SMART-MIG: A Learning Framework for Scalable and Energy-Efficient GPU Scheduling

***Abstract*—The emergence of Multi-Instance GPU (MIG) technology enables us to run smaller machine learning models on partitions of a GPU rather than the entire device, thus improving utilization and reducing energy consumption, albeit with potential performance trade-offs. Meanwhile, the growing energy demands of GPU-equipped data centers motivate the development of online partitioning and scheduling schemes that not only ensure fast job processing but also achieve high energy efficiency. However, achieving energy–tardiness efficiency with manageable algorithmic complexity in large-scale scheduling remains a great challenge, due to the dual objectives of deciding on the GPU partitions and scheduling jobs onto the slices of the heterogeneous partitions. To address this challenge, we propose SMART-MIG, a parallel computing system that combines Mean-Field Multi-Agent Reinforcement Learning (MF-MARL) for large-scale MIG repartitioning with tailored heuristic algorithms for job scheduling. We demonstrate that the complexity of the repartitioning component remains constant even as the number of jobs and GPUs increases. We also establish theoretical lower bounds on energy consumption and tardiness to rigorously benchmark system performance. Finally, extensive experiments show that SMART-MIG improves the energy–tardiness efficiency by 18% compared to its corresponding static partitioning counterpart, while being only 27% above the theoretical lower bound on energy consumption.**

***Index Terms*—Scheduling, Energy-Efficiency, Repartitioning, Tardiness, GPU, MIG, PPO, Mean Field, Multi-Agent Reinforcement Learning, A100, Algorithms, Lower Bounds**

I. Introduction

In recent years, the increasing adoption of artificial intelligence (AI) and machine learning (ML), especially driven by large language models has advanced the deployment of GPU clusters. These clusters form an important component of modern data centers, and they often run longer-running training jobs and latency-sensitive inference jobs in a colocated manner to increase utilization and improve throughput. According to [1], the data centers consumed 4.4% of the total electricity in the United States in 2023 (without accounting for cryptocurrency), and this number is expected to increase to 12% in 2028. About 40% of this energy was used by computing systems such as CPUs and GPUs, and another 40% was used for cooling. The tremendous scale of this issue creates a pressing need for a GPU scheduling framework that balances energy efficiency with SLO compliance to create sustainable, high-performance AI infrastructure for the future.

There are several reasons for the high energy consumption of AI workloads on GPUs, including (i) underutilization of GPUs, (ii) fragmented or poor job allocations, (iii) fluctuations in demand, and (iv) the requirements of complex cooling systems. In this work, some of our scheduling algorithms are designed to address the first two issues by intelligently assigning jobs to slices, and our RL-based repartitioning scheme helps resolve the third issue by learning patterns in arrival rates.

For multi-tenant environments, NVIDIA recently introduced Multi-Instance GPU (MIG) technology in several of its GPUs, such as A100, H100, etc. MIG technology allows a GPU to be partitioned into up to seven isolated instances or slices with their own compute, cache, and memory resources. A scheduler for MIG has to make two intertwined sets of decisions: (i) which job to allocate to which slice of which MIG partition, and (ii) when and how to repartition a GPU to be more suitable for the job mix. Repartitioning adds a temporal dimension that can reduce energy at the cost of possible disruption of the job.

Prior work on GPU scheduling has emphasized throughput, fairness, utilization, and SLOs [2], while energy studies largely rely on DVFS or cluster-level schedulers that treat GPUs as uniform units. Such approaches overlook MIG, which enables dynamic resource splitting, which is particularly useful for today's smaller models like Phi-4 [3] and Gemma-2 [4]. Recent studies [5]–[8] explore MIG-based placement and repartitioning, showing the feasibility of runtime adaptation, but they primarily target utilization and performance rather than jointly addressing energy and performance trade-offs.

The repartitioning of MIGs is highly complex, as it must consider online decision-making, the nonlinear throughputs of workloads, and the trade-offs between repartitioning overhead and potential performance gains, all while managing energy efficiency. Furthermore, the nature of data centers necessitates scheduling at a massive scale – a large number of GPUs must handle extremely diverse jobs characterized by heterogeneity, high uncertainty, and large problem sizes. The challenge is to deliver high-quality solutions to this NP-hard problem (given that restricted versions of it are also NP-hard [9]) in seconds.

Previous studies have primarily relied on simple heuristics or restricted the number of reconfigurable MIGs to enable exhaustive search or basic linear programming (LP) formulations. However, these approaches are inadequate in highly uncertain environments, as they cannot effectively capture workload characteristics, adapt to diverse configurations, or scale to large jobs and GPU pools without severe performance degradation and unacceptable computation time. A recent work has explored reinforcement learning (RL) to address online scheduling, using Deep Q-Networks to learn task characteristics and maintain service quality [10]. However, since the computational complexity of traditional RL grows exponentially with the number of GPUs and jobs, such approaches

can only handle very small-scale scenarios (such as scheduling on a single GPU).

Motivated by these challenges, we pose a fundamental research question: **Is it possible to design an efficient, multi-objective and scalable MIG-based scheduling framework for industrial-scale computing that can adapt to uncertainty and real-time demands, while producing high-quality global optimization solutions in seconds?**

We address this challenge by introducing **SMART-MIG: S**cheduling with **M**ean-field Multi-**A**gent **RL** using **T**op-k sampling for **MIG**, an intelligent real-time scheduling system. The pipeline of SMART-MIG is depicted in Figure 1. It builds on the key observation that all MIGs are interchangeable; the scheduling process only needs to determine how many MIGs are assigned to each valid configuration, rather than tracking the details of each MIG. Accordingly, the input to the central controller is represented as the distribution of GPU and job states, while the output is a randomized policy that can be uniformly applied to all MIGs.

This formulation bounds the size of the input–output space, making it much less dependent on the number of GPUs and jobs. Within this tractable space, the strong representational power of neural networks enables the learning of highly performant and generalizable RL policies for repartitioning MIGs, regardless of the system scale. Once the configuration of each MIG is decided, since the subsequent multi-machine job scheduling problem presents a complex action space, we employ a carefully designed efficient heuristic algorithm that achieves competitive performance.

To rigorously evaluate the system performance, we investigate a theoretical lower bound for the MIG scheduling problem in terms of energy and tardiness, where tardiness is defined as the maximum of 0 and the difference between the job completion time and the deadline. (In other words, jobs that finish before their deadlines have 0 tardiness, while jobs that are late by x units of time, have tardiness of x.) We then compare algorithmic outcomes with the corresponding bounds, which substantially reduces evaluation biases arising from dataset differences. This contribution establishes **a standardized benchmark** for fair and consistent assessment.

In summary, our contributions are as follows.

1) We propose **SMART-MIG**, a large-scale MIG scheduling framework that integrates methodologies from ML and OR. The framework consists of two dimensions:
   a) We employ Mean-Field Multi-Agent Reinforcement Learning (MF-MARL), suitable for large-scale problems, together with Top-k sampling to repartition GPUs.
   b) We design EDF-based scheduling algorithms for assigning jobs to MIG slices across multiple GPUs (without repartitioning), exploiting both job throughput characteristics and the MIG's concave power curve.
2) We derive energy and tardiness lower bounds that serve as a reference for evaluating our scheduling policies.
3) We observe how energy and tardiness vary when we increase the number of GPUs and note diminishing returns in the trade-off.
4) We conduct extensive experiments against theoretical lower bounds to validate the effectiveness of our method. Under high job arrival rates, the traditional no-partition GPU model rapidly deteriorates in performance. In contrast, our static repartitioning scheduler achieves robust results, operating within around 30% of the energy and tardiness lower bounds on average. Moreover, incorporating MF-MARL–based dynamic repartitioning yields an additional 18% improvement in energy–tardiness efficiency. Importantly, our method demonstrates consistent advantages across diverse workloads: even under low arrival rates, it reduces tardiness by 47% compared to the no-MIG baseline.

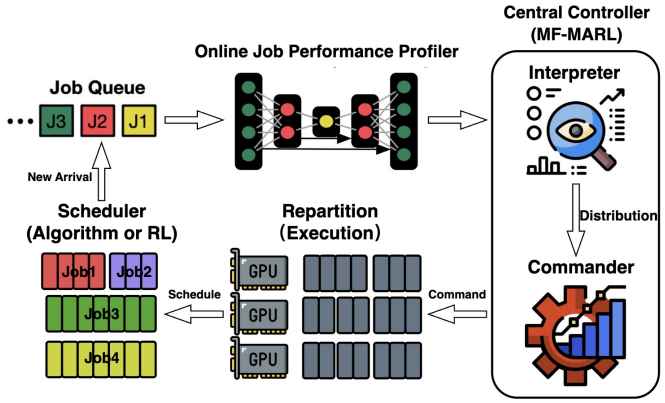

Fig. 1. Method Pipeline Overview. For each task, we extract the job profile, then deploy the MF-MARL model at the central controller to read the state distributions of jobs and MIGs. The controller issues repartitioning instructions, after which a scheduling algorithm assigns jobs to different slices.

II. Related Work

In this section, we further elaborate on the related work by categorizing it and noting our added contributions.

A. Energy Aware GPU Scheduling for AI/ML Workloads

GPU scheduling has been widely studied in both single-node and cluster contexts. Previously, Gandiva [11] introduced cluster scheduling for heterogeneous workloads with time-sharing and job packing to improve utilization by making use of job migration, while recent surveys such as [2] highlight scheduling approaches for AI training and inference that focus on meeting performance targets (SLOs) and efficient job placement in modern GPU data centers. DVFS [12] and PowerFlow [13] reduce energy consumption by integrating energy budgets into scheduling, enabling faster job completion within power limits. However, most of these methods treat GPUs as static resources and overlook dynamic partitioning capabilities like NVIDIA's MIG. Our work builds on these ideas by addressing tardiness while explicitly prioritizing energy efficiency in dynamically reconfigurable GPU environments.

B. MIG-Aware Scheduling and Repartitioning

There has been extensive research recently in MIG-based scheduling. MISO [5], for instance, makes use of Multi-Process Service (MPS) to decide the best MIG configurations by profiling the job throughputs before making costly switches. MIG-Serving [6] looks at deep neural network serving as a scheduling challenge where GPUs can be reshaped on the fly with a two-phase algorithm. MIGER [7] combines MIG and MPS within a GPU to better handle a mixture of online and offline jobs. Other studies [8] explore scheduling while accounting for PCIe bandwidth limits or allowing dynamic layout changes, while mapping out MIG's valid setups and practical repartitioning constraints.

[14] introduces a 3-phase approach called FAR that aims to optimize for makespan, whereas [15] provides a method of first assigning jobs to slices and then rearranging them as needed (along with repartitioning) in order to minimize the number of GPUs. [16] also optimizes for minimizing the number of GPUs while maintaining the SLO requirements.

C. RL for Resource Management

RL has often been used in resource management, starting with [17]. Mean Field Multi Agent RL gives tractable approximations for large populations of agents (here, GPUs) [18], and has been used for mean field games [19], while Proximal Policy Optimization (PPO) provides stable on-policy learning [20]. We use them together to get scalable, feedback-driven repartitioning policies in a collection of MIGs.

D. Lower Bounds in Scheduling

Lower bounds help us benchmark the performance of our scheduling policies. There are several classical results to bound flow time and tardiness under various scheduling policies [21]. We derive new lower bounds for energy and tardiness, where the former reduces to makespan minimization, and the latter employs a time-indexed mixed integer program (MIP).

III. Background

In this section, we briefly explain the existing technology and techniques used in this paper.

A. Multi-Instance GPUs

We first introduce Multi-Instance GPU, a form of resource sharing in GPUs as an alternative to Multi-Process Service (MPS). We focus on the A100-40 GB GPUs, but our techniques can be extended to other similar MIG-enabled GPUs.

NVIDIA's Multi-Instance GPU (MIG), available on the A100-40GB GPU, partitions the device into up to seven independent and isolated slices. A MIG-enabled GPU supports several partitioning configurations by fusing slices into groups of sizes 1, 2, 3, 4, or 7. Each slice is allocated its dedicated hardware resources, including streaming multiprocessors, memory, and cache, ensuring predictable performance and strong fault isolation. This makes MIG essential for multi-tenant cloud environments and workloads requiring guaranteed Quality of Service (QoS), as a fault in one instance does not impact others. In this work, we focus on configurations that fully utilize the 40GB memory capacity of the A100-40GB (Figure 2), while ignoring redundant permutations of identical slices in different orders.

Config	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
1	7g.40gb						
2	4g.20gb				3g.20gb		
3	4g.20gb				2g.10gb		1g.10gb
4	4g.20gb				1g.5gb	1g.5gb	1g.10gb
5	3g.20gb				3g.20gb		
6	2g.10gb		2g.10gb		3g.20gb		
7	2g.10gb		1g.5gb	1g.5gb	3g.20gb		
8	1g.5gb	1g.5gb	1g.5gb	1g.5gb	3g.20gb		
9	2g.10gb		2g.10gb		2g.10gb		1g.10gb
10	2g.10gb		2g.10gb		1g.5gb	1g.5gb	1g.10gb
11	2g.10gb		1g.5gb	1g.5gb	1g.5gb	1g.5gb	1g.10gb
12	1g.5gb	1g.5gb	1g.5gb	1g.5gb	1g.5gb	1g.5gb	1g.10gb

Fig. 2. Nontrivial Configurations of A100-40GB MIG

Complementing MIG is the Multi-Process Service (MPS), a software technology that operates at a different level. While MIG constructs safe hardware slices for different users or large tasks, MPS can be enabled within a single MIG slice or on the entire GPU. It allows multiple CUDA processes from a single user to share that instance's resources concurrently, reducing kernel launch overhead and maximizing utilization.

B. Proximal Policy Optimization

In the actor-critic framework, the critic estimates the value function to guide updates, while the actor learns the policy that maps states to actions. Building on this framework, the policy-gradient RL algorithm Proximal Policy Optimization (PPO) optimizes a clipped surrogate objective function to iteratively improve the agent's policy. This approach balances exploration and exploitation while avoiding excessively large policy updates that could destabilize training.

C. Multi-Agent Reinforcement Learning

We consider a cooperative MARL setting with N agents and discount factor γ. At each timestep t, every agent observes the global state $\boldsymbol{s}_t \in S$, where $S = S_1 \times \cdots \times S_N \times S_{sys}$ and S_i denotes the local state of agent i, and S_{sys} is the system state. Each agent then selects an action from its local action space A_i, yielding the joint action $\boldsymbol{a}_t = (a_t^1, \cdots, a_t^N) \in A = A_1 \times \cdots \times A_N$. The system transitions according to a stochastic kernel, and all agents share a joint reward $r_t^i(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The objective is to find a Pareto-optimal joint policy that maximizes the long-term expected reward:

$$\boldsymbol{\pi}^\star = \arg\max_{\boldsymbol{\pi}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]. \tag{1}$$

The joint state and action spaces in MARL grow exponentially with the number of agents N, because:

$$|S| \times |A| = \left(\prod_{i=1}^{N} |S_i|\right) \times \left(\prod_{i=1}^{N} |A_i|\right) \times |S_{sys}|. \quad (2)$$

This is the well-known curse of dimensionality. Such a rapidly growing state-action space makes data sampling extremely difficult (there can be millions of states and thousands of agents in practice), and a huge amount of data is required for policy evaluation, which makes such algorithms extremely difficult to scale. In practical data center scheduling, where large numbers of MIGs and jobs must be managed, this challenge motivates us to adopt advances from operations research, specifically Mean Field MARL [18].

D. Mean Field MARL

If all agents are identical, indistinguishable, and interchangeable, that is, we care only about whether an appropriate action is taken, rather than which agent takes it, then we can adopt the Cooperative Mean Field MARL (MF-MARL) framework. In this framework, all agents share the same state space S and action space A. The anonymity of agents allows us to represent their state and action information as distributions, i.e., the number of agents occupying each state. Each agent's decision depends on other agents only through the empirical distribution of their state–action pairs. Denote $\mathcal{P}(S)$ and $\mathcal{P}(A)$ as the probability measure spaces over the state space S and the action space A, respectively. The empirical distribution of the states is $\mu_t(s) = \frac{\sum_{j=1}^{N} \mathbf{1}(s_t^j = s)}{N} \in \mathcal{P}(S)$ and the empirical distribution of the actions is $\nu_t(a) = \frac{\sum_{j=1}^{N} \mathbf{1}(a_t^j = a)}{N} \in \mathcal{P}(A)$.

Formally, at each timestep t, the central controller observes the state distribution μ_t and outputs a policy $\pi_t(\cdot, \mu_t)$, which is a mapping from S to $\mathcal{P}(A)$. Since all agents are interchangeable, we consider the representative agent who selects an action $a_t \sim \pi_t(s_t, \mu_t)$ based on its local state s_t, receives a reward $r_t(s_t, \mu_t, a_t, \nu_t)$, and transitions to the next state $s_{t+1} \sim P_t(s_t, \mu_t, a_t, \nu_t)$. Here, both the transition probability P_t and the reward r_t depend on the state and action distributions $\nu_t(\cdot) := \int_{\mathcal{S}} \pi_t(s, \mu_t)(\cdot)\, \mu_t(s)\, ds$.

For notational simplicity, let $h_t(\cdot) = \pi_t(\cdot, \mu_t)$, and let $\mathcal{H} := \{h : S \to \mathcal{P}(A)\}$ be the function space of $h_t(\cdot)$. The policy learned by the central controller thus belongs to the space

$$\Pi := \{\pi = \{\pi_t\}_{t=0}^{\infty} \mid \pi_t : \mathcal{P}(S) \to \mathcal{H} \text{ is measurable}\}. \quad (3)$$

Importantly, the input of the central controller is the space $\mathcal{P}(S)$, and the output is $\mathcal{H} = \mathcal{P}(A)^{|S|}$, which implies that the dimensionality of the input–output mapping is independent of the number of agents. In fact, prior work [22] shows that under Pareto-optimality criteria, cooperative MF-MARL can approximate cooperative MARL up to an error of $\mathcal{O}(\frac{1}{\sqrt{N}})$.

IV. Problem Formulation and Methodology

Let J be a collection of n training or inference workloads (represented as integers in $[n]$). They arrive in an online fashion such that a workload $j \in J$ has a release time of r_j, a soft deadline of d_j, and $p_{j,k}$ is the processing time of workload j on a MIG slice of size k, where $k \in \{1, 2, 3, 4, 7\}$. The processing time of each online job could be obtained from a job performance estimator such as MISO [5], but in this work, they are obtained from a probability distribution (along with the deadlines). There is a collection of m MIG-enabled A100-40GB GPUs, indexed in $[m]$. The goal is to decide which job to allocate to which GPU and to which slice at any point in time (with preemption being permitted). Given the partitioning and scheduling strategy, we compute the total energy consumption e by making use of the concave power characteristics of MIG on A100-40GB at 250W power cap. The precise values are $P = [40, 119, 160, 205.3, 243.9, 247.7, 248.5, 248.5]$, where the i-th entry in this 0-indexed list corresponds to i MIG slices being utilized [10]. The average tardiness t is defined as $(\sum_{i=j}^{n} \max(C_j - d_j, 0))/n$, where C_j is the completion time of job j. This problem models a data center, where the jobs are arriving in real time, and the decision to repartition and schedule on each GPU needs to be made in order to optimize for both energy and performance.

Most workloads consist of linear and sublinear (we consider capped jobs as sublinear) throughput jobs [6], [23] scaling with slice size, as shown in Figure 3. An effective GPU repartitioning and scheduling strategy can allocate sublinear jobs appropriately, avoiding oversized slices that waste resources and undersized slices that cause delays. To analyze SMART-MIG under different sublinearity levels of throughput, we define the degree of sublinearity for a group of jobs as the average of their throughputs across 1g, 2g, 3g, 4g, and 7g slices, effectively converting the group into a single "average job". If the group were fully linear, its throughput curve would form a straight line with slope equal to the throughput on a 1g slice; we denote this slope as k_{linear}. For any real group of jobs, we fit a line to its actual throughput curve by minimizing the mean squared error, and denote the slope of this fitted line as $k_{sublinear}$. Clearly, for the same group of jobs, the slope of the fitted line is always smaller than the slope of the linearized throughput line. We denote $ss = \frac{k_{sublinear}}{k_{linear}} \in [0, 1]$ as the sublinearity score for the group of jobs.

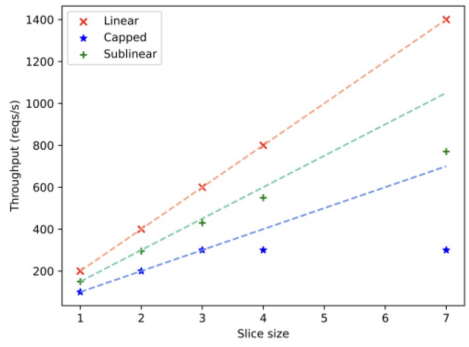

Fig. 3. Thoughput curves of different types of jobs

At the same time, the power consumption of MIGs grows in a concave manner with the number of active slices as seen in [10], which motivates us to maximize GPU utilization to

reduce energy costs. These insights demonstrate the feasibility of designing a system that optimizes for both energy and tardiness. To this end, we aim to minimize the ET value across simulations, as defined in [10]:

$$ET = \frac{1}{N}\sum_{k=1}^{N}\frac{ae_k + t_k}{a+1}, \tag{4}$$

where N is the number of simulations, e_k, t_k are the total energy and the average tardiness of the simulation $k \in [N]$, and a is a scaling parameter that measures the relative importance of energy and tardiness. Let $\bar{e}$ and $\bar{t}$ be the average of the total energy and average tardiness over N simulations, respectively.

A. Tardiness-Energy Curves with Increasing Number of GPUs

The first point to consider is the number of GPUs required to support the maximum *load* on the GPU system while preventing resource waste. We experiment by increasing the number of GPUs incrementally. We first observe the Pareto curves of $\bar{t}$ versus $\bar{e}$ as the number of GPUs increases at various arrival rates (Figure 4) by scheduling jobs using any given algorithm. As expected, we observe diminishing returns with a higher

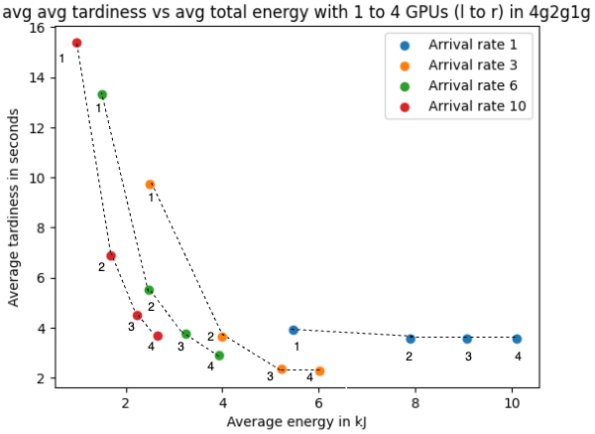

Fig. 4. Tardiness-Energy curves with varying arrival rates

number of GPUs. In particular, as we continue to increase the GPU count, the energy keeps increasing with incrementally smaller tardiness savings. The number of GPUs required for scheduling (which increases with increasing arrival rate) can be estimated from the knee of the plots (similar to the idea in [24], but in a different context).

B. SMART-MIG Overview

The general workflow of SMART-MIG is as follows. Whenever a new job arrives, we first obtain its profile, which includes the expected completion time in slices of different sizes. In a typical workflow, a throughput profiler such as MISO [5] can be used, but here we generate them from distributions based on real-world data as explained in Section VI. Next, when a new job arrives or an existing job completes (or the controller may be invoked less frequently, for example, once every five arrival or completion events to reduce repartition overhead), the MF-MARL model deployed at the central controller reads the state distributions of jobs and MIGs and outputs reallocation instructions to guide MIG repartitioning. Once the configuration of each MIG is fixed, the problem reduces to a multi-machine scheduling problem, where we employ a carefully designed scheduling algorithm to assign jobs across different slices of different GPUs. We first present scheduling algorithms under static configurations in Section IV-C, followed by the central controller that directs dynamic MIG repartitioning in Section IV-D.

C. Scheduling Algorithms

We design algorithms that leverage the throughput characteristics of jobs and the concave power curve of GPUs when making assignment decisions. Unlike the similar single-machine problem studied in [10], the presence of multiple GPUs significantly complicates the scheduling decisions, which necessitates the use of more intelligent scheduling algorithms. The pseudocodes of algorithms depicted here outline the choices of slices and GPUs, and not the entire scheduling algorithms. The ordering of the jobs in each of them is done in an earliest deadline first (EDF) manner.

We consider the following algorithms:

1) **EDF Most Packed (EDF)**: The jobs in the ready queue are sorted in increasing order of deadlines, and each job is assigned one at a time to the slowest slice of the most fractionally packed GPU that still finishes the job on time. If all the GPUs are occupied, then the job is assigned to the GPU slice with the fastest job completion time.

Algorithm 1 EDF Most Packed (EDF)

```
Sort GPUs by occupied slices (desc), Jobs by deadline (asc)
for job in Jobs, gpu in GPUs do
    for slice in free slices of gpu do
        t ← now + p_{job,slice}
        if t ≤ d_job then
            assign job to (gpu, slice); return
        end if
        if t < best_finish then
            best_choice ← (gpu, slice)
        end if
    end for
end for
assign job to best_choice (earliest finish)
```

2) **EDF Marginal ET (MET)**: The jobs in the ready queue are sorted in increasing order of deadlines, and each job is assigned to the GPU that has the lowest marginal ET gain, where ET is defined in equation 4. The marginal ET is calculated by assuming that no more jobs will be scheduled on that GPU.

Algorithm 2 EDF Marginal ET

Sort Jobs by deadline (asc)
for job in Jobs **do**
  tardiness $\leftarrow \max(0, now + p_{job,slice} - d_{job})$
  marginal $\leftarrow$ extra energy if the slice used until finish
  chosen $\leftarrow \arg\min_{s \in \text{free slices}} \frac{a \cdot \text{marginal} + \text{tardiness}}{a+1}$
  assign job to chosen slice
end for

3) **Categorical EDF (CEDF)**: This algorithm classifies the jobs into four categories, ranging from least sublinear to most sublinear. The GPUs are sorted in decreasing order of the average slice size. The jobs are then sorted first by these categories and within categories by deadlines. They are then packed GPU-wise in the slowest slice that completes the job on time, otherwise the fastest slice. The procedure for choosing the slice is similar to EDF Most Packed, but the GPUs are ordered by average slice size rather than the fractional utilization.

Algorithm 3 Categorical EDF (CEDF)

procedure CLASSIFYJOB(job)
$$\begin{cases} \text{if } thr(4g) < 0.8 \cdot thr(7g), thr\_class = 1 \\ \text{elif } thr(2g) < 0.8 \cdot thr(7g), thr\_class = 2 \\ \text{elif } thr(1g) < 0.8 \cdot thr(7g), thr\_class = 3 \\ \text{otherwise}, thr\_class = 4 \end{cases}$$
end procedure
Sort GPUs by average slice size (desc)
Sort Jobs by thr_class and deadline (asc)
Assign Jobs using steps 2-13 of Algorithm 1

D. MF-MARL based Central Controller

As the MIGs are identical, indistinguishable, and interchangeable, the key is to check whether a valid MIG configuration exists for a given task, not which MIG executes it. The jobs share the same property: we are concerned with the characteristics of pending jobs, not their individual identities. So, for any job, as long as the GPU configuration is appropriate, the execution outcome remains the same regardless of which GPU processes it; likewise, given a fixed GPU configuration, jobs with the same characteristics yield identical results.

This symmetry motivates the use of Mean-Field MARL, where the state information of all GPUs and jobs can be represented as distributions. Concretely, GPUs can take on 12 possible configurations, denoted as $c_1, \cdots, c_{12}$. The current configuration of each MIG defines its state. We define this moment of a job arrival or completion as step t. Assuming there are m MIGs in total, the state of the i^{th} MIG is denoted as $s_t^{i,cfg}$ for each $i \in [m]$. The state distribution of the MIGs can then be computed as: $\mu_t^{MIGs}(c_q) = \frac{\sum_{i=1}^{m} \mathbf{1}\left(s_t^{i,cfg} = c_q\right)}{m}$.

The state of a job is defined by its duration and deadline, denoted for the j^{th} job as $(s_t^{j,du}, s_t^{j,dl})$ in step t. Since these states are continuous, they must be discretized in order to be represented as distributions. Specifically, let $du_1, \cdots, du_V$ denote the discretized intervals of job duration. Then $s_t^{j,du} = du_v$ if and only if the duration of job j falls into interval du_v. Similarly, let $dl_1, \cdots, dl_O$ represent the discretized intervals of job deadlines. To simplify the problem, we consider the duration distribution and the deadline distribution of jobs separately. They can be computed as: $\mu_t^{Du}(du_v) = \frac{\sum_{j=1}^{n} \mathbf{1}\left(s_t^{j,du} = du_v\right)}{n}$ and $\mu_t^{DL}(dl_o) = \frac{\sum_{j=1}^{n} \mathbf{1}\left(s_t^{j,dl} = dl_o\right)}{n}$, assuming there are n jobs,

Following the MF-MARL paradigm described in Section III-D, the central controller takes the above distributions as input and outputs a policy h_t applicable to all MIGs, guiding their repartitioning. For any given MIG, this policy takes its current configuration as input and returns a probability distribution over the next possible configurations. The next configuration of the MIG is then sampled from this distribution. The form of the policy can be expressed as follows:

$$h_t = \begin{pmatrix} p(c_1 \mid c_1) & \cdots & p(c_{12} \mid c_1) \\ \vdots & \ddots & \vdots \\ p(c_1 \mid c_{12}) & \cdots & p(c_{12} \mid c_{12}) \end{pmatrix}, \tag{5}$$

where $p(c_q \mid c_p) = p(s_{t+1}^{i,cfg} = c_q \mid s_t^{i,cfg} = c_p)$ for all $i \in [m]$.

After the central controller issues repartitioning instructions, the configurations of all MIGs are updated based on the transition distribution corresponding to its current state. We adopt top-k sampling, a technique widely used in large language models, to improve the stability of the system. Specifically, top-k sampling selects the k configurations with the highest probabilities from the distribution, re-normalizes their probabilities, and then randomly samples the next configuration from this restricted set. The system then applies the scheduling algorithm to assign jobs across different slices, and the GPUs process these tasks until either a new job arrives or an existing job is completed. At this point, the central controller receives a reward defined as

$$r_t = -\frac{\text{tardiness}_t + a \times \text{energy\_consumption}_t}{1+a}, \tag{6}$$

where tardiness_t and $\text{energy\_consumption}_t$ denote the newly incurred job delay and energy cost during this processing interval. The parameter a is set empirically such that the weight of the tardiness term is approximately twice that of the energy consumption term in order to prioritize tardiness (after normalizing both to the same order of magnitude). Summing over all timesteps, the cumulative reward is set to the negative of the total ET value. Therefore, the reward of the central controller is directly tied to the downstream objectives. This stepwise reward formulation alleviates the issue of reward sparsity, thereby facilitating effective learning.

Then, we use PPO to train the model to maximize the expected cumulative reward. We know that there is a 4-second repartitioning penalty [5], which is incorporated into the model indirectly by reducing the deadlines by this amount.

V. LOWER BOUNDS

In this section, we show lower bound results for energy and tardiness. We first *linearize* the sublinear jobs to their linear

counterparts and then show that running each of them on $7g$ MIGs yields lower bounds for both energy and tardiness.

For the energy lower bound, we reduce our problem to a makespan minimization problem that is polynomially solvable, and for the tardiness lower bound, we use a mixed integer program, which is numerically approximated (it is NP-hard).

Definition 1. *For a job j with processing time $p_{j,1}$ on $1g$, we define its linearization as a job $L(j)$ with processing times $p_{j,i} = p_{j,1}/i$ for each $i \in \{2, 3, 4, 7\}$.*

For a collection of jobs J, let $L(J)$ represent the corresponding collection of linearized jobs.

A. Energy Lower Bound

Lemma 1. *Any schedule Π with a combination of linear and sublinear throughput jobs J can be converted to a corresponding schedule Π' on the linearized job set $L(J)$ without increasing the energy.*

Proof. The above result follows from the power curve's concavity and the throughput curves' sub-linearity, implying that the jobs' linear extensions will consume less energy running on a corresponding schedule with the same starting times as the original schedule (and will finish no later). □

From now on, we will consider all jobs to be linear, and let the energy used by a schedule Π' on a linearized job set $L(J)$ be $E^{\Pi'}(L(J))$.

Lemma 2. *Any schedule Π' on a linearized job set $L(J)$ on m machines with any configurations can be converted to a schedule Π'' on m $7g$ machines with no increase in energy.*

Proof. Consider the first machine without loss of generality, with Π' restricted to it being denoted as Π'_1. Let the finishing time of Π' be T. For ease of calculation, we deduct P_0T from the total energy $E^{\Pi'_1}(L(J))$ and account for it later.

We divide the schedule Π' on the first machine into k segments, where a new segment begins whenever a job starts or completes (or there is a reconfiguration). Consider a segment $i \in [k]$, and let its duration be t_i and starting time and finish times be s_i and f_i respectively. Let there be l jobs running on slices of sizes y_{ij} for each $j \in [l]$, such that the $\sum_j y_{ij} = S_i$. The power consumed by these jobs is $P_{S_i}t_i$. Since the idle power must be deducted, the active power during this time is $(P_{S_i} - P_0)t_i$. Construct a corresponding segment on $7g$ where the parts of jobs in segment i are scheduled consecutively, one after the other. Its duration will be $\frac{S_i t_i}{7}$ which is no greater than t_i. The active power consumed will be $(P_7 - P_0)\frac{S_i t_i}{7} = P_7 t_i \frac{S_i}{7} + P_0 t_i \left(1 - \frac{S_i}{7}\right) - P_0 t_i \leq (P_{S_i} - P_0)t_i$, using the concavity of the power curve. Finally, since each new segment takes no more duration than the original segment, the reduction is feasible and cannot result in an increase in the active energy by summing up these segments. The idle power is P_0T, as we are constructing a total schedule Π''_1 of duration T, so it can be added in the end. Finally, repeating this process for all m machines proves the lemma. □

Lemma 3. *To compute a lower bound on the total energy, we calculate the energy of a minimum makespan schedule using jobs $L(J)$ on m $7g$ machines.*

Proof. It can be shown that for jobs $L(J)$ running using any schedule Π on m $7g$ machines, the total energy consumption is given by adding the active energy to the idle energy:

$$E^{\Pi}(L(J)) = \sum_{j \in J} \frac{p_j}{7} P_7 + (mC^{\Pi} - \sum_{j \in J} \frac{p_j}{7}) P_0, \qquad (7)$$

where $[n]$ is the index set of jobs and C^{Π} is the makespan of a given schedule Π. Replacing C^{Π} with the minimum makespan C^* in (7) yields the energy lower bound (the active energy cannot be decreased, and minimizing the makespan minimizes the idle energy). Note that C^* can be computed in $O(n^2)$ time using the staircase algorithm for makespan minimization with preemption on uniform machines from [25]. □

B. Tardiness Lower Bound

It can be shown that Lemmas 1 and 2 hold for tardiness as well (using similar proofs). In this subsection, we replace J with $L(J)$ and derive a tardiness lower bound result for $L(J)$. We frame our problem as a parallel identical machine scheduling problem and develop a mixed integer program (MIP) with a just-in-time formulation.

Let there be m machines and n linearized jobs. For each job, $j \in [n]$, let the release time be r_j, the processing time be p_j, and the deadline be d_j. After setting an ϵ that indicates time step size, we set the maximum time horizon at $\tau = \lfloor (\max_j r_j + \sum_j p_j)/\epsilon \rfloor$. The time horizon is discretized into time slots from 0 to τ with time slot s corresponding to $[s\epsilon, (s+1)\epsilon)$.

- Let $x_{j,s} \geq 0$ be the amount of job j processed in time slot s for each $j \in [n]$ and for each $s \in [0, \tau]$.
- Let $y_{j,s} \in \{0, 1\}$ be a binary variable that is 1 if job j has been completed until time slot s, and 0 otherwise.
- Let C_j be the completion time of job j for each $j \in n$.
- Let $T_j = \max\{0, C_j - d_j\}$ be the tardiness of job j for each $j \in [n]$.

Objective: $\min \sum_{j=1}^{n} T_j$

Constraints:

1) Processing requirement: $\sum_{s=0}^{\tau} x_{j,s} \leq p_j \quad \forall j \in [n]$.
2) Machine capacity: $\sum_{j=1}^{n} x_{j,s} \leq m\epsilon \quad \forall s \in [0, \tau]$.
3) Job processing rate: $x_{j,s} \leq \epsilon \quad \forall j \in [n], \forall s \in [0, \tau]$.
4) Release time: $x_{j,s} = 0 \quad \forall j, \forall s < \lfloor r_j/\epsilon \rfloor$.
5) Completion indicator: $y_{j,s} \leq y_{j,s+1} \quad \forall j, \forall s \in [0, \tau - 1]$ and $y_{j,t} \leq \frac{\sum_{s=0}^{t} x_{j,s}}{p_j} \quad \forall j, \forall t \in [0, \tau]$.
6) Completion time: $C_j = \epsilon \sum_{s=0}^{\tau} (1 - y_{j,s}) \quad \forall j \in [n]$.
7) Tardiness: $T_j \geq C_j - d_j,\ T_j \geq 0 \quad \forall j \in [n]$.
8) Domain constraints: $y_{j,s} \in \{0, 1\}, \quad x_{j,s} \geq 0 \quad \forall j \in [n], \forall s \in [0, \tau]$.

The constraints ensure $y_{j,t}$ is non-decreasing and switches from 0 to 1 when $\sum_{s=0}^{t} x_{j,s} = p_j$ because the tardiness minimization in the objective pushes the indicator to be 1 as soon as possible (if the completion time is after the due date).

$C_j = \epsilon \sum_{t=0}^{\tau}(1 - y_{j,t})$ counts the number of slots where $y_{j,t} = 0$, multiplied by the slot width, equaling ϵ times the smallest t where $y_{j,t} = 1$, which approximates when job j completes. This procedure will lower-bound job tardiness with an error up to twice the time slot width (errors arising from both release time and completion time discretization). Hence, the lower bound can underestimate the actual tardiness by at most $2n\epsilon$ when accounting for n jobs.

However, in practice, for both energy and tardiness, there is additional error from the jobs being sublinear in throughput. But since most of the common inference jobs are observed to be of linear throughput [6], we observe that we are empirically not too far from both the lower bounds (refer to Section VI).

VI. Experimental Results

Here, we explain the processes that generate our workloads and the parameters that we vary to perform experiments on MIGs to demonstrate improvements in energy and tardiness.

A. Experimental Settings

The jobs and machines are as defined previously. For the MIG experiments without repartitioning, we consider the release times of the workloads to be simulated from a fixed-rate Poisson arrival process. We then split them into training and inference jobs. The processing time characteristics of the workloads are generated from a lognormal distribution [26] for training jobs and an exponential distribution for inference jobs [27], with their slicewise processing times being determined by using the throughput curves of Resnet-50 and BERT-Base from [23] and [6]. Since deadline information is not generally available in data center traces, we model it as $d_j \sim Unif(1, 1.5) * p_{j,7}$, which is similar to the deadline distribution in [28].

For our repartitioning experiments, we generate workloads with a base arrival rate that varies during the day [10], [29] as in Figure 5, which is multiplied by a multiplier. Next, we split each of these workloads into training and inference types in a fixed proportion, and again consider their throughput characteristics from the experiments in [6], [23], [30], like in the no-repartitioning case.

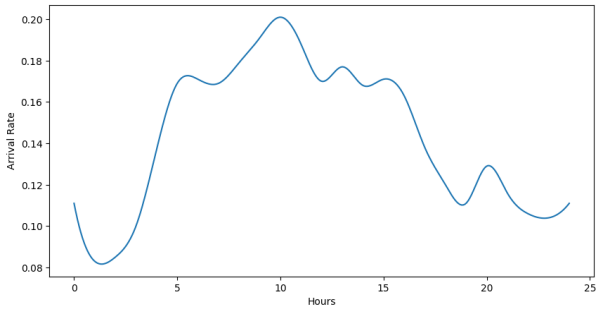

Fig. 5. Arrival rate variation during a day

We do not consider distributed sliced training because, according to the NVIDIA documentation [31], MIG-enabled GPUs do not have P2P communication, which is necessary for such tasks, and it is beyond our scope.

Using the described experimental setup, we evaluate the performance gains of MIGs compared to standard GPUs without MIG. To emulate multi-tenant, large-scale workloads with high job counts and short-lived tasks, we set the job arrival rate to be 20 times the base rate. Under such conditions, non-partitioned GPUs fail to keep up, whereas MIGs achieve 60–100× lower tardiness at high arrival rates (Table I), since most training jobs have sublinear throughput curves that degrade efficiency and cause severe backlog. Although the performance of non-MIG GPUs can be partially improved by dedicating some GPUs to inference and others to training (still yielding a 3–4× improvement), their performance remains significantly worse than that of MIG-enabled GPUs. These results strongly motivate our focus on MIGs.

TABLE I
Performance comparison under different arrival rates

		20x	**16x**	**12x**
Energy	SMART-MIG	2.5M	2.3M	2M
	No Partition	3.1M	2.4M	2M
Tardiness	SMART-MIG	1.0037	0.44	0.18
	No Partition	96.67	16.58	0.34

We compare the energy and tardiness from our repartitioning algorithm on the best scheduling algorithm in practice to the lower bound, as well as to the naive results obtained by scheduling on GPUs each having fixed configurations.

B. Results for Scheduling Algorithms without Repartitioning

We first evaluate different scheduling algorithms without repartitioning by running experiments with varying (but fixed per run) arrival rates, and configurations, aiming to characterize their relative performance, strengths, and suitable scenarios. Each job queue contains 1000 jobs with a $4:1$ inference–training job ratio. We then simulate a full day of jobs (as in Section VI.C) to identify the best-performing algorithm, which is subsequently used to train the central controller model. Experiments are conducted on 8 GPUs without penalizing preemption. We compare the three algorithms from Section IV-C against a simple 'Random' baseline that assigns the earliest-deadline job to a random available slice.

- **Effect of varying arrival rates:** At lower arrival rates with smaller queues, we generally see Most Packed EDF performing well in terms of energy (Figure 6), whereas at higher arrival rates, Marginal ET performs the best (Figure 7). The energy is around 100% worse than the lower bound at arrival rate 1, whereas it is only 20-30% worse for the others, mainly because the loose lower bound introduces larger errors, and we cannot pack so many GPUs well with such a low arrival rate.
- **Effect of varying the configuration:** This change is most prominent for CEDF as it sorts GPUs by average slice size (Figure 8), and hence benefits from having partitions with all slices having similar sizes. With such

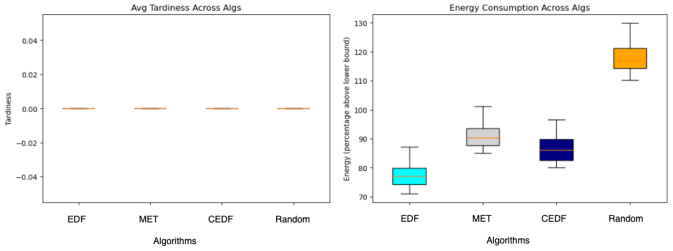

Fig. 6. Average tardiness and energy consumption at arrival rate 1

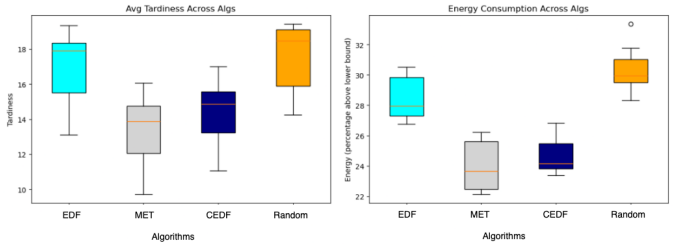

Fig. 7. Average tardiness and energy consumption at arrival rate 8

'good' configurations, CEDF performs the best in terms of energy and tardiness.

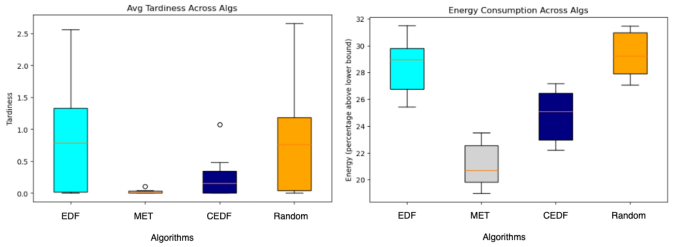

Fig. 8. Average tardiness and energy consumption for configurations where slice size varies within a GPU

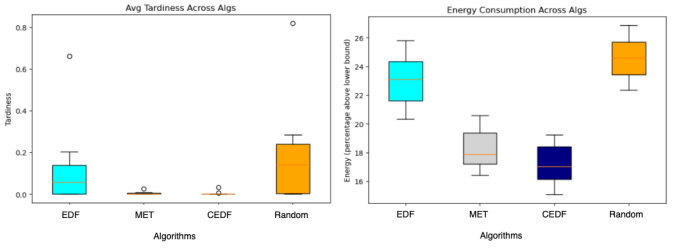

Fig. 9. Average tardiness and energy consumption for configurations where slice sizes are similar within a GPU

- **Results on realistic 24-hour queues:** These queues have varying arrival rates through the queues, and are also longer, with over 4000 jobs. Here, we find that CEDF largely performs the best (Figure 10), even when tested on varying configurations.
- **Tardiness lower bound results:** Since computing the tardiness lower bound is computationally complex, we present tardiness results for running 50 simulations of 50 mixed throughput jobs and 50 linear throughput jobs, each using MP-EDF on 4 GPUS with configuration 8 and arrival rate 8 (Figure 11). The average tardiness is above the lower bound by $\sim 34\%$ for mixed throughput jobs and $\sim 17\%$ for linear throughput jobs, on average.

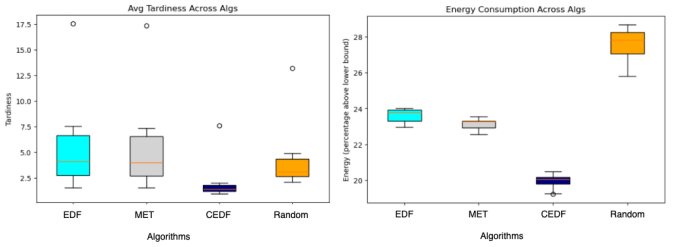

Fig. 10. Average tardiness and energy consumption for realistic 24-hour queues

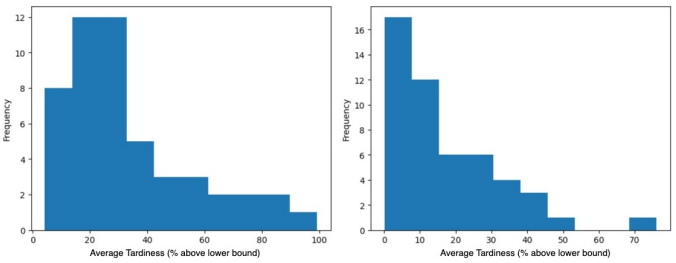

Fig. 11. Average tardiness compared with lower bounds

C. Results for SMART-MIG

In our evaluation, we consider a node with 8 MIGs and set the job arrival rate to 20 times the base rate. Throughout these experiments, the ratio of training-to-inference jobs is fixed at $1:4$. For SMART-MIG, the central controller is invoked every 5 steps with top-3 sampling (as explained in Section IV-D). Once the configurations are fixed, we use CEDF, which generally yields the best energy–tardiness trade-off among our tested algorithms. The central controller model is trained on $\sim 5M$ jobs, equivalent to 5000 days of workload at this arrival rate. We select the static configuration [1, 1, 2, 3, 3, 5, 5, 10] as a baseline for comparison with SMART-MIG since we find that CEDF demonstrates the best performance with it.

Figure 12 shows that SMART-MIG reduces average tardiness by 25% compared to CEDF with a static configuration and by 40% compared to EDF with a static configuration. In terms of energy consumption, SMART-MIG achieves improvements of 1.2% over CEDF and 7% over EDF. Evaluated with the ET metric, SMART-MIG delivers overall gains of 18% and 32% relative to CEDF and EDF, respectively. It is important to note that in this study, SMART-MIG was trained with CEDF as the scheduling algorithm, though it can, in principle, be paired with any mature multi-machine scheduler. The observed improvement of SMART-MIG when compared with its static configuration counterpart stems from its ability to dynamically repartition MIGs based on the evolving job queue, allowing the system to better handle varying urgency

and throughput demands. In the figure, the red line marks the theoretical lower bound of average energy consumption. Comparing this lower bound with our three evaluated methods, we find that RL is 27% worse and CEDF is approximately 29% worse in terms of energy efficiency. Recall that this lower bound is not tight, as the calculation involves a linearization of jobs, which underestimates the true energy lower bound. Moreover, our focus is on multi-objective optimization, where the goal is to balance energy consumption with job tardiness. Under these considerations, we argue that all three algorithms achieve reasonably effective energy optimization.

We also notice that SMART-MIG cannot substantially reduce energy consumption via repartitioning. A plausible explanation is: given the requirement to maintain low job tardiness, the baseline CEDF without repartitioning is already close to optimal and leaves little room for further improvement. However, the improvement in tardiness is quite large.

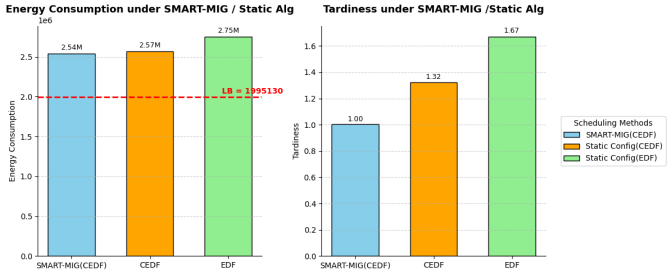

Fig. 12. Energy consumption and average tardiness results

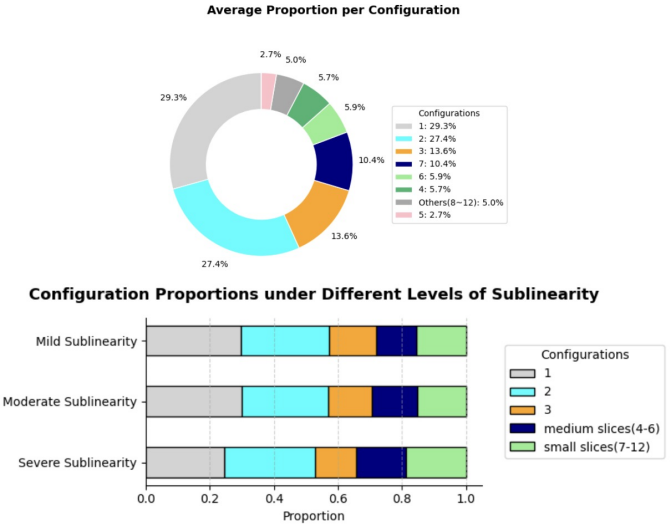

Fig. 13. Configuration characterization while repartitioning

Next, we analyze the repartitioning behavior of SMART-MIG. It is clear from [10] that if all jobs were perfectly linear, then MIGs would provide no benefit. MIGs become meaningful only when sublinear jobs are present, since in such cases, the system must balance the energy against tardiness. Thus, effective repartitioning essentially amounts to intelligently scheduling jobs with different throughput curves.

In our experiments, the sublinearity scores (as defined in section IV) of job groups encountered by SMART-MIG at different steps range from 0.67 to 0.84. Based on this range, we categorize job groups into equally divided intervals:

- Severe Sublinearity: $ss \in [0.67, 0.72]$
- Moderate Sublinearity: $ss \in [0.72, 0.78]$
- Mild Sublinearity: $ss \in [0.78, 0.84]$

The central controller encounters job groups with varying degrees of sublinearity across different experiments and timesteps. For each of these three categories, we measure the system's average repartitioning response, namely, the distribution of configurations selected. For comparison, we also report the global average distribution of configurations across all cases (Figure 13).

The most frequently selected configurations are 1, 2, 3, and 7. Meanwhile, several configurations containing small slices (e.g., 8–12) collectively account for 5%, demonstrating that a reasonable GPU configuration should maintain a balance among large, medium, and small sliced GPUs. The specific balance should adapt dynamically to variations in job arrival rate and throughput. Furthermore, as the central controller encounters an increasing number of sublinear jobs, it tends to select configurations with larger indices, which contain a greater number of small slices. This strategy increases task parallelism without undermining overall job execution efficiency, thereby reducing latency. Such variations do not cause substantial deviation in the average proportion of each configuration, thus indicating stability in job processing.

Put differently, SMART-MIG smartly balances harder-to-complete jobs (those more prone to incurring delays) with easier workloads by dynamically adjusting the configurations.

VII. Conclusion and Future Work

We propose SMART-MIG, a scalable MIG-based parallel computing framework that jointly optimizes job performance efficiency and data center energy consumption. To enable fair system evaluation, we establish theoretical lower bounds on energy and tardiness. Our experiments first demonstrate the superiority of MIGs over traditional GPUs under machine learning workloads, where the latter nearly break down (with 60-100x worse tardiness). We also design a scheduling algorithm (CEDF) that leverages the nonlinearity of job and power curves, achieving performance within $\sim 27\%$ of the energy lower bound. Our dynamic repartitioning approach based on MF-MARL, when combined with CEDF reduces the tardiness further by $\sim 25\%$ while also decreasing the energy consumption. These results highlight the effectiveness of using MF-MARL with intelligent scheduling for achieving energy-tardiness gains in MIG-based systems.

There are several promising future directions. Some of them include (i) applying RL for job scheduling in conjunction with repartitioning, (ii) further tightening the lower bounds (by avoiding linearization), (iv) extending our work to heterogeneous GPUs, and (iv) exploring geometrical job alignment within and across GPUs to minimize heat dissipation and the energy needed for cooling.

REFERENCES

[1] M. C. Offutt and L. Zhu, "Data centers and their energy consumption: Frequently asked questions," Tech. Rep. R48646, Congressional Research Service, August 2025.

[2] Z. Ye, W. Gao, Q. Hu, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

[3] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[4] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, *et al.*, "Gemma 2: Improving open language models at a practical size," *arXiv preprint arXiv:2408.00118*, 2024.

[5] B. Li, T. Patel, S. Samsi, V. Gadepally, and D. Tiwari, "Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters," in *Proceedings of the 13th Symposium on Cloud Computing*, pp. 173–189, 2022.

[6] C. Tan, Z. Li, J. Zhang, Y. Cao, S. Qi, Z. Liu, Y. Zhu, and C. Guo, "Serving dnn models with multi-instance gpus: A case of the reconfigurable machine scheduling problem," *arXiv preprint arXiv:2109.11067*, 2021.

[7] B. Zhang, S. Li, and Z. Li, "Miger: Integrating multi-instance gpu and multi-process service for deep learning clusters," in *Proceedings of the 53rd International Conference on Parallel Processing*, pp. 504–513, 2024.

[8] Y.-M. Tang, W.-F. Sun, H.-T. Ting, M.-H. Chen, I.-H. Chung, and J. Chou, "Pcie bandwidth-aware scheduling for multi-instance gpus," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, pp. 43–51, 2025.

[9] R. Sitters, "Complexity of preemptive minsum scheduling on unrelated parallel machines," *Journal of Algorithms*, vol. 57, no. 1, pp. 37–48, 2005.

[10] E. Lipe, N. Karia, C. Espenshade, C. Stein, A. Tantawi, and O. Tardieu, "Energy efficient scheduling of ai/ml workloads on multi instance gpus with dynamic repartitioning," in *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 53–62, IEEE, 2025.

[11] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 595–610, 2018.

[12] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "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*, pp. 315–325, 2019.

[13] D. Gu, X. Xie, G. Huang, X. Jin, and X. Liu, "Energy-efficient gpu clusters scheduling for deep learning," *arXiv preprint arXiv:2304.06381*, 2023.

[14] J. Villarrubia, L. Costero, F. D. Igual, and K. Olcoz, "Leveraging multi-instance gpus through moldable task scheduling," *Journal of Parallel and Distributed Computing*, p. 105128, 2025.

[15] B. Turkkan, P. Murali, P. Harsha, R. Arora, G. Vanloo, and C. Narayanaswami, "Optimal workload placement on multi-instance gpus," *arXiv preprint arXiv:2409.06646*, 2024.

[16] M. Lee, S. Seong, M. Kang, J. Lee, G.-J. Na, I.-G. Chun, D. Nikolopoulos, and C.-H. Hong, "Parvagpu: Efficient spatial gpu sharing for large-scale dnn inference in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, pp. 1–14, 2024.

[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, pp. 50–56, 2016.

[18] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *International conference on machine learning*, pp. 5571–5580, PMLR, 2018.

[19] J. Subramanian and A. Mahajan, "Reinforcement learning in stationary mean-field games," in *Proceedings of the 18th international conference on autonomous agents and multiagent systems*, pp. 251–259, 2019.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] M. L. Pinedo, *Scheduling*, vol. 29. Springer, 2012.

[22] H. Gu, X. Guo, X. Wei, and R. Xu, "Mean-field controls with q-learning for cooperative marl: convergence and complexity analysis," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 4, pp. 1168–1196, 2021.

[23] H. Zhang, Y. Li, W. Xiao, Y. Huang, X. Di, J. Yin, S. See, Y. Luo, C. T. Lau, and Y. You, "Migperf: A comprehensive benchmark for deep learning training and inference workloads on multi-instance gpus," *arXiv preprint arXiv:2301.00407*, 2023.

[24] Y. Kim, Y. Choi, and M. Rhu, "Paris and elsa: An elastic scheduling algorithm for reconfigurable multi-gpu inference servers," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 607–612, 2022.

[25] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Preemptive scheduling of uniform machines subject to release dates," in *Progress in combinatorial optimization*, pp. 245–261, Elsevier, 1984.

[26] H. Li, D. Groep, and L. Wolters, "Workload characteristics of a multi-cluster supercomputer," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 176–193, Springer, 2004.

[27] S. Cruzes, "Revolutionizing optical networks: The integration and impact of large language models," *Optical Switching and Networking*, p. 100812, 2025.

[28] Y. Zhang, L. Luo, G. Sun, H. Yu, and B. Li, "Deadline-aware online job scheduling for distributed training in heterogeneous clusters," *IEEE Transactions on Cloud Computing*, 2025.

[29] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "{MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 945–960, 2022.

[30] C. Espenshade, R. Peng, E. Hong, M. Calman, Y. Zhu, P. Parida, E. K. Lee, and M. 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*, pp. 47–55, 2024.

[31] "MIG User Guide 2014; NVIDIA Multi-Instance GPU User Guide r580 documentation — docs.nvidia.com." https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html#application-considerations.