Energy-efficient GPU SM allocation

Bing-Shiun Han, Kunaal Parekh, Wan-Chu Lin, Tathagata Paul, Anshul Gandhi, Zhenhua Liu
Stony Brook University, Stony Brook, New York, USA
{bing-shiun.han,kunaal.parekh,wanchu.lin,tathagata.paul,anshul.gandhi,zhenhua.liu}@stonybrook.edu

ABSTRACT

GPU sharing between workloads is an effective approach to increase GPU utilization and reduce idle power waste. To minimize resource contention under GPU sharing, current architectures allow users to allocate core GPU compute resources exclusively to workloads. However, identifying the most efficient GPU compute resource allocation for colocated workloads is challenging, as it requires balancing potential performance degradation and power savings.

This paper presents a framework for finding the most energy-efficient compute allocation for colocated workload pairs under NVIDIA MPS using lightweight prediction models. Experimental results, using a range of training, inference, and general CUDA workloads, demonstrate that our solution outperforms the equal sharing strategy by 35%, on average, and is within 1.5% of the offline optimal strategy.

1 Introduction

The growing workload demand for power-hungry GPUs has significantly strained data centers [9]. With newer, often larger, AI models requiring even more computational resources, there is now even greater reliance on energy-intensive GPUs. It is thus important to fully utilize available GPUs.

A common approach to improve the efficiency of GPUs is to colocate multiple workloads together on a GPU to improve utilization [13, 22, 15, 5, 4]; this is particularly useful for workloads that do not fully saturate the GPU's compute capacity. The performance interference between the workloads can then be regulated by partitioning the resources between them. For example, the widely employed NVIDIA MPS (Multi-Process Service) [2] enables partitioning of GPU resources between colocated workloads. In particular, MPS is a CUDA API implementation that allows each workload to exclusively occupy a Streaming Multiprocessor (SM), which is the compute unit of a GPU. By configuring the `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` parameter, MPS prevents process interference within a single SM by limiting the percentage of available threads a workload can use. Using this parameter, for example, we can reserve 50% of SMs for one workload and the remaining 50% for another colocated workload; we refer to this as the "equal partitioning" strategy.

One key problem in using MPS is to ***find the optimal SM partition configuration for energy efficiency***. This problem is non-trivial and *workload-dependent*. For instance, when colocating an ALBERT training workload with the more compute-intensive FastWalshTransform CUDA workload [17], we can reduce GPU energy consumption by 36% *and* improve system throughput by 8% by allocating 10% of the SMs to ALBERT and 90% to FastWalshTransform, compared to an equal partitioning approach (50% SMs to each workload). Likewise, when colocating ALBERT with the lightweight RESNET workload, we can reduce GPU energy usage by 10% (while providing the same throughput), but this time by allocating 60% of the SMs to ALBERT and 40% to RESNET, compared to the equal partitioning approach. Thus, while significant benefits can be achieved with optimal SM partitioning, finding this optimal partition configuration can be challenging.

In light of the above discussion, we pose our problem statement as: ***"Given a pair of GPU workloads, how can the optimal SM partition be determined?"*** In this work, we limit our scope to a workload pair; we will consider colocation of more than two workloads in future work.

This workshop paper presents *a lightweight, application-level framework that predicts the optimal SM partition configuration.* We employ the *throughput-per-watt* metric, which has been used in prior optimization works [3, 10, 14] as it represents a single metric that balances energy efficiency and performance. We *predict* colocated power consumption and throughput using machine learning (ML) approaches, then leverage these predictions at runtime to determine the optimal SM partition that maximizes throughput per watt.

Through our experimental analysis, we find that a simple Extra Trees model, with minimal training time, can accurately predict the optimal SM partitioning ratio for a given workload pair. Our solution involves a one-time, offline run of individual (not colocated) workloads to extract meaningful metrics. We find that employing *NVIDIA DCGM* [19] metrics as features to our ML model provides valuable insights for SM partitioning while incurring low profiling overhead.

We implement our solution framework in Python and evaluate its performance against other baselines using several deep learning (DL) and compute-intensive CUDA Sample workloads [18]. We find that our solution is able to accurately predict colocated throughput and colocated GPU power (5%–10% prediction error) using limited training data. Our experimental results on an NVIDIA V100 GPU with MPS support show that the throughput per watt achieved by our predictive solution is within 1.5% of that achieved by an offline optimal strategy. Further, our solution outperforms the equal partitioning strategy by 35%, on average, across all workloads we tested.

2 Background and Related Work

MIG [1] and MPS [2] are spatial sharing frameworks provided by NVIDIA that support GPU resource partitioning. While MIG offers hardware-level isolation with both compute and memory partitioning, it is only available on a few of the latest high-end GPU architectures and supports less

than 20 fixed partition configurations, limiting its general applicability. Additionally, reconfiguring MIG partitions incurs significant time and restarting overheads [15]. In contrast, MPS is supported on most NVIDIA GPUs and allows user-defined SM compute partitioning across concurrent processes with low reconfiguration overheads, offering greater flexibility and adaptability. In this workshop paper, we adopt MPS for SM allocation and partitioning.

We now discuss related works that leverage GPU partitioning to improve energy efficiency. Vamja et al. [23] use power models to predict the power consumption of MIG. Espenshade et al. [8] scale deep learning (DL) training workloads to fit various MIG partition sizes while maintaining energy efficiency. MISO [15] predicts optimal MIG partitions by running actual colocated runs (requiring significant training effort) under different MPS partition ratios. However, as discussed above, MIG is available only on a limited set of high-end GPUs.

Weaver et al. [24] predicts interference and rearranges scheduling workflows to identify the optimal number of concurrent MPS clients to launch; However, they do not search for the optimal SM partition for each colocated workload. KRISP [6] reduces energy usage for ML inference workloads by resizing a model's spatial partitions at GPU kernel level with AMD Compute Unit (CU) masking; however, its implementation is limited to environments with AMD ROCm runtime. EaCO [12] predicts the placement of DL training tasks under spatial sharing to reduce GPU provisioning and idle power waste. However, it does not incorporate partitioning techniques and may result in workload interference. μ-Serve [20] is a DL inference serving framework that combines model multiplexing with DVFS to achieve power savings while minimizing SLO violations. Part of its design relies on output-token length predictions and DL operator profiling, limiting its applicability to autoregressive models and DL inference workloads.

Based on the above discussion, we believe that ***there is a gap in the literature on identifying the most efficient SM partition when colocating workloads using MPS***. This is the gap we aim to bridge in this work.

3 System Design

This section describes the system design of our solution. The key components of our solution are (1) offline performance metric profiling, (2) offline model training, and (3) online SM partition prediction. The system operates as follows: when a new (unseen) workload pair arrives, we profile the individual workload's metrics offline (Section 3.1) and use the pretrained throughput and power prediction models (Section 3.2) to predict the optimal SM partition for colocation (Section 3.3). The prediction models do not need to be retrained for each new workload.

3.1 Performance metric profiling (offline)

A natural approach to determining the optimal SM partition configuration for a given workload pair is to experiment with various SM partition configurations and then find the best among them. This approach has two significant problems: (i) the obvious problem is the high profiling/experimentation overhead of actually trying out various partitions; and (ii) the less obvious, but still critical problem of having to repeat this profiling and experimentation for every new workload pair encountered.

To overcome these problems, we design a predictive approach. We start by profiling GPU metrics offline for *individual* workloads, i.e., without colocation, under 9 different SM allocations (10%, 20%, ..., 90%). These metrics are collected one-time only. This approach crucially avoids the costly profiling required for all possible colocated pairs.

To profile individual workloads offline with minimal overhead, we leverage multiple NVIDIA GPU monitoring tools, including `nvidia-smi` [16] and `NVIDIA DCGM` [19], to collect overall system metrics such as compute and memory utilization. We incorporate DCGM metrics because `nvidia-smi` can be inaccurate in SM measurements [7, 22]. In contrast, DCGM offers more detailed insights, including SM warp activities, SM occupancy, and floating point pipeline usage. Some previous works have collected performance metrics through fine-grained kernel profiling [13]. However, kernel profiling an entire workload could take several hours, which becomes impractical, especially when multiple SM allocations must be evaluated. By contrast, DCGM metrics can be gathered with real-time monitoring at 1-second intervals, significantly reducing profiling time.

To initiate the profiling process, we perform a single offline profiling run for each workload, as both `nvidia-smi` and `DCGM` can be collected simultaneously. Each DL workload is executed for 100 steps (averaging about one minute), while the entire runtime of each scientific CUDA sample workload is profiled, since we cannot guarantee periodic execution as we can with DL workloads. However, all profiling is performed offline and completes in minutes, ensuring it is not on the critical path.

The metrics collected for power and throughput modeling differ slightly. For power-related modeling, we collect only the essential GPU performance metrics. For the more challenging and workload-dependent task of throughput prediction, we additionally include the throughput of each individual workload (obtained from an exclusive run without colocation) as a feature, since it serves as a strong performance indicator. We also compute *throughput sensitivity* for each workload, defined as the slope of a linear fit of individual throughput across different SM partitions; this sensitivity feature guides our SM allocation by suggesting which workloads can make better use (in terms of additional throughput achieved) of additional SMs.

The final set of metrics we employ is: SM warp activity rate, SM occupancy rate, FP32 engine activity rate, memory bandwidth utilization, memory busy rate, individual workload sensitivity, throughput of individual workload without colocation, PCIe transmission bytes, and PCIe receiving bytes.

3.2 Model training (offline)

To train (and test) the colocated throughput and colocated GPU power models for predicting the throughput per watt of colocated workloads, we conduct experiments where pairs of workloads are colocated with different SM allocation partitions. To inspect the impact of interference, we focus on evaluating the interval when both workloads are executing. We show in Section 5 that a small training set size suffices for accurate model predictions.

For the training dataset, we combine the features of colocated workloads. Specifically, we sum the raw metrics (e.g., PCIe transmission bytes) and average the metrics reported as percentages (e.g., SM occupancy). Finally, all features are

Workload	Compute	Memory
FWT_samples [25]	95.6% (H)	80.4% (H)
Reduction_samples [25]	93.2% (H)	74.4% (H)
Transpose_samples [25]	84.3% (H)	68.0% (H)
Sorting_samples [25]	83.4% (H)	62.5% (H)
Gemm_samples [25]	98.5% (H)	12.8% (L)
ALBERT-train	88.7% (H)	31.8% (L)
BERT-inf	87.4% (H)	24.1% (L)
ViT-train	87.2% (H)	37.3% (L)
Whisper-inf [21]	87.1% (H)	23.8% (L)
BERT-train	85.5% (H)	34.4% (L)
Wav2Vec2-inf	80.2% (H)	25.2% (L)
ViT-inf	33.1% (L)	4.9% (L)
ResNet50-train	15.1% (L)	7.1% (L)
ResNet50-inf	10.2% (L)	2.7% (L)
Mobile-train	4.1% (L)	0.9% (L)
Mobile-inf	3.3% (L)	1.2% (L)

Table 1: Workloads employed, categorized by compute and memory bandwidth usage: (H)igh, (L)ow.

normalized using a min-max scaling method before training. We evaluate three different ML prediction techniques for our solution; see Section 5.1.

3.3 Optimal SM partition prediction (online)

At runtime, when a workload pair is encountered for execution, we use the trained throughput and power models to predict (without actually executing) the throughput sum and GPU power for the given pair under different SM partition configurations; we limit our configuration space to (10%,90%), (20%,80%), ..., (90%,10%). Based on our predictions, we choose the partition that maximizes the ratio of predicted throughput sum to predicted power (see Section 4 for more details). The workload pair is then executed by setting the `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` parameter in MPS to the chosen SM partition.

4 Methodology

We conduct our evaluation on a server in the Chameleon Cloud equipped with 2 Intel Xeon Gold 6230 CPUs, 128GB RAM, and a 32GB NVIDIA V100 GPU. We use PyTorch 1.13 and CUDA12.3 for our experiments.

For evaluation, we use a total of 16 workloads, consisting of 11 deep learning training and inference models (with a batch size of 2) and 5 compute-intensive NVIDIA CUDA CODE Samples [18] representing common scientific computations. Detailed workload characteristics, including compute and memory utilization, are provided in Table 1. By considering all workload pairs that fit within GPU memory, we generate 121 valid colocation pairs. For each pair, we evaluate 9 SM partitions: (10%,90%), (20%,80%), ..., (90%,10%). Each partition is treated as a separate data point, with its corresponding performance metrics recorded. In this workshop paper, we only consider pairs of workloads colocated together. With some effort, our approach can be extended to more than two workloads. However, as the size of GPU models continues to increase, we do not expect that too many colocated workloads will be needed to saturate a GPU while maintaining performance efficiency [4].

Our evaluation metric is **throughput per watt**, a commonly employed metric for system optimization [3, 10, 14] as it captures the trade-off between power consumption and performance. In our colocated setting, we specifically define throughput per watt as $(X_1+X_2)/(\text{GPU Power})$, where X_1 and X_2 denote the throughput of each workload when colocated. To estimate this metric under different SM partitions, we use the throughput and power prediction models presented in Section 3 to predict $(X_1 + X_2)_{pred}/\text{Power}_{pred}$. We *predict* the throughput per watt for all 9 SM partition configurations and employ, for execution, the partition that maximizes the predicted throughput per watt.

For comparison, we consider the *equal partition* strategy that employs the (50%,50%) SM partition for the colocated pair workloads. We also consider the impractical and offline *Oracle* strategy which picks the best (throughput per watt maximizing) SM partition (after the fact) among the 9 partitions we experiment with for each pair.

5 Evaluation

This section presents our experimental results. We first discuss our throughput and power prediction results under different ML models and training set sizes. Then we present our workload colocation results with throughput per Watt, highlighting the efficiency achieved by our solution compared to the equal partition and (offline) Oracle strategies.

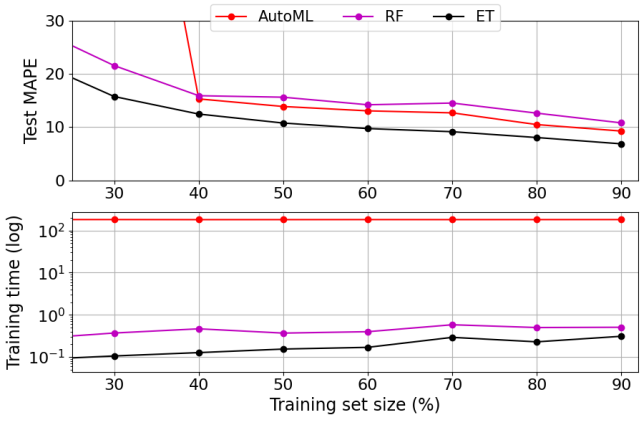
5.1 Throughput and power predictions

To assess the accuracy of our predicted throughput sum ($(X_1 + X_2)_{pred}$) and our predicted GPU power (Power_{pred}), we evaluate across all 9 SM partition configurations that we experiment with for all valid (121) colocated workload pairs. For the predictions, we consider three different ML models: (i) H2O Automatic Machine Learning (*AutoML*) [11]; (ii) Random Forest (*RF*); and (iii) Extra Trees Regressor (*ET*). AutoML is an automated ML framework that selects the best model from a set including Distributed RF, Gradient Boosting, Deep Learning, and ensemble models within a user-defined time constraint. We allocate 3 minutes for AutoML to select its optimal model.

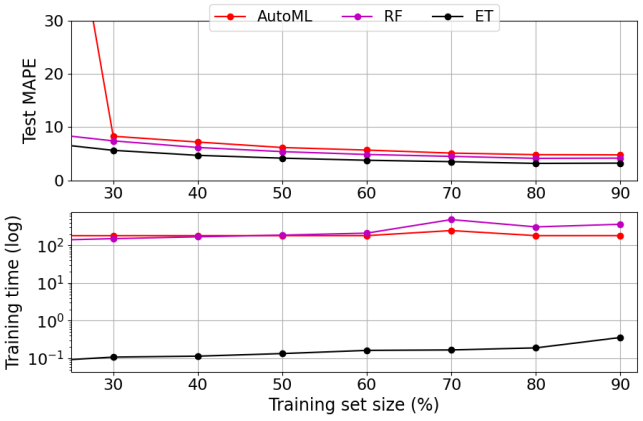
To evaluate ML model predictions, we test on different training set sizes. The full dataset (121 pairs, each under all 9 partition configurations) is split into 10 subsets. For each training dataset size (e.g., 70%), we randomly sample 7 subsets for training and 3 for testing, repeating this process 10 times to report averaged accuracy.

Colocated throughput sum predictions. Figure 1(a) shows (on top) the Mean Absolute Percentage Error (MAPE) of throughput sum predictions and (on bottom) the corresponding training time across different dataset sizes. Among the three models, Extra Trees (ET) consistently achieves the lowest MAPE, especially for smaller training sizes, while Random Forest (RF) performs the worst. ET outperforms RF due to reduced overfitting, as it selects split points randomly rather than optimizing them, making it more robust to limited training data. In contrast, both RF and AutoML are more prone to overfitting due to their complexity. AutoML also struggles with training set sizes under 40%, as its ensemble methods require more training data. Notably, ET is not part of the AutoML framework, yet maintains MAPE below 10% for training sizes above 50%. ET also has the smallest training time requirement, since it avoids exhaustive optimization of split points. AutoML understandably has high training time as it involves hyperparameter tuning and model selection overhead.

Colocated GPU power predictions. Figure 1(b) shows our GPU power prediction results. All models achieve better accuracy for power than for throughput, since GPU power



(a) Throughput sum prediction: $(X_1 + X_2)_{pred}$



(b) GPU Power prediction: Power_{pred}

Figure 1: MAPE and training time for colocated throughput sum predictions (left) and colocated GPU power predictions (right) for different ML models as a function of training dataset size.

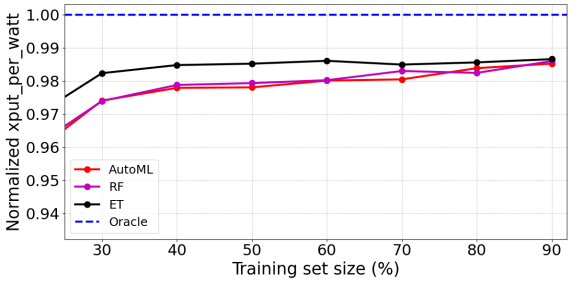


Figure 2: Normalized throughput per watt achieved by our solution using different ML models.

is naturally correlated with GPU-level metrics that we use as features (e.g., SM activity, memory usage). ET again achieves the lowest MAPE, although the difference between the models is much smaller compared to the throughput prediction results. In terms of training time, ET remains the quickest, while AutoML is the slowest; RF incurs high training time due to grid search.

Based on its low prediction errors and small training time for both colocated throughput sum and GPU power, we conclude that, for our experiments, *Extra Trees offers the best trade-off between accuracy and training efficiency.*

5.2 Workload colocation results

We now evaluate our solution in terms of achieved throughput per watt. For our solution, we used the methodology described in Section 4 to combine the predicted throughput sum and predicted GPU power to obtain the predicted throughput per watt, and then set the SM partition configuration that maximizes the predicted throughput per watt.

Throughput per watt under different ML models. Figure 2 shows the throughput per watt, normalized by that under Oracle (maximum throughput per watt across all 9 partition configurations), achieved by our solution as a function of training dataset size for different ML models. As expected, the normalized throughput per watt increases as the training size increases, reflecting improved model accuracy with more data. Interestingly, for all ML models, the achieved values are close and impressive—within 4% of the Oracle—even though the prediction accuracy results (from Figure 1) varied noticeably. This suggests that the profiled features used for training are sufficient for accurately estimating throughput per watt, even without the need for complex ML techniques. ET regressor typically obtains the best results, with the estimated throughput per watt consistently within 2% of that of the Oracle. Since *ET provides the most accurate predictions at all training set sizes and incurs the lowest training overhead*, we select ET as the prediction model for our solution.

Evaluating throughput per watt results. We now compare the throughput per watt obtained by our predictive solution (using ET as the ML model) with that obtained by (i) the *equal partition* strategy, which uses a (50%,50%) allocation to split the SMs between the pair of colocated workloads; and (ii) the offline *Oracle* strategy.

Our evaluation in Figure 3 shows the normalized (by Oracle) throughput per watt achieved by our solution and equal partitioning for all test set workload colocations. The '_bx' notation in the workload name indicates the batch size (x) and the '-train'/'-inf'/'-samples' refers to the workload type (training, inference, and CUDA CODE Samples [18], respectively). Each bar group represents the results averaged over all test set workload pairs of the form $(w, *)$, where workload w is unseen in the training set. This "unseen workload" scenario is more challenging than the randomized training split used in Figure 1 and 2, as we explicitly ensure that the training data does not include any colocated pair involving w. Unseen workloads can occur in practice either when a new workload is encountered at runtime (without any profiled data) or when the training set is limited in size. Finally, the bar groups are arranged in descending order of SM warp activity rate of workload w. All values are normalized with that under Oracle, similar to Figure 2.

We see that ***our solution outperforms equal partitioning in all cases, with a maximum relative gain of 60.6% and an average gain of 35%***. We find that our solution almost always selects the best configuration (the Oracle configuration), with ***only a 1.5% lower throughput per watt on average compared to the offline Oracle***. In general, our solution outperforms equal partitioning the most in throughput per watt for high-compute work-

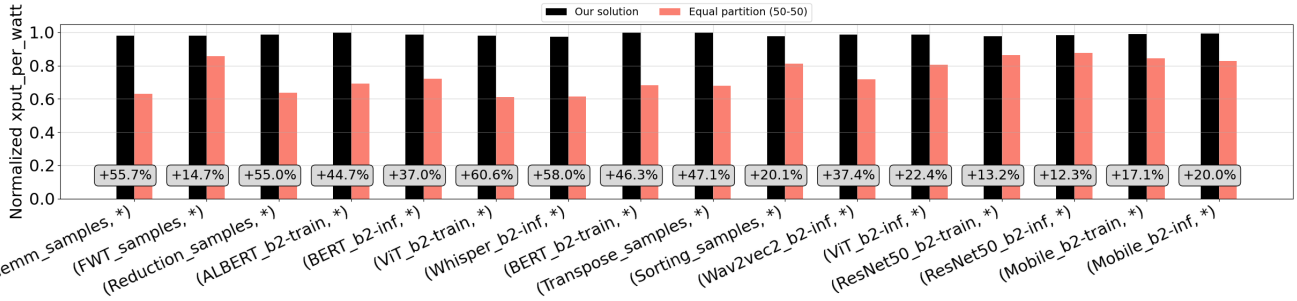


Figure 3: Normalized throughput per watt (normalized by that of Oracle) of our solution and equal partitioning (50%-50%) for all test cases, sorted by compute requirements. Relative gain is shown in text boxes.

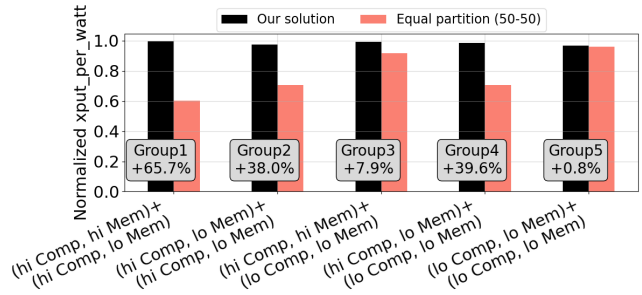


Figure 4: Throughput per watt results grouped by different workload resource categories.

loads (left of Figure 3), with the improvement decreasing for low-compute workloads (right of Figure 3). This is because high-compute workloads benefit from more dominant resource allocation (e.g., 90% SMs allocated), which significantly accelerates its throughput. The (static) equal partition strategy of 50%-50% cannot realize such opportunities.

Some workloads, however, do not follow the above discussed pattern. For instance, FWT_samples (FastWalshTransform) has the second-heaviest compute SM activity but only achieves a 14.7% improvement under our solution compared to equal partitioning. This is because FWT's performance does not improve significantly even with more SM allocated. As a result, the Oracle configuration is closer to equal partitioning, where allocating more SMs to the other workload in the pair is optimal. We find that 45% of the Oracle partitions for test workload pairs that include FWT as one workload are not 90%-10%, which results in our solution (which performs similarly to Oracle) being closer to equal partitioning. A similar argument holds for the Sorting_samples workload pairs (10th bar group). Nonetheless, these cases show a 14%–20% improvement in throughput per watt, suggesting that our solution can outperform equal partitioning even in cases where the Oracle partition is non-trivial.

Further analysis of results based on workload resource requirements. Figure 4 analyzes how our solution performs for workload pairs cateogrized by their resource requirements. We categorize our 16 workloads into four groups based on their SM activity and memory bandwidth utilization: (high compute, high memory); (high compute, low memory); and (low compute, low memory). The categories are defined using a 50% threshold for high (hi) and low (lo) compute and memory requirements, listed in Table 1. We then organize all colocated results from Figure 3 into these categories, as shown in Figure 4.

Our solution performs differently across workload resource categories, with *a high relative gain in throughput per watt of 65.7% (compared to equal partitioning) for the most resource-intensive pairs (Group 1) and a negligible 0.8% gain for pairs with both low compute and low memory requirements (Group 5)*. Group 1 has larger gain because the (high compute, high memory) category consists of high-throughput CUDA sample workloads, where optimal configurations consistently allocate 90% of SMs to those workloads to maximize overall throughput, making 50%–50% equal partition sub-optimal. In contrast, Group 5 shows very little gain as both colocated workloads are similarly light, and so allocating additional resources (beyond 50% SMs) to one of these light workloads at the expense of the other does not significantly improve throughput sum. So, equal partitioning works well here.

To isolate the impact of resource requirements of colocated workload pairs, we compare Groups 2 and 5, where the colocated workloads share the same resource category. The results show a substantial difference: our results for Group 2 (pair of high compute, low memory) outperform the baseline by 38%, with 82% of the Oracle configurations favoring a 90%–10% SM partition. In contrast, our results for Group 5 (pair of low compute, low memory) show minimal improvement, suggesting that a 50%–50% SM allocation partition suffices for this group. This suggests that workloads with higher compute demands should be carefully allocated SMs per their sensitivity.

Group 4 presents a non-trivial case: compared to other groups that include (low compute, low memory) workloads (i.e., Groups 3 and 5), our solution performs significantly better for group 4. This discrepancy occurs because Group 4 includes (high compute, low memory) workloads—primarily DL training workloads, whose throughput does not improve as significantly from greater SM allocation as the throughput of its colocated (low compute, low memory) light DL inference workloads. As a result, 92% of the Oracle configurations for this group are 10%–90%, prioritizing the lighter inference workloads. This observation highlights the importance of workload characteristics in determining energy efficiency. Combined with our earlier comparison of Groups 2 and 5, we conclude that *energy efficiency gains depend not only on compute and memory demands, but also on the nature of the colocated workloads.*

6 Conclusion and Future Work

This paper presents a prediction-based approach to identify the most efficient SM partition configuration for a workload pair when using MPS. Using a limited set of offline GPU system metrics profiled for training, our solution achieves a 12%–60% increase in throughput per watt, with an average improvement of 35%, compared to the equal partitioning baseline; further, our solution's performance is within 1.5% of that of the offline optimal strategy. As part of future work, we will expand our approach to multiple colocated workloads and extend our solution to multi-GPU setups.

7 References

[1] Nvidia multi-instance gpu v560, 2024. Official Documentation.

[2] Nvidia multi-process service v570, 2025. Official Documentation.

[3] Tahmid Abtahi, Colin Shea, Amey Kulkarni, and Tinoosh Mohsenin. Accelerating convolutional neural network with fft on embedded hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(9):1737–1749, 2018.

[4] Wenyan Chen, Zizhao Mo, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 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 '23, Denver, CO, USA, 2023.

[5] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 199–216, Carlsbad, CA, July 2022. USENIX Association.

[6] Marcus Chow, Ali Jahanshahi, and Daniel Wong. Krisp: Enabling kernel-wise right-sizing for spatial partitioned gpu inference servers. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 624–637, 2023.

[7] Paul Elvinger, Foteini Strati, Natalie Enright Jerger, and Ana Klimovic. Measuring gpu utilization one level deeper. `https://arxiv.org/abs/2501.16909`, 2025.

[8] Connor Espenshade, Rachel Peng, Eumin Hong, Max Calman, Yue Zhu, Pritish Parida, Eun Kyung Lee, and Martha A. Kim. Characterizing training performance and energy for foundation models and image classifiers on multi-instance gpus. In *Proceedings of the 4th Workshop on Machine Learning and Systems*, EuroMLSys '24, page 47–55, New York, NY, USA, 2024. Association for Computing Machinery.

[9] Hugging Face. The llama 3 models were trained on 15 trillion tokens with 24,000 gpus, 2023. Accessed: 2024-07-03.

[10] Brett Foster, Shubbhi Taneja, Joseph Manzano, and Kevin Barker. Evaluating energy efficiency of gpus using machine learning benchmarks. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 42–50, 2023.

[11] H2O.ai. *H2O.ai AutoML Documentation*, 2024. `https://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html`.

[12] Kawsar Haghshenas and Mona Hashemi. Eaco: Resource sharing dynamics and its impact on energy efficiency for dnn training, 2024.

[13] Bing-Shiun Han, Tathagata Paul, Zhenhua Liu, and Anshul Gandhi. Kace: Kernel-aware colocation for efficient gpu spatial sharing. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, SoCC '24, page 460–469, New York, NY, USA, 2024. Association for Computing Machinery.

[14] Xin He, Jiawen Liu, Zhen Xie, Hao Chen, Guoyang Chen, Weifeng Zhang, and Dong Li. Enabling energy-efficient DNN training on hybrid GPU-FPGA accelerators. In *Proceedings of the 35th ACM International Conference on Supercomputing*, ICS '21, pages 227–241, Virtual Event, 2021.

[15] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. MISO: exploiting multi-instance GPU capability on multi-tenant GPU clusters. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 173–189, San Francisco, CA, USA, 2022.

[16] NVIDIA Corporation. *NVIDIA System Management Interface*, August 2016. Version 367.38.

[17] NVIDIA Corporation. Cuda samples: fastwalshtransform. CUDA Toolkit Samples Repository, 2024. Accessed: March 22, 2025.

[18] NVIDIA Corporation. Cuda toolkit 12.8 samples. CUDA Toolkit Samples Repository, 2024. Accessed: March 24, 2025.

[19] NVIDIA Corporation. Nvidia data center gpu manager (dcgm). Online, 2025. Accessed: March 24, 2025.

[20] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Power-aware deep learning model serving with μ-Serve. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 75–93, Santa Clara, CA, USA, 2024.

[21] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[22] Foteini Strati, Xianzhe Ma, and Ana Klimovic. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 1075–1092, Athens, Greece, 2024.

[23] Tirth Vamja, Kaustabha Ray, Felix George, and UmaMaheswari C Devi. On the partitioning of gpu power among multi-instances, 2025.

[24] Alex Weaver, Krishna Kavi, Dejan Milojicic, Rolando Pablo Hong Enriquez, Ninad Hogade, Alok Mishra, and Gayatri Mehta. Granularity-and interference-aware gpu sharing with mps. In *Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '24, page 1630–1637. IEEE Press, 2025.

[25] zyjopensource. geepafs: Cuda samples. GitHub Repository, 2025. Accessed: March 30, 2025.