

Beyond Utilization: Energy-Conscious GPU Sharing for Inference Serving

Prasoon Sinha Dimitrios Liakopoulos Nathan Lemma Neeraja J. Yadwadkar

The University of Texas at Austin

Abstract

GPUs are expensive, yet inference-serving GPU clusters remain heavily underutilized. To improve utilization, state-of-the-art systems adopt GPU multiplexing. However, optimizing solely for utilization can counterintuitively increase energy consumption. Designing policies that treat power and energy as first-order metrics requires understanding how deployment decisions—GPU allocation size, operating frequency, and batch size—affect energy, latency, and throughput. These relationships are complex, leading existing approaches to rely on extensive profiling. At scale, profiling becomes prohibitively expensive: each model can be deployed under hundreds of configurations, and profiling itself incurs significant energy cost, necessitating accurate yet energy-conscious methods. Further, such systems must model the power draw of colocated models on shared GPUs and adapt to dynamic workload fluctuations. We present EnerTune, an inference serving system that reduces energy consumption while meeting performance SLOs. EnerTune introduces analytical models to estimate per-model performance and power, and the power draw of colocated models on shared GPUs, and uses them in an energy-aware bin-packing algorithm to jointly determine model placement and configuration. EnerTune meets performance SLOs while reducing energy consumption by 1.4–2.3× and power draw by 1.3–2.6× over state-of-the-art baselines.

CCS Concepts: • Hardware → Power estimation and optimization; • Computing methodologies → Machine learning; • Computer systems organization → Cloud computing.

ACM Reference Format:

Prasoon Sinha Dimitrios Liakopoulos Nathan Lemma Neeraja J. Yadwadkar. 2026. Beyond Utilization: Energy-Conscious GPU Sharing for Inference Serving. In *ACM SIGOPS 32nd Symposium on Operating Systems Principles (SOSP '26)*, September 29–October 02, 2026, Prague, Czech Republic. ACM, New York, NY, USA, 25 pages. <https://doi.org/10.1145/3830418.3843913>



This work is licensed under a Creative Commons Attribution 4.0 International License.

SOSP '26, Prague, Czech Republic

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2585-2/2026/09

<https://doi.org/10.1145/3830418.3843913>

1 Introduction

Despite the ever-growing machine learning (ML) demand, production hyperscalers with massive GPU fleets remain severely underutilized—averaging only 10–50% utilization [27, 61, 66, 140, 141, 143, 148]. To improve utilization, a line of recent works multiplex multiple inference jobs on a single GPU, either temporally or spatially [20, 23, 75, 117, 127, 140, 141, 158, 161]. However, being energy-agnostic in their placement and GPU allocation decisions, state-of-the-art serving systems inflate energy consumption by 1.6× by accidentally multiplexing *energy-incompatible* models (models that, when colocated, consume more energy than serving them on dedicated GPUs §2.1) [120]. As available power is becoming a limiting resource [29, 59, 67, 86, 122, 132], we must treat reduction in power and energy consumption, in addition to improving utilization, as first-class objectives in today's inference-serving systems.

Prior work, including DynamoLLM [124], μ Serve [112], and others [22, 25, 26, 111, 116, 125, 145, 149, 152, 163], proposes power and energy management techniques, but targets large models that require one or more full GPUs, leaving limited opportunity for spatial multiplexing. Other works focus on small, resource-constrained hardware (e.g., edge or small GPUs) [41, 60, 65, 131], where limited resources similarly leave little opportunity for spatial multiplexing. In datacenter settings, however, smaller models remain widely used. For example, Alibaba [128, 140, 141], Meta [27], and private datacenters [61] report serving millions of inference requests for smaller CNNs/NLPs/LLMs that underutilize GPUs. This renders multiplexing both feasible and necessary for improving overall datacenter efficiency. However, existing state-of-the-art inference serving systems that spatially multiplex GPUs [75, 77, 114, 117, 141, 158, 161] do not treat energy as a first-order objective, leading to placement and configuration decisions that increase energy consumption. *In this work, we improve the energy efficiency of ML serving systems that spatially multiplex GPUs.* However, doing so introduces several challenges.

Challenge 1: Energy-conscious profiling at scale. Designing energy-aware policies requires understanding how deployment decisions—GPU allocation, frequency, and batch size—affect energy, latency, and throughput. Each model can be deployed under hundreds of configurations, with a non-trivial impact on these metrics. For example, Stable Diffusion has >450 configurations that exhibit 90.3× latency variation

(0.036–3.25 min), 16.9× throughput variation (9.83–166.3 requests/min), and 13.4× variation in energy per request (353–4747 J). While prior work studies the impact of individual knobs on performance [23, 28, 33, 47, 75, 112, 115, 117], they do not capture the joint impact of all knobs on power/energy, thereby necessitating extensive profiling. However, naively profiling all configurations is expensive: profiling only 11 small models (<6B parameters) takes > 60 GPU-hours (5–6 GPU-hours per model) and consumes > 5MJ of energy (0.1–0.9 MJ per model, 0.53MJ on average). This profiling is impractical in production: cloud platforms serve hundreds to thousands of models (e.g., AWS Bedrock [5], Azure AI Foundry [90], Meta’s production fleet [89]), and must frequently re-profile them [10, 16, 52, 53] as hardware/software stacks change [19, 166], or model variants change by compression [42, 51, 84] or fine-tuning [35, 81], making exhaustive profiling prohibitively expensive. *We need accurate yet time-efficient and energy-conscious methods that capture these power–performance tradeoffs with minimal profiling overhead.*

Challenge 2: Accurate power modeling of shared GPUs.

Designing energy-aware placement policies also requires understanding how colocated models jointly affect GPU power draw. Simply summing per-model power profiles, proposed by prior work [61], leads to inaccurate power estimation (§7.3), as it does not capture interactions through shared GPU components (e.g., memory controllers) and device-level power-gating mechanisms. Prior work in power modeling [2, 6, 8, 39, 56, 70, 76, 83, 94, 121, 138, 144] relies heavily on low-level hardware metrics. Extending these approaches to shared GPUs would require profiling all model colocations, rendering them impractical. Thus, *we need scalable and accurate methods to estimate the power draw of colocated models without exhaustive joint profiling.*

Challenge 3: Dynamically changing runtime conditions.

Making initial cluster-wide resource management decisions is already challenging due to the combinatorially large deployment space. Runtime fluctuations, such as changes in request arrival rates, further complicate these decisions (§2.4). However, simply augmenting existing placement algorithms, like those used by Usher [117], FGD [141], GPULets [23], and others [73, 75, 77, 80, 112, 148, 158, 159, 161, 168], with accurate colocation power models is non-trivial. We find that maximizing energy efficiency requires frequency-aware colocation. This is challenging because colocated models share a single GPU frequency, yet have heterogeneous performance characteristics and frequency requirements, making it difficult to select a frequency that balances energy efficiency with SLO compliance (see §2.4, 5.1). Moreover, we must adapt these decisions under dynamic conditions, requiring potentially frequent re-evaluation. *Thus, we need low-overhead placement and runtime policies that can efficiently adapt decisions while maximizing energy efficiency and meeting SLOs.*

We build EnerTune, a multi-tenant inference serving system that rethinks GPU multiplexing through the lens of energy and power efficiency. EnerTune operates in two phases. In the *offline* phase, EnerTune performs energy-conscious profiling to efficiently build power–performance profiles for individual models across hundreds of deployment configurations (Challenge 1). EnerTune profiles only a small subset of configurations and uses our proposed analytical formulations to estimate each model’s power draw, latency, and throughput across the full configuration space. EnerTune then uses these profiles to accurately estimate the power draw of GPUs serving colocated models (Challenge 2) by analytically synthesizing individual model profiles into GPU power estimates under concurrent execution. With these estimates, EnerTune uses our energy-aware bin-packing algorithm to jointly determine model placement and deployment configurations that minimize energy and cost. Finally, EnerTune dynamically adjusts deployment configurations at runtime as workloads fluctuate (Challenge 3), maintaining energy efficiency while meeting performance SLOs.

We summarize our novelty and contributions as follows:

- We develop analytical models that capture the impact of GPU allocation size, frequency, and batch size on per-model power. Using these models, our energy-conscious profiling reduces profiling time by 7.3× on average compared to prior systems while retaining high accuracy.
- We introduce a novel analytical model that accurately estimates the power draw of spatially multiplexed GPUs from individual model power profiles. Our methodology eliminates tens of hours of joint profiling overhead and enables EnerTune to reduce power and energy consumption by 1.3× over prior work.
- We develop an energy-aware bin-packing algorithm and build EnerTune on top of PyTorch. Compared to four state-of-the-art performance-driven baselines, EnerTune reduces energy and power draw by 1.4–2.3× and 1.3–2.6×, respectively, without violating SLOs, across load. We open-source EnerTune at <https://github.com/UT-SysML/EnerTune>.

2 Motivation and Challenges

2.1 Energy as a First-Order Metric

Prior works show many ML inference workloads underutilize modern GPUs [23, 27, 75, 117, 126, 141] and hence spatial multiplexing is required to improve utilization. However, most prior inference serving systems that multiplex ignore the implications of their decisions on energy consumption. Existing energy-efficient serving systems instead target LLMs that require dedicated GPUs [112, 116, 124, 163] or operate on older, small hardware (e.g., edge, small GPUs) [41, 60, 65, 131], rendering multiplexing unnecessary. A few workshop papers [120, 136] provide preliminary investigations into the impact of multiplexing on energy consumption. We

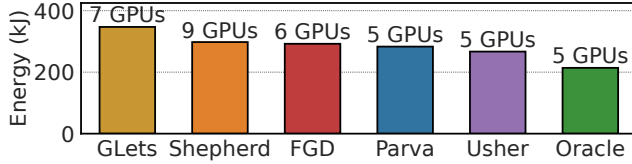


Figure 1. Energy-agnostic placement and allocations by prior systems increase energy by 1.25–1.62 $\times$ over an energy-optimal policy (Oracle), despite all systems meeting latency SLOs. Each system serves 11 models under a 3-minute window of the Azure trace [159]. See §2.1.

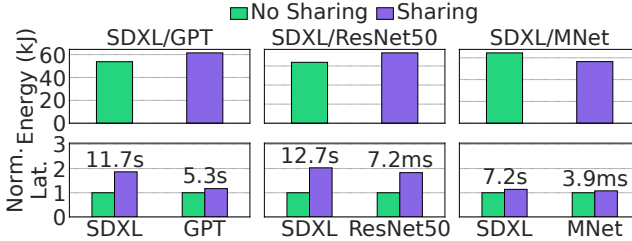


Figure 2. Energy consumption is sensitive to model allocation and placement decisions. Stable Diffusion XL (SDXL) appears in all cases, but different models colocated with it change the benefits of sharing due to latency inflation. Latency is normalized by no sharing. See §2.1.

provide further analysis in this section by studying the energy impact of four state-of-the-art inference serving systems that multiplex GPUs: GPULets [23], ParvaGPU [75], FGD [141], and Usher [117]. We compare them with Shepherd [159], which does not spatially multiplex GPUs, and an Oracle that minimizes cluster-wide energy while meeting latency SLOs. All systems serve identical workloads across 11 models (seven CNNs, two flavors of the image-generation model Stable Diffusion XL, Whisper speech-to-text, and GPT-J-6B LLM) at high load using the Azure Function trace [159]. We keep the trace short (fixed 3-minute window) to enable an exhaustive search over placement and allocation choices, yielding the energy-minimal, SLO-compliant Oracle that would otherwise be intractable to compute over longer traces. §7 provides further methodology details and evaluation with longer traces.

Figure 1 shows that all systems inflate energy consumption by 1.25–1.62 $\times$ over the Oracle. Shepherd [159] uses the most GPUs: it serves 11 models across 9 GPUs by sharing GPUs over time (time multiplexing) instead of spatially multiplexing them, leaving many GPUs underutilized and incurring higher static energy from the extra GPUs. The remaining systems spatially multiplex and use 5–7 GPUs, increasing utilization while reducing serving cost and static power draw. However, these systems make suboptimal placement decisions that, for certain model sets, increase energy consumption even compared to Shepherd, which does not multiplex GPUs (Figure 2). For example, FGD and GPULets place Stable Diffusion XL (SDXL throughout the paper) with

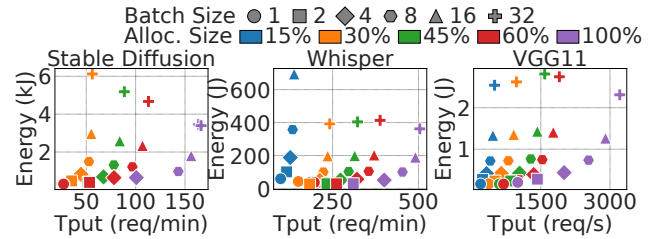


Figure 3. Throughput vs. energy for three models across GPU allocations (color) and batch sizes (shape). See §2.2.

GPT; Usher and ParvaGPU place SDXL with ResNet50. Figure 2 (left and middle plots) shows that these placement decisions increase latency ($> 1.7\times$) and thus energy consumption ($E = \int P dt$) for these model sets by 15% compared to Shepherd (no sharing). Meanwhile, the Oracle multiplexes SDXL with MobileNetV3, cutting energy consumption by 13% over Shepherd (no sharing, Figure 2 right), as it reduces total GPU power draw by 1.3 $\times$ while incurring only a 1.14 $\times$ latency increase.

We explain these findings by analyzing each model’s resource utilization. SDXL is compute-intensive (52.9% compute vs. 42.9% memory-pipeline utilization), while GPT-J and ResNet50 are more memory-intensive. However, memory utilization remains substantial across all three models (35.2–43.2%), making the shared memory subsystem the bottleneck when SDXL is colocated with GPT-J or ResNet50. The resulting contention increases latency enough to outweigh the power savings from sharing, consuming more energy than dedicating a GPU to each model. In contrast, MobileNetV3 uses $<15\%$ of both compute and memory resources, leaving ample headroom when colocated with SDXL; here, the power savings outweigh the modest latency increase. Thus, *intelligent* placement reduces both serving cost (only 5 GPUs used) and cluster-wide energy (Figure 1).

Observation #1: Spatial multiplexing guided solely by GPU utilization can increase energy consumption; instead, it must be guided by placement decisions that treat energy as a first-order objective. This enables multiplexing to reduce energy while improving utilization and lowering serving cost.

2.2 Joint Allocation and Batch Size Decisions

When spatially multiplexing, the system must decide the GPU allocation (i.e., fraction of GPU resources) and batch size per model. Prior systems typically allocate resources before choosing batch size, creating smaller allocations with larger batches. However, we observe many cases where slightly larger allocations with smaller batches reduce energy while providing the same throughput. For example, the Oracle in Figure 1 assigns Whisper a 45% allocation with batch 2 for 4 requests per second (RPS), reducing energy by 1.71 $\times$ over Usher and GPULets, which allocate 30% with batch 4. Although the Oracle’s larger allocation increases power by 1.27 $\times$, the 2.18 $\times$ latency reduction yields lower net energy consumption. Figure 3, presenting throughput versus energy

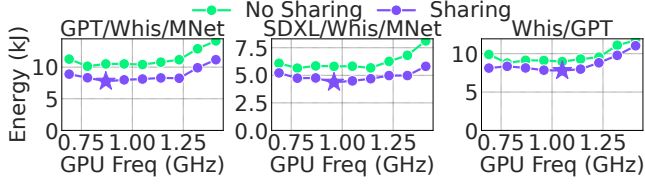


Figure 4. Tuning the frequency of shared GPUs to the model set can further reduce energy consumption. See §2.3.

under unique allocation/batch size combinations, shows similar trends across models (Appendix B shows data for all models): for a given throughput (e.g., SDXL at 52 req/min), the points in the figure that are most energy-efficient tend to be smaller batches (circles, squares, diamonds) with slightly larger allocations. Our measurements of isolated models thus suggest that selecting batch size first and then allocating resources is energy-efficient. However, this decoupled approach reduces packing density—larger allocations are required to sustain arrival rates—increasing GPUs used and cluster-wide energy (GPUlets and Shepherd’s larger allocations use 2–4 more GPUs in Figure 1, inflating energy by 1.4–1.7×). Instead, to minimize cluster-wide energy, we must jointly reason over allocations and batch sizes across models. **Observation #2:** For cluster-wide energy efficiency, GPU allocations and batch sizes must be jointly determined across models to balance power, latency, and the number of GPUs.

2.3 Tuning the Frequency of Shared GPUs

Prior works show that tuning GPU frequency can reduce energy consumption [112, 124]. Lowering frequency reduces power draw but increases latency. Hence, in a shared GPU setting, which also inflates latency relative to isolated execution, frequency tuning risks SLO violations and higher energy consumption. However, Figure 4 shows that reducing the frequency of shared GPUs can still reduce energy while meeting SLOs (latency data omitted for brevity). For example, reducing the frequency of GPT/Whis/MNet multiplexed (purple line) from 1.41 GHz (1.41 GHz is the maximum compute frequency A100s support) to 0.87 GHz reduces energy consumption by 1.43×. When SLOs have slack, the increase in latency can be tolerated, allowing the power reduction to dominate and lower overall energy consumption, as $E = \int P dt$. We observe similar trends across model sets, although the optimal frequency depends on the model combination (e.g., 0.96 GHz for SDXL/Whis/MNet). Thus, frequency must be tuned to the model set, balancing power reduction against latency inflation to achieve energy efficiency.

Observation #3: Reducing GPU frequency under spatial multiplexing can lower energy consumption when SLO slack exists, but requires per-model-set frequency tuning to balance power reduction against latency inflation.

2.4 Design Challenges

Our observations show that reducing energy in multi-tenant inference serving requires *jointly* reasoning about placement,

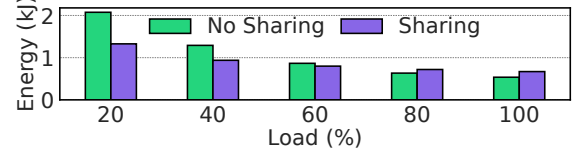


Figure 5. Fluctuations in runtime load require readjustment of deployment configurations to optimize energy consumption. See §2.4 (C3).

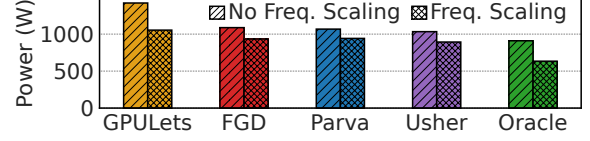


Figure 6. Prior frequency-agnostic placement policies colocate models with varying frequency needs, limiting power/energy savings from frequency scaling. See §2.4 (C4).

configuration, and frequency decisions while balancing GPU power draw against inference latency. Realizing such policies requires addressing four key challenges, discussed below:

C1. Accurate and energy-conscious profiling. The per-model deployment configuration space is large—over 450 configurations across > 15 GPU frequencies (§6 specifies the frequency levels we use), 5–10 allocation sizes, 6 batch sizes (≤ 32)—with wide variation in latency, throughput, and GPU power (§1). Understanding this space is essential: optimal configurations depend on latency SLOs, where looser SLOs expand the feasible region and enable smaller batches, lower frequencies, and smaller allocations that improve energy efficiency. Moreover, many configurations achieve similar throughput and satisfy SLOs but differ significantly in energy consumption (§2.2), requiring configuration choices to adapt across models. Prior multiplexing systems explore a small subset of this space (e.g., allocation size) using brute-force profiling [20, 23, 75], sometimes with kernel-level profiling [23, 50, 117, 126]. However, brute-force profiling, requiring 6 GPU-hours on average and substantial energy per model, becomes prohibitive at scale. Thus, *we need accurate, low-overhead methods to capture power–performance behavior across the deployment space.*

C2. Scalable power modeling for shared GPUs. Placement and allocation decisions require estimating the power draw of colocated models, which directly impacts energy (Appendix A). This is non-trivial: simply summing each model’s isolated dynamic power, proposed in prior work [61], overestimates (§7.3), as it ignores overlapping activity of auxiliary GPU components and GPU power-capping strategies. Meanwhile, accurate prior methods rely on costly low-level hardware profiling [2, 6, 8, 39, 70, 76, 83, 94, 121, 138], which is prohibitive in shared GPU settings, as each model combination must be profiled to obtain input data needed for the respective modeling strategy. Thus, *we need accurate yet time-efficient methods to estimate the power draw of spatially multiplexed GPUs.*

C3. Accounting for runtime workload fluctuations.

Making initial scheduling decisions is already challenging due to the combinatorially large space of placement and deployment configuration options. Runtime fluctuations further complicate decisions by altering the benefits of spatial multiplexing. For example, Figure 5 shows that at low load, sharing reduces energy by 1.08–1.56×: sharing increases completion time by only 5% while reducing GPU power by 1.54×, avoiding an extra GPU incurring static power overheads. At high load, however, sharing increases energy by 1.14–1.25×: although isolated execution incurs the static power cost of an extra GPU, it reduces completion time by 1.33×, reducing energy consumption ($E = \int P dt$).

C4. Architectural constraints limit the benefits of frequency scaling. Realizing the benefits of frequency scaling shared GPUs (§2.3) is non-trivial due to a fundamental constraint imposed by GPU architectures: frequency is a device-level setting that all multiplexed models share. This inflates energy consumption when models with significantly different performance requirements and hence frequency requirements are multiplexed on a GPU. To demonstrate this, we run an oracular frequency scaling policy (built using GEEPAFS [163], which transparently scales frequency without violating SLOs) on top of four frequency-agnostic placement policies (GPUlets, FGD, Usher, ParvaGPU). For each model set, the scaling policy determines the minimum frequency that does not violate SLOs. Figure 6 compares the power draw between these four placement policies and an optimal, frequency-aware placement policy when serving 11 models at high load. This analysis quantifies the headroom for jointly optimizing cluster and GPU-level decisions. While frequency scaling reduces each system’s power, the Oracle reduces power by 1.4–1.6×, as the prior systems colocate models with unaligned frequency needs. This forces models to run at higher frequencies dictated by the most demanding model, even when others could meet SLOs with lower frequencies. *Thus, placement and frequency decisions must be jointly optimized, as decoupling them and running frequency scaling tools [163] on top of frequency-agnostic placement policies sacrifices power/energy-efficiency.*

3 EnerTune’s Design

Motivated by our observations and challenges, we present EnerTune, a multi-tenant inference serving system that re-thinks GPU multiplexing through the lens of energy and power-efficiency.

Workflow. Figure 7 shows EnerTune’s architecture. Offline, the Config Characterizer builds accurate power-performance profiles per model by sampling a small set of deployment configurations and extrapolating to the full configuration space. In the online phase, EnerTune uses these profiles to construct energy-efficient deployment plans. Given a set of models, the Energy and Power-Aware (EPA) Resource Planner jointly determines model placement and deployment

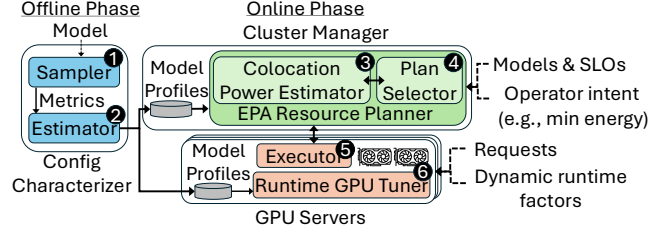


Figure 7. EnerTune’s system architecture. See §3.

configurations (GPU allocation, batch size, and frequency). Then, on each GPU server, the Executor enacts the plan, while the Runtime GPU Tuner adapts frequency and per-model batch sizes to workload fluctuations. We describe EnerTune’s key components next.

Configuration Characterizer. The Config Characterizer builds power-performance profiles per model via two components. The Sampler is energy-conscious and measures the GPU power, inference latency, and throughput over only a small subset (15%) of deployment configurations (addressing C1, §2.4). The Estimator then estimates these measurements for the remaining configurations using our polynomial analytical models. We detail the Config Characterizer in §4.1.

Energy and Power-Aware (EPA) Resource Planner. The EPA Resource Planner is a cluster-level controller that determines model placement and deployment configurations for a workload. It uses the Colocation Power Estimator to synthesize individual model profiles into accurate estimates of the power draw of multiplexed GPUs (addressing C2, §2.4). Using these estimates, the Plan Selector applies an energy-aware multi-dimensional bin-packing algorithm to efficiently derive execution plans (addressing C4, §2.4). §4.2 details the Colocation Power Estimator; §5.1 details the Plan Selector.

Runtime GPU Tuner. The Runtime GPU Tuner is a device-level controller that dynamically adapts GPU frequency and per-model batch sizes to fluctuating workloads without interrupting execution, meeting SLOs while optimizing for energy (addressing C3, §2.4). The Tuner is applicable only within certain workload fluctuations. Larger changes, driven by significant shifts in traffic to models, may require replanning via our Resource Planner. §5.2 details the GPU Tuner.

EnerTune’s general applicability. While §2 analyzes energy consumption to motivate our work, EnerTune can optimize for the cluster operator’s intent to minimize power, energy, or carbon emissions while meeting latency SLOs. We show efficiency improvements for both power and energy in this paper, with a discussion about extending to carbon emissions in §9.

EnerTune’s GPU multiplexing mechanism. We predominantly use NVIDIA’s Multi-Instance GPUs (MIG) to spatially multiplex models, ensuring strong isolation and fault tolerance for safe/reliable multiplexing. MIG is widely adopted across inference serving systems [71, 75, 77, 127, 139, 141, 157, 158, 161] and continues to be featured in the latest

NVIDIA hardware [103, 104]. However, we also show compatibility with Multi Process Service (MPS), NVIDIA’s other multiplexing mechanism, in §7.4.

4 Estimating Power Draw

GPU power is the fundamental metric underlying all downstream metrics (energy, carbon emissions), making accurate power estimation central to EnerTune. However, exhaustively profiling all models, deployment configurations, and colocated workloads to measure power is prohibitively time- and energy-intensive (§7.3). Instead, we devise analytical models to estimate per-model power across configurations (§4.1) and the power draw of colocated models (§4.2). We report average power estimates here; maximum power results are in Appendix D.2, along with latency and throughput formulations (§C).

4.1 Estimating Per-Model Power Draw

Traditionally, total GPU power P_T is broken down as:

$$P_T = P_S + P_D \quad (1)$$

where P_S is the workload-agnostic static power and P_D is workload-driven dynamic power (i.e., from models serving requests). P_D (and latency/throughput) depends on a model’s deployment configuration (GPU allocation size, frequency, batch size). With 450+ configurations per model, exhaustive profiling is infeasible. Instead, we profile a small subset of configurations to measure P_D and use an analytical model to estimate the model’s power for the remaining configurations. **Impact of configuration knobs on GPU power.** To guide our analytical formulation, we study how GPU allocation size, frequency, and batch size each impacts a model’s dynamic power draw P_D . Prior work analyzes their isolated impact on performance [23, 28, 33, 47, 75, 112, 115, 117], but not their joint impact on GPU power. Understanding these effects enables us to build an analytical model that estimates power across deployment configurations.

Across models, frequency exhibits a near cubic relationship with power (Figure 8). This is consistent with CMOS theory: dynamic power $P_D \propto V^2 f$, where V and f are GPU voltage and frequency, respectively, and voltage scales roughly linearly with frequency [70], yielding $P_D \propto f^3$. Batch size increases power sublinearly. Larger batches eventually saturate compute or memory pipelines; beyond this point, additional batching yields little extra work per cycle, producing a flattened, logarithmic-like trend. Finally, GPU allocation effects are model-specific: some models show little change (MobileNetV3) as they are small and do not require additional GPU resources, while wider and deeper models (VGG19, SDXL) exploit larger allocations, increasing power draw.

Analytical model. Using our findings from Figure 8, we devise an analytical formulation to estimate P_D . To estimate P_D across configurations, our model must capture three properties that impact power draw (Figure 8): (a) cubic dependence

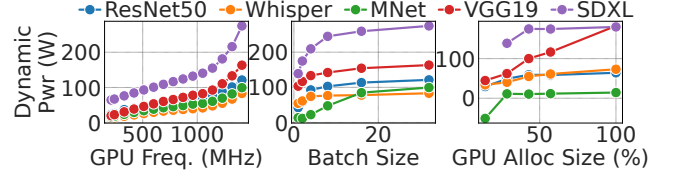


Figure 8. Impact of each deployment configuration knob on GPU dynamic power draw. See §4.1.

on GPU frequency, (b) sublinear impact of batch size, and (c) model-specific effects of GPU allocation and the interactions among these knobs.

To capture these requirements, we model dynamic power with a third-degree multivariate polynomial over frequency, batch size, and allocation. This formulation directly encodes the cubic dependence on GPU frequency. Moreover, it fits the sublinear, logarithmic-shaped impact of batch size well over $b \in \{1, 32\}$. Finally, cross-terms in the polynomial expansion capture interactions between configuration knobs and model-specific effects of allocation size. We thus estimate P_D as:

$$P_D(f, b, a) = \sum_{i=1}^3 (\beta_{i,f} f^i + \beta_{i,b} b^i + \beta_{i,a} a^i) + \sum_{\substack{i+j+k \leq 3 \\ \text{at least two of } i,j,k > 0}} \beta_{ijk} f^i b^j a^k \quad (2)$$

where $f, b, a > 0$ denote frequency, batch size, and allocation. The first term captures the individual nonlinear effects of each knob in isolation (knobs are not cross-multiplied in the first term). Index i denotes the polynomial degree, $i \in \{1, 2, 3\}$, capturing linear, quadratic, and cubic effects of each knob. Each coefficient $\beta_{i,f}$, $\beta_{i,b}$, and $\beta_{i,a}$ modulates model-specific sensitivity to each knob at degree i . The final term captures pairwise and three-way interactions between knobs (e.g., fb , f^2b , fba). Indices i, j, k denote the degrees of frequency, batch size, and allocation size in each interaction monomial, with $i + j + k \leq 3$ to maintain a cubic model and at least two of i, j, k nonzero such that the second term does not double-count the individual impact of knobs already captured by the first term.

Since the impact of configuration knobs is model-specific, we fit β coefficients per model by profiling a subset of configurations and applying Ordinary Least Squares (OLS) [93]. Rather than randomly select configurations, we use Latin Hypercube Sampling, as it provides better stratification and lower variance than random sampling [87]; other robust sampling methods (e.g., Bayesian optimization) are also applicable. We find that profiling only 15% of configurations accurately estimates power draw for the remaining 85% (§7.3).

4.2 Estimating Power Draw of Shared GPUs

While Equation 2 estimates an individual model’s dynamic power draw, placement decisions require understanding the

power impact of colocated models. We find that prior solutions that aggregate the individual dynamic power draw of each model [61] overestimate the power draw of shared GPUs (§7.3). Overestimating shared GPU power prohibits EnerTune from maximizing energy savings, since it leads to conservative packing decisions (i.e., models are packed less densely, increasing power draw §7.2). We thus develop an analytical model grounded in a deep understanding of the power delivery across GPU hardware components to synthesize individual per-model power profiles into accurate shared GPU power estimates.

Analytical model. We formulate our model by understanding how GPUs dynamically draw power P_D . Modern GPUs power-gate their compute units [70]. Hence, when profiling models in isolation under fractional allocations, unused compute units generally do not draw power. Instead, the gated power draw of GPU compute depends on the model and its compute usage given a GPU allocation. Other GPU components are not power-gated and remain active regardless of the allocation size. Some of these components (e.g., L2 cache, memory controllers) draw power in two parts: the idle power draw, incurred whenever any model runs on the GPU, and active power draw, which depends on the model’s usage of these shared components [70]. Other components of the GPU (e.g., board fan, supporting auxiliary circuitry) draw power whenever any process executes, generally independent of which model is running [70].

This breakdown highlights why adding the individually profiled P_D of colocated models overestimates shared GPU power draw: it counts the power draw of model-agnostic terms once per model rather than once per GPU. Hence, to accurately estimate shared GPU power draw, we formally decompose dynamic power as the power draw of the components that depend on the model P_M (gated compute units, active shared components) and the model-agnostic contributions P_O (idle shared-component draw, fan/auxiliary circuitry draw) incurred once per GPU. Formally,

$$P_D = P_M + P_O. \quad (3)$$

Accurately estimating shared GPU power requires counting model-agnostic components once while preserving dynamic model-specific contributions. Thus, we model total GPU power draw P_T when concurrently serving N models $M_1, \dots, M_N$ as:

$$P_T(M_1, \dots, M_N) = P_S + P_O + \sum_{i=1}^N P_M(i) \quad (4)$$

where P_S is the model-agnostic static power draw (Equation 1). Modern GPUs do not expose telemetry that isolates P_M from P_O , so neither can be measured directly. Instead, we can derive P_M per model using Equation 3 and the per-model dynamic power estimation P_D from Equation 2 in §4.1: $P_M = P_D - P_O$. With knowledge of P_M per model and the model-agnostic term, we can estimate the net power

draw of shared GPUs. We describe how we obtain the two model-agnostic terms P_S and P_O next.

Obtaining P_S . P_S is frequency dependent [46, 70]. Obtaining P_S is simple: at each frequency, we measure GPU power when no processes execute. This is a one-time cost per GPU type.

Obtaining P_O . P_O is the model-agnostic power draw of shared GPU components. The exact composition of P_O depends on the GPU type, as different GPU types can have different components, each with varying power draw. For example, modern GPUs include L2 caches and memory controllers, but can differ substantially in cooling architecture (e.g., A100s are passively cooled by chassis-level fans external to the GPU module, while A6000s have onboard fans powered via the same power rail as the GPU). Consequently, P_O is not a fixed physical quantity that transfers across GPU types; instead, its value and which components contribute to it differ by GPU architecture.

Because of these differences in GPU composition, we derive P_O empirically per GPU type, rather than formally decomposing power draw per component in the architecture (we leave that breakdown to future work as instrumentation for directly measuring each component improves). Specifically, we rely on a high-level profiling methodology to estimate P_O . At each frequency, we deploy one and two GEMM instances, each given a 15% allocation, and measure GPU power. By Equation 4, each measurement is $P_T(n) = P_S + P_O + n \cdot p$ for $n \in \{1, 2\}$ concurrent instances, where p is a single GEMM instance’s model-specific power draw. With P_S already measured, there remain two unknowns, p and P_O . By solving the 2×2 equations with $n = 1$ and $n = 2$ measurements, we obtain P_O ’s value.

Putting it all together. With knowledge of P_S , P_O , and each model’s P_M , we can use Equation 4 to accurately estimate the GPU power draw of colocated models.

5 Deployment and Runtime Decisions

Leveraging our power estimation models (§4), EnerTune jointly reasons about model placement, deployment configurations, and GPU frequency to make energy-aware deployment and runtime decisions. We next describe how EnerTune’s EPA Resource Planner computes initial placement and configuration decisions for models with expected load (which are obtainable via historical request rates). The Runtime GPU Tuner adjusts configurations as runtime load deviates from the expected load to maintain latency SLOs while maximizing energy efficiency.

5.1 Generating Energy-Aware Execution Plans

Given models with latency SLOs and expected request arrival rates, EnerTune’s EPA Resource Planner generates execution plans, specifying deployment configuration and placement decisions, to minimize “cost”: energy or power. Naive brute-force is infeasible: with K GPUs, N models, and $B \times A \times F$ deployment configurations (B batch, A allocation size, and F

Algorithm 1: Generating execution plans (§5.1).

```
1 Function createExecutionPlans(freqBins, costMetric):
2   bins  $\leftarrow \emptyset$ 
3   foreach freq  $\in$  freqBins do
4     foreach model  $\in$  freqBins[freq] do
5       bin  $\leftarrow$  GETCOMPATIBLEBIN(model, bins[freq])
6       if bin not None then
7         bin.models  $\leftarrow$  bin.models  $\cup$  {model}
8       else bins.add(NEWBIN(model, freq))
9   bins  $\leftarrow$  SortByFreqDescending(bins)
10  foreach bin  $\in$  bins and bin.util  $<$  100% do
11    foreach lfb  $\in$  LowerFreqBins(bin.freq) do
12      if lfb.util  $<$  100% then bins  $\leftarrow$  TRYMERGE(bin, lfb)
13  bins  $\leftarrow$  SortByCostSpreadAscending(bins, costMetric)
14  foreach bin  $\in$  bins and bin.util  $<$  100% do
15    foreach hfb  $\in$  HighFreqBins(bin.freq) do
16      if hfb.util  $<$  100% then bins  $\leftarrow$  TRYMERGE(bin, hfb)
17  return bins
```

frequency options) per model, the search space is exponential $O((BAFK)^N)$. Mixed-integer linear program formulations are also infeasible, as multivariate bin-packing is NP-hard and MILP solvers are slow at scale [23, 80, 112, 117, 124]. Prior work for cloud-native workloads [15, 30, 31, 72, 73, 79, 118, 119, 130], ML training [24, 45, 57, 73, 92, 97, 147], and inference serving [23, 73, 75, 77, 80, 112, 114, 117, 141, 142, 148, 158, 159, 161, 168] propose time-efficient bin-packing algorithms, but optimize utilization, performance, or cost rather than energy or power. Simply augmenting existing algorithms with our colocation power model (Equation 4) is insufficient: maximizing gains requires frequency-aware placement decisions (C4, §2.4). Thus, we design a new placement algorithm whose bin-packing decisions are driven by energy and frequency (Algorithm 1). Our algorithm consists of two main steps: (1) initial frequency-aware model binning, and then (2) cost and fragmentation reduction.

Step 1: Initial frequency-aware model binning. To reduce the complexity of our problem, we prune the search space via frequency-aware binning (lines 3–8). We initially group models with the same frequency requirements into a bin, since frequency is a device-level setting shared among colocated models (C4, §2.4). For each model, we select the lowest-cost configuration that meets its latency SLO, using per-model profiles generated by the Config Characterizer (§4.1). We then group models by frequency and place them in the same “bin” if they are *compatible* (line 5). We consider models *compatible* if they fit on a GPU and multiplexing them reduces cost compared to isolated execution, determined with Equation 4 and $E = \int P dt$.

Step 2: Cost and fragmentation reduction. Naively deploying the initial execution plan, constructed from per-model optimal configurations, may be globally suboptimal: models with differing frequency needs create underutilized

bins, increasing both GPU fragmentation and cost from the static power draw of extra GPUs. Thus, in this step, we reduce fragmentation and cost by merging bins across frequency as long as we continue meeting SLOs. We first merge higher-frequency bins into lower-frequency ones, since reducing frequency generally reduces power and energy (Figure 8). We sort bins by frequency (line 9), traverse them in descending order (lines 10–11), and evaluate merging opportunities in each iteration (line 12). When merging bins, we consider per-model allocation and batch size combinations to identify configurations that jointly fit on a GPU, meet SLOs, and minimize cost. As before, we only merge bins if *compatible*.

To further reduce cost and fragmentation, we make an additional pass that merges lower-frequency bins into higher-frequency bins. This step reduces cost and fragmentation, as increasing frequency preserves SLOs (latency decreases monotonically [112]), enabling more merging opportunities. However, naively iterating from the lowest-frequency bin can inflate cluster cost, as moving them to higher frequencies can increase power and energy. Instead, we sort bins by their cost spread (the cost difference between a bin’s current frequency and the highest-frequency, line 13) and traverse them in ascending order (lines 14–15). This prioritizes bins whose cost increases the least when moved to higher frequencies, reducing overall cluster cost. For each bin, we greedily attempt merges from the highest-frequency bin downward. As before, merges occur only if bins are *compatible*.

Satisfying cluster size constraints. After an iteration, the plan may exceed the GPU budget. Therefore, we must reduce bins even when no cost-compatible merges remain. We merge the two lowest-frequency bins, minimizing cost increase, and rerun Step 2 (lines 9–16). This reduces GPU count by one per iteration with minimal cost increase.

Efficiency of our algorithm. We reduce time complexity from exponential to polynomial: $O(N^3(BA)^4)$. The dominant cost arises in lines 12/16, which enumerate batch/allocation combinations for up to four models, yielding $(BA)^4$. This packing procedure runs up to N^2 times across nested loops (lines 10–11, 14–15). Finally, the two-pass merging may repeat up to $N - K \approx N$ times to meet GPU constraints.

5.2 Adapting to Workload Fluctuations

EnerTune’s EPA Resource Planner makes placement and configuration decisions given an expected load. However, workloads can deviate from expectation during runtime, potentially requiring adjustment to GPU frequency, batch size, or even model allocation/placement. To minimize disruption, we prioritize maintaining model placement/allocation decisions and instead dynamically tune GPU frequency and batch sizes in response to fluctuating workloads.

Runtime GPU-level tuning. The Runtime GPU Tuner monitors each model’s load λ and, upon $> \alpha\%$ or $< \beta\%$ deviations, adjusts batch size and/or GPU compute frequency with no downtime to meet demand while minimizing energy

Algorithm 2: Frequency tuning and batching (§5.2).

```
1 Function tuneDevice(gpu):  
2    $inc \leftarrow (\exists m \in gpu.models : \lambda_m > (1 + \alpha)\hat{\lambda}_m)$   
3    $dec \leftarrow (\exists m \in gpu.models : \lambda_m < (1 - \beta)\hat{\lambda}_m)$   
4    $C_{best} \leftarrow \infty, best_{cfgs} \leftarrow \emptyset$   
5   foreach  $f \in getFreqSpace(gpu.freq, inc \text{ or } dec)$  do  
6     if  $ALLSLOSMET(gpu.models, f)$  then  
7        $batches \leftarrow CHOOSEBATCHES(gpu, f)$   
8        $C \leftarrow COST(f, batches, gpu.models)$   
9       if  $C < C_{best}$  then  
10       $C_{best} \leftarrow C \ \& \ best_{cfgs} \leftarrow (f, cfgs)$ 
```

or power. EnerTune directly pins GPU compute frequency, bypassing GPU dynamic voltage frequency scaling (DVFS); thus, changing batch size does not impact compute frequency. We do not control GPU memory frequency, leaving memory-frequency tuning as future work. Algorithm 2 details this process. As load changes, the Tuner explores increasing or decreasing frequency (lines 2–3). Using our shared GPU power draw estimations (Equation 4) and individual model latency and throughput estimations (Equations 8–9), the Tuner analytically evaluates feasible frequency/batch-size combinations within the fixed allocation size. We only consider configurations that are realizable (e.g., sufficient GPU memory) and that satisfy SLOs, before selecting the minimum-cost configuration (lines 5–10). If no feasible local configuration exists, the Tuner escalates to the EPA Resource Planner. We use a standard grid search with three models colocated to tune $\alpha = 5\%$ and $\beta = 10\%$ (§7.4 presents a sensitivity study). **Escalation triggers.** Although we prioritize preserving existing placement and allocation decisions, large load shifts can render them suboptimal, insufficient (e.g., SLOs are violated despite local frequency/batch-size tuning), or infeasible (e.g., insufficient memory for the required batch size). Hence, following prior work [23, 80, 112, 117], we escalate to the EPA Resource Planner if (1) cluster load deviates $> \gamma\%$ or $< \theta\%$, or (2) local frequency/batch-size adjustments are insufficient or infeasible. We tune $\gamma = 20\%$ and $\theta = 40\%$ using a standard grid search (§7.4 presents a sensitivity study).

6 Implementation

We build EnerTune in Python (1426 LoC), C++ (209 LoC), and Bash (894 LoC). We use NVIDIA MIG [105], a widely adopted mechanism [71, 75, 77, 127, 139, 141, 157, 158, 161] that provides strong isolation and fault tolerance, for safe/reliable multiplexing. We show compatibility with MPS [101] in §7.4.

EnerTune comprises two entities: a Cluster Manager (logically centralized controller), hosting our EPA Resource Planner, and GPU servers, hosting the Executor and Runtime GPU Tuner. The Config Characterizer is lightweight and can run anywhere, simply requiring one GPU for quick profiling. Components communicate via gRPC [43].

Config Characterizer. We use SciPy’s Latin Hypercube Sampling [129] to select configurations, profiling them for

5s with closed-loop request arrivals for stable measurements. We sample power every 100 ms via NVIDIA Management Library [102]. The Estimator produces power-performance profiles (10.8 KB), stored in an in-memory SQLite store [55].

EPA Resource Planner. The EPA Resource Planner is a CPU process that uses model profiles to generate execution plans and sends them via gRPC to each GPU server’s Executor.

Executor. The Executor maintains a thread per GPU. It deploys a process per model, using vLLM [74] for all LLMs and TensorRT [107] for other models (CNNs, Whisper, SDXL). Model weights are downloaded from HuggingFace [58].

Runtime GPU Tuner. The Tuner, a CPU process with one thread per GPU, monitors per-model request queues to track load. To minimize frequency-switching overheads, the Tuner runs as a persistent privileged process that keeps the clock-management utility (NVIDIA System Management Interface [106]) in memory, eliminating repeated launch and privilege escalation overheads. The Tuner selects among 15 streaming multiprocessor (SM) frequencies spanning the GPU’s range (A100 210–1410 MHz in ~ 90 MHz increments). Although GPUs support finer-grained frequencies, they increase configuration and profiling complexity without significant energy benefits. However, our modular design enables EnerTune to support finer-grained frequencies if desired.

7 Evaluation

We evaluate EnerTune against five state-of-the-art baselines using real-world traces. Our key findings are:

1. Across baselines and load, EnerTune reduces power by 1.3–2.6 $\times$ and energy by 1.4–2.3 $\times$ while meeting latency SLOs (§7.1).
2. Each of EnerTune’s components is critical for improving energy and power-efficiency of inference (§7.2).
3. EnerTune reduces profiling time by 1.32–17.3 $\times$ and maintains high power estimation accuracy: 4–7% error (§7.3).
4. EnerTune is robust to varying arrival patterns and SLOs, generalizes to other GPU types and LLM-only workloads, and scales as the number of models to deploy grows (§7.4).
5. EnerTune introduces no additional inference overhead, matching the minimal overheads of prior systems (§7.6).

Models. We use 22 models (two instances of 11 models) spanning domains, architectures, and sizes (5.5M–6B parameters) used in recent work [27, 117, 126, 161]: seven CNNs for image classification (VGG11/19, MobileNetV3-Large, ResNet50/152, DenseNet161/169), two SDXL variants generating 1024x1024 and 256x256 images, Whisper speech-to-text, and GPT-J-6B text generation. These reflect the smaller CNNs/NLPs/LLMs served by hyperscalers (Alibaba [128, 140, 141], Meta [27]), with a few in MLPerf’s Datacenter Inference Benchmark v5 [91] (ResNet50, GPT-J-6B, SDXL, Whisper). Appendix D.1 further details models and datasets.

Workload. We follow the methodology of prior GPU multiplexing works [50, 117, 126, 161] to generate our workloads.

We use two Azure Function traces: MAF1 with dense, steady request arrival [159] and MAF2 with bursty arrival [162]. Each model’s 100% load is the RPS of the 22 functions with the highest average RPS in the traces, like prior work [47, 75, 117, 126, 159]. We scale the trace to 25–125% load relative to the peak capacity, yielding cluster arrival rates of 5K–25K RPS. We execute traces for ~1 hour; each system serves the same number of requests (identical workloads) for fair comparison of energy consumption.

Baselines. We compare with five state-of-the-art baselines.

- (1) *Usher* [117]: a serving system that aggressively packs models using MPS and selectively replicates them to further increase packing density and utilization.
- (2) *FGD* [141]: a serving system that greedily packs models to minimize cluster-wide fragmentation.
- (3) *GPUlets* [23]: a serving system that uses MPS to pack models while meeting SLOs. To mitigate interference and SLO violations, GPUlets packs at most two models per GPU.
- (4) *ParvaGPU* [75]: a serving system that applies MPS within MIG slices to tightly pack GPUs and improve utilization.
- (5) *Jacquet et al.* [61] (*Additive*): a state-of-the-art power estimator that sums per-model isolated dynamic power with GPU static power to estimate shared GPU power. We compare EnerTune’s estimation accuracy against it (§7.3).

Testbed. Our testbed has 16 NVIDIA A100 80 GB GPUs (4 GPUs/server), equipped with AMD EPYC 9554P (64 Core) CPUs. We also show EnerTune’s generalizability to GPU types with 8 A6000s and RTX 4090s in §7.4. We use Ubuntu 22.04 and NVIDIA Drivers 575.57.08 with CUDA 12.9.

Metrics. We compare the power and energy consumption across systems. For performance, following prior work [80, 112, 161], we report each system’s p99 latency normalized to per-model SLOs: values <1 meet the SLO with $\geq 99\%$ attainment, while >1 indicate violations. We follow industry methodologies [44, 112, 124] and set each model’s SLO (see Appendix D.1) as its p90 latency when run in isolation on a single GPU under Poisson arrivals; we evaluate tighter and looser SLOs in §7.4. Finally, we evaluate our analytical models with profiling time/energy and estimation accuracy.

7.1 E2E Power and Energy Gains

As cluster operators may have different intents (minimize power or energy), we evaluate EnerTune with each intent while meeting latency SLOs.

Power savings. Figure 9 (top) shows average cluster power draw (Appendix D.2 shows max power) and p99 latency of each system serving all models (MAF1 trace). Across load, EnerTune reduces power by 1.3–2.6 $\times$ versus the baselines while meeting SLOs (Appendix D.3 shows per-model latency). At low loads (25–50%), EnerTune opportunistically reduces frequency (240–690 MHz). Although this increases latency, EnerTune tunes frequency to meet SLOs and support load such that request queues do not build up (queuing delay

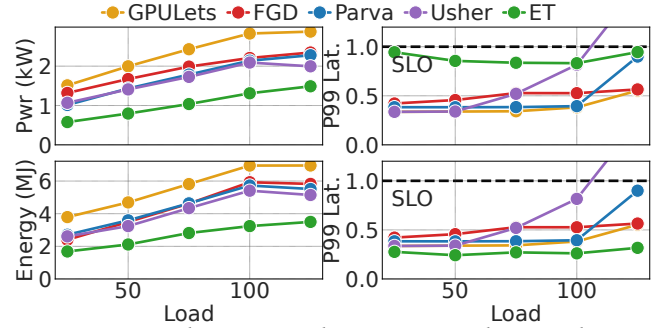


Figure 9. ET reduces power by 1.3–2.6 $\times$ and energy by 1.4–2.3 $\times$ while consistently meeting SLOs. See §7.1.

across models is 0.02–0.08 ms at low load). Moreover, with accurate latency and throughput estimates from the Config Characterizer, EnerTune reduces GPU allocation and batch size without violating SLOs: for example, EnerTune allocates ResNet152 30% of a GPU with batch size 2, while the baselines allocate 45% with batch size 4–16. This enables denser packing and power reduction: EnerTune uses only 8 GPUs while baselines use 9–12.

At higher loads ($\geq 75\%$), EnerTune increases frequency (780–1140 MHz) and readjusts deployment configurations to meet SLOs. For example, it increases ResNet152’s allocation to 60% and batch size 8 at 125% load. However, it continues to select configurations that minimize power given performance constraints. It also colocates models with similar frequency needs (e.g., DenseNet161 and Whisper at 780 MHz), enabling a shared frequency that avoids excess power draw at high frequencies and SLO violations at low frequencies.

Usher is next best in reducing power, as its aggressive packing reduces GPUs used. However, this packing violates SLOs at high load (125% load). GPUlets is conservative, placing only two performance-compatible models on a GPU. Hence, it consistently meets SLOs, but inflates power by using the most GPUs (12–16) across all loads. FGD and ParvaGPU adopt more balanced strategies, but ParvaGPU (and Usher) deploy multiple instances of the same model under small allocations to pack GPUs instead of one larger instance; this increases power due to replicated model weights occupying GPU memory and traversing the GPU memory hierarchy.

Energy savings. We next analyze EnerTune’s ability to adapt and minimize energy. Figure 9 (middle) shows that across loads, EnerTune reduces energy by 1.4–2.3 $\times$ while meeting latency SLOs. EnerTune makes different decisions when minimizing energy, as it balances GPU power and latency ($E = \int P dt$). For example, at 100% load, EnerTune uses 10 GPUs (versus 8 for power optimization) as it allocates slightly larger GPU shares with smaller batches: when minimizing power, it allocates ResNet152 a 30% GPU share with batch 16, whereas for energy, it allocates 45% with batch 8, reducing latency by 2.9 $\times$ and energy per request by 2 $\times$.

EnerTune also adapts placement and frequency when optimizing for energy. For example, we place SDXL with

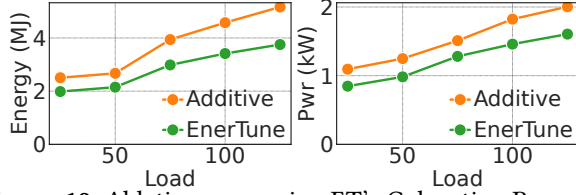


Figure 10. Ablation comparing ET’s Colocation Power Estimator with a state-of-the-art additive estimator [61]. ET’s accurate estimations reduce power and energy. See §7.2.

VGG19/DenseNet161 when optimizing power. This combination is energy-incompatible, inflating energy by 1.3× relative to isolated execution. Instead, we place SDXL with MobileNetV3 for energy compatibility (Figure 2, §2.1). We also reduce frequency conservatively to balance power reduction against latency inflation: it operates GPUs at 780–1050 MHz at low load, versus 240–690 MHz for power optimization.

Despite decreasing frequency, EnerTune reduces latency compared to the baselines. GPUlets slightly inflates latency due to interference; Usher’s aggressive replication and dense packing increases contention. FGD and Parva minimize allocation size and increase batch size to support load, which meets SLOs but slightly raises latency. EnerTune instead increases allocation size to reduce energy (e.g., EnerTune allocates DenseNet161 60% GPU share with batch 4, versus 30% with batch 8 from FGD and Parva). Moreover, reducing frequency from 1410 MHz to 780–1240 MHz has little impact on latency, which typically degrades sharply <600 MHz [112]. Thus, modestly larger allocations offset lower frequencies, enabling EnerTune to achieve lower latency and energy.

7.2 Performance Breakdown

We next break down EnerTune’s power and energy gains into the contributions of the Colocation Power Estimator, the EPA Resource Planner, and the Runtime GPU Tuner.

Ablation study: Colocation Power Estimator. Our Colocation Power Estimator estimates the power draw of shared GPUs (§4.2). We evaluate its accuracy in §7.3. Here, we show our accurate estimates are critical for reducing power and energy. We compare against an Additive estimator proposed in prior work [61], which sums each model’s isolated dynamic power with GPU static power. We keep all other system components unchanged. Figure 10 shows that accurate estimates enable EnerTune to reduce energy by 1.2–1.4× (1.3× avg) and power by 1.2–1.3× (1.3× avg). The Additive estimator greatly overestimates (§7.3). This causes the EPA Resource Planner to incorrectly deem many colocations incompatible since their estimated shared power/energy exceeds the cost of serving them on separate GPUs. Hence, the Additive estimator uses 2–3 more GPUs, increasing power and energy.

Ablation study: EPA Resource Planner. The Resource Planner generates execution plans specifying frequency and placement/allocation decisions. Here, we isolate gains from our placement/allocation decisions by disabling frequency

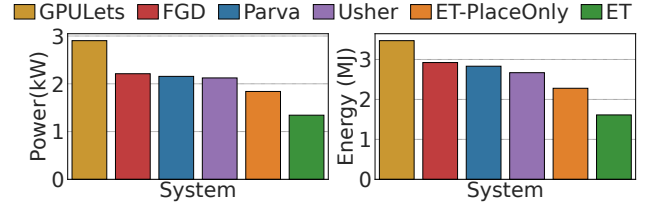


Figure 11. Ablation of ET’s EPA Resource Planner shows that its placement and allocation decisions alone (no frequency tuning) reduce power and energy. See §7.2.

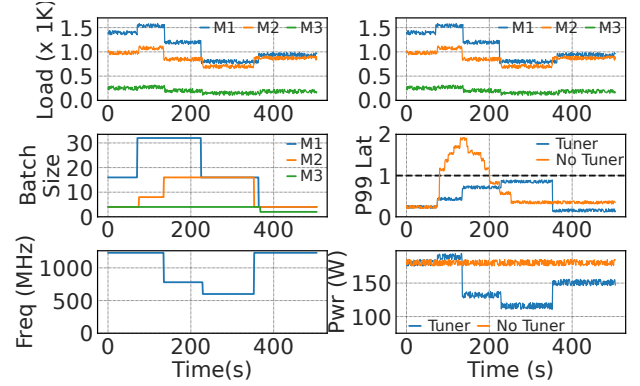


Figure 12. Ablation of ET’s Runtime GPU Tuners shows they are critical for meeting SLOs and reducing power by adjusting batch size and frequency under varying load (§7.2).

tuning (all GPUs run at max frequency). Figure 11 shows our placement decisions alone (ET-PlaceOnly) reduce power by 1.13–1.52× and energy by 1.17–1.53× over the baselines at high load (30-minute trace). This is because we reduce GPUs used (only 8, baselines ≥ 9) and ensure multiplexed models consume less power or energy than isolated execution. In contrast, baselines make power/energy-incompatible decisions (e.g., GPUlets places SDXL with GPT, §2.1). Jointly optimizing placement with frequency by colocating models with similar frequency requirements further reduces power by 1.42× and energy by 1.41× over ET-PlaceOnly.

Ablation study: Runtime GPU Tuner. We next evaluate the impact of EnerTune’s Runtime GPU Tuner, which dynamically adjusts frequency and batch size under fluctuating load. We randomly choose a 500-second window from the trace and analyze the changes the Tuner makes for a set of three models multiplexed on a GPU (Figure 12). At 70s when load for M1 (ResNet50) and M2 (MobileNetV3) slightly increases, we only increase batch size (not frequency) to minimize the increase in power draw while supporting the higher load. At 134s and 226s, noticeable load drops trigger a ~ 600 MHz frequency reduction, reducing power draw by 50–60W (bottom right). At 362s when load grows, we increase frequency while reducing batch size to meet demand efficiently. These dynamic adjustments by the Runtime GPU Tuner improve EnerTune’s SLO compliance (middle right) while reducing GPU draw by 30–60W (bottom right).

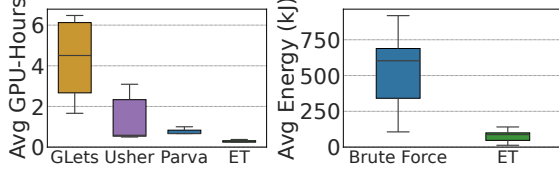


Figure 13. ET’s profiling cost, measured with GPU-hours and energy consumption. See §7.3.

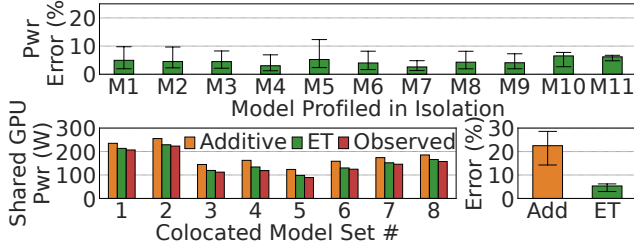


Figure 14. ET’s power estimation accuracy. See §7.3.

7.3 Efficacy of Profiling & Power Estimation

Profiling cost. As our goal is to improve energy efficiency, in addition to profiling time, we evaluate the energy consumed during profiling (Figure 13). We exclude FGD since it only makes placement decisions, not deployment configuration decisions. Despite a $15\times$ larger search space from including frequency as a configuration knob, EnerTune reduces profiling time by $1.32\text{--}17.3\times$ ($7.3\times$ avg) and profiling energy consumption by $6.52\text{--}8.29\times$ ($7.4\times$ avg) by measuring high-level metrics (GPU power draw, latency) for a small subset of configurations and then estimating the rest. ParvaGPU’s brute-force profiling increases profiling time by $2.67\times$ on average. Usher requires 2–3 hours to collect kernel-level traces for its latency/memory predictors, increasing profiling time by $2.32\times$ on average. GPUlets profiles kernel-level metrics per model (7 hours for models with thousands of kernels, e.g., GPT-J-6B), increasing profiling time by $17.3\times$ over EnerTune.

Estimation accuracy. Figure 14 (top) shows that across 11 diverse model types (CNNs, image generation, speech-to-text, LLMs), we accurately estimate the average power draw of individual models under 450 deployment configurations, with median absolute percentage error (MAPE) $<5\%$ and maximum error $<10\%$ (Appendix D.4 shows latency accuracy). Figure 14’s bottom row shows that for 8 representative model sets colocated on a GPU (2–4 models per GPU under high load), we accurately estimate the shared GPU power draw compared to recent prior work [61] that sums each model’s individual dynamic power profiled in isolation with static power. Across 150 model sets, we only have 6.3% MAPE. The additive estimator overestimates by double-counting shared components (§4.2). Instead, EnerTune decomposes power into model-dependent power draw of GPU components and model-agnostic contributions (shared auxiliary components, peripherals), enabling accurate synthesis of per-model estimates into shared GPU power.

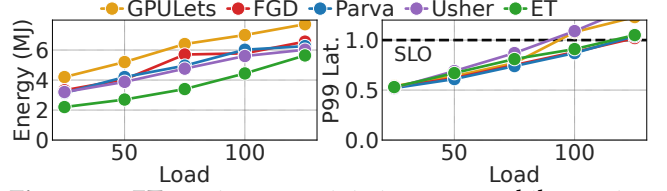


Figure 15. ET continues to minimize energy while meeting SLOs under bursty arrival (MAF2 trace [162]). See §7.4.

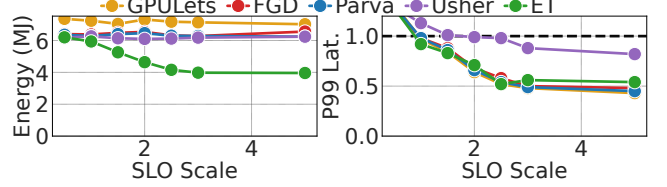


Figure 16. ET is robust to different SLOs. See §7.4.

Ablation study of analytical model components. We conduct a study to understand the impact of each model component on estimation accuracy. Specifically, in Equation 2, we observe the increase in estimation error as we remove each model component. Not accounting for batch size, frequency, or allocation size increases the average power estimation error of individual models from 5.2% to 24.3%, 42.9%, and 21.3%, respectively. We find similar accuracy degradation with Equation 4 when estimating the power draw of shared GPUs. Removing P_S or P_O increases average estimation error from 6.3% to 33%. Thus, the model components in Equations 2 and 4 are vital for high estimation accuracy.

7.4 Robustness, Generalizability, and Scalability

Robustness to arrival patterns. We evaluate EnerTune’s robustness to arrival patterns with MAF2 bursty trace [162]. For brevity, we report energy (similar trends hold for power). Figure 15 shows EnerTune reduces energy by $1.13\text{--}1.92\times$ while meeting SLOs across load. Unlike the MAF1 trace, at high load (125%) EnerTune only decreases energy by $1.17\times$ on average compared to the baselines. This is because the GPU Tuner increases batch size and GPU frequency to handle bursts and meet SLOs (minimum frequency 1050 MHz for MAF2 versus 780 MHz for MAF1).

Robustness to SLO scales. Figure 16 shows EnerTune is robust to SLOs (MAF2 trace, high load). With tighter SLOs, EnerTune increases frequency and allocation sizes to meet SLOs, thereby increasing energy consumption closer to the baselines. With loose SLOs, EnerTune is conservative in reducing frequency and allocation sizes to avoid significantly inflating latency, which would raise net energy consumption: EnerTune reduces energy consumption by $1.64\times$ on average under looser SLOs while maintaining low inference latency. When optimizing for power (Appendix D.8), under looser SLOs, EnerTune greatly reduces frequency to reduce power by $2.6\times$ on average without violating SLOs.

Sensitivity to load hyperparameters. EnerTune’s Runtime GPU Tuner adjusts frequency and batch size when a

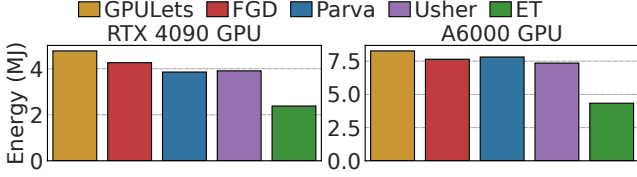


Figure 17. ET generalizes across GPU types. See §7.4.

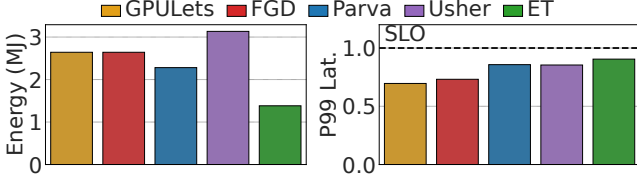


Figure 18. Evaluation of ET’s efficacy on LLM-only workloads. ET generalizes to LLMs, reducing energy consumption while maintaining SLO compliance. See §7.4.

model’s load increases by $\alpha\%$ or decreases by $\beta\%$ relative to its current load. Larger α values delay adjustments and increase SLO violation risk. In contrast, SLO violations are largely insensitive to β , since it only triggers adjustments when load decreases; β primarily governs power savings, with larger values making the Tuner more conservative. Via a grid search (5–50%, 5% increments) with 3 models deployed on a single GPU and load varying between 50–100%, we find $\alpha = 5\text{--}20\%$ maintains SLO compliance, while $\beta = 10\text{--}40\%$ maximizes power savings (see Appendix D.6 for data).

Generalizability to GPU types. We evaluate EnerTune’s generalizability to GPU types with eight A6000s and RTX 4090s. On each type, we deploy all models (MAF1 trace, high load) and use MPS for multiplexing (these GPUs do not support MIG), like prior work [23, 117]. EnerTune generalizes to GPU types (Figure 17): we reduce energy by 1.56–2.12 $\times$ with energy-aware placement and configuration decisions.

Generalizability to LLM workloads. Our E2E workload (§7.1) mixes LLMs with SDXL, Whisper, and CNNs. Here, we evaluate EnerTune on LLM-only workloads, deploying 2 $\times$ each of Llama-3.1-8B, Mistral-Nemo-12B, Qwen2.5-14B, InternLM2.5-20B, Gemma-2-27B, and Yi-1.5-34B with int4 quantization. We use vLLM, WikiText [88], 512 output tokens, and the MAF2 high-load trace. Figure 18 shows that EnerTune reduces energy by 1.65–2.27 $\times$ while meeting SLOs despite a slight increase in latency by 1.06–1.30 $\times$. Per-model and shared GPU power estimation errors also remain low at 4.63% and 9.7%, respectively. We assume each LLM and its maximum KV cache fit within its GPU allocation; we leave handling dynamically growing KV caches under multiplexed environments as future work.

Scalability to # of models. We evaluate EnerTune’s scalability as the number of models grows (10–1000 models) by analyzing the number of GPUs used and power draw of each system (Appendix D.7). EnerTune matches the next best baselines (Usher, ParvaGPU) in GPUs used (148–150 GPUs for 500 models, 290–294 for 1000 models). However,

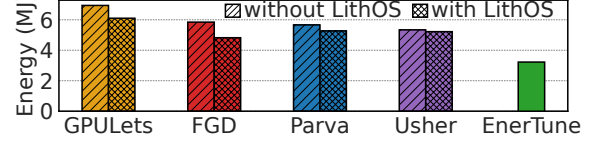


Figure 19. Energy consumption of cluster-level serving systems with and without LithOS, a device-level tuner adjusting GPU allocation and frequency. EnerTune jointly reasons about placement, allocation, batch size, and frequency decisions to maximize energy efficiency. See §7.5.

we reduce power draw by 2–2.7 $\times$. We note it is prohibitive for us to deploy 1000 models across 250+ GPUs; thus, we estimate the power draw of each system’s execution plan with EnerTune’s accurate Colocation Power Estimator.

7.5 Joint versus Decoupled Optimization

EnerTune is the first inference serving system that jointly optimizes model placement, GPU allocation, batch size, and GPU frequency. Prior systems only optimize the first three knobs without reasoning about GPU frequency. Meanwhile, a few device-level schedulers optimize allocation (LithOS [27], Orion [126], REEF [50]) and/or frequency (LithOS [27], GEEP-AFS [163]) for models already assigned to a GPU. Hence, a plausible alternative to EnerTune is to compose cluster- and device-level schedulers. We evaluate this design by combining each serving system (Usher, GPUlets, ParvaGPU, FGD) with LithOS [27]: the serving system determines model placement and batch size, after which LithOS optimizes GPU allocation and frequency to minimize energy while meeting SLOs. We compare EnerTune’s energy consumption against each system with and without LithOS using the MAF1 high-load trace. Figure 19 shows that despite combining serving systems with device-level optimizers, EnerTune still significantly reduces energy consumption by 1.5–1.89 $\times$. While LithOS reduces each serving system’s energy consumption, it is limited in its benefits since the serving systems make frequency-incompatible placements that force LithOS to run at higher frequencies dictated by the most demanding model. Instead, by optimizing placement, allocation, batch size, and frequency jointly, EnerTune maximizes energy efficiency while meeting latency SLOs.

7.6 EnerTune Incurs Minimal Overheads

EnerTune incurs minimal overheads. The Colocation Power Estimator estimates shared GPU power in 400 μ s. The Resource Planner takes <1.7s to generate execution plans (1.5 min for 1000 models). Model loading time ranges from 0.39s (MobileNetV3) to 1.53s (GPT-J-6B), and GPU configuration time (allocation, frequency, batch size) is <0.1s. Hence, total overhead is 2.24–3.38s, similar to baselines (1.83–3.98s). Obtaining P_S and P_O (§4.2) takes a few minutes (~ 3 min) per GPU type; this is a one-time cost off the critical path.

8 Related Work

Energy-aware serving. While several works make energy-aware scheduling and frequency scaling decisions [14, 60, 65, 68, 69, 96, 111, 112, 116, 124, 125, 131, 137, 155, 163, 167], they assume GPUs are not multiplexed. These works either target LLMs that require dedicated GPUs [68, 69, 111, 112, 116, 124, 125, 155] or operate on limited hardware resources (edge, small GPUs) [60, 65, 131, 137]. Both preclude GPU sharing, and hence, these systems do not need to make energy- and frequency-aware GPU-binning decisions. *Our work is the first to introduce energy-aware GPU sharing for ML inference.*

GPU sharing. Prior GPU sharing work falls into two categories: device-level schedulers and cluster-level serving systems. Device-level works [4, 27, 37, 40, 50, 54, 63, 64, 85, 99, 113, 126, 142, 151, 153, 154, 160, 161, 164, 165] propose new mechanisms for sharing and controlling GPU resources across concurrent processes. These works are orthogonal to EnerTune, as they do not make model placement and colocation decisions. Moreover, aside from LithOS [27], these works require expensive kernel-level profiling that takes hours and consumes substantial energy. Cluster-level systems make placement and colocation decisions, sharing GPUs via TM, MPS, or MIG. TM-based systems [28, 47, 62, 96, 108, 115, 142, 147, 148, 156] face poor utilization, as only a single task executes at a time. Systems using MPS [11, 20, 23, 33, 34, 38, 49, 117, 135, 146, 150] or MIG [21, 71, 75, 77, 127, 139, 157, 158, 161] bin-pack GPUs. However, their placement is energy-agnostic; we show energy-aware placement is critical (§2.1). Clover [78] reduces energy by trading accuracy for lower-quality model variants. Then, it multiplexes GPUs. Clover is orthogonal and can prune models before energy-aware placement and configuration.

GPU power modeling. Previous works estimating GPU power [2, 3, 6, 8, 39, 56, 70, 76, 83, 94, 121, 138, 144] rely on expensive profiling of low-level hardware usage (e.g., memory bandwidth utilization, L2 cache utilization). Though accurate, these approaches are impractical for shared GPUs, as they require profiling all model combinations to obtain inputs (e.g., hardware usage). Instead, our analytical model synthesizes isolated profiles into accurate shared GPU power.

Energy-aware cloud resource management. Prior systems reduce datacenter energy through server provisioning, application/VM consolidation and placement, power allocation, and CPU DVFS [7, 13, 17, 18, 36, 82, 98, 123, 133, 134]; several systems jointly optimize placement/consolidation and frequency [7, 36, 134]. These systems target CPU-centric servers and therefore do not model how ML-specific controls (batch size), GPU resource allocation, or a shared GPU frequency jointly affect power and performance. Their placement problem also differs fundamentally: CPU applications can execute on separate cores with independently controlled frequencies, whereas spatially colocated GPU models receive separate resource allocations but share one GPU frequency

and power domain. Thus, the models colocated determine the frequency that can satisfy all their SLOs, while their allocations and batch sizes jointly determine power, latency, and the required GPU count. EnerTune captures these GPU- and ML-specific interactions (§4.1–4.2) and jointly optimizes placement, allocation, batch size, and frequency (§5.1).

9 Discussion: Carbon-Efficiency

While we evaluate EnerTune for power and energy efficiency, it readily extends to carbon optimization, since carbon derives directly from power and energy (Appendix A). Carbon optimization is becoming increasingly important [120], as the Information and Communication Technology sector accounts for an estimated 1.5–4% of global carbon emissions [1, 9, 12, 95, 100]. Supporting carbon simply requires extending EnerTune’s EPA Resource Planner to derive carbon emissions from the Colocation Power Estimator and setting the operator objective to minimize carbon. Appendix A provides the corresponding analytical formulations.

10 Conclusion

EnerTune is a sustainability-aware inference serving system. It combines energy-conscious profiling with scalable power estimation using our analytical models to accurately estimate the power draw of colocated models from a small number of observations. Using our analytical formulations, it derives deployment plans that meet latency SLOs while minimizing serving cost, and dynamically adapts GPU frequency, batching, and allocations to changing workloads. Across models, workloads, and traces, EnerTune reduces GPU power and energy consumption compared to state-of-the-art systems.

11 Acknowledgments

We thank the SOSP’26 anonymous reviewers and our shepherd, Mosharaf Chowdhury, for their insightful comments. We thank Vivek Chawda, Mattan Erez, and John Wilkes for their invaluable feedback on the paper. We thank the UT-SysML research group for helpful discussions. This work was supported by the UT ECE junior faculty start-up fund, UT iMAGiNE consortium and its industrial affiliates, an award from the UT Machine Learning Lab (MLL), the AMD Chair Endowment, and the Amazon Research Award.

References

- [1] ACM Technology Policy Council. 2021. ACM TechBrief: Computing and Climate Change. <https://dl.acm.org/doi/pdf/10.1145/3483410>.
- [2] Gargi Alavani, Jineet Desai, Snehashu Saha, and Santonu Sarkar. 2023. Program Analysis and Machine Learning-based Approach to Predict Power Consumption of CUDA Kernel. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 8, 4, Article 10 (July 2023), 24 pages. doi:10.1145/3603533
- [3] Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. 2023. Performance-Aware Energy-Efficient GPU Frequency Selection using DNN-based Models. In *Proceedings of the 52nd International Conference on Parallel Processing* (Salt Lake City,

- UT, USA) (*ICPP '23*). Association for Computing Machinery, New York, NY, USA, 433–442. doi:10.1145/3605573.3605600
- [4] Tyler Allen, Xizhou Feng, and Rong Ge. 2019. Slate: Enabling Workload-Aware Efficient Multiprocessing for Modern GPGPUs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 252–261. doi:10.1109/IPDPS.2019.00035
 - [5] Amazon Web Services. 2024. Amazon Bedrock Model Choice. <https://aws.amazon.com/bedrock/model-choice/>.
 - [6] Oscar Antepara, Zhengji Zhao, Brian Austin, Nan Ding, Leonid Oliker, Nicholas J. Wright, and Samuel Williams. 2025. Benchmark-driven Models for Energy Analysis and Attribution of GPU-Accelerated Supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*. Association for Computing Machinery, New York, NY, USA, 888–904. doi:10.1145/3712285.3759815
 - [7] Patricia Arroba, José M. Moya, José L. Ayala, and Rajkumar Buyya. 2015. DVFS-Aware Consolidation for Energy-Efficient Clouds. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 494–495. doi:10.1109/PACT.2015.59
 - [8] Akhil Arunkumar, Evgeny Bolotin, David Nellans, and Carole-Jean Wu. 2019. Understanding the Future of Energy Efficiency in Multi-Module GPUs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 519–532. doi:10.1109/HPCA.2019.00063
 - [9] Seth Ayers, Sara Ballan, Vanessa Gray, and Rosie McDonald. 2024. *Measuring the Emissions & Energy Footprint of the ICT Sector: Implications for Climate Action*. Technical Report. World Bank.
 - [10] Amir Ayupov, Maksim Panchenko, and Sergey Pupyrev. 2024. Stale Profile Matching. In *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction (Edinburgh, United Kingdom) (CC 2024)*. Association for Computing Machinery, New York, NY, USA, 162–173. doi:10.1145/3640537.3641573
 - [11] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 499–514. <https://www.usenix.org/conference/osdi20/presentation/bai>
 - [12] Lotfi Belkhir and Ahmed Elmeligi. 2018. Assessing ICT Global Emissions Footprint: Trends to 2040 and Recommendations. *Journal of Cleaner Production* 177 (2018), 448–463. doi:10.1016/j.jclepro.2017.12.239
 - [13] Anton Beloglazov and Rajkumar Buyya. 2012. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. *Concurrency and Computation: Practice and Experience* 24, 13 (2012), 1397–1420. doi:10.1002/cpe.1867
 - [14] Srikant Bharadwaj, Shomit Das, Kaushik Mazumdar, Bradford M. Beckmann, and Stephen Kosonocky. 2023. Predict; Don't React for Enabling Efficient Fine-Grain DVFS in GPUs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4 (Vancouver, BC, Canada) (ASPLOS '23)*. Association for Computing Machinery, New York, NY, USA, 253–267. doi:10.1145/3623278.3624756
 - [15] Vivek M. Bhasi, Jashwant Raj Gunasekaran, Aakash Sharma, Mahmut Taylan Kandemir, and Chita Das. 2022. Cypress: Input Size-Sensitive Container Provisioning and Request Scheduling for Serverless Platforms. In *Proceedings of the 13th Symposium on Cloud Computing (San Francisco, California) (SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 257–272. doi:10.1145/3542929.3563464
 - [16] Tawhid Bhuiyan, Sumya Hoque, Angélica Moreira, and Tanvir Ahmed Khan. 2026. Wax: Optimizing Data Center Applications With Stale Profile. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (Pittsburgh, PA, USA) (ASPLOS '26)*. Association for Computing Machinery (ACM).
 - [17] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (Banff, Alberta, Canada) (SOSP '01)*. Association for Computing Machinery, New York, NY, USA, 103–116. doi:10.1145/502034.502045
 - [18] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. 2008. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 337–350.
 - [19] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: an automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (Carlsbad, CA, USA) (OSDI'18)*. USENIX Association, USA, 579–594.
 - [20] Wenyan Chen, Chengzhi Lu, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2025. Multiplexing Dynamic Deep Learning Workloads with SLO-awareness in GPU Clusters. In *Proceedings of the Twentieth European Conference on Computer Systems (Rotterdam, Netherlands) (EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 589–604. doi:10.1145/3689031.3696074
 - [21] Yukang Chen, Weihao Cui, Han Zhao, Ziyi Xu, Xiaozhe Fan, Xusheng Chen, Yangjie Zhou, Shixuan Sun, Bingsheng He, and Quan Chen. 2026. Towards High-Goodput LLM Serving with Prefill-decode Multiplexing. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (USA) (ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 2030–2047. doi:10.1145/3779212.3790236
 - [22] Sangjin Choi, Inhoe Koo, Jeongseob Ahn, Myeongjae Jeon, and Youngjin Kwon. 2023. EnvPipe: Performance-preserving DNN Training Framework for Saving Energy. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 851–864. <https://www.usenix.org/conference/atc23/presentation/choi>
 - [23] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 199–216. <https://www.usenix.org/conference/atc22/presentation/choi-seungbeom>
 - [24] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, Ritesh Tijoriwala, Denis Samoylov, and Chunqiang Tang. 2024. MAST: Global Scheduling of ML Training across Geo-Distributed Datacenters at Hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 563–580. <https://www.usenix.org/conference/osdi24/presentation/choudhury>
 - [25] Jae-Won Chung, Yile Gu, Insu Jang, Luoxi Meng, Nikhil Bansal, and Mosharaf Chowdhury. 2024. Reducing Energy Bloat in Large Model Training. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (Austin, TX, USA) (SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 144–159. doi:10.1145/3694715.3695970
 - [26] Jae-Won Chung, Jeff J. Ma, Ruofan Wu, Jiachen Liu, Oh Jun Kweon, Yuxuan Xia, Zhiyu Wu, and Mosharaf Chowdhury. 2025. The ML.ENERGY Benchmark: Toward Automated Inference Energy Measurement and Optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks*

- Track. <https://openreview.net/forum?id=CXqq2v6qUv>
- [27] Patrick H. Coppock, Brian Zhang, Eliot H. Solomon, Vasilis Kypriotis, Leon Yang, Bikash Sharma, Dan Schatzberg, Todd C. Mowry, and Dimitrios Skarlatos. 2025. LithOS: An Operating System for Efficient Machine Learning on GPUs. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (Lotte Hotel World, Seoul, Republic of Korea) (SOSP '25). Association for Computing Machinery, New York, NY, USA, 1–17. doi:10.1145/3731569.3764818
 - [28] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: a low-latency online prediction serving system. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) (NSDI'17). USENIX Association, USA, 613–627.
 - [29] Kate Crawford. 2024. Generative AI's environmental costs are soaring – and mostly secret. <https://www.nature.com/articles/d41586-024-00478-x>. *Nature* 626, 8000 (2024), 693. doi:10.1038/d41586-024-00478-x
 - [30] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) (ASPLOS '13). Association for Computing Machinery, New York, NY, USA, 77–88. doi:10.1145/2451116.2451125
 - [31] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) (ASPLOS '14). Association for Computing Machinery, New York, NY, USA, 127–144. doi:10.1145/2541940.2541941
 - [32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. doi:10.1109/CVPR.2009.5206848
 - [33] Aditya Dhakal, Sameer G Kulkarni, and K. K. Ramakrishnan. 2020. GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) (SoCC '20). Association for Computing Machinery, New York, NY, USA, 492–506. doi:10.1145/3419111.3421284
 - [34] Aditya Dhakal, K. K. Ramakrishnan, Sameer G. Kulkarni, Puneet Sharma, and Junguk Cho. 2022. Slice-Tune: A System for High Performance DNN Autotuning. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference* (Quebec, QC, Canada) (Middleware '22). Association for Computing Machinery, New York, NY, USA, 228–240. doi:10.1145/3528535.3565247
 - [35] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence* 5, 3 (2023), 220–235. doi:10.1038/s42256-023-00626-4
 - [36] Youwei Ding, Xiaolin Qin, Liang Liu, and Taochun Wang. 2015. Energy Efficient Scheduling of Virtual Machines in Cloud with Deadline Constraint. *Future Generation Computer Systems* 50 (2015), 62–74. doi:10.1016/j.future.2015.02.001
 - [37] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2021. IOS: Inter-Operator Scheduler for CNN Acceleration. In *Proceedings of Machine Learning and Systems 2021 (MLSys 2021)*. mlsys.org.
 - [38] Jiaang Duan, Shenglin Xu, Shiyu Qian, Dingyu Yang, Kangjin Wang, Chenzhi Liao, Yinghao Yu, Qin Hua, Hanwen Hu, Qi Wang, Wenchao Wu, Dongqing Bao, Tianyu Lu, Jian Cao, Guangtao Xue, Guodong Yang, Liping Zhang, and Gang Chen. 2026. GFS: A Preemption-aware Scheduling Framework for GPU Clusters with Predictive Spot Instance Management. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (USA) (ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 117–131. doi:10.1145/3760250.3762231
 - [39] Bishwajit Dutta, Vignesh Adhinarayanan, and Wu-chun Feng. 2018. GPU power prediction via ensemble machine learning for DVFS space exploration. In *Proceedings of the 15th ACM International Conference on Computing Frontiers* (Ischia, Italy) (CF '18). Association for Computing Machinery, New York, NY, USA, 240–243. doi:10.1145/3203217.3203273
 - [40] Paul Elvinger, Foteini Strati, Natalie Enright Jerger, and Ana Klimovic. 2025. Understanding GPU Resource Interference One Level Deeper. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC'25)*. Association for Computing Machinery, New York, NY, USA, 687–694. doi:10.1145/3772052.3772270
 - [41] Kaijie Fan, Marco D'Antonio, Lorenzo Carpentieri, Biagio Cosenza, Federico Ficarella, and Daniele Cesarini. 2023. Synergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) (SC '23). Association for Computing Machinery, New York, NY, USA, Article 69, 13 pages. doi:10.1145/3581784.3607055
 - [42] Elias Frantar and Dan Alistarh. 2023. SparseGPT: massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) (ICML '23). JMLR.org, Article 414, 15 pages.
 - [43] Google. 2015. gRPC. <https://grpc.io/>.
 - [44] Google SRE Team. 2019. Google SRE Workbook: Implementing SLOs. <https://sre.google/workbook/implementing-slos/>.
 - [45] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 485–500. <https://www.usenix.org/conference/nsdi19/presentation/gu>
 - [46] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 789–800. doi:10.1109/HPCA.2018.00072
 - [47] Arpan Gujari, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 443–462. <https://www.usenix.org/conference/osdi20/presentation/gujarati>
 - [48] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (New York, New York) (ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. doi:10.1145/3470496.3527408
 - [49] Bing-Shiun Han, Tathagata Paul, Zhenhua Liu, and Anshul Gandhi. 2024. KACE: Kernel-Aware Colocation for Efficient GPU Spatial Sharing. In *Proceedings of the 2024 ACM Symposium on Cloud Computing (Redmond, WA, USA) (SoCC '24)*. Association for Computing Machinery, New York, NY, USA, 460–469. doi:10.1145/3698038.3698555
 - [50] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 539–558. <https://www.usenix.org/conference/osdi22/presentation/han>

- [51] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.
- [52] Wenlei He. 2020. [RFC] Context-sensitive Sample PGO with Pseudo-Instrumentation. https://groups.google.com/g/llvm-dev/c/1p1rdYbL93s/m/AN_hgPmnAwAJ. Google Groups discussion on LLVM-dev.
- [53] Wenlei He, Julián Mestre, Sergey Pupyrev, Lei Wang, and Hongtao Yu. 2022. Profile inference revisited. *Proc. ACM Program. Lang.* 6, POPL, Article 52 (Jan. 2022), 24 pages. doi:10.1145/3498714
- [54] Yongjun He, Haofeng Yang, Yao Lu, Ana Klimović, and Gustavo Alonso. 2025. Resource multiplexing in tuning and serving large language models. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) (*USENIX ATC '25*). USENIX Association, USA, Article 97, 17 pages.
- [55] D. Richard Hipp. 2024. SQLite. <https://www.sqlite.org>.
- [56] Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) (*ISCA '10*). Association for Computing Machinery, New York, NY, USA, 280–289. doi:10.1145/1815961.1815998
- [57] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Lucid: A Non-intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) (*ASPLOS 2023*). Association for Computing Machinery, New York, NY, USA, 457–472. doi:10.1145/3575693.3575705
- [58] Hugging Face, Inc. 2026. Hugging Face: The AI Community Building the Future. <https://huggingface.co/>.
- [59] International Energy Agency. 2025. Energy and AI. <https://www.iea.org/reports/energy-and-ai>.
- [60] Md. Najrul Islam, Rahul Shrestha, and Shubhajit Roy Chowdhury. 2024. Energy-Efficient and High-Throughput CNN Inference Engine Based on Memory-Sharing and Data-Reusing for Edge Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers* 71, 7 (2024), 3189–3202. doi:10.1109/TCSI.2024.3392807
- [61] Pierre Jacquet, Maxime Agusti, Eddy Caron, Camille Coti, Marcos Dias de Assuncao, Laurent Lefèvre, and Anne-Cécile Orgerie. 2026. Untangling GPU Power Consumption: Job-Level Inference in Cloud Shared Settings. In *EUROSYS 2026 - European Conference on Computer Systems*. ACM, Edinbourg, Ecosse, United Kingdom. doi:10.1145/3767295.3769333
- [62] Ali Jahanshahi, Mohammadreza Rezvani, and Daniel Wong. 2024. WattWiser: Power & Resource-Efficient Scheduling for Multi-Model Multi-GPU Inference Servers. In *Proceedings of the 14th International Green and Sustainable Computing Conference* (Toronto, ON, Canada) (*IGSC '23*). Association for Computing Machinery, New York, NY, USA, 39–44. doi:10.1145/3634769.3634807
- [63] Saksham Jain, Iljoo Baek, Shige Wang, and Ragunathan Rajkumar. 2019. Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 29–41. doi:10.1109/RTAS.2019.00011
- [64] Hyeon-Jun Jang, Sang-Jae Kim, Weikuan Yu, and Hyun-Wook Jin. 2026. Spatio-Temporal Resource Control for Cloud-Native GPU Provisioning. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC '25)*. Association for Computing Machinery, New York, NY, USA, 695–707. doi:10.1145/3772052.3772271
- [65] Jihoon Jang, Hyeon Lee, and Hyun Kim. 2024. EDen: Enabling Low-Power CNN Inference on Edge Devices Using Prefetcher-assisted NVM Systems. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design* (Newport Beach, CA, USA) (*ISLPED '24*). Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/3665314.3670801
- [66] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 947–960. <https://www.usenix.org/conference/atc19/presentation/jeon>
- [67] Josh Gabbatiss. 2025. AI: Five charts that put data-centre energy use – and emissions – into context. <https://www.carbonbrief.org/ai-five-charts-that-put-data-centre-energy-use-and-emissions-into-context/>
- [68] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutos, Sotirios Xydis, and Dimitrios Soudris. 2025. throttLL'eM: Predictive GPU Throttling for Energy Efficient LLM Inference Serving. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1363–1378. doi:10.1109/HPCA61900.2025.00103
- [69] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2024. SLO-Aware GPU DVFS for Energy-Efficient LLM Inference Serving. *IEEE Comput. Archit. Lett.* 23, 2 (July 2024), 150–153. doi:10.1109/LCA.2024.3406038
- [70] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavell. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) (*MICRO '21*). Association for Computing Machinery, New York, NY, USA, 738–753. doi:10.1145/3466752.3480063
- [71] Yunseong Kim, Yujeong Choi, and Minsoo Rhu. 2022. PARIS and ELSA: an elastic scheduling algorithm for reconfigurable multi-GPU inference servers. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) (*DAC '22*). Association for Computing Machinery, New York, NY, USA, 607–612. doi:10.1145/3489517.3530510
- [72] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 427–444. <https://www.usenix.org/conference/osdi18/presentation/klimovic>
- [73] Neeraj Kumar, Pol Mauri Ruiz, Vijay Menon, Igor Kabiljo, Mayank Pundir, Andrew Newell, Daniel Lee, Liyuan Wang, and Chunqiang Tang. 2024. Optimizing Resource Allocation in Hyperscale Datacenters: Scalability, Usability, and Experiences. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 507–528. <https://www.usenix.org/conference/osdi24/presentation/kumar>
- [74] 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* (Koblenz, Germany) (*SOSP '23*). Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165
- [75] Munkyu Lee, Sihoon Seong, Minki Kang, Jihyuk Lee, Gap-Joo Na, In-Geol Chun, Dimitrios Nikolopoulos, and Cheol-Ho Hong. 2024. ParvaGPU: Efficient Spatial GPU Sharing for Large-Scale DNN Inference in Cloud Environments. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, Los Alamitos, CA, USA, 1–14. doi:10.1109/SC41406.2024.00048
- [76] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013.

- GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) (ISCA '13). Association for Computing Machinery, New York, NY, USA, 487–498. doi:10.1145/2485922.2485964
- [77] 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* (San Francisco, California) (SoCC '22). Association for Computing Machinery, New York, NY, USA, 173–189. doi:10.1145/3542929.3563510
- [78] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) (SC '23). Association for Computing Machinery, New York, NY, USA, Article 20, 15 pages. doi:10.1145/3581784.3607034
- [79] Suyi Li, Wei Wang, Jun Yang, Guangzhen Chen, and Daohe Lu. 2023. Golgi: Performance-Aware, Resource-Efficient Function Scheduling for Serverless Computing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) (SoCC '23). Association for Computing Machinery, New York, NY, USA, 32–47. doi:10.1145/3620678.3624645
- [80] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 663–679. <https://www.usenix.org/conference/osdi23/presentation/li-zhouhan>
- [81] Baohao Liao, Yan Meng, and Christof Monz. 2023. Parameter-Efficient Fine-Tuning without Introducing New Latency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 4242–4260. doi:10.18653/v1/2023.acl-long.233
- [82] Harold Lim, Aman Kansal, and Jie Liu. 2011. Power Budgeting for Virtualized Data Centers. In *Proceedings of the 2011 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association.
- [83] Jieun Lim, Nagesh B. Lakshminarayana, Hyesoon Kim, William Song, Sudhakar Yalamanchili, and Wonyong Sung. 2014. Power Modeling for GPU Architectures Using McPAT. *ACM Trans. Des. Autom. Electron. Syst.* 19, 3, Article 26 (June 2014), 24 pages. doi:10.1145/2611758
- [84] Ji Lin et al. 2024. AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. In *MLSys*.
- [85] Zejia Lin, Hongxin Xu, Guanyi Chen, Zhiguang Chen, Yutong Lu, and Xianwei Zhang. 2026. Bullet: Boosting GPU Utilization for LLM Serving via Dynamic Spatial-Temporal Orchestration. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (USA) (ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 290–306. doi:10.1145/3779212.3790135
- [86] Sasha Luccioni, Boris Gamazaychikov, Sara Hooker, Régis Pierrard, Emma Strubell, Yacine Jernite, and Carole-Jean Wu. 2024. Light bulbs have energy ratings — so why can't AI chatbots? <https://www.nature.com/articles/d41586-024-02680-3>. *Nature* 632, 8026 (2024), 736–738. doi:10.1038/d41586-024-02680-3
- [87] M. D. McKay, R. J. Beckman, and W. J. Conover. 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* 21, 2 (1979), 239–245. <http://www.jstor.org/stable/1268522>
- [88] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. WikiText Language Modeling Dataset. <https://huggingface.co/datasets/Salesforce/wikitext>. <https://huggingface.co/datasets/Salesforce/wikitext> Accessed: 2026-01-25.
- [89] Meta AI. 2021. PyTorch Builds the Future of AI and Machine Learning at Facebook. <https://ai.meta.com/blog/pytorch-builds-the-future-of-ai-and-machine-learning-at-facebook/>.
- [90] Microsoft. 2024. Microsoft Foundry Models Overview. <https://learn.microsoft.com/en-us/azure/foundry/concepts/foundry-models-overview>.
- [91] MLCommons. 2026. MLPerf® Inference Benchmark Suite. <https://github.com/mlcommons/inference>.
- [92] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 579–596. <https://www.usenix.org/conference/osdi22/presentation/mohan>
- [93] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. 2012. *Introduction to Linear Regression Analysis*. Wiley.
- [94] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. 2022. Performance and Power Prediction for Concurrent Execution on GPUs. *ACM Trans. Archit. Code Optim.* 19, 3, Article 35 (May 2022), 27 pages. doi:10.1145/3522712
- [95] David Mytton and Masao Ashtine. 2022. Sources of Data Center Energy Estimates: A Comprehensive Review. *Joule* 6, 9 (2022), 2032–2056. doi:10.1016/j.joule.2022.07.011
- [96] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2021. BatchSizer: Power-Performance Trade-off for DNN Inference. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 819–824. doi:10.1145/3394885.3431535
- [97] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 481–498. <https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak>
- [98] Ripal Nathuji and Karsten Schwan. 2007. VirtualPower: coordinated power management in virtualized enterprise systems. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA) (SOSP '07). Association for Computing Machinery, New York, NY, USA, 265–278. doi:10.1145/1294261.1294287
- [99] Kelvin K. W. Ng, Henri Maxime Demoulin, and Vincent Liu. 2023. Paella: Low-latency Model Serving with Software-defined GPU Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) (SOSP '23). Association for Computing Machinery, New York, NY, USA, 595–610. doi:10.1145/3600006.3613163
- [100] Thong Trung Nguyen, Thu Anh Thi Pham, and Huong Thi Xuan Tram. 2020. Role of Information and Communication Technologies and Innovation in Driving Carbon Emissions and Economic Growth in Selected G-20 Countries. *Journal of Environmental Management* 261 (2020), 110162. doi:10.1016/j.jenvman.2020.110162
- [101] NVIDIA. 2014. NVIDIA Multi-Process Service (MPS). <https://docs.nvidia.com/deploy/mps>.
- [102] NVIDIA. 2025. NVIDIA Management Library (NVML). <https://developer.nvidia.com/management-library-nvml>.
- [103] NVIDIA. 2026. Inside NVIDIA Rubin GPU Architecture: Powering the Era of Agentic AI. <https://developer.nvidia.com/blog/inside-nvidia-rubin-gpu-architecture-powering-the-era-of-agentic-ai/>.
- [104] NVIDIA. 2026. NVIDIA Blackwell Architecture. <https://www.nvidia.com/en-us/data-center/technologies/blackwell-architecture/>.
- [105] NVIDIA. 2026. NVIDIA Multi Instance GPU. <https://www.nvidia.com/en-us/technologies/multi-instance-gpu/>.
- [106] NVIDIA. 2026. System Management Interface SMI. <https://developer.nvidia.com/system-management-interface>.
- [107] NVIDIA Corporation. 2026. NVIDIA TensorRT. <https://github.com/NVIDIA/TensorRT>.

- [108] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. arXiv:1712.06139 [cs.DC] <https://arxiv.org/abs/1712.06139>
- [109] George Ostrouchov, Don Maxwell, Rizwan A. Ashraf, Christian Engelmann, Mallikarjun Shankar, and James H. Rogers. 2020. GPU lifetimes on titan supercomputer: survival analysis and reliability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) (SC '20). IEEE Press, Article 41, 14 pages.
- [110] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An ASR Corpus Based on Public Domain Audio Books. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP). IEEE, 5206–5210.
- [111] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warriar, Nithish Mahalingam, and Ricardo Bianchini. 2024. Characterizing Power Management Opportunities for LLMs in the Cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) (ASPLOS '24). Association for Computing Machinery, New York, NY, USA, 207–222. doi:10.1145/3620666.3651329
- [112] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Basar, and Ravishankar K. Iyer. 2024. Power-aware Deep Learning Model Serving with μ -Serve. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 75–93. <https://www.usenix.org/conference/atc24/presentation/qiu>
- [113] John Ravi, Tri Nguyen, Huiyang Zhou, and Michela Becchi. 2021. PILOT: a Runtime System to Manage Multi-tenant GPU Unified Memory Footprint. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE Computer Society, Los Alamitos, CA, USA, 442–447. doi:10.1109/HiPC53243.2021.00063
- [114] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411. <https://www.usenix.org/conference/atc21/presentation/romero>
- [115] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) (SOSP '19). Association for Computing Machinery, New York, NY, USA, 322–337. doi:10.1145/3341301.3359658
- [116] Tianyao Shi, Yanran Wu, Sihang Liu, and Yi Ding. 2025. Disaggregated Speculative Decoding for Carbon-Efficient LLM Serving. *IEEE Computer Architecture Letters* 24, 2 (2025), 369–372. doi:10.1109/LCA.2025.3630094
- [117] Sudipta Saha Shubha, Haiying Shen, and Anand Iyer. 2024. USHER: Holistic Interference Avoidance for Resource Optimized ML Inference. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 947–964. <https://www.usenix.org/conference/osdi24/presentation/shubha>
- [118] Arjun Singhvi, Arjun Balasubramanian, Kevin Houck, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2021. Atoll: A Scalable Low-Latency Serverless Platform. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) (SoCC '21). Association for Computing Machinery, New York, NY, USA, 138–152. doi:10.1145/3472883.3486981
- [119] Prasoon Sinha, Kostis Kaffes, and Neeraja J. Yadwadkar. 2025. ALAP: Intent-Based Serverless Computing via Delayed Decision-Making. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC'25)*. Association for Computing Machinery, New York, NY, USA, 831–846. doi:10.1145/3772052.3772262
- [120] Prasoon Sinha, Dimitrios Liakopoulos, Ruihao Li, and Neeraja J. Yadwadkar. 2025. The Utilization Fallacy and the Real Drivers of Carbon-Efficient Inference Serving. *SIGENERGY Energy Inform. Rev.* 5, 2 (Aug. 2025), 76–83. doi:10.1145/3757892.3757903
- [121] Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS '13)*. IEEE Computer Society, USA, 673–686. doi:10.1109/IPDPS.2013.73
- [122] Morgan Stanley. 2024. Powering the AI Revolution. <https://www.morganstanley.com/ideas/ai-energy-demand-infrastructure>.
- [123] Jan Stoess, Christian Lang, and Frank Bellosa. 2007. Energy Management for Hypervisor-Based Virtual Machines. In *Proceedings of the 2007 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 1–14.
- [124] Jovan Stojkovic, Chaojie Zhang, Inigo 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)*. IEEE Computer Society, Los Alamitos, CA, USA, 1348–1362. doi:10.1109/HPCA61900.2025.00102
- [125] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Esha Choukse, Haoran Qiu, Rodrigo Fonseca, Josep Torrellas, and Ricardo Bianchini. 2025. TAPAS: Thermal- and Power-Aware Scheduling for LLM Inference in Cloud Platforms. Association for Computing Machinery, New York, NY, USA, 1266–1281. <https://doi.org/10.1145/3676641.3716025>
- [126] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) (EuroSys '24). Association for Computing Machinery, New York, NY, USA, 1075–1092. doi:10.1145/3627703.3629578
- [127] Cheng Tan, Zhichao Li, Jian Zhang, Yu Cao, Sikai Qi, Zherui Liu, Yibo Zhu, and Chuanxiong Guo. 2021. Serving DNN models with multi-instance gpus: A case of the reconfigurable machine scheduling problem. *arXiv preprint arXiv:2109.11067* (2021).
- [128] Xiaochuan Tang, Hao Qi, Jianbo Dong, Yinghao Yu, Zhennan Xue, Zhengyu Zhang, Daocheng Ying, Zheng Cao, and Xiaoyi Lu. 2026. eGPU: Production-Scale Elastic Sharing Over 10,000 GPUs. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. doi:10.1109/HPCA68181.2026.11408556
- [129] The SciPy Community. 2021. LatinHypercube. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.LatinHypercube.html>.
- [130] Huangshi Tian, Suyi Li, Ao Wang, Wei Wang, Tianlong Wu, and Hao-ran Yang. 2022. Owl: Performance-Aware Scheduling for Resource-Efficient Function-as-a-Service Cloud. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) (SoCC '22). Association for Computing Machinery, New York, NY, USA, 78–93. doi:10.1145/3542929.3563470
- [131] Mohammed F. Tolba, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, Ayman Hroub, and Thanos Stouraitis. 2024. Energy Efficient and Fast CNN Inference by Exploring Weight Approximation and Computational Reuse. *IEEE Transactions on Artificial Intelligence* 5, 6 (2024), 2678–2692. doi:10.1109/TAI.2023.3324315
- [132] UNESCO. 2025. AI Large Language Models: new report shows small changes can reduce energy use by 90%. <https://www.unesco.org/en/articles/ai-large-language-models-new-report-shows-small-changes-can-reduce-energy-use-90>.

- [133] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (Leuven, Belgium) (Middleware '08). Springer-Verlag, Berlin, Heidelberg, 243–264.
- [134] Gregor von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. 2009. Power-Aware Scheduling of Virtual Machines in DVFS-Enabled Clusters. In *Proceedings of the IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*. doi:10.1109/CLUSTER.2009.5289182
- [135] Jiali Wang, Yankui Wang, Mingcong Han, and Rong Chen. 2025. Colocating ML inference and training with fast GPU memory handover. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) (USENIX ATC '25). USENIX Association, USA, Article 98, 19 pages.
- [136] 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. doi:10.1145/3731599.3767396
- [137] Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2020. Energy-efficient Inference Service of Transformer-based Deep Learning Models on GPUs. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. 323–331. doi:10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00067
- [138] Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian. 2025. Using Analytical Performance/Power Model and Fine-Grained DVFS to Enhance AI Accelerator Energy Efficiency. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) (ASPLOS '25). Association for Computing Machinery, New York, NY, USA, 1118–1132. doi:10.1145/3669940.3707231
- [139] Xinpeng Wei, Zhichao Li, and Cheng Tan. 2024. Optimizing GPU Sharing for Container-Based DNN Serving with Multi-Instance GPUs. In *Proceedings of the 17th ACM International Systems and Storage Conference* (Virtual, Israel) (SYSTOR '24). Association for Computing Machinery, New York, NY, USA, 68–82. doi:10.1145/3688351.3689156
- [140] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 945–960. <https://www.usenix.org/conference/nsdi22/presentation/weng>
- [141] Qizhen Weng, Lingyun Yang, Yinghao Yu, Wei Wang, Xiaochuan Tang, Guodong Yang, and Liping Zhang. 2023. Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 995–1008. <https://www.usenix.org/conference/atc23/presentation/weng>
- [142] Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. 2023. Transparent GPU Sharing in Container Clouds for Deep Learning Workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 69–85. <https://www.usenix.org/conference/nsdi23/presentation/wu>
- [143] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. In *Proceedings of Machine Learning and Systems* (2022), D. Marculescu, Y. Chi, and C. Wu (Eds.), Vol. 4. 795–813.
- [144] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 564–576. doi:10.1109/HPCA.2015.7056063
- [145] Ruofan Wu, Jae-Won Chung, and Mosharaf Chowdhury. 2026. Kareus: Joint Reduction of Dynamic and Static Energy in Large Model Training. arXiv:2601.17654 [cs.LG] <https://arxiv.org/abs/2601.17654>
- [146] Yuxing Xiang, Xue Li, Kun Qian, Yufan Yang, Diwen Zhu, Wenyan Yu, Ennan Zhai, Xuanzhe Liu, Xin Jin, and Jingren Zhou. 2025. Aegaeon: Effective GPU Pooling for Concurrent LLM Serving on the Market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (Lotte Hotel World, Seoul, Republic of Korea) (SOSP '25). Association for Computing Machinery, New York, NY, USA, 1030–1045. doi:10.1145/3731569.3764815
- [147] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 595–610. <https://www.usenix.org/conference/osdi18/presentation/xiao>
- [148] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 533–548. <https://www.usenix.org/conference/osdi20/presentation/xiao>
- [149] Kaiqiang Xu, Decang Sun, Han Tian, Junxue Zhang, and Kai Chen. 2025. GREEN: carbon-efficient resource scheduling for machine learning clusters. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation* (Philadelphia, PA, USA) (NSDI '25). USENIX Association, USA, Article 53, 16 pages.
- [150] Chunyu Xue, Yi Pan, Weihao Cui, Quan Chen, Shulai Zhang, Bingsheng He, and Minyi Guo. 2026. MuxTune: Efficient Multi-Task LLM Fine-Tuning in Multi-Tenant Datacenters via Spatial-Temporal Backbone Multiplexing. arXiv:2603.02885 [cs.DC] <https://arxiv.org/abs/2603.02885>
- [151] Yanan Yang, Zhengxiong Jiang, Meiqi Zhu, Hongqiang Xu, Yujun Wang, Liang Li, Jiansong zhang, and Jie Wu. 2026. gShare: Efficient GPU Sharing with Aggressive Scheduling in Multi-tenant FaaS platform. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) (ASPLOS '26). Association for Computing Machinery, New York, NY, USA, 838–859. doi:10.1145/3779212.3790168
- [152] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2023. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 119–139. <https://www.usenix.org/conference/nsdi23/presentation/you>
- [153] Fuxun Yu, Shawn Bray, Di Wang, Longfei Shangguan, Xulong Tang, Chenchen Liu, and Xiang Chen. 2021. Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD51958.2021.9643501
- [154] Haoxuan Yu, Sheng Yao, and Wei Wang. 2025. ZipBatch: Multi-Tenant GPU Batching with Dual-Resource Regulation. In *Proceedings of the*

- 2025 ACM Symposium on Cloud Computing (SoCC'25). Association for Computing Machinery, New York, NY, USA, 240–254. doi:10.1145/3772052.3772229
- [155] Junyeol Yu, Jongseok Kim, and Euseong Seo. 2023. Know Your Enemy To Save Cloud Energy: Energy-Performance Characterization of Machine Learning Serving. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 842–854. doi:10.1109/HPCA56546.2023.10070943
- [156] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-Grained GPU Sharing Primitives for Deep Learning Applications. In *Proceedings of Machine Learning and Systems 2020 (MLSys 2020)*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2020/file/d9cd83bc91b8c36a0c7c0fcca59228f2-Paper.pdf
- [157] Bowen Zhang, Shuxin Li, and Zhuozhao Li. 2024. MIGER: Integrating Multi-Instance GPU and Multi-Process Service for Deep Learning Clusters. In *Proceedings of the 53rd International Conference on Parallel Processing (Gotland, Sweden) (ICPP '24)*. Association for Computing Machinery, New York, NY, USA, 504–513. doi:10.1145/3673038.3673089
- [158] Bowen Zhang, Yuhang Wang, and Zhuozhao Li. 2025. BOER: Enhancing Resource Utilization for Deep Learning Inference with Hybrid Spatial GPU Sharing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*. Association for Computing Machinery, New York, NY, USA, 519–532. doi:10.1145/3712285.3759857
- [159] 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)*. USENIX Association, Boston, MA, 787–808. <https://www.usenix.org/conference/nsdi23/presentation/zhang-hong>
- [160] Shulai Zhang, Quan Chen, Weihao Cui, Han Zhao, Chunyu Xue, Zhen Zheng, Wei Lin, and Minyi Guo. 2025. Improving GPU Sharing Performance through Adaptive Bubbleless Spatial-Temporal Sharing. In *Proceedings of the Twentieth European Conference on Computer Systems (Rotterdam, Netherlands) (EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 573–588. doi:10.1145/3689031.3696070
- [161] Shulai Zhang, Ao Xu, Quan Chen, Han Zhao, Weihao Cui, Zhen Wang, Yan Li, Limin Xiao, and Minyi Guo. 2025. Efficient performance-aware GPU sharing with compatibility and isolation through kernel space interception. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference (Boston, MA, USA) (USENIX ATC '25)*. USENIX Association, USA, Article 59, 17 pages.
- [162] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (Virtual Event, Germany) (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 724–739. doi:10.1145/3477132.3483580
- [163] Yijia Zhang, Qiang Wang, Zhe Lin, Pengxiang Xu, and Bingqiang Wang. 2024. Improving GPU Energy Efficiency through an Application-transparent Frequency Scaling Policy with Performance Assurance. In *Proceedings of the Nineteenth European Conference on Computer Systems (Athens, Greece) (EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 769–785. doi:10.1145/3627703.3629584
- [164] Wei Zhao, Anand Jayarajan, and Gennady Pekhimenko. 2025. Tally: Non-Intrusive Performance Isolation for Concurrent Deep Learning Workloads. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (Rotterdam, Netherlands) (ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1052–1068. doi:10.1145/3669940.3707282
- [165] Zhihe Zhao, Neiwen Ling, Nan Guan, and Guoliang Xing. 2022. Aaron: Compile-time Kernel Adaptation for Multi-DNN Inference Acceleration on Edge GPU. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (Boston, Massachusetts) (SenSys'22)*. Association for Computing Machinery, New York, NY, USA, 802–803. doi:10.1145/3560905.3568050
- [166] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: generating high-performance tensor programs for deep learning. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 49, 17 pages.
- [167] Zhong Zheng, Seyfal Sultanov, Michael E. Papka, and Zhiling Lan. 2025. Minimizing Power Waste in Heterogenous Computing via Adaptive Uncore Scaling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*. Association for Computing Machinery, New York, NY, USA, 505–518. doi:10.1145/3712285.3759879
- [168] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation (Santa Clara, CA, USA) (OSDI'24)*. USENIX Association, USA, Article 11, 18 pages.

A Sustainability Metrics

We provide a brief background on different sustainability metrics cluster operators may consider optimizing for. Common metrics include power draw (measured in Watts) and energy consumption (measured in Joules), where $E = \int P dt$. Recently, several works focus on reducing carbon emissions, as the Information and Communication Technology sector accounts for roughly 1.5–4% of worldwide carbon emissions [1, 9, 12, 95, 100]. Unlike power, GPUs do not capture carbon emissions. Therefore, we derive the carbon footprint of serving inference requests with analytical models established by prior work [48]. Formally, the total carbon footprint C_T is:

$$C_T = C_O + C_E \quad (5)$$

where C_O and C_E are the operational and embodied emissions to serve the inference workload, respectively. Operational emissions measure the carbon footprint associated with the electricity consumed during inference, which depends on both the net energy E consumed and the carbon intensity CI of the power grid:

$$C_O = CI \times E = CI \times \int P dt. \quad (6)$$

Embodied emissions capture the upstream carbon costs of manufacturing the GPU hardware itself. Prior works leverage ACT’s customer-based carbon-accounting methodology [48] to attribute manufacturing emissions to individual workloads¹ by the ratio between the workload’s execution time (T) and the GPU’s lifetime (LT) scaled by the manufacturing emissions of the GPU (C_M). Therefore, the embodied footprint for serving a workload of inference requests is given by:

$$C_E = \frac{T}{LT} \times C_M. \quad (7)$$

In this work, we assume a total GPU lifetime of 5 years, typical of datacenter components [48, 109]. Moreover, as operators may choose to optimize for different metrics (power, energy, or carbon emissions), we build a flexible system that can naturally adapt to the operator’s intent.

B Joint Allocation & Batching

Impact of configuration knobs on latency. In §2.2, we show the throughput versus energy under unique allocations/batch size combinations for three models. Here, we show the data for all models in Figure 20. Frequency is fixed to 1410 MHz, the maximum frequency. The data shows that for a given throughput (e.g., ResNet50 at 750 req/sec), the points in the figure that are most energy-efficient tend to be smaller batches (circles, squares, diamonds) with slightly larger allocations. This suggests that selecting batch size first and then allocating resources is energy-efficient. However, this decoupled approach reduces packing density—larger

¹Given the debate over including embodied emissions during runtime, EnerTune supports both operational-only and operational+embodied modes.

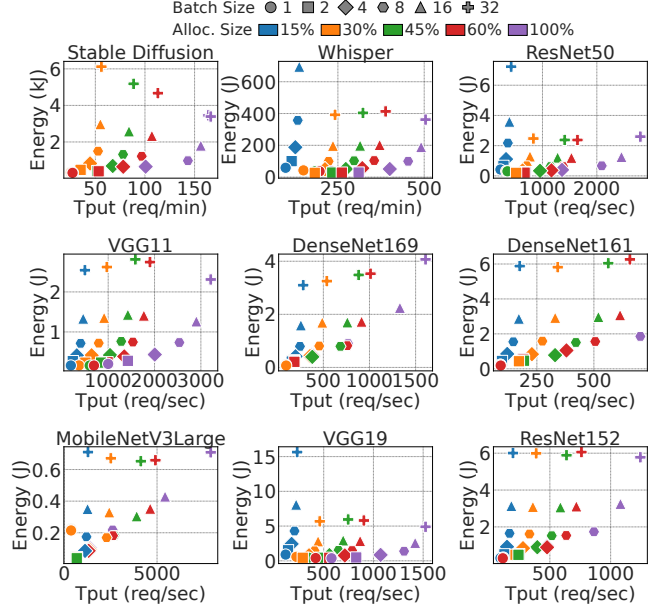


Figure 20. Throughput vs. energy across GPU allocations (color) and batch sizes (shape). See §B.

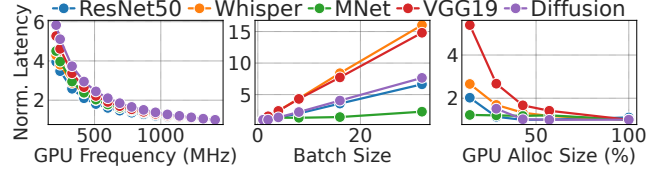


Figure 21. Impact of each deployment configuration knob on inference latency. See §C.

allocations are required to sustain arrival rates—leading to more GPUs used and high cluster-wide energy. Thus, to minimize cluster-wide energy, EnerTune jointly reasons over allocations and batch sizes across models.

C Estimating Per-Model Latency & Throughput

In §4.1, we present our analytical model for estimating the power draw of individual models under different deployment configurations (GPU allocation size, batch size, GPU frequency). Here, we present our analytical formulation for latency. Then, we explain how we use these latency estimations to derive throughput.

Impact of configuration knobs on latency. To guide our analytical formulation, we study how GPU allocation size, frequency, and batch size, each impacts a model’s inference latency. Across models, frequency exhibits a hyperbolic decay relationship with latency (Figure 21), as shown in prior work [112]. Batch size increases latency superlinearly: each additional request in a batch contributes additional sequential work, and at larger batch sizes, memory

bandwidth and compute pipeline pressure compound, accelerating the latency increase. Finally, GPU allocation effects are model-specific and exhibit a sharp initial drop that rapidly plateaus: models with larger memory footprints (e.g., VGG19) suffer disproportionately at small allocations due to memory-bandwidth pressure, while smaller models (e.g., MobileNetV3) are largely insensitive to allocation size.

Analytical model. To estimate latency across configurations, our model must capture three properties (Figure 21): (a) hyperbolic decay with GPU frequency, (b) superlinear growth with batch size, and (c) model-specific effects of GPU allocation size and the interactions among these knobs.

To capture these requirements, similar to power draw, we model latency with a third-degree multivariate polynomial over frequency, batch size, and GPU allocation size. However, unlike power, where the raw value of configuration knobs aligns naturally with the polynomial’s representational capacity, latency’s hyperbolic decay with frequency $L \propto 1/f$ and sharp drop with allocation size lie fundamentally outside what a polynomial can approximate well over a wide input range. A polynomial fit over raw frequency would require high-degree terms to chase the steep low-frequency region while simultaneously fitting the flat high-frequency tail, producing poor generalization. Instead, we apply input transformations—substituting $1/f$ for frequency and $\log(a)$ for GPU allocation size—that approximate these dominant nonlinearities. This reduces the hyperbolic and saturating relationships to shapes the polynomial handles naturally, while retaining cross-terms to capture model-specific interactions and the superlinear growth with batch size. Formally, we estimate latency L as:

$$L(f, b, a) = \sum_{i=1}^3 \left[\beta_{i,f} \left(\frac{1}{f} \right)^i + \beta_{i,b} b^i + \beta_{i,a} (\log a)^i \right] + \sum_{\substack{i+j+k \leq 3 \\ \text{at least two of } i,j,k > 0}} \beta_{ijk} \left(\frac{1}{f} \right)^i b^j (\log a)^k \quad (8)$$

where $f, b, a > 0$ denote frequency, batch size, and GPU allocation size. The first term captures the individual nonlinear effects of each knob under their respective transformations: $1/f$ encodes the hyperbolic decay of latency with frequency, b captures the superlinear growth with batch size, and $\log(a)$ encodes the sharp saturating drop with allocation size. Index i denotes the polynomial degree, $i \in \{1, 2, 3\}$, capturing linear, quadratic, and cubic effects of each transformed knob. Each coefficient $\beta_{i,f}$, $\beta_{i,b}$, and $\beta_{i,a}$ modulates model-specific sensitivity to each knob at degree i . The final term captures pairwise and three-way interactions between knobs. Indices i, j, k denote the degrees of $1/f$, b , and $\log a$ in each interaction monomial, with $i + j + k \leq 3$ to maintain a cubic model and at least two of i, j, k nonzero such that the second

Table 1. Model descriptions. See §D.1.

Model	Description	Params	SLO
ResNet50	Image classification	25.6M	47.6 ms
MobileNetV3	Image classification	5.5M	26.0 ms
VGG11	Image classification	132.9M	12.5 ms
ResNet152	Image classification	60.2M	128.1 ms
DenseNet169	Image classification	14.1M	93.7 ms
DenseNet161	Image classification	28.7M	97.3 ms
VGG19	Image classification	143.7M	17.5 ms
Whisper	Speech2Text	244M	5.51 s
GPT-J-6B	Text generation	6B	12.4 s
SDXL-1024	Image generation	3.5B	25.4 s
SDXL-256	Image generation	3.5B	8.28 s

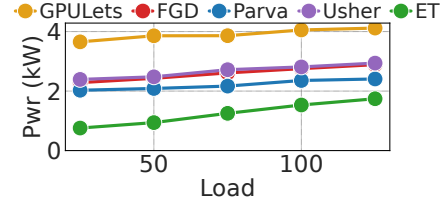


Figure 22. ET reduces maximum cluster power draw by 1.4–4.8× (2.6× average) across loads. Load 100% = 20K RPS. See §D.2.

term does not double-count the individual impact of knobs already captured by the first term.

Similar to power draw, we fit β coefficients per model. We use the same subset of configurations as those used when sampling power draw. Then, we apply Ordinary Least Squares [93].

Deriving throughput from latency estimations. Rather than modeling throughput directly, we derive it from our latency estimations. Throughput is defined as the number of requests completed per unit time; given a batch of b requests processed in $L(f, b, a)$ seconds (Equation 8), throughput follows as:

$$T(f, b, a) = \frac{b}{L(f, b, a)}. \quad (9)$$

While we could construct a separate analytical model for throughput, this simple derivation achieves high estimation accuracy (§D.4), making additional modeling unnecessary.

D Further Evaluation

D.1 Table of Model Information

In our evaluation, we use 11 models, each with two instances. Table 1 describes the models, including the description, number of parameters per model, and SLOs. We note that for SDXL, we use 50 denoising steps. Moreover, for GPT-J-6B, we assume a max output length, like prior work (AlpaServe [80], μ Serve [112], Orion [126]), of 1024 tokens. Finally, for CNN models, we use the ImageNet [32] dataset for image classification. For Whisper, we use the LibriSpeech [110] dataset. For GPT-J-6B, we use the WikiText dataset [88].

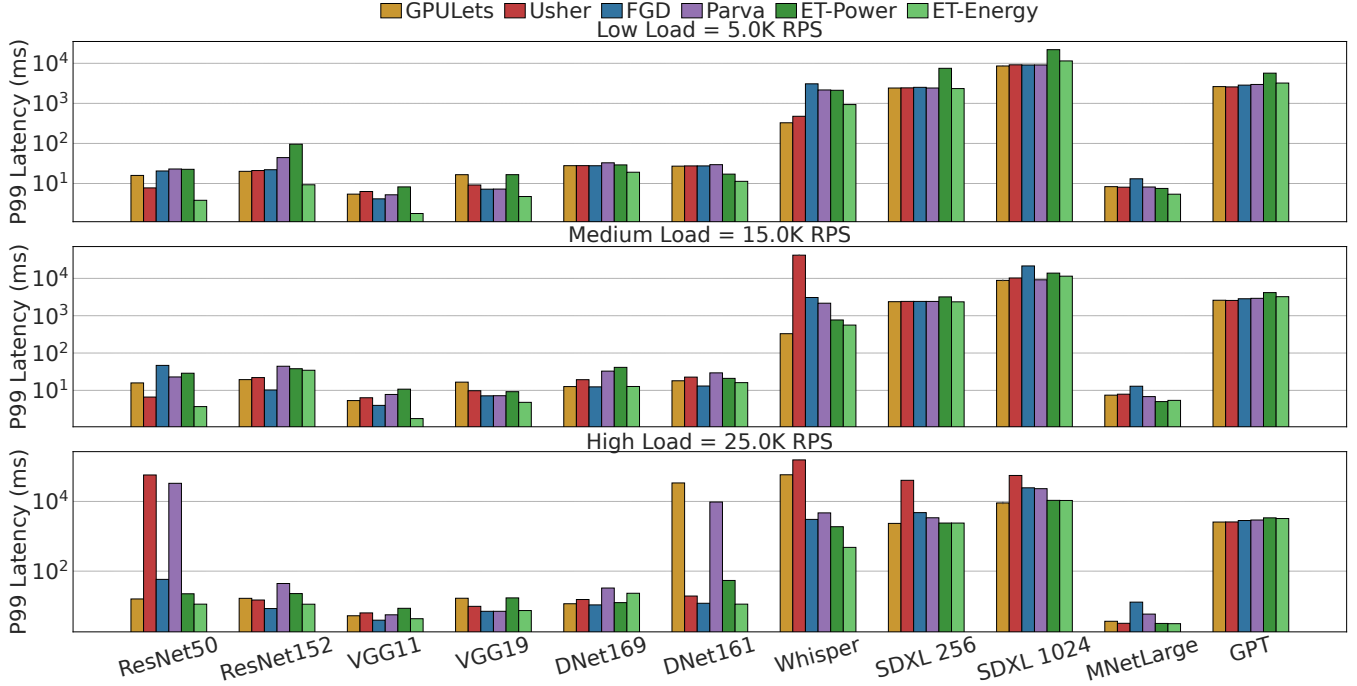


Figure 23. P99 latency of each model across low, medium, and high cluster load. We include the latency for ET when it optimizes for power and energy. See §D.3.

D.2 Comparison of Max Power Draw

In §7.1, we show that across load, EnerTune reduces average GPU power draw by 1.3–2.6 $\times$ compared to the baselines while meeting latency SLOs. Here, we show that EnerTune also reduces maximum cluster power draw by 1.4–4.8 $\times$ (2.6 $\times$ average) across loads compared to the baselines (Figure 22). This is because EnerTune makes power-aware placement, allocation, frequency tuning, and batch size decisions while meeting latency SLOs.

D.3 Comparison of Per-Model Latency

In our E2E evaluation (§7.1), we show that EnerTune consistently meets latency SLOs. Here, we detail the latency of each model under low, medium, and high load (5K–25K RPS). Figure 23 shows that across loads, EnerTune maintains latency similar to the baselines. ET-Power slightly raises latency, compared to the baselines and ET-Energy, because it opportunistically increases packing and reduces GPU frequency to reduce power draw. However, we continue to meet SLOs. A few of the baselines (GPUlets, Usher) at high load greatly inflate latency due to poor packing decisions, thereby inflating resource contention.

D.4 Latency and Throughput Estimation Accuracy

Figure 24 presents the latency and throughput estimation error of our formulations described in §C. Across models under 400+ deployment configurations, our estimations remain highly accurate, with maximum error 8%.

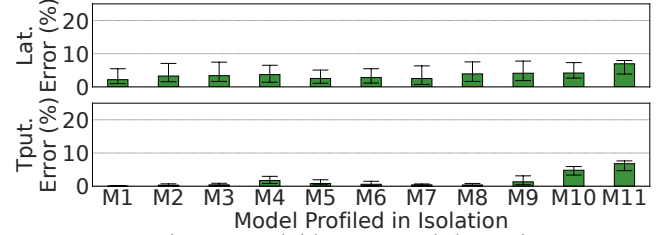


Figure 24. ET's per-model latency and throughput estimation accuracy. See §D.4.

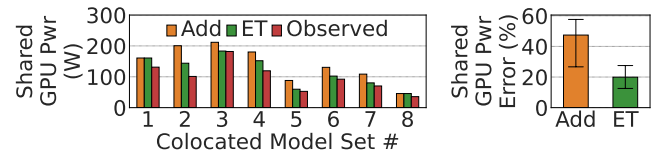


Figure 25. ET's power estimation error of shared GPUs spatially multiplexed with MPS. See §D.5.

D.5 Shared GPU Power Estimation Error with MPS

In §7.4, we showed EnerTune generalizes across GPU types with eight A6000s and RTX 4090s. We used MPS to multiplex GPUs since MIG is not offered on those GPU types, and we showed that E2E, EnerTune reduces energy consumption by 1.56–2.12 $\times$ while meeting latency SLOs. Here, we show the accuracy of EnerTune's estimations of the power draw of shared GPUs when using MPS for multiplexing. Figure 25 shows MAPE error of 20%, with maximum error up to 28%. This is higher than our error under MIG (6.3%, §7.3); MPS is known to exhibit interference across colocated

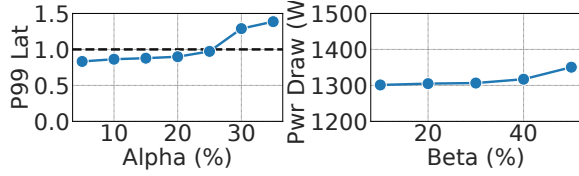


Figure 26. Sensitivity study to α and β , which trigger the Runtime GPU Tuner to dynamically adjust GPU frequency and batch size when model load fluctuates. See §D.6.

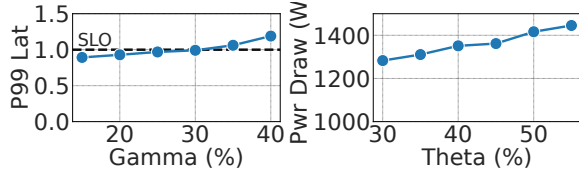


Figure 27. Sensitivity study to γ and θ which trigger the EPA Resource Planner to regenerate execution plans when cluster load significantly fluctuates. See §D.6.

models [27, 77, 126, 161, 164], making accurate shared GPU power estimation more challenging. Despite this, our estimates are significantly more accurate than state-of-the-art approaches [61]. Lastly, using these estimates, we show that EnerTune minimizes energy consumption across the state-of-the-art serving systems (Figure 17, §7.4). We leave improving the accuracy of the power draw of shared GPUs leveraging MPS as future work.

D.6 Load Hyperparameter Sensitivity

EnerTune’s Runtime GPU Tuner adjusts GPU frequency and batch size when a model’s load increases by $\alpha\%$ or decreases by $\beta\%$ relative to its current load. Larger α values delay adjustments and increase SLO violation risk. In contrast, SLO violations are largely insensitive to β , since it only triggers adjustments when load decreases; instead, β primarily governs power savings, with larger values making the Tuner more conservative. Via a grid search (5–50%, increments of 5%) with 3 models deployed on a single GPU and load varying between 50–100%, Figure 26 shows that $\alpha = 5$ –20% maintains SLO compliance, while $\beta = 10$ –40% maximizes power savings.

Following prior work [23, 80, 112, 117], if cluster load drastically deviates $> \gamma\%$ or $< \theta\%$, we escalate to the EPA Resource Planner to readjust the execution plan. While the Runtime GPU Tuners may meet SLOs with local adjustments, drastic load shifts can render current placement/allocation decisions suboptimal. Larger γ values delay adjustments and increase SLO violation risk. In contrast, SLO violations are largely insensitive to θ , since it only triggers adjustments when load decreases; instead, θ primarily governs power savings, with larger values making adjustments more conservative. Via a grid search (15–50%, increments of 5%) with five models deployed across two GPUs and load varying

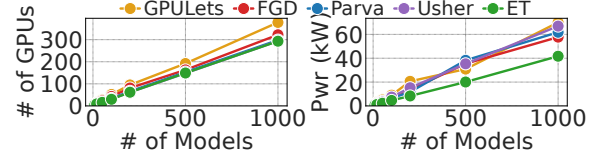


Figure 28. ET scales as the number of models to serve in the cluster grows. See §D.7.

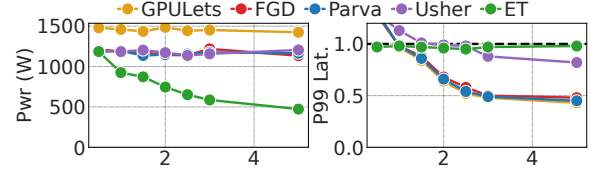


Figure 29. ET is robust to different SLOs. See §D.8.

between 100–150%, Figure 27 shows that $\gamma = 15$ –30% maintains SLO compliance, while $\theta = 30$ –45% maximizes power savings.

D.7 EnerTune’s Scalability

In §7.4, we describe that EnerTune scales as the number of models to deploy in the cluster grows from 10 to 1000 models. We analyzed the number of GPUs used and the power draw of each system. Here, we present the data. Figure 28 shows that EnerTune matches the next best baselines (Usher, ParvaGPU) in GPUs used (e.g., 148–150 GPUs for 500 models, 290–294 for 1000 models). However, we reduce power draw by 1.98–2.67 $\times$.

D.8 Robustness to SLO Scales

In §7.4, we show that EnerTune is robust to SLOs by analyzing its energy consumption under looser and tighter SLOs. Here, we present a similar analysis, but for power draw. Figure 29 shows that across SLOs, EnerTune reduces power draw by 2.6 $\times$ on average without violating SLOs. Unlike energy consumption, EnerTune opportunistically reduces GPU frequency and packs more models, thereby just meeting SLOs while greatly reducing power draw.