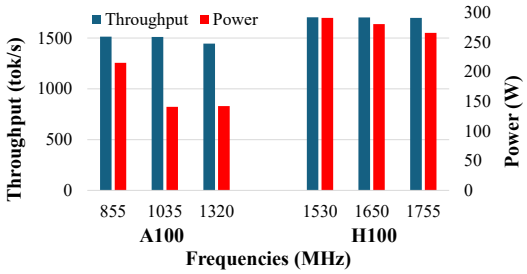


Figure 4: Hardware comparison: Yi-6B performance

4.2.5 MIG-Based Multi-Tenancy Analysis. Consider a cloud provider serving multiple customers who require both Llama3-8B and Qwen-7B simultaneously. We compare two deployment strategies:

(1) **Separate Full GPUs:** Dedicate one A100 per model.
(2) **MIG-Based Sharing:** Partition a single A100 into MIG instances, running both models on shared hardware.

Table 6 compares the two approaches for LISO workloads at 50 concurrent requests per model.

Table 6: Multi-tenancy comparison: running both models simultaneously (LISO workload)

Setup	Model	P95 Lat. (s)	Throughput	Concurrency	Power (W)
Full GPU	Llama3-8B	6.85	7.59	50	235
Full GPU	Qwen-7B	7.44	6.80	50	232
				Total:	**467**
MIG	Llama3-8B	13.87	3.69	50	270
MIG	Qwen-7B	19.96	3.17	50	270
				Total:	**270**

LISO Workload Insights:

- **Separate full GPUs:** Consume **467W total** (235W + 232W) but provide superior latency (6.85s and 7.44s) and throughput (7.59 and 6.80 req/s).
- **MIG configuration:** Consumes only **270W**, achieving **42% power reduction** compared to separate GPUs.
- **Performance trade-off:** MIG configuration has higher latency (13.87s and 19.96s) and lower throughput (3.69 and 3.17 req/s), but still **meets SLO requirements**.
- **Energy savings:** When both models must run continuously, MIG saves **197W**, equivalent to approximately **4.7 kWh per day** or **142 kWh per month**.

Table 7 shows the same comparison for SILO workloads, which are more resource-intensive.

Table 7: Multi-tenancy comparison: running both models simultaneously (SILO workload)

Setup	Model	P95 Lat. (s)	Throughput	Concurrency	Power (W)
Full GPU	Llama3-8B	12.20	0.25	3	234
Full GPU	Qwen-7B	12.24	0.25	3	233
				Total:	**467**
MIG	Llama3-8B	21.85	0.14	3	265
MIG	Qwen-7B	20.01	0.15	3	265
				Total:	**265**

SILO Workload Insights:

- **Power consumption:** Despite low concurrency, full GPUs still consume **467W total**. MIG reduces this to **265W** (**43% savings**).
- **SLO compliance:** MIG configuration for SILO workloads operates at the edge of SLO compliance (21.85s and 20.01s, both just above 20s threshold). For strict SLO requirements, full GPU deployment would be necessary.
- **Throughput impact:** SILO throughput is low (0.14-0.25 req/s) regardless of configuration, highlighting the resource intensity of large-output generation.

Takeaway 5. *MIG-based multi-tenancy achieves 42-43% power reduction (197W savings) when running two models simultaneously. For LISO workloads with moderate demand, MIG meets SLO while drastically reducing energy costs. SILO workloads operate near SLO limits with MIG, requiring careful provisioning. Figure 5 shows the difference in power consumption between full GPU and MIG configurations for both LISO and SILO workloads.*

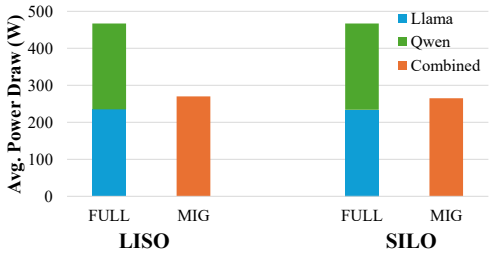


Figure 5: Multi-tenancy (simultaneous run) comparison

4.3 Phase 2: Runtime Adaptation with Real Workload

To validate OASIS's runtime adaptativeness (Phase 2), we conduct a comprehensive evaluation comparing full GPU and MIG configurations under real-world workload conditions.

4.3.1 Experimental Setup.

Hardware configuration. We use an NVIDIA A100X-80GB GPU for both configurations: (1) *Full GPU*: entire 80GB device allocated to a single vLLM instance, and (2) *MIG*: Using half of the GPU's hardware.

Software stack. We deploy vLLM v0.6.3 serving the Meta-Llama-3-8B-Instruct model with identical inference parameters: temperature 0.8, top-p 0.95, and max tokens 512.

4.3.2 Workload Characteristics and Context. **Real-World campus deployment.** We evaluate OASIS using a 4-hour daytime trace from the BurstGPT dataset [29], which provides actual Azure OpenAI GPT service logs collected over 213 days from a regional deployment. This dataset captures authentic user behavior during peak hours, representing the type of bursty traffic one would expect from a campus environment serving over 3,000 students.

The selected trace contains 153 requests from an ostensibly small user population, typical of early deployment phases or specialized campus services. *Critically, even during peak request periods, the moderate traffic volume (average 41 requests/hour) demonstrates that provisioning full GPUs would waste substantial resources.* The workload distribution is heavily skewed toward short-output queries: 64.7% SISO (Short Input, Short Output) and 29.4% LISO (Long Input, Short Output), totaling 94.1% of all requests. Only 5.9% are SILO queries, with no LILO requests observed. This pattern reflects common student use cases such as quick Q&A and code snippet generation demonstrating that *MIG partitions can efficiently serve typical campus workloads without over-provisioning.*

Table 8: Runtime performance: full GPU vs MIG

Metric	Full GPU	MIG	Delta (%)
Completed Requests	312	312	0.0
Token Throughput (tok/s)	15.45	15.44	-0.1
Mean TTFT (ms)	117.5	204.4	+74.0
P99 TTFT (ms)	289.7	502.3	+73.4
Mean E2E Latency (s)	2.29	3.83	+66.8
P99 E2E Latency (s)	6.78	11.50	+69.6
Mean Power (W)	188.2	164.9	-12.4
Total Energy (Wh)	750.9	657.8	-12.4

4.3.3 Temporal Dynamics. The workload exhibits bursty arrival patterns with a coefficient of variation (CV) of 1.70, indicating high variability in request inter-arrival times. Figure 6 shows periods of intense activity interspersed with quieter intervals—characteristic of campus usage where demand surges during class transitions and deadlines.

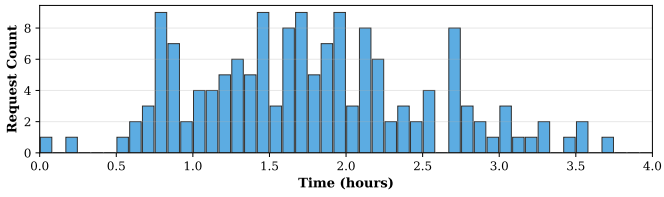


Figure 6: Request arrival pattern over 4-hour trace (Day 2, hours 8-12) with 5-minute bins

Power consumption traces reveal the energy efficiency gains from MIG adaptation. Figure 1 shows instantaneous power draw (60-second rolling mean) for both configurations, demonstrating a consistent 20-30W reduction during active serving periods. The MIG configuration maintains power consumption around 165W while the full GPU fluctuates between 185-195W. The energy consumption analysis reveals that MIG achieves 93.1 Wh savings (12.4% reduction) over the 4-hour period.

4.3.4 Performance and SLO Compliance. Despite using only half the GPU resources, the MIG configuration maintains SLO compliance while achieving significant energy savings. Although MIG exhibits 69.6% higher P99 latency (11.5s vs 6.8s), both configurations comfortably meet the P99 SLO requirement of 20s, with all requests completing well within bounds.

Table 8 summarizes key performance metrics. The MIG configuration maintains identical throughput (15.44 vs 15.45 tokens/s) and request completion rate (312 requests), while reducing mean power consumption from 188.2W to 164.9W—a statistically significant 12.4% reduction ((t-test $p < 10^{-18}$). Both TTFT and E2E latency increase moderately (+74% and +69.6% respectively), but remain within acceptable bounds for interactive applications.

4.3.5 Energy Efficiency Analysis. As shown in Figure 1 and quantified in Table 8, the MIG configuration consistently operates at lower power levels (median 164.9W) compared to full GPU (median

Table 9: Energy efficiency and cost savings

Metric	Full GPU	MIG	Savings
Energy per Request (Wh)	2.41	2.11	0.30 (12.4%)
Daily Cost @ $0.12/kWh ($)	86.01	75.34	10.67
Daily CO_2 Emissions (kg)	281.2	246.4	34.8

188.2W), with minimal overlap in distributions. This translates to 93.1 Wh total energy savings over the 4-hour period.

Table 9 quantifies the economic and environmental impact at scale. For a campus serving 3,000 students, the 12.4% energy reduction would save $10.67/day in electricity costs and avoid 34.8 kg CO_2/day emissions. These benefits compound over academic terms, making OASIS's runtime adaptation a practical approach to sustainable LLM serving.

4.3.6 Key Findings. Our runtime adaptation evaluation demonstrates that: (1) **MIG sufficiency prevents resource waste**: even during peak morning hours (Day 2, 8-12) with bursty workload patterns (CV=1.70), a 1g.10gb MIG partition successfully serves all requests while maintaining SLO compliance. Provisioning full GPUs for such moderate traffic (41 req/hr) would waste 85% of GPU resources. (2) **Substantial energy savings**: the 12.4% power reduction achieved without degrading throughput translates to meaningful cost and environmental benefits at deployment scale. (3) **Practical deployment viability**: rather than over-provisioning full GPUs for workloads with moderate user counts, OASIS can dynamically adapt to traffic patterns, allocating MIG partitions during typical usage and reserving full GPU capacity only when necessary.

5 Conclusion

We present OASIS, a two-phase methodology for efficient SLM inference provisioning that combines systematic pre-deployment profiling with runtime workload adaptation. Our extensive evaluation on NVIDIA A100 and H100 GPUs with real-world BurstGPT traces demonstrates OASIS's effectiveness across multiple dimensions.

Phase 1 pre-deployment profiling reveals critical parameter sensitivities: (1) GPU frequency scaling achieves 39% power reduction (239.4W→145.2W) with only 7% throughput loss, improving energy efficiency by 34%; (2) query-type classification shows LISO workloads achieve 2.1× higher throughput than SILO, with maximum sustainable request rates of 6.77 vs 1.84 req/s for Qwen-7B; (3) cross-model comparison demonstrates 1.4× energy efficiency variation (86.1-118.4 mJ/tok) at identical GPU frequency; (4) hardware platform analysis shows H100 provides 12% higher throughput but consumes 89% more power than A100; and (5) MIG-based multi-tenancy achieves 42-43% power reduction (467W→265W) when running two models simultaneously while meeting SLO requirements.

Phase 2 runtime adaptation validates OASIS's dynamic provisioning capability using a 4-hour real-world campus workload trace (153 requests, CV=1.70 burstiness). MIG configuration (1g.10gb partition) achieves statistically significant 12.4% power reduction (188.2W→164.9W, t-test $p < 10^{-18}$, Mann-Whitney $p < 10^{-19}$) while maintaining identical throughput (15.44 tok/s) and SLO compliance (P99 latency: 11.5s vs 20s threshold). This translates to 93.1 Wh energy savings over 4 hours, equivalent to $10.67/day cost savings and 34.8 kg CO_2/day emission reduction at campus scale.

Future work: While OASIS demonstrates significant efficiency gains through single-parameter optimization, several directions warrant further investigation:

Multi-parameter joint optimization. OASIS currently performs single-parameter sweeps (frequency, batch size, etc.). A comprehensive multi-parameter optimization framework could explore joint configurations (e.g., frequency + batch size + scheduling policy) to identify global optima rather than local maxima from individual parameter tuning.

Workload prediction and proactive adaptation. Phase 2 reacts only to observed workload patterns. Time-series forecasting (LSTM, ARIMA) could enable proactive resource provisioning based on predicted traffic patterns, reducing transition overhead and improving energy efficiency.

Cross-GPU architecture support. We evaluated A100 and H100 GPUs, but emerging architectures have different power profile. Automated profiling pipelines that generalize across GPU vendors and generations would enhance OASIS's applicability.

References

[1] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2024. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems* 6 (2024), 1–13.

[2] Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. 2024. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1362–1379.

[3] Aditya Dhakal, Pedro Bruel, Gourav Rattihalli, Sai Rahul Chalamalasetti, and Dejan Milojicic. 2024. LLM Serving With Efficient KV-Cache Management Using Triggered Operations. *Memory* 100 (2024), 200.

[4] Jingzhi Gong, Vardan Voskanyan, Paul Brookes, Fan Wu, Wei Jie, Jie Xu, Rafail Giavrimis, Mike Basios, Leslie Kanthan, and Zheng Wang. 2025. Language models for code optimization: Survey, challenges and future directions. *arXiv preprint arXiv:2501.01277* (2025).

[5] Kristen Howell, Gwen Christian, Pavel Fomitchov, Gitit Kehat, Julianne Marzulla, Leanne Rolston, Jadin Tredup, Ilana Zimmerman, Ethan Selfridge, and Joseph Bradley. 2023. The economic trade-offs of large language models: A case study. *arXiv preprint arXiv:2306.07402* (2023).

[6] Bodun Hu, Jiamin Li, Le Xu, Myungjin Lee, Akshay Jajoo, Geon-Woo Kim, Hong Xu, and Aditya Akella. 2024. Blockllm: Multi-tenant finer-grained serving for large language models. *arXiv preprint arXiv:2404.18322* (2024).

[7] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems* 3 (2021), 711–732.

[8] Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Guoliang He, Xupeng Miao, Ana Klimovic, Bin Cui, Binhang Yuan, and Eiko Yoneki. 2025. Demystifying cost-efficiency in llm serving over heterogeneous gpus. *arXiv preprint arXiv:2502.00722* (2025).

[9] Christoforos Kachris. 2025. A survey on hardware accelerators for large language models. *Applied Sciences* 15, 2 (2025), 586.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[12] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters. In *Proceedings of the 13th Symposium on Cloud Computing*. 173–189.

[13] Chao Li, Yi Yang, Min Feng, Srimat Chakradhar, and Huiyang Zhou. 2016. Optimizing memory efficiency for deep convolutional neural networks on GPUs. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 633–644.

[14] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, et al. 2024. Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466* (2024).

[15] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. 2023. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 663–679.

[16] Qunyou Liu, Darong Huang, Marina Zapater, and David Atienza. 2025. GreenLLM: SLO-Aware Dynamic Frequency Scaling for Energy-Efficient LLM Serving. arXiv:2508.16449

[17] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. 2019. Optimizing {CNN} model inference on {CPUs}. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1025–1040.

[18] Lixian Ma, Haoruo Chen, En Shao, Leping Wang, Quan Chen, and Guangming Tan. 2024. ElasticRoom: Multi-Tenant DNN Inference Engine via Co-design with Resource-constrained Compilation and Strong Priority Scheduling. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*.

[19] Guanzhong Pan, Vishal Chodnekar, Abinas Roy, and Haibo Wang. 2025. A Cost-Benefit Analysis of On-Premise Large Language Model Deployment: Breaking Even with Commercial LLM Services. *arXiv preprint arXiv:2509.18101* (2025).

[20] Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2024. Queue management for SLO-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 18–35.

[21] Gourav Rattihalli, Ninad Hogade, Aditya Dhakal, Eitan Frachtenberg, Rolando Pablo Hong Enriquez, Pedro Bruel, Alok Mishra, and Dejan Milojicic. 2023. Fine-Grained Heterogeneous Execution Framework with Energy Aware Scheduling. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 35–44. doi:10.1109/CLOUD60044.2023.00014

[22] Zhyar Rzgar K Rostam, Sándor Szénási, and Gábor Kertész. 2024. Achieving peak performance for large language models: A systematic review. *IEEE access* (2024).

[23] Mohammad Sadrosadati, Seyed Borna Ehsani, Hajar Falahati, Rachata Ausavarungnirun, Arash Tavakkol, Mojtaba Abaee, Lois Orosa, Yaohua Wang, Hamid Sarbazi-Azad, and Onur Mutlu. 2019. ITAP: Idle-time-aware power management for GPU execution units. *ACM Transactions on Architecture and Code Optimization (TACO)* 16, 1 (2019), 1–26.

[24] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 590–606.

[25] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1348–1362. doi:10.1109/HPCA61900.2025.00102

[26] Niki van Stein, Diederick Vermetten, and Thomas Bäck. 2024. In-the-loop hyperparameter optimization for llm-based automated design of heuristics. *ACM Transactions on Evolutionary Learning* (2024).

[27] Leyuan Wang, Zhi Chen, Yizhi Liu, Yao Wang, Lianmin Zheng, Mu Li, and Yida Wang. 2019. A unified optimization approach for cnn model inference on integrated gpus. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.

[28] Tianyu Wang, Gourav Rattihalli, Aditya Dhakal, Xulong Tang, and Dejan Milojicic. 2025. WAGES: Workload-Aware GPU Sharing System for Energy-Efficient Serverless LLM Serving. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25)*. Association for Computing Machinery, New York, NY, USA, 508–515. https://doi.org/10.1145/3731599.3767396

[29] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. 2025. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 5831–5841.

[30] Yi Xiong, Hao Wu, Changxu Shao, Ziqing Wang, Rui Zhang, Yuhong Guo, Junping Zhao, Ke Zhang, and Zhenxuan Pan. 2024. Layerkv: Optimizing large language model serving with layer-wise kv cache management. *arXiv preprint arXiv:2410.00428* (2024).

[31] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[32] Hengrui Zhang, August Ning, Rohan Prabhakar, and David Wentzlaff. 2023. A hardware evaluation framework for large language model inference. *arXiv preprint arXiv:2312.03134* (2023).

[33] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 787–808.

[34] Longteng Zhang, Xiang Liu, Zeyu Li, Xinglin Pan, Peijie Dong, Ruibo Fan, Rui Guo, Xin Wang, Qiong Luo, Shaohuai Shi, et al. 2023. Dissecting the runtime performance of the training, fine-tuning, and inference of large language models. *arXiv preprint arXiv:2311.03687* (2023).