CATS: Correlation-aware Task Scheduling for GPU Power Optimization in AI Data Centers

Srinivasan Subramaniyan
Electrical and Computer Engineering
The Ohio State University
Columbus, Ohio, USA
subramaniyan.4@buckeyemail.osu.edu

Xiaorui Wang
Electrical and Computer Engineering
The Ohio State University
Columbus, Ohio, USA
wang.3596@osu.edu

Abstract

GPUs have been increasingly deployed in the data centers of big IT companies to enhance their AI/ML infrastructures. Since GPUs typically consume significantly more power than CPUs, it is crucial to optimize the power consumption of GPU data centers. Task consolidation has been demonstrated to be an effective way to reduce GPU power consumption by consolidating ML tasks onto a smaller set of GPUs and putting unused GPUs and servers into sleep. Unfortunately, existing work on GPU sharing and consolidation assumes that the GPU utilization of each ML task can be approximated as a constant during consolidation. This is in contrast to our analysis of real-world traces, which shows the GPU utilization of ML workloads fluctuates significantly over time.

In this paper, we propose **CATS**, a novel power optimization framework designed for GPU data centers. CATS features *correlation-aware scheduling* to analyze GPU utilization patterns across different ML tasks. By consolidating tasks with negatively correlated GPU utilization on the same GPUs, CATS reduces the number of active GPUs required and puts unused GPUs to sleep, thereby lowering idle power consumption without significantly increasing the job completion time (JCT) of the ML tasks. Additionally, CATS integrates *dynamic frequency scaling* to reduce the dynamic power consumption of active GPUs. We formulate correlation-aware ML task consolidation as a constrained optimization problem with a costly optimal solution. We then design a light-weight heuristic algorithm that is practical for real data centers. Our experiments, conducted on a hardware testbed and through extensive simulations, demonstrate that CATS reduces GPU power consumption by up to 24.7% compared to the baseline, while maintaining comparable JCTs.

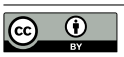
CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Cloud computing**.

Keywords

GPU Scheduling, task correlation, power optimization, machine learning training, AI data center

ACM Reference Format:
Srinivasan Subramaniyan and Xiaorui Wang. 2026. CATS: Correlation-aware Task Scheduling for GPU Power Optimization in AI Data Centers.



ICS '26, Belfast, United Kingdom

ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3800546

In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3797905.3800546

1 Introduction

The recent success of AI/ML development has resulted in a high demand for Graphics Processing Units (GPUs). As an arms race, major IT companies (e.g., OpenAI, Meta, Microsoft, Google) are all rushing to purchase/build more GPU chips to significantly enhance their AI/ML infrastructures [1]. Hence, it can be well anticipated that the number of GPUs used in the data centers of those Big Tech companies will increase sharply in the next few years. Although GPUs are generally more energy-efficient than CPUs, a GPU can consume significantly more power than a CPU due to their higher density of memory and computing units that operate in parallel. For example, the power consumption of a high-end GPU used for ML Training (e.g., Nvidia H100) can be as high as 700 Watts [49]. Based on Nvidia's estimate, approximately 1.5 to 2 million H100 GPUs were sold in 2024. Thus, the total power consumption of just those H100 GPUs can reach 13.1 TWh annually [34], which, if compared to residential power consumption of large cities in the US, would rank as the fifth-largest residential power consumer and is higher than the entire city of Phoenix, AZ [56]. Therefore, it is crucial to optimize the power consumption of these massive GPU data centers used for AI/ML development.

Most existing GPU power optimization research relies on offline profiling of the target ML applications on a specific GPU to identify the most energy-efficient scheduling (e.g., [3, 52]). However, they may incur high profiling overheads and cannot adapt to online changes in program behaviors. Some studies aim to modify the internal mechanisms of specific ML models to improve energy efficiency [47], but these approaches may not apply to all ML applications. Online system-level GPU power optimization has been proposed for ML inference [9, 33, 35, 37, 51, 53] and training tasks [61, 65], respectively. However, they mainly depend on GPU frequency scaling to achieve the desired power-performance trade-offs. While frequency scaling can provide prompt GPU power management, it cannot eliminate GPU idle power consumption, which can be significant for many GPUs, and thus achieves only inferior power optimization results.

A well-known approach to minimizing idle power consumption in data centers is to consolidate workloads onto fewer computational resources, such as GPUs or CPUs, to optimize resource utilization [27][69]. Then, unused computational resources can be turned off or put into sleep [27][43], reducing the data center's power consumption. For example, task consolidation for data center server power optimization has been studied before to make servers

more energy-proportional to their workloads [26, 64]. Likewise, the traffic flows in a data center network can also be consolidated to put unused network devices into sleep for energy savings [24]. For GPU data centers, recent research has proposed various methods for packing more ML tasks onto fewer GPUs [5, 15, 74] with the goal of increasing GPU utilization, because GPUs in production data centers are reported to be severely underutilized [70]. With such GPU-sharing or consolidation methods, unused GPUs can be powered down or placed in sleep mode to minimize their idle power consumption. Additionally, when no GPUs in a server are active, we can further reduce idle power consumption by putting unused servers to sleep [27][43].

Unfortunately, existing GPU consolidation solutions assume that the GPU utilization of each ML task can be approximated as a constant (e.g., by taking either the peak or average value) during the consolidation process. Such a simplistic assumption may not accurately capture the dynamic resource demands of ML tasks. For example, our analysis of the real-world traces from Microsoft (Philly [31]) reveals that the GPU utilization of ML workloads fluctuates significantly over time, with usage levels varying throughout the day. Therefore, conservatively using the peak values for GPU consolidation can result in an unnecessarily higher number of active GPUs and thus more power consumption, while using the average values may cause some co-located ML tasks to peak at the same time and therefore have longer Job Completion Times (JCTs) due to undesired GPU resource competition. In sharp contrast to existing solutions, we argue if the correlations among ML tasks are considered in GPU consolidation, more power savings can be achieved for GPU data centers. In particular, compared with ML inference tasks that last only milliseconds to seconds, ML training tasks are better candidates for correlation analysis, because the consolidation period cannot be too short due to overhead considerations [64].

In this paper, we propose CATS, a power optimization framework specifically designed for GPU data centers that run ML training tasks. CATS leverages *correlation-aware scheduling*, which analyzes GPU utilization patterns among various ML tasks to efficiently consolidate them onto fewer GPUs. By scheduling tasks with complementary resource demands, we can fully utilize active GPUs and put unused GPUs to sleep, thereby saving idle power. CATS is also integrated with dynamic GPU frequency scaling to maximize dynamic power savings without much impact on the JCTs of ML training tasks. CATS focuses on scheduling training tasks, since inference tasks typically have very short durations, usually in milliseconds [8][4][58]. Therefore, we focus on scheduling **training tasks** with longer durations [63][62] and provide multiple data points for correlation analysis. To this end, we first model the impact of frequency scaling on GPU power consumption and task JCTs. We then formulate GPU consolidation as a constrained optimization problem, known to be NP-hard. Then, we derive an *optimal* solution that determines the best consolidation strategy. To significantly reduce computational overhead, we design a heuristic algorithm to efficiently consolidate **ML training tasks** based on the analyzed correlation of their GPU utilization. Finally, we discuss practical considerations for implementing CATS in real-world data centers, including scalability, GPU utilization prediction, and integration with existing infrastructure.

The major contributions of this paper are as follows:

- We examine real-world traces and observe that ML tasks generally exhibit weak pairwise correlations and do not peak simultaneously. Leveraging this insight, we propose correlation-aware scheduling that strategically consolidates ML training tasks to minimize the number of required GPUs, thereby reducing overall power consumption.
- We formulate ML task consolidation as a constrained optimization problem. Given the NP-hard nature of this problem, we mathematically derive an *optimal* solution. Although this solution is computationally intensive and impractical for real-time data center operations, it serves as an upper bound to guide the development of efficient heuristic algorithms.
- We validate the effectiveness of CATS through experiments on our GPU hardware testbed and trace-driven simulations. Our testbed results demonstrate a substantial reduction in the number of required GPUs. Furthermore, simulation evaluations show that CATS outperforms several baseline solutions, achieving up to an 24.7% decrease in power consumption.

The remainder of this paper is structured as follows: In Section 2, we formalize the optimization problem. Section 3 motivates utilization-based correlation and details the design of the CATS framework. Section 4 describes the experimental setup. Hardware and simulation evaluations are covered in Sections 5 and 6, respectively. Section 7 discusses the related work. Finally, Section 8 summarizes our conclusions.

2 Problem Formulation

In this section, we develop a model to analyze how GPU frequency scaling affects the power consumption and JCT of ML training tasks. We then propose an *optimal* solution to optimize data center power consumption without degrading performance. Finally, we discuss the practical challenges associated with computing this optimal solution.

2.1 Modeling

2.1.1 Power Model. We first introduce the power model and then formulate the power optimization as a constrained problem. We use the following notations:

- D: Datacenter.
- N: Total number of servers in D.
- w_i: Total number of GPUs on server i.
- G_{ij}: The jth GPU on the ith server.
- P_{gs}: Per-GPU static power (idle) when a GPU is powered on (W).
- α: Linear coefficient of dynamic power vs. frequency (W/MHz).
- f_{ij}: SM (Streaming Multiprocessor) frequency of GPU j running on server i (MHz).
- P_i^{static}: Server i's static power draw when the server is on (independent of GPU load) (W).
- c_{ij}^{mem}: DRAM capacity of GPU j on server i.
- L: Total number of tasks.
- d_l: Memory consumption of task t_l.
- v_l: Number of GPUs required by task t_l.
- κ_l: Maximum allowed Job Completion Time (JCT) extension factor for task l (e.g., $\kappa_l = 1.2$ for a 20% tolerance).
- $x_{lij} \in \{0, 1\}$: 1 if task l is assigned to GPU j on server i, 0 otherwise.

- $y_{ij} \in \{0, 1\}$: 1 if GPU j on server i is active, 0 otherwise.
- $z_i \in \{0, 1\}$: 1 if server i is active, 0 otherwise.

The power consumption of a GPU consists of both static (idle) and dynamic components. The static power consumption is the baseline power drawn by the GPU when idle, including components such as the power supply and fans. The dynamic power consumption varies with workload and is directly proportional to the GPU's operating frequency. Thus, the total power consumption of a GPU is represented in Equation (1), where P_{gpu} is the total power consumption in Watts. P_{gs} represents the static power consumption (obtained from measurements of the GPU's power draw at idle), α is the coefficient for linear dynamic power contribution, and f is the GPU's operational frequency in Megahertz.

$$P_{\text{gpu}} = P_{gs} + \alpha f. \tag{1}$$

For a datacenter (D) comprising a homogeneous GPU cluster consisting of N servers, where each server has w_i GPUs, the total power consumption is shown in Equation (2). To derive the linear coefficient in Equation (2), we characterize (P_{gs}, α) for our NVIDIA V100 GPUs by running ML training workloads, specifically *ResNet50* and *VGG16* with a batch size of 60, on frequencies ranging from 135 MHz to 1350 MHz. For each frequency setting, we measure power consumption per training epoch and compute the average across workloads. The GPU power model is expressed in Equation (3). In this model, 23.3 W represents the static power consumption, obtained from measurements of the GPU's idle power draw, while 0.09 W/MHz is the coefficient that captures the linear contribution of dynamic power. This model achieves a high degree of accuracy, with an R^2 score of approximately 0.95, as illustrated in Figure 1a.

$$P_{\text{gpu,total}} = \sum_{i=1}^{N} \sum_{j=1}^{w_i} y_{ij} \left(P_{gs} + \alpha f_{ij}\right). \tag{2}$$

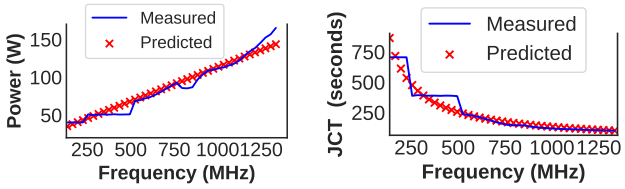
$$P_{\text{gpu,total}} = \sum_{i=1}^{N} \sum_{j=1}^{w_i} y_{ij} \left(23.3 + 0.09 f_{ij}\right). \tag{3}$$

A typical server used for ML training features multiple GPUs (often 1–8 [8, 30, 32]). The total *server* power for server i includes its static base draw when active, the sum of its GPU power, and the power drawn from other components as shown in Equation (4).

$$P_{\text{server},i} = \begin{cases} P_i^{\text{static}} + \sum_{j=1}^{w_i} P_{\text{gpu},ij}(f_{ij}) + P_{other} & \text{if } z_i = 1, \\ 0, & \text{if } z_i = 0 \end{cases} \tag{4}$$

In Equation (4), P_{other} captures the non-GPU power components, including CPU power, memory power, chipset power, and power consumption of the other components. Since ML training tasks predominantly use GPUs, GPUs are the primary contributors to the overall power consumption.

In cloud services, providers often anticipate user requests for machine learning services. To process these requests promptly and satisfy service-level agreements (SLAs), providers keep GPUs active and operate them at their maximum frequency to avoid latency violations [52]. However, keeping GPUs continuously powered on to meet latency requirements results in significant idle power consumption when no requests are being processed. For example, on an $A100$-PCIE-40GB (250W TDP), the idle power draw for the smallest compute slice is already around $40 - 50$ W. For an $H100$-PCIE-80GB (350W TDP), the idle power draw for the smallest slice is even higher, at approximately 70-80 W [29]. Moreover, the idle power percentage will increase for future GPUs. Therefore, when GPUs are not needed to serve active tasks, unused GPUs can be transitioned into low-power sleep states to reduce idle power consumption [27]. In addition, when no tasks are assigned to any GPU on the server (i.e., $z_i = 0$), the server can be powered off or put to sleep to further reduce its static power consumption.



(a) Power consumption **(b) Job completion time**

Figure 1: Measured and predicted GPU power consumption and job completion time (JCT) under different SM frequencies: (a) power consumption and (b) JCT.

2.1.2 Latency Model. Training tasks do not have strict Service Level Agreements (SLAs) as inference tasks [8]. However, training latency cannot be extended indefinitely. The GPU frequency affects training latency [22]: reducing it can reduce power consumption but may prolong the JCT [18, 45]. Gao et al. [18] conducted a survey involving 103 researchers to assess their tolerance for deadline extensions in Training tasks. The findings revealed that approximately 3%, 12%, 21%, and 22% of respondents were willing to accept deadline extensions of 0%, 5%, 10%, and 20%, respectively. In this paper, we establish a 20% threshold for determining SLA violations. To understand the impact of frequency scaling on JCT, we analyze how the GPU frequency affects JCT. By varying the frequency of the GPU as described in Section 2.1.1, we measure the average JCT between workloads in Section 2.1.1 and fit a curve to model the relationship between the two as shown in Equation (5).

$$e_l(f) = e_{\min,l} \times \left(\frac{f_{\max}}{f}\right)^{\beta} \tag{5}$$

Through curve fitting, we obtain $\beta = 0.91$, which captures the nonlinear scaling behavior and yields $R^2 \approx 0.97$ (Fig. 1b).

2.2 Problem Formulation

Deep learning training clusters are typically constrained by the *power budget* of the data center rather than the total energy consumed by individual tasks. Consequently, our objective is to minimize the *instantaneous* power draw while ensuring that each training task does not exceed its SLA. We emphasize that this work optimizes *power* (Watts), because instantaneous power directly affects the rack-level electrical provisioning, cooling capacity, and cluster-wide admission control.

The optimization problem is shown in Equation (6).

$$\min \sum_{i=1}^{N} P_{\text{server},i} \tag{6}$$

$$\text{s.t.} \sum_{i=1}^{N} \sum_{j=1}^{w_i} x_{lij} = v_l, \qquad \forall l, \tag{7}$$

$$\sum_{l=1}^{L} d_l \, x_{lij} \le c_{ij}^{\text{mem}} \, y_{ij}, \qquad \forall i, j, \tag{8}$$

$$x_{lij} \le y_{ij}, \qquad \forall l, i, j, \tag{9}$$

$$z_i \ge y_{ij}, \qquad \forall i, j, \tag{10}$$

$$z_i \le \sum_{j=1}^{w_i} y_{ij}, \qquad \forall i, \tag{11}$$

$$x_{lij},\ y_{ij},\ z_i \in \{0, 1\}, \qquad \forall l, i, j \tag{12}$$

$$f_{\min} \, y_{ij} \le f_{ij} \le f_{\max} \, y_{ij}, \qquad \forall i, j, \tag{13}$$

$$f_{ij} \ge f_{\max} \, \kappa_l^{-1/\beta} - M(1 - x_{lij}), \qquad \forall l, i, j \tag{14}$$

where $M \ge f_{\max}$.

Consolidating tasks onto fewer GPUs reduces the number of active GPUs (i.e., those with $y_{ij} = 1$), which in turn enables more servers to be powered down/put to sleep (so $z_i = 0$). Constraint (7) denotes the distributive constraint used to determine the number of GPUs assigned to a training task. If $v > 1$, distributed training is performed. Equation (8) ensures that the task's memory consumption does not exceed the DRAM capacity of the GPU. Constraints (9), (10), (11) represent the activation constraints. Constraint (13) specifies the operating frequency range of the GPU. Constraint (14) enforces the SLA by ensuring $e_l(f_{ij}) \le \kappa_l e_{\min,l}$, where M ($\ge f_{\max}$) is a large constant used to deactivate the constraint when task l is not assigned to GPU G_{ij}. The SLA constraint introduced in Equation (14) ensures that frequency scaling does not degrade training performance beyond the specified tolerance.

Computational Complexity and Solver Limitations: We use PuLP [44] to obtain the optimal solution. The solver's tolerance is set to 1×10^{-6}. Because the power minimization formulation is a mixed-integer linear program (MILP) with binary placement and activation variables, the problem is NP-hard. Consequently, the number of decision variables and constraints increases rapidly with the number of tasks (L), resulting in exponential growth in solver runtime, as shown in Table 1.

Table 1: Computation Time for Different Numbers of Tasks

Tasks	2	4	8	16	32	64	128
Solve Time (s)	0.02	0.12	0.24	0.66	4.7	16.5	80.9

To overcome this, we propose a lightweight heuristic for online task placement that achieves near-optimal power savings with significantly reduced computational overhead.

3 Design of CATS

In this section, we begin by discussing GPU utilization and analyzing the correlations in GPU utilization across ML training tasks. We then discuss the consolidation analysis of ML tasks. Building on this analysis, we detail the design and functional components of the CATS framework. We then discuss the practical considerations associated with CATS.

3.1 GPU Utilization

There are different granularities of utilization in GPUs. The GPU utilization measured by the `nvml` interface represents the percentage of time over the past sample period during which one or more kernels are executing on the GPU [13]. At the streaming multiprocessor (SM) [14] level, utilization measures the distribution and execution efficiency of thread blocks across SMs. At the instruction level, utilization is calculated as the ratio of successfully issued warp instructions to stalled cycles, indicating the warp scheduler's effectiveness. Finally, tensor core utilization focuses on the performance of specialized cores for matrix operations, evaluating the dominance and throughput of tensor core instructions. Although finer-grained utilization is useful for application tuning, it incurs significant profiling overhead. In contrast, GPU and SM-level utilization is relatively easy to obtain and is therefore widely used in data center schedulers [14]. To obtain temporal GPU utilization information for our analysis, we collect periodic utilization samples using the NVIDIA Management Library (`nvml`). Specifically, utilization is sampled asynchronously by a dedicated background thread that invokes `nvml` APIs at regular intervals. This approach provides low-overhead access to GPU utilization without interfering with the training execution. While `nvidia-smi` can also be used to query the same `nvml` interfaces, running it as a subprocess adds unnecessary overhead; therefore, our implementation interacts directly with `nvml` in-process. Although more advanced profilers such as NVIDIA Nsight Systems (nsys) can capture extremely fine-grained information (e.g., kernel-level traces, memory transactions, and tensor-core activity), they introduce substantial runtime overhead and are not suitable for continuous or online monitoring in production training clusters. Our coarse-grained, low-overhead sampling is consistent with how GPU utilization is monitored in real-world cloud environments.

GPU utilization prediction has been extensively studied before [2, 11, 19, 27, 50, 73]. For example, some research has achieved a low prediction error, with a Root Mean Squared Logarithmic Error (RMSLE) of 0.154 [73]. Since utilization prediction is *not* the focus of this work, we assume that GPU utilization can be predicted using such existing techniques. It is also important to note that CATS can still work effectively even when the predicted GPU utilization is inaccurate (See Section 5). While task memory usage may vary with batch size or degree of parallelism, both parameters are available at the submission time, and the scheduler can account for them when estimating memory demand. Since CATS relies on coarse-grained correlation trends rather than exact point predictions, moderate deviations do not significantly change scheduling decisions. Inaccurate predictions may slightly increase latency or reduce consolidation, while remaining superior to correlation-agnostic baselines.

3.2 Utilization Correlation Analysis

To motivate the design methodology of CATS in the subsequent sections, we present our analysis of the Philly trace from Microsoft's

data center [31]. The trace includes training tasks from production groups that develop products using models for image classification, speech recognition, and other applications. The trace is collected from clusters equipped primarily with Nvidia $P-100$ GPUs. We select 600 tasks from the middle of the trace. We map each task's machine ID in the job trace file to the corresponding machine IDs in the GPU utilization trace file. We calculate the aggregate GPU utilization for each job by summing utilization across all GPUs assigned to the task for each minute of execution. For the selected 600 tasks, we compute the correlation coefficient from the utilization values to quantify the similarity in GPU utilization patterns. To quantify the correlations among tasks, we compute the Pearson correlation coefficient for each pair of tasks, as shown in Equation (15). The correlation coefficient for two distributions, $x = \{x_1, \ldots, x_n\}$ and $y = \{y_1, \ldots, y_n\}$, is given by:

$$\rho(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{15}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the sample means of x and y, respectively. The correlation coefficient ρ_{xy} usually ranges from -1 to 1. We plot the Cumulative Distribution Function (CDF) of the correlation coefficient for all pairs of tasks in the *Philly trace*, as shown in Figure 2. From this figure, we observe that 46% of task pairs exhibit negative correlations, indicating that tasks are negatively correlated. Meanwhile, 34% fall between 0 and 0.3, suggesting a weak positive correlation. Overall, approximately 80% of task pairs exhibit correlations $|r_{xy}| \leq 0.3$, indicating that most task pairs are only weakly correlated. Only a small fraction of task pairs show stronger correlations, with $r_{xy} > 0.3$.

Observation 1: The GPU utilization of most training tasks is loosely correlated, so they usually do not peak at the same time.

It is important to note that the correlation values reported here are computed from utilization samples extracted from the trace. Consequently, the metric captures *coarse-grained temporal trends* rather than fine-grained variations at the level of kernels, warps, caches, or memory-access behavior. As shown in later subsections, this coarse-grained signal is sufficient to guide consolidation decisions that reduce power consumption without significantly degrading JCT. Since the correlation analysis above was performed on 600 tasks, it is essential to quantify the computational overhead of computing correlation coefficients, particularly as the sample size increases. To evaluate this cost, we measure the time required to compute the Pearson correlation for vectors of different lengths. As shown in Table 2, the computation of correlations incurs minimal overhead. Even with 1 million samples, the runtime is under 13 ms. This confirms that correlation analysis imposes negligible overhead and can be performed online without affecting scheduling responsiveness.

Table 2: Overhead for Correlation Computation

Samples	1K	10K	100K	500K	1000K
Time (ms)	0.295	0.325	1.325	6.469	12.999

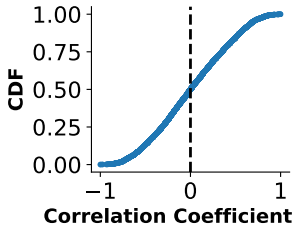


Figure 2: The CDF of the correlation coefficient for the 600 tasks chosen from the Philly trace.

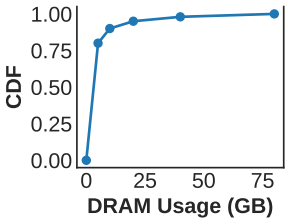


Figure 3: The CDF of DRAM usage for the 600 selected tasks from the GPUs in the PAI trace.

3.3 Consolidation Analysis

Several works discuss multiplexing approaches, such as GPUlet [12], IADeep [10], and Orion [57], which explore task consolidation on the same GPU. However, *all of these focus on inference tasks*. However, a recent study analyzes GPU workload consolidation for deep learning training tasks on NVIDIA GPUs [55]. It shows that for small and medium-sized training jobs that underutilize a GPU, running multiple jobs concurrently (e.g., via MPS or MIG) can significantly increase overall training throughput, in some cases by up to 3×. To determine whether it is feasible to schedule multiple DNN workloads on the same GPU, we analyze real-world traces (the PAI trace [66]). This analysis serves two main purposes: first, it assesses whether GPUs have enough DRAM capacity to accommodate the tasks; second, it verifies that tasks can be scheduled on the same GPU without exceeding that DRAM capacity. This step is crucial because the correlation analysis focuses on scheduling multiple tasks or layers to a single GPU.

The PAI trace from Alibaba [66] captures a mix of ML training and inference jobs executed on a large-scale cluster comprising over 6,500 GPUs during July and August 2020. Each job in the trace is associated with a unique identifier, start time, duration, requested GPU memory, requested GPU utilization, and job status. We preprocess the trace by applying the following filters: (1) we exclude jobs marked as failed, (2) we remove jobs with zero GPU memory or GPU utilization, and (3) we filter out jobs with execution times shorter than 20 seconds, as these typically correspond to inference workloads [8].

Figure 3 presents the CDF of DRAM usage, showing that most workloads consume significantly less than 16 GB, as indicated by the steep increase in the CDF. Additionally, most workloads require less than 32 GB. Therefore, scheduling multiple workloads on the same GPU is feasible when the aggregate working sets (including short-term peaks) fit within a GPU with 32 GB or more of DRAM, with sufficient headroom to avoid allocator fragmentation and framework/runtime reservations.

Observation 2: GPU memory consumption is fragmented, providing an opportunity to consolidate tasks.

Consolidating multiple tasks can significantly impact the JCT of each task [8, 14, 16, 57, 60, 73]. Hence, we need to ensure that consolidation does not exceed the task's SLA [18]. For example, Orion [57] suggests that consolidating a compute-oriented task with a memory-bound task reduces contention. Similarly, Yeung

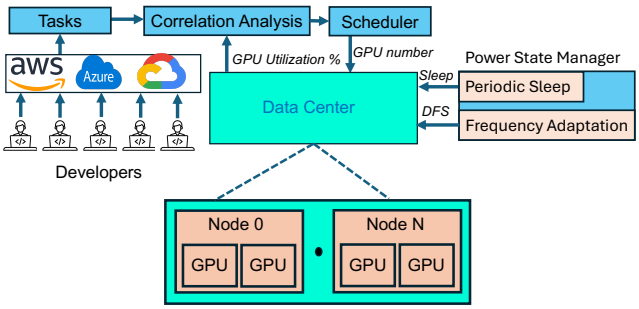


Figure 4: Architecture of the proposed CATS framework. The framework comprises correlation analysis, a scheduler, and a power-state manager that performs periodic sleep and frequency adaptation.

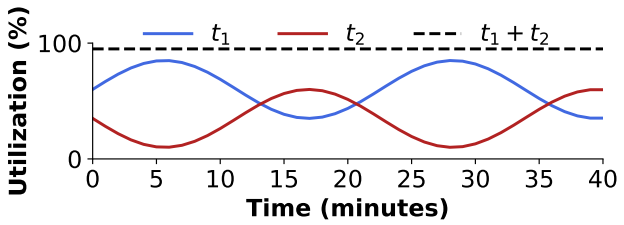


Figure 5: The two tasks are negatively correlated, and the aggregated utilization is less than 100%.

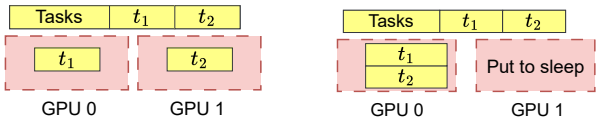
et al. [73] propose proactively predicting GPU utilization before scheduling so that jobs are consolidated only when their combined utilization is unlikely to exceed GPU capacity. Motivated by our empirical analysis of GPU utilization and memory usage in production traces, we design CATS to favor consolidation of tasks with complementary utilization patterns. Specifically, CATS consolidates tasks only when (1) the aggregate DRAM footprint of the consolidated tasks fits within the capacity of a single GPU, and (2) their GPU utilization patterns are complementary such that their resource demands do not peak simultaneously. By consolidating tasks with non-overlapping utilization phases, CATS mitigates resource contention and avoids excessive increases in JCT. In the next subsection, we present our CATS framework.

3.4 CATS Framework

The design of our proposed CATS framework is illustrated in Figure 4. CATS includes a *correlation analysis* module that performs correlation analysis of task GPU utilization. It has a *scheduler* that dispatches tasks to datacenter GPUs. The *power state manager* runs periodically and consists of two modules: a *sleep module*, which puts unused GPUs and servers to sleep, and a *frequency adaptation module*, which dynamically adjusts the frequency of active GPUs.

Our framework has several benefits. (1) It can integrate services based on ML, such as a resource manager aware of burstiness [75] and a network-aware job scheduler [54]. (2) It offers high usability: The proposed scheduling approach can be integrated with systems such as Kubernetes or YARN using existing plugin interfaces or custom schedulers. The algorithm relies solely on task metadata and lightweight computations, allowing it to operate efficiently within control loops. Users do not need to provide extra information or specifications. (3) It incurs low overhead: computation for correlation takes only milliseconds, as discussed in Section 3.2, which is negligible for DL workloads.

We now illustrate a simple example to demonstrate how CATS works. In this example, we consider two training tasks: t_1 with 12 GB of memory and t_2 with 10 GB of memory from the Philly trace [31]. The average GPU utilization of t_1 and t_2 is 52.23% and 42.7%, respectively. The utilization patterns of these tasks are shown in Figure 5. In this example, we consider our testbed setup with 2 GPUs, each having 32 GB of memory. Figure 6a shows the default approach [45][48], where each task is assigned to a separate GPU. We now compare it with CATS. For CATS at time 0, when t_1 arrives, we assign it to GPU 0. At time 1 second, task t_2 arrives. We compute the correlation coefficient ($\rho(t_1, t_2)$ using Equation (15)) between the task already running on GPU 0 and the incoming task t_2, which is -1. Since the correlation coefficient is negative, and their total memory usage (22 GB) is less than the GPU's DRAM capacity (32 GB), we consolidate t_1 and t_2 on GPU 0 as shown in Figure 6b.



(a) Default **(b) CATS**

Figure 6: An example of ML task consolidation by CATS. (a) shows the initial task assignment. (b) shows the aggregated task consolidation after applying CATS. We put the unused GPU (GPU 1), to sleep to reduce power consumption.

While distributing each task to its individual GPU results in the shortest JCT for all tasks, the JCT for our proposed method, CATS, is only slightly longer, as we utilize correlation and the tasks do not contend for resources. For the CATS GPU 1 can be put to sleep to reduce idle power consumption. Additionally, if a server is equipped with only one GPU, as in Alibaba's infrastructure [70], the server can be shut down or put to sleep when the GPU is no longer in use, reducing idle server power consumption. If a server contains two or more GPUs, it can be transitioned to a sleep state only after all GPUs on that server have been put into sleep. We discuss how this is enforced in Section 3.5. In addition, to reduce dynamic power consumption for CATS, GPU 0 can adjust its operating frequency via DFS at runtime as discussed in Section 3.5.

3.5 Correlation-Aware Consolidation Algorithm

We introduced in Section 2 that using mathematical tools to solve the optimal task consolidation is infeasible mainly due to its high computational complexity, as the task consolidation problem is *NP-hard*. We now propose a lightweight heuristic algorithm that achieves correlation-aware consolidation with low computational overhead. The algorithm we designed is based on the bin-packing principle. CATS comprises three key components: correlation analysis (Algorithm 1), scheduler (Algorithm 2), and power state manager (Algorithm 3 and Algorithm 4). We now describe each in detail.

Correlation Analysis. Algorithm 1 performs correlation analysis for each incoming task t_l. In lines 1-4 it begins by initializing an empty buffer $\mathcal{B}$ and iterates over all servers $i \in [1, N]$ and their GPUs G_{ij}, where $j \in [1, w_i]$. If GPU G_{ij} has sufficient memory available (that is, $d_l < c_{ij}^{\text{mem}}$), we compute two metrics (i) the correlation coefficient (δ_g) between the tasks running on the GPU G_{ij} and the task t_l on line 6, (ii) the absolute difference in the GPU utilization (δ_μ) between the incoming tasks and existing tasks running on the GPU on line 7. Before computing δ_g, the utilization values are truncated to the length of the shorter one since the lengths must be the same to compute the correlation as stated in Equation (15).

We perform correlation analysis immediately at job arrival (submission time). We do not require a full initial run; instead, we obtain the utilization trace using existing prediction methods for recurring tasks or, at startup, for new tasks via short-term profiling, as detailed in Section 3.1. We then compute correlations between the arriving task and the currently running tasks and use these correlations to select the target GPU/server. We choose to use absolute utilization as an additional threshold, as occasional capacity exceedance is acceptable. If the combined resource demand of two jobs temporarily exceeds capacity, it may result in a minor delay in execution time, but it will not cause failure or instability. It is beneficial to consolidate tasks with the largest absolute difference in their GPU utilization [20] to reduce contention during consolidation. We add the GPU used, the server used, the correlation coefficient, and the average utilization to the *buffer*. If G_{ij} is not already present in the active GPU tracker $\mathcal{A}$, it is added; similarly, if server i is not present in the server tracker $\mathcal{S}$, it is also added. If the buffer $\mathcal{B}$ is not empty, the scheduler is invoked (line 14) to evaluate each candidate GPU in $\mathcal{B}$ to determine the most suitable placement for task t_l.

Algorithm 1: Correlation Analysis

Input: Incoming task t_l with memory d_l and utilization $u(t_l)$; GPU utilizations $u(G_{ij})$; thresholds β, α; servers $i \in N$; trackers $\mathcal{A}, \mathcal{S}$

1 $\mathcal{B} \leftarrow \emptyset$;
2 **for** *each incoming task* t_ℓ **do**
3 **for** *each server* $i \in [1, N]$ **do**
4 **for** *each GPU* $G_{ij},\ j \in [1, w_i]$ **do**
5 **if** $d_l < c_{ij}^{mem}$ **then**
6 $\delta_g \leftarrow \rho(u(G_{ij}),\ u(t_l))$;
7 $\delta_\mu \leftarrow ||u(G_{ij}) - u(t_l)||$;
8 $\mathcal{B} \leftarrow \mathcal{B} \cup \{(i, j, \delta_\mu, \delta_g)\}$;
9 **if** $G_{ij} \notin \mathcal{A}$ **then**
10 $\mathcal{A} \leftarrow \mathcal{A} \cup \{G_{ij}\}$;
11 **if** $i \notin \mathcal{S}$ **then**
12 $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$;
13 **if** $\mathcal{B} \neq \emptyset$ **then**
14 **Scheduler**$(S, \mathcal{A}, \mathcal{B}, t_l)$;
15 **break**;

Scheduler. Algorithm 2 schedules the incoming task using the candidates in $\mathcal{B}$. In lines 1, 2, the scheduler initializes the candidate list C and sets the weights (λ_1 and λ_2). Currently, we assign equal weights because the scheduler allows it to balance two objectives: maximizing the difference in aggregate utilization (δ_μ) and minimizing correlation (δ_g). Since oversubscribing GPUs can affect JCT [73], we jointly select the candidate with the largest absolute difference in utilization and the lowest correlation to reduce JCT. In lines $3-6$, the scheduler computes the weighted score and stores it in a temporary buffer C. The scheduler then sorts the buffer C by s and schedules the task to the chosen GPU with the lowest score, as shown in lines $(7, 8)$.

Algorithm 2: Scheduler

Input: $\mathcal{B}$: candidate GPU tuples; $\mathcal{S}$: server tracker; $\mathcal{A}$: GPU tracker; incoming task t_l

Output: $\mathcal{K}$: selected GPU

1 $C \leftarrow \emptyset$;
2 Initialize weights (λ_1, λ_2);
3 **for** *each* $(i, j, \delta_\mu, \delta_g) \in \mathcal{B}$ **do**
4 score $\leftarrow \lambda_1 \cdot \delta_g$ - $\lambda_2 \cdot \delta_\mu$
5 $C \leftarrow C \cup \{(i, j, \text{score})\}$;
6 Sort C by score ascending;
7 $(\mathcal{K}, \text{gpu}) \leftarrow$ GPU associated with the smallest score;
8 Dispatch task t_l to GPU $\mathcal{K}$;

Power State Manager. The power state manager in CATS minimizes the average power consumption over the upcoming scheduling window, thereby reducing the total energy expenditure within that interval. Because the JCT is constrained by a service-level agreement (SLA), as specified in Algorithm 4, a reduction in average power consumption translates directly into proportionally lower energy usage. The power state manager comprises two components: (1) the periodic sleep module and (2) the frequency adaptation modules. We describe each module in detail below. The *periodic sleep* module is described in Algorithm 3. This module runs on a background thread and executes at regular intervals of length τ. In this paper, we set τ to 100 ms to match the frequency adjustment overhead of `nvidia-smi` [42]. Synchronizing the sleep interval with the frequency adjustment overhead ensures that both mechanisms operate efficiently without introducing additional delays. The periodic sleep module in Algorithm 3 scans all servers (line 1). For each server, it checks whether all GPUs have no active tasks or are idling. If they are idling, it puts them to sleep (lines 4-7). After scanning all GPUs, if none are active, the entire server is put to sleep (lines 8-10).

The *Frequency Adaptation* module, stated in Algorithm 4, runs in the background at regular intervals of length τ and performs DFS. As discussed in Section 2, lowering GPU frequency reduces power consumption but can increase job completion time (JCT). Accordingly, our goal is to balance performance and power efficiency while keeping execution times within acceptable bounds.

In Algorithm 4, the minimum and maximum frequencies (f_{min} and f_{max}) of the GPU and the SLA factor (γ) for all tasks are the inputs provided when the frequency adaptation module is launched. For each active GPU g, the algorithm iterates over all tasks $\mathcal{T}_g$ currently running on it. In line 8, for every task $t \in \mathcal{T}_g$, we estimate the prediction time of the task based on the model from Section 2. If the predicted time exceeds the maximum allowed time (line 10),

Algorithm 3: Periodic Sleep

Input: Runs every interval τ; servers $i \in [1, N]$ with w_i GPUs each; active server set $\mathcal{S}$

```
for i = 1 to N do
    server_active ← false;
    for j = 1 to w_i do
        if G_ij has running tasks then
            server_active ← true;
        else
            sleep G_ij;
    if server_active = false then
        sleep server i;
        S ← S \ {i};
```

a violation is recorded, and *violation* is set to true. If the predicted time is within 5% of the limit (lines 12-13), then *slack* is set to false. After all tasks are checked, the GPU frequency is updated (lines 14–17). If any task violates its bound, the frequency increases by Δf but is capped at f_{max}. If no violations occur and all tasks have slack, the frequency is reduced by Δf but not below f_{min}. If tasks are close to the limit (within 5% of e_t^{max}), the frequency remains unchanged to avoid oscillations.

Algorithm 4: Frequency Adaptation

Input: Runs every interval τ; GPU set $\mathcal{K}$; SLA factor γ; minimum and maximum GPU frequencies $f_{\min}, f_{\max}$

```
for each GPU g ∈ K do
    f_g ← current frequency of GPU g;
    β ← curve-fitting constant;
    T_g ← tasks running on GPU g;
    violation ← false;
    slack ← true;
    for each task t ∈ T_g do
        e_t^pred ← e_min,t · (f_max / f_g)^β;
        e_t^max ← γ · e_min,t;
        if e_t^pred > e_t^max then
            violation ← true;
        else if e_t^pred ≥ 0.95 · e_t^max then
            slack ← false;
    if violation then
        f_g ← min(f_max, f_g + Δf);
    else if slack then
        f_g ← max(f_min, f_g − Δf);
```

Complexity Analysis. The time complexity of *Correlation Analysis (Algorithm 1)*, for each incoming task, is that the algorithm scans every GPU in the server. For each GPU, it computes the correlation coefficient and the difference in utilization, both of which require O(L) operations on the utilization vectors of length L. Thus, the time complexity per invocation is $O(G \cdot L)$, where G is the total number of GPUs. For the *Scheduler (Algorithm 2)*. the scheduler evaluates each candidate GPU in the buffer $\mathcal{B}$ and assigns it a score,which costs $O(|\mathcal{B}|)$, and then sorts the scores, which costs $O(|\mathcal{B}| \log |\mathcal{B}|)$. Therefore, the total time complexity is $O(|\mathcal{B}| \log |\mathcal{B}|)$ which is at most $O(G \log G)$. The *Periodic Sleep (Algorithm 3)* procedure scans every server and every GPU once per interval. Each check is constant time, so the total time complexity is $O(G)$. For the *Frequency Adaptation (Algorithm 4)*. This module iterates over all active GPUs and over all tasks running on them. Let $|\mathcal{K}|$ be the number of active GPUs and T the total number of active tasks. The total time complexity per invocation is $O(|\mathcal{K}| + T)$.

3.6 Discussion

Scalability. To address scalability concerns in large data centers with thousands of GPUs, CATS can be deployed at three levels of granularity: *data center level*, *cluster level*, and *server level*. In *Data Center Level* managing all GPUs through a single centralized controller can lead to significant overhead. To mitigate this, correlation analysis can be performed less frequently, such as every 10 minutes, and simpler task allocation methods like first-fit can be utilized. This approach reduces computational complexity while still enabling power optimization across the entire data center. For *Cluster Level* we divide the data center into smaller clusters, each managed by its own CATS controller, which lowers the computational burden compared to a fully centralized approach. Within each cluster, correlation analysis can be conducted more regularly, for example, every 5 minutes, allowing for more responsive task dispatching and frequency scaling. *Server Level* has the highest level of granularity. Here CATS can be implemented on individual servers. Each server's controller manages only the GPUs within that server, enabling real-time task scheduling and frequency adjustments with minimal overhead.

Task Migration. CATS does not perform Dynamic migration because it's expensive for ML training and requires checkpointing, transferring, and restoring GPU-resident model and optimizer state, which can take seconds to minutes and scales with model size. Migration also disrupts placement locality in distributed training and can affect performance [38].

Multidimensional Correlation. CATS can be extended to a multidimensional resource model (e.g., Memory utilization, HBM bandwidth, interconnect) by representing each job as a resource vector and performing per-dimension correlation checks subject to capacity constraints. However, such an extension scales linearly with the number of dimensions, preserving lightweight execution within the sub-millisecond window. Currently, CATS conducts correlation analysis only for GPU utilization and considers other resource demands, such as memory capacity, as hard constraints.

Extension to LLM workloads. Although our evaluation focuses on conventional DNN training jobs, CATS naturally extends to large language model (LLM) workloads. Compared to traditional DNN training, LLM execution exhibits more structured and predictable phases, such as attention computation and KV-cache updates, which often lead to bursty and memory-bound behavior. Prior work shows that these phases can be accurately predicted [51]. Because CATS identifies tasks whose coarse-grained utilization patterns do not peak simultaneously, these predictable execution phases make LLM workloads particularly well suited for consolidation. For example, a compute-intensive prompt-processing

task can be consolidated with a memory-bound token-generation task when their utilization phases complement each other, and their combined memory footprint fits within the GPU capacity.

4 Experimental Setup

Hardware Testbed. Our testbed consists of a single node-server with six Nvidia Volta-V100 GPUs. Table 3 describes the DRAM consumption of the GPUs on the server. The server runs on Ubuntu 20.04 LTS and is powered by a 1.7 GHz CPU, and the GPUs operate at frequencies between 135 MHz and 1350 MHz. Task scheduling is performed online using PyTorch 1.8.1. To enable spatial sharing, the CUDA Multiprocessing Service (MPS) is used. The MPS thread percentage is set to 100% to ensure complete GPU provisioning. In all experiments, we set the correlation threshold to 0 and the utilization threshold to 100%. We measure GPU power consumption and utilization using the `nvidia-smi` interface.

Table 3: GPU Configuration in the Hardware Testbed

GPU	I	II	III	IV	V	VI
GPU DRAM Capacity (GB)	16	16	32	32	16	32

Workloads. The workloads for the experiments are as follows: t_1 corresponds to DenseNet with a batch size of 50, t_2 is GoogleNet with a batch size of 30, t_3 is ConvNeXt [71] with a batch size of 50, t_4 is Vision Transformer (ViT) with a batch size of 60, t_5 is Swin Transformer [40] with a batch size of 40, t_6 is ResNet [40] with a batch size of 30, t_7 is MobileNet [25] with a batch size of 50, and finally t_8 is SqueezeNet [28] with a batch size of 60. The workloads are derived from the PAI trace and were selected to (i) reflect realistic GPU utilization levels observed in production traces and (ii) ensure diverse memory footprints that enable meaningful consolidation under DRAM constraints [8][59].

Baselines. The baselines used for comparison are `(1) Hotcloud` and `(2) First`. `Hotcloud` calculates the sum of the average utilization of the existing and incoming tasks on the GPU. If the combined average utilization of the two is below the utilization threshold, `Hotcloud` schedules them on the same GPU. The GPU utilization prediction follows the approach suggested by Yeung et al. [73]. `First` computes the sum of GPU utilization during the first second of the incoming task and adds it to the aggregated GPU utilization of tasks already running on the GPU. If their total utilization is less than the threshold (q), it schedules the tasks to the same GPU.

Trace-driven Simulation. To address the limitations of our hardware testbed, we developed a Python simulator to evaluate `CATS` on a large scale using real-world traces. The simulator is built from the open-source code of Gavel [48] and GPUColo [8], and is designed to emulate a heterogeneous data center. The simulator features several key components: a simulated clock module, a scheduler, a server, a GPU manager, and a power module. Before the simulation begins, all jobs are sorted by their release times in ascending order. Jobs are then assigned to GPUs using the scheduling strategy specific to each baseline; for each baseline, we apply its corresponding scheduling solution as described in the previous paragraph. The simulator also includes a power management component that continuously monitors the power of active GPUs and servers. The power manager records power consumption over time, applies consolidation policies such as dynamic frequency scaling or periodic sleep, and ensures that idle resources do not consume unnecessary power.

5 Hardware Evaluation

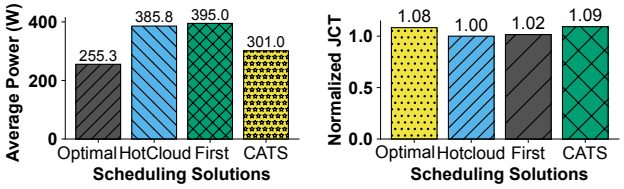
We evaluate `CATS` on our GPU testbed using workloads sampled from production traces. We use 14 machine learning (ML) tasks, sampled from the PAI trace [66], selected based on their GPU utilization and memory requirements. To reflect the bursty arrival patterns observed in the PAI trace, we structure the experiment into two stages, each consisting of a group of tasks released within a short time window, as summarized in Table 4. Stage 1 comprises eight tasks t_1 to t_8 and stage 2 comprises six tasks t_9 to t_{14}. Initially, four tasks (t_1–t_4) are released sequentially at 0, 1, 2, and 3 seconds. Four additional tasks (t_5–t_8) are then launched at $20 - 23$ seconds. In Stage 2 at time 500 seconds, four tasks (t_9, t_{10}, t_{11}, and t_{12}) are released. Two more tasks (t_{13} and t_{14}) are released at 520 and 522 seconds. The workloads described in Section 4 are reused across the tasks. Specifically, tasks t_1 to t_8 use the workloads of Section 4, while tasks t_9 to t_{14} replicate the same workloads as t_1 to t_8.

Table 4: Task start time, GPU utilization (%), and memory consumption (GB).

Task	Start time	GPU util. (%)	Mem (GB)
t_1	0	45.84	6.28
t_2	1	44.44	6.30
t_3	2	40.00	2.20
t_4	3	46.00	1.80
t_5	20	49.00	3.90
t_6	21	47.00	1.60
t_7	22	45.00	2.20
t_8	23	44.00	3.50
t_9	500	45.84	6.28
t_{10}	500	44.44	6.30
t_{11}	500	40.00	2.20
t_{12}	500	46.00	1.80
t_{13}	520	49.00	3.90
t_{14}	522	47.00	1.60

We design this experiment to evaluate the robustness of CATS under model prediction error. The objective is to demonstrate that the correlation-aware scheduler remains effective even when task utilization predictions are noisy or partially inaccurate. To achieve this, we construct a synthetic GPU utilization trace for the selected 14 tasks. For each task, we sample from a Gaussian distribution with mean (μ) and variance (σ^2) matching those observed in real ML training workloads. To emulate prediction uncertainty, we add additive white noise to the generated utilization profiles, introducing controlled perturbations to the predicted task behavior.

In the first step, we measure power consumption across all stages and calculate the mean power consumption for each solution, as illustrated in Figure 7a. The figure indicates that the `optimal` solution has the lowest power consumption, averaging 255.3 W. In comparison, `CATS`, `HotCloud`, and `First` consume 301 W, 385.8 W, and 395 W, respectively. The `optimal` solution achieves this lower power consumption by utilizing the fewest GPUs. Although the `optimal` solution offers an upper bound for comparison, its high computational overhead limits its scalability. Therefore, we only



(a) Power (b) Normalized JCT

Figure 7: (a) The `optimal` solution has the smallest power consumption. (b) The normalized JCT is similar across all solutions, and the `optimal` solution remains within the SLA threshold.

present results for the `optimal` solution in our hardware testbed experiments and exclude it from larger-scale simulations.

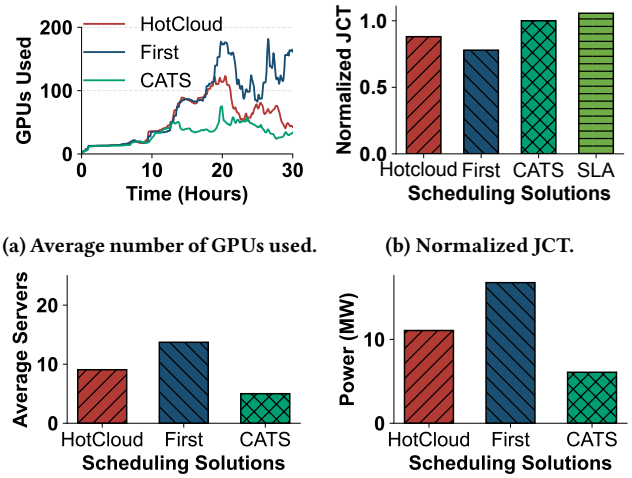
CATS achieves significantly lower power consumption compared to HotCloud and `First`. This is due to CATS consolidating several tasks onto fewer GPUs, placing unused GPUs into a low-power sleep state, and applying frequency scaling to reduce GPU dynamic power consumption. Reducing power consumption is typically achieved by decreasing the number of GPUs, which may increase job completion time (JCT). To assess whether this increase violates the Service Level Agreement (SLA), we compute the normalized JCT for all tasks and normalize it with respect to CATS. We set the SLA threshold to 20% of the fastest baseline as mentioned in Section 2. Figure 7b presents the normalized JCT for all techniques. We observe that CATS maintains its JCT within the SLA. This is mainly because CATS consolidates tasks with negative correlations, thereby reducing resource contention.

6 Simulation Evaluation

In this section, we conduct simulation experiments across all baselines and CATS using the simulator mentioned in Section 4. As discussed in Section 2, we do not perform the simulation experiment for the `optimal` solution because determining the optimal consolidation is computationally expensive.

We perform simulation experiments on the two traces (*PAI* and *Philly*) mentioned in Section 2. To ensure a fair comparison at the start of the simulation, all GPUs and servers are put to sleep and activated only when a task is dispatched. Whenever the GPU or a server is idle, the *periodic* sleep module is activated and puts unused GPUs and servers to sleep for all the solutions. In all the experiments, each server is equipped with 8 GPUs.

PAI Trace. We choose 5000 tasks from the *PAI* trace to perform the simulation. Figure 8a shows the average number of GPUs used throughout the experiment. CATS requires fewer GPUs because it consolidates more tasks per GPU than `HotCloud`. On average, CATS uses 37.32 GPUs, `First` uses 92.94 GPUs, and `HotCloud` uses 61.94 GPUs. CATS uses 39.75% fewer GPUs than `HotCloud`. The reduction in the number of GPUs must not exceed the SLA. As shown in Figure 8b, CATS exhibits a slightly higher JCT than the baseline approaches, `HotCloud` and `First`, which have JCT values of 0.87 and 0.77, respectively. In contrast, CATS's JCT is 1, which remains within the due time threshold of $1.2 \times 0.87 = 1.044$, where 0.87 is `HotCloud`'s JCT. CATS reduces resource contention by consolidating tasks with negative correlations. CATS also uses fewer servers compared to other solutions as shown in Figure 8c. CATS has the lowest power consumption because it puts unused servers and GPUs to sleep to reduce idle power, and it applies DFS to reduce dynamic power. CATS's power consumption is 6.08 MW, which is 63.59% less than `First`'s 16.7 MW as shown in Figure 8d.



(a) Average number of GPUs used. (b) Normalized JCT.

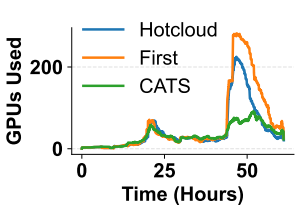
(c) Average number of servers used. (d) Average power consumption.

Figure 8: Resource utilization, performance, and power consumption results for the PAI trace simulation. (a) CATS uses the fewest GPUs. (b) The normalized job completion time (JCT) of CATS remains within the SLA. (c) CATS uses the fewest servers. (d) CATS achieves the lowest average power consumption.

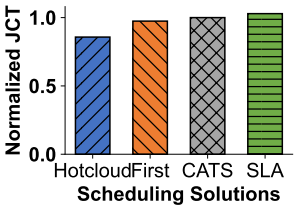
Philly Trace. We choose 2000 tasks from the *Philly* trace to perform the simulation. Figure 9a shows the average number of GPUs used throughout the experiment. CATS requires fewer GPUs because it consolidates more tasks per GPU than `HotCloud`.

Reducing the number of GPUs must not exceed the SLA. As shown in Figure 9b, CATS exhibits a slightly higher JCT than the baseline approaches, `HotCloud` and `First`. Normalized to CATS, the JCT values are 0.857 for `HotCloud`, 0.974 for `First`, and 1.0 for CATS. CATS's normalized JCT remains within the due time threshold of $1.2 \times 0.857 = 1.028$, where 0.857 corresponds to `HotCloud`'s normalized JCT. CATS reduces resource contention by consolidating tasks with negative correlations. CATS also uses fewer servers compared to other solutions as shown in Figure 9c. CATS has the lowest power consumption because it puts unused servers and GPUs to sleep to reduce idle power and applies DFS to reduce dynamic power consumption. The average power consumption of CATS is 6.77 MW, compared to 8.30 MW for `HotCloud` and 8.99 MW for `First`. This corresponds to a reduction of 18.4% compared to `HotCloud` and 24.7% compared to `First` as shown in Figure 9d.

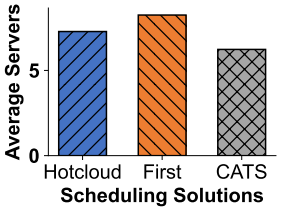
Next, we compare CATS with other task consolidation solutions that perform consolidation based on resource attributes such as memory consumption. These approaches are orthogonal to utilization-aware GPU scheduling techniques such as CATS, but they influence how tasks are colocated on shared resources and can therefore be used as baselines for comparison. The consolidation solutions we



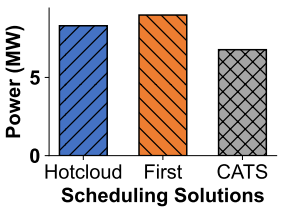
(a) Average number of GPUs used.



(b) Normalized JCT.



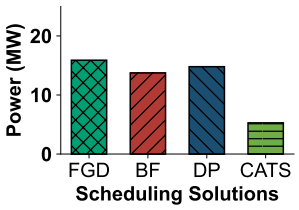
(c) Average number of servers used.



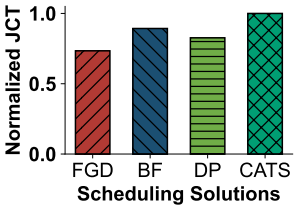
(d) Average power consumption.

Figure 9: Resource utilization, performance, and power consumption results for the Philly trace simulation. (a) CATS uses the fewest GPUs. (b) The normalized job completion time (JCT) of CATS remains within the SLA. (c) CATS uses the fewest servers. (d) CATS achieves the lowest average power consumption.

consider are `Best-Fit (BF)` [46], `Dot-Product (DP)` [21], and FGD [70]. `Best-Fit (BF)` assigns a task to the node that results in the least remaining resources after placement, computed as a weighted sum across resource dimensions. `Dot-Product (DP)` selects the node that minimizes the dot product between the task's resource demand vector and the node's available resource vector. FGD mitigates GPU fragmentation by placing tasks along the steepest descent of fragmentation, thereby improving allocation efficiency.



(a) Power consumption.



(b) Normalized JCT.

Figure 10: (a) CATS achieves the lowest power consumption. (b) Normalized JCT for all the solutions.

Figure 10a shows the average power consumption across all solutions. Among all the solutions, CATS achieves the lowest power consumption, using approximately 5.28 kW on average, while FGD, BF, and DP consume 15.89 kW, 13.73 kW, and 14.79 kW, respectively. CATS reduces power consumption by up to 61.6%. This reduction is primarily due to two factors. First, CATS uses fewer GPUs by consolidating tasks with negative performance correlations. Second, CATS applies DFS to reduce GPU dynamic power consumption further. Figure 10b presents the normalized job completion time (JCT). The other solutions have a better JCT because these solutions are optimized for the shortest makespan, without accounting for power consumption.

7 Related Work

The related work can be classified into two categories: (1) Schedulers for AI Training in data centers, and (2) Power optimization for GPU clusters.

Schedulers for AI Training in data centers: Early efforts used cluster managers like Kubernetes or YARN to schedule DL jobs in the cloud without considering the characteristics of DL jobs, which results in low performance [31][6]. Recent efforts proposed specialized cluster schedulers for DL training jobs [45][72][8][23][70][21]. These efforts focus on optimizing for JCT, fairness, or GPU utilization while ignoring the energy consumption of DL jobs and the GPU cluster.

Power optimization for GPU clusters. Prior research on power optimization has predominantly centered around CPU-based systems, where strategies such as dynamic power management, server consolidation, and CPU dynamic voltage and frequency scaling (DVFS) have been extensively investigated [36, 67, 68]. However, with the rapid growth of AI workloads, recent studies have begun to focus on GPUs, which are significant contributors to data center energy consumption due to their high power density and substantial idle power. Consequently, several GPU-focused initiatives have explored DVFS-based power control [7, 9, 39, 41, 42], strategies for workload deferral to sustain high GPU utilization [17], and predictive techniques for powering GPU nodes on and off to mitigate idle energy waste [27]. More recent work has additionally investigated energy-aware GPU scheduling and dynamic configuration tuning for deep learning training workloads [22, 51, 65].

All the above solutions consider GPU utilization as static, overlooking temporal fluctuations or correlations among tasks. This oversight can lead to suboptimal consolidation decisions and unnecessary activation of GPUs or servers.

8 Conclusion and Future Work

In this paper, we present CATS, a novel power-optimization framework for GPU data centers. In sharp contrast to existing work that assumes GPU utilization for each ML task can be approximated as a constant, CATS features correlation-aware scheduling to analyze GPU utilization patterns across ML tasks. By consolidating tasks with negatively correlated GPU utilization on the same GPUs, CATS reduces the number of active GPUs required, thereby lowering overall power consumption without significantly increasing the job completion time (JCT) of the ML tasks. We formulate correlation-aware ML task consolidation as a constrained optimization problem with a costly optimal solution. We then design a light-weight heuristic algorithm that is practical for real data centers. Our experiments, conducted through extensive simulations, demonstrate that CATS reduces power consumption by up to 24.7% relative to the baselines while maintaining comparable JCTs.

Acknowledgments

This work was supported, in part, by the U.S. NSF under Grant CNS-2336886. We would like to thank the anonymous reviewers for

their valuable comments. We also want to thank Keegan Freyhof for his assistance with the simulation experiments.

References

[1] [n. d.]. Tech's splurge on AI chips has companies in 'arms race' that's forcing more spending. https://www.cnbc.com/2024/07/25/techs-splurge-on-ai-chips-has-meta-alphabet-tesla-in-arms-race.html.

[2] Hadeel Albahar, Shruti Dongare, Yanlin Du, Nannan Zhao, Arnab K Paul, and Ali R Butt. 2022. Schedtune: A heterogeneity-aware gpu scheduler for deep learning. In *22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE.

[3] Yehia Arafa, Ammar ElWazir, Abdelrahman ElKanishy, Youssef Aly, Ayatelrahman Elsayed, Abdel-Hameed Badawy, Gopinath Chennupati, Stephan Eidenbenz, and Nandakishore Santhi. 2020. Verified instruction-level energy consumption measurement for NVIDIA GPUs. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*.

[4] 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)*.

[5] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep learning-based job placement in distributed machine learning clusters. In *IEEE conference on computer communications (INFOCOM)*.

[6] Scott Boag, Parijat Dube, Benjamin Herta, Waldemar Hummer, Vatche Ishakian, K Jayaram, Michael Kalantar, Vinod Muthusamy, Priya Nagpurkar, and Florian Rosenberg. 2017. Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs. In *Workshop on ML Systems, NIPS*.

[7] Robert A Bridges, Neena Imam, and Tiffany M Mintz. 2016. Understanding GPU power: A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys (CSUR)* 49, 3 (2016).

[8] Guoyu Chen, Srinivasan Subramaniyan, and Xiaorui Wang. 2024. Latency-guaranteed co-location of inference and training for reducing data center expenses. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*.

[9] Guoyu Chen and Xiaorui Wang. 2022. Performance optimization of machine learning inference under latency and server power constraints. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*.

[10] Wenyan Chen, Zizhao Mo, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2023. Interference-aware multiplexing for deep learning in gpu clusters: A middleware approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[11] Xiaoming Chen, Danny Z Chen, and Xiaobo Sharon Hu. 2018. moDNN: Memory optimal DNN training on GPUs. In *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[12] 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)*.

[13] NVIDIA Corporation. 2023. *NVIDIA System Management Interface*.

[14] Paul Delestrac, Debjyoti Battacharjee, Simei Yang, Diksha Moolchandani, Francky Catthoor, Lionel Torres, and David Novo. 2024. Multi-level analysis of gpu utilization in ml training workloads. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[15] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. 2020. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC)*.

[16] Paul Elvinger, Foteini Strati, Natalie Enright Jerger, and Ana Klimovic. 2025. Measuring GPU utilization one level deeper. *arXiv preprint arXiv:2501.16909* (2025).

[17] Federica Filippini, Danilo Ardagna, Marco Lattuada, Edoardo Amaldi, Maciek Riedl, Katarzyna Materka, Paweł Skrzypek, Michele Ciavotta, Fabrizio Magugliani, and Marco Cicala. 2021. ANDREAS: Artificial intelligence traiNing scheDuler foR accElerAted resource clusterS. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE.

[18] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[19] Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. 2020. Estimating GPU memory consumption of deep learning models. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*.

[20] Guin Gilman and Robert J Walls. 2022. Characterizing concurrency mechanisms for NVIDIA GPUs under deep learning workloads. *ACM SIGMETRICS Performance Evaluation Review* 49, 3 (2022).

[21] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014).

[22] Diandian Gu, Xintong Xie, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. Energy-efficient GPU clusters scheduling for deep learning. *arXiv preprint arXiv:2304.06381* (2023).

[23] Zhenhua Han, Haisheng Tan, Shaofeng H-C Jiang, Xiaoming Fu, Wanli Cao, and Francis CM Lau. 2020. Scheduling placement-sensitive BSP jobs with inaccurate execution time estimation. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*.

[24] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. Elastictree: Saving energy in data center networks.. In *NSDI*.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[26] Ching-Hsien Hsu, Kenn D Slagter, Shih-Chang Chen, and Yeh-Ching Chung. 2014. Optimizing energy consumption with task consolidation in clouds. *Elsevier Information Sciences* (2014).

[27] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[28] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[29] Pierre Jacquet, Maxime Agusti, Eddy Caron, Camille Coti, Marcos Dias De Assunção, Laurent Lefèvre, and Anne-Cécile Orgerie. 2026. Untangling GPU Power Consumption: Job-Level Inference in Cloud Shared Settings. In *European Conference on Computer Systems (Eurosys)*.

[30] Arpan Jain, Ammar Ahmad Awan, Asmaa M Aljuhani, Jahanzeb Maqbool Hashmi, Quentin G Anthony, Hari Subramoni, Dhableswar K Panda, Raghu Machiraju, and Anil Parwani. 2020. Gems: Gpu-enabled memory-aware model-parallelism system for distributed dnn training. In *SC20: international conference for high performance computing, networking, storage and analysis*.

[31] 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 *USENIX Annual Technical Conference (USENIX ATC)*.

[32] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems* (2019).

[33] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2024. Slo-aware gpu dvfs for energy-efficient llm inference serving. *IEEE Computer Architecture Letters* (2024).

[34] Beth Kindig. 2024. AI Power Consumption: Rapidly Becoming Mission-Critical. https://www.forbes.com/sites/bethkindig/2024/06/20/ai-power-consumption-rapidly-becoming-mission-critical/.

[35] 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*.

[36] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. 2007. Server-level power control. In *IEEE Fourth International Conference on Autonomic Computing (ICAC'07)*.

[37] 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 (SC)*.

[38] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[39] Wenjie Liu, Zhihui Du, Yu Xiao, David A Bader, and Chen Xu. 2011. A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*.

[40] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*.

[41] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *41st international conference on parallel processing (ICPP)*. IEEE.

[42] Yuan Ma, Srinivasan Subramaniyan, and Xiaorui Wang. 2025. Power capping of gpu servers for machine learning inference optimization. In *Proceedings of the 54th International Conference on Parallel Processing (ICPP)*.

[43] David Meisner, Brian T Gold, and Thomas F Wenisch. 2009. Powernap: eliminating server idle power. *ACM SIGARCH Computer Architecture News* (2009).

[44] Stuart Mitchell, Michael OSullivan, and Iain Dunning. 2011. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand* (2011).

[45] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2021. Synergy: Resource sensitive DNN scheduling in multi-tenant clusters. *arXiv preprint arXiv:2110.06073* (2021).

[46] 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)*.

[47] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2021. Batchsizer: Power-performance trade-off for dnn inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*.

[48] 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)*.

[49] Nvidia. [n. d.]. NVIDIA H100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/h100/.

[50] Adrian Osterwind, Julian Droste-Rehling, Manoj-Rohit Vemparala, and Domenik Helms. 2022. Hardware execution time prediction for neural network layers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer.

[51] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, 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*. 207–222.

[52] Pratyush Patel, Zibo Gong, Syeda Rizvi, Esha Choukse, Pulkit Misra, Thomas Anderson, and Akshitha Sriraman. 2023. Towards improved power management in cloud gpus. *IEEE Computer Architecture Letters* (2023).

[53] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2024. Power-aware deep learning model serving with $\{\mu$-Serve$\}$. In *USENIX Annual Technical Conference (USENIX ATC)*.

[54] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. CASSINI:Network-Aware job scheduling in machine learning clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[55] Ties Robroek, Ehsan Yousefzadeh-Asl-Miandoab, and Pınar Tözün. 2024. An analysis of collocation on GPUs for deep learning training. In *Proceedings of the 4th Workshop on Machine Learning and Systems*.

[56] Anton Shilov. 2023. Nvidia's H100 GPUs will consume more power than some countries — each GPU consumes 700W of power, 3.5 million are expected to be sold in the coming year. https://www.tomshardware.com/.

[57] 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*.

[58] Srinivasan Subramaniyan, Rudra Joshi, Xiaorui Wang, and Marco Brocanelli. 2025. SEEB-GPU: Early-Exit Aware Scheduling and Batching for Edge GPU Inference. In *Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing*.

[59] Srinivasan Subramaniyan and Xiaorui Wang. 2025. Exploiting ML Task Correlation in the Minimization of Capital Expense for GPU Data Centers. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*.

[60] Srinivasan Subramaniyan and Xiaorui Wang. 2025. FC-GPU: Feedback Control GPU Scheduling for Real-time Embedded Systems. *ACM Transactions on Embedded Computing Systems* 24, 5s (2025).

[61] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy)*.

[62] Balavignesh Vemparala, Ming Yang, and Soheil Soghrati. 2024. Deep learning-driven domain decomposition (DLD3): A generalizable AI-driven framework for structural analysis. *Computer Methods in Applied Mechanics and Engineering* 432 (2024).

[63] Balavignesh Vemparala Narayana Murthy. 2024. *Advanced Computational and Deep Learning Techniques for Modeling Materials with Complex Microstructures*. Ph. D. Dissertation. The Ohio State University.

[64] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. 2009. Server workload analysis for power minimization using consolidation. In *Proceedings of the conference on USENIX Annual technical conference (USENIX ATC)*.

[65] Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. 2021. Dynamic GPU energy optimization for machine learning training workloads. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2021).

[66] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. 2019. Characterizing deep learning training workloads on alibaba-pai. In *IEEE international symposium on workload characterization (IISWC)*.

[67] Xiaorui Wang and Ming Chen. 2008. Cluster-level feedback power control for performance optimization. In *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)*.

[68] Xiaorui Wang, Ming Chen, Charles Lefurgy, and Tom W Keller. 2009. Ship: Scalable hierarchical power control for large-scale data centers. In *18th International Conference on Parallel Architectures and Compilation Techniques*.

[69] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. 2012. Carpo: Correlation-aware power optimization in data center networks. In *Proceedings IEEE INFOCOM*.

[70] 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 *USENIX Annual Technical Conference (USENIX ATC)*.

[71] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. 2023. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

[72] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2024. Deep learning workload scheduling in gpu datacenters: A survey. *Comput. Surveys* 56, 6 (2024).

[73] Gingfung Yeung, Damian Borowiec, Adrian Friday, Richard Harper, and Peter Garraghan. 2020. Towards GPU utilization prediction for cloud deep learning. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[74] Daegun Yoon and Sangyoon Oh. 2023. Deft: Exploiting gradient norm difference between model layers for scalable gradient sparsification. In *Proceedings of the 52nd International Conference on Parallel Processing*.

[75] Sheng Zhang, Zhuzhong Qian, Zhaoyi Luo, Jie Wu, and Sanglu Lu. 2015. Burstiness-aware resource reservation for server consolidation in computing clouds. *IEEE Transactions on Parallel and Distributed Systems* 27, 4 (2015).