GREEN: Carbon-efficient Resource Scheduling for Machine Learning Clusters

Kaiqiang Xu[1], Decang Sun[1], Han Tian[2], Junxue Zhang[1], Kai Chen[1]
[1]*iSING Lab, Hong Kong University of Science and Technology* [2]*USTC*

Abstract

This paper explores the problem of scheduling machine Learning (ML) jobs while also taking into account the reduction of carbon emissions in the cluster. Traditional cluster schedulers for ML jobs mainly focus on optimizing job completion time (JCT), but do not consider the environmental impact of their decisions, resulting in a suboptimal carbon footprint. To address this issue, we propose GREEN, an ML cluster scheduler that is both time-efficient and carbon-efficient. At its core, GREEN uses a unique carbon-aware scheduling algorithm that reduces carbon footprint with minimized impact on JCT. Additionally, it leverages the temporal flexibility of ML jobs to reduce carbon emissions by shifting workloads to less carbon-intensive times, while still maintaining overall daily capacity. Our experiments using real ML jobs workload demonstrate that GREEN can achieve up to 41.2% reduction in cluster-wide carbon footprint and 12% reduction in peak power consumption, while incurring 3.6%-5.9% time efficiency tradeoff compared to existing methods.

1 Introduction

Machine Learning (ML) workloads currently constitute 8% of the 54 GW global data center demand and are projected to increase to 15-20% by 2028 [3]. Current production clusters handle a large volume of ML jobs, supported by fast parallel computing infrastructures using GPU devices. These setups, termed ML-as-a-Service (MLaaS) clusters, enhance utilization and reduce costs by allowing multiple users to share resources [33]. Given the rising concern about AI's environmental impact, there is a pressing need for a new cluster resource management strategy to mitigate the environmental footprint of ML processing, especially in the pursuit of global carbon neutrality [1, 26].

Existing cluster resource schedulers for ML jobs primarily focus on optimizing metrics related to job completion time (JCT) for time efficiency. However, their optimization practices may result in inefficient energy usage [39]. For example, schedulers like Pollux [20], Optimus [19], and Themis [14] optimize throughput by dynamically adjusting GPU allocation and job hyperparameters but neglect variations in jobs' power consumption during scaling and reconfiguration. Zeus [39] balances the tradeoff between energy consumption and time efficiency, but it lacks consideration for resource allocation among multiple jobs with different energy characteristics.

More importantly, energy usage is *not* the only factor that contributes to the carbon footprint. Carbon intensity, representing carbon emissions per unit of energy consumed, can vary significantly over time [18] and considering this temporal variation is crucial in reducing the carbon footprint of ML clusters [34]. ML training jobs often span days or weeks, during which they are intermittently paused and resumed based on their priority [33]. Therefore, these jobs have a level of temporal flexibility – the timing of job execution can be shifted as long as the total allocated resources are preserved within a specified time frame (e.g., 24 hours). By exploiting this flexibility, schedulers can align job power consumption with carbon intensity fluctuations, effectively reducing the overall carbon footprint. However, existing energy-aware optimizations [23, 39] overlooked carbon intensity and job flexibility's combined effect on cluster-wide environmental impact.

This paper presents GREEN, a novel carbon-efficient cluster scheduler that aims to reduce the overall carbon footprint of the cluster while achieving comparable time efficiency performance to prior work. GREEN utilizes a carbon-aware scheduling algorithm that balances the cluster-wide JCT and carbon footprint by considering various carbon-related factors, including the *energy efficiency* of each job, which derives from how effectively energy usage contributes to a job's training progress. Additionally, the algorithms exploit the temporal flexibility of ML jobs to reduce grid carbon emissions by automatically shifting workloads to greener hours with lower carbon intensity while still maintaining overall daily capacity.

GREEN makes carbon-aware decisions on scheduling by taking into account carbon-related factors. Specifically, GREEN employs a carbon tracker (§4) to monitor the energy consumption and other runtime metrics of ML jobs throughout their execution, and uses a factor model to compute the *energy efficiency* (the change in energy usage as the job progresses) for each job and *carbon footprint* (the accumulated carbon emissions based on energy usage).

GREEN proposes two optimizers (§5) for optimizing energy efficiency and reducing carbon footprint in ML clusters. The *Energy Efficiency Optimizer* scales resources allocated to jobs, taking into account the scalability demonstrated by the amount of energy consumption for a job when scaling, while the *Carbon Footprint Optimizer* dynamically rearranges job priorities for peak load shifting, minimizing cluster-wide carbon emissions while ensuring fairness in resource allocation.

To enable the co-optimization of these two optimizers, GREEN's scheduling algorithm (§6) incorporates a Multi-

level Feedback Queue (MLFQ) mechanism. The upper queue, guided by the *Energy Efficiency Optimizer*, emphasizes energy efficiency by scaling out jobs that achieve greater progress with lower energy consumption. The lower queue retains the progress and decisions made in the upper queue and utilizes the *Carbon Footprint Optimizer* to minimize the carbon footprint by relocating high-power-consuming jobs to "greener" hours with lower carbon intensity.

GREEN optimizes cluster-wide carbon footprint but has a non-goal: GREEN avoids altering job-level configurations (e.g. model hyperparameters) as in some existing ML schedulers [20, 39]. This design choice makes GREEN non-intrusive to the user's ML implementation and becomes orthogonal to optimization techniques that are provided in ML frameworks at the single-job level, such as adaptive hyperparameters and learning rate decay, which would otherwise conflict with job-level optimizations in cluster schedulers.

We evaluate GREEN using two types of workloads: (1) a production ML cluster used by a research institution with over 500 active academic users, and (2) a real-world machine learning workload trace from Alibaba [33]. We compare the performance of GREEN to state-of-the-art cluster schedulers under these workloads, and report on time metrics including JCT and makespan, as well as energy metrics including carbon footprint and peak power usage. Our results show that GREEN achieves significant reductions in the cluster's carbon footprint (up to 41.2%) and peak power draw (up to 12%), while maintaining comparable time efficiency to prior work, with a tradeoff of 3.6%-5.9% in average JCT.

We summarize our contributions as follows:

- We present a factor model to calculate carbon-related metrics and factors, including Carbon Intensity Curve (CIC), carbon footprint, and energy efficiency, providing important signals to the scheduler.
- We propose a carbon-efficient scheduler that scales out jobs while ensuring energy efficiency, and moves high-power jobs to greener hours to reduce carbon footprint without compromising fairness in resource allocation.
- We implement and evaluate GREEN with the real workload in a 32-node production cluster and the ML job trace from a large-scale enterprise cloud, and results show that GREEN can achieve up to 41.2% carbon footprint reduction, by trading $<$5.9% in average JCT.

2 Background and Motivations

This section discusses related work on ML cluster schedulers and energy-aware systems under a wider context.

2.1 ML Cluster Scheduling

A machine learning (ML) cluster is a shared environment where multiple users can submit training jobs that compete for limited resources. The cluster scheduler is responsible for managing job queues and allocating resources, with the goal of optimizing performance metrics such as job completion time (JCT) and resource utilization. This setup is commonly referred to as Machine Learning-as-a-Service (MLaaS) [33].

Schedulers	Scale-Adapt.	Energy-Aware	Carbon-Aware	Model-Agnostic
Gandiva / AntMan	✗	✗	✗	✗
Tiresias	✗	✗	✗	✓
Optimus	✓	✗	✗	✓
Pollux	✓	✗	✗	✗
Zeus	✗	✓	✗	✗
GREEN	✓	✓	✓	✓

Table 1: Characteristics of representative cluster schedulers.

Studies in ML job clusters explore optimization techniques in different directions. We classify these works based on key characteristics and present representative ones in Table 1.

① **Scale-Adaptive (or Elastic)**. Scale-adaptive schedulers dynamically adjust resource allocation (i.e., number of GPUs) for jobs based on their efficiency in speeding up tasks. Pollux [20] assesses statistical efficiency in ML training, measuring progress per unit of data processed. Optimus [19] builds predictive models for job-specific system throughput to determine scalability.

In the energy-aware context, jobs have different energy characteristics when scaling out, therefore, varying scheduling decisions can lead to different outcomes in energy consumption, affecting overall power consumption. This renders some scheduling decisions less favorable than others in the energy context. For instance, Pollux may prioritize scaling out jobs with lower speedup potential, preventing stragglers and enhancing average JCT. However, this approach directs more resources to jobs with lower energy utilization.

Non-scale-adaptive schedulers overlook the scalability of ML job performance under allocated resources. Tiresias [7] and Gandiva [35], for instance, require users to set a fixed number of GPUs throughout execution, limiting cluster-side performance optimization possibilities.

② **Model-Agnostic**. Some schedulers control model hyperparameters or require prior knowledge of a job (e.g., a specific pattern). Pollux [20] dynamically adjusts DNN hyperparameters (e.g., learning rate, per-GPU batch size) for improved statistical efficiency. Gandiva [35] uses a time-slicing approach for a group of jobs running the same DNN model. AntMan [36] uses a specialized framework for fine-grained GPU-sharing. A recent work, GreenFlow [6], dynamically allocates GPUs and optimizes job-level configurations to reduce average JCT under a carbon emission budget.

However, modifying job configurations introduces compatibility and usability challenges. Key hyperparameters, such

as per-GPU batch size and learning rate, are highly sensitive in ML workloads and exhibits intricate behaviors, especially in modern model architecture such as transformer-based models. Enforcing control over these hyperparameters not only requires additional engineering overhead on users to integrate with external schedulers but also conflicts with model-level optimizations in ML frameworks. Similar concerns have been highlighted in recent scheduler research [11].

③ **Energy-Aware.** GPU power consumption, which varies with utilization, is a key driver of ML models' overall energy usage. For instance, an NVIDIA H100 GPU consumes between 150W at low utilization and over 700W at peak load. Optimizing job energy efficiency thus requires accounting for GPU utilization. Zeus [39] optimizes the trade-off between energy consumption and time efficiency for individual jobs but does not address cluster-wide resource allocation across jobs with varying energy profiles. Meanwhile, prior research on energy-aware optimizations primarily targets generic workloads without leveraging ML-specific characteristics and requirements, as discussed in §2.2.

Design Space of GREEN. GREEN embodies all three attributes discussed above: scale-adaptive, model-agnostic, and energy-aware. Specifically, GREEN achieves energy and time efficiency comparable to prior work, without modifying job-level settings such as model hyperparameters. This approach preserves the non-intrusiveness of GREEN within the user's ML implementation and ensures GREEN's compatibility with both existing and future job-level optimization techniques in ML frameworks (as discussed in §9).

2.2 Carbon-aware Resource Scheduling

When addressing carbon-aware scheduling, it is important to recognize that reducing energy consumption may not be the sole goal. In optimizing for a lower carbon footprint, the focus shifts to carbon intensity, which measures emissions per kWh of power consumption.

Carbon Intensity (kgCO_2/kWh) signifies carbon emission per kilowatt-hour of electricity. It can be publicly obtained from public sources as a grid characteristic. Figure 1 illustrates hourly carbon intensity averages using data from UK [18], revealing daily variation.

The conceptual model below calculates carbon footprint.

$$\texttt{FOOTPRINT}_{\texttt{job}} = \int \mathit{Power}_{\mathit{host}} \times \mathit{Carbon\ Intensity}\ dTime$$

The model tracks device power and then integrates the product of power and carbon intensity over time. This integral is essential as both power and carbon intensity are time-varying.

An intuition is that during low carbon intensity time, more optimal scheduling decisions might be possible by moving high-power jobs to these lower periods. Certain ML training jobs exhibit flexibility for this time-shifting. This is because ML model training can span over weeks, or even months for large models, during which they are intermittently paused and resumed based on their scheduling priority [33]. The timing of job execution can be shifted as long as the total allocated resources are preserved within a specified time frame (e.g., 24 hours). By exploiting this flexibility, schedulers can align job power consumption with carbon intensity fluctuations, effectively reducing the overall carbon footprint.

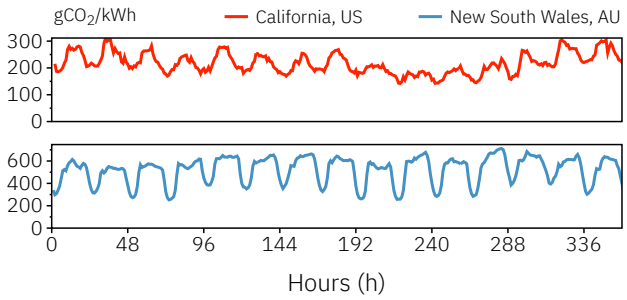


Figure 1: Daily pattern of historical carbon intensity, using data from August 2023 [16]. The carbon intensity of energy supplied by the electric grid depends on the energy sources.

Next, we briefly discuss existing work in carbon-aware scheduling, focusing on the limitations of these approaches in the ML cluster context.

Power and Resource Throttling. Google's CICS [23] imposes a limit on cluster capacity during periods of high carbon intensity. Some approaches [6, 39] utilize Dynamic Voltage and Frequency Scaling (DVFS) to enforce GPU power limits, which directly caps energy usage. However, these methods reduce energy consumption essentially by *scaling back the work performed*. In the MLaaS context, where ML clusters often operate at high occupancy, imposing power or resource limits reduce cluster capacity, leading to a negative impact on overall job completion time (JCT).

Lack of Capacity Constraints. EcoVisor [28] and WaitAWhile [34] use scheduling strategies that do not account for resource competition between jobs. CarbonScaler [8] requires a user-specified job deadline and can scale jobs using unlimited, on-demand cloud resources to meet the deadline. GAIA [9] focuses on cloud environments with on-demand resources, aiming to balance carbon footprint, time performance, and resource cost. In contrast, the MLaaS setting involves static cluster resources, leading to competition between jobs. In this context, the scheduler must make dynamic decisions on resource allocation while adhering to the cluster's resource constraints.

In summary, we observe that prior approaches to carbon-aware scheduling often overlook resource constraints, either throttling jobs during periods of high carbon intensity—leaving resources underutilized—or elastically scaling jobs with unconstrained on-demand capacity, disregarding resource limitations. Meanwhile, some systems require additional user input, such as predefined job deadlines [8] or

carbon budgets [6], which are often difficult for ML workloads due to their unpredictable execution characteristics.

To bridge this gap, we propose GREEN, a resource scheduler for ML jobs that operates strictly within available resource capacity. GREEN exploits characteristics of ML workloads to monitoring and reduce the cluster-wide carbon footprint while preserving time efficiency.

2.3 Motivation and Challenges

We consider the opportunities missed by existing job schedulers to reduce cluster-wide carbon footprint.

Motivation 1: Available resources are not allocated in a way that optimizes energy utilization. Prior ML cluster schedulers focuses on JCT optimization without addressing cluster-wide energy efficiency. As discussed in §2.2, these approaches are either energy-unaware [7, 14, 20] or emphasize at the single-job level [39]. Consequently, they do not effectively optimize for clusters where jobs' energy characteristics vary and resource allocation impacts cluster-wide energy consumption.

Opportunity: The cluster scheduler should determine the appropriate level of resources to allocate to a job, taking into account its scalability in both speed and energy consumption.

Motivation 2: Temporal flexibility of workload is not exploited to reduce carbon footprint. As ML models may take days or weeks to train, ML jobs are generally not sensitive to minor variations in progress during the process. This indicates that they have a certain degree of temporal flexibility. While we also observe the fact that carbon intensity (described in §2.2) varies substantially over time of day, we can exploit the temporal elasticity to consume energy when the electricity grid is less carbon-intensive.

Opportunity: The cluster scheduler should apply temporal shifting to flexible workloads, moving up or delaying them to "greener" hours with lower carbon intensity.

Challenges: The key challenge lies in making carbon-aware scheduling decisions without introducing significant performance degradation. To achieve this, the scheduler must:

- Preserve cluster efficiency: New scheduling and scaling mechanisms must maintain overall cluster efficiency, ensuring job execution speed remains comparable to state-of-the-art schedulers.
- Ensure resource fairness in temporal shifting: When shifting flexible workloads temporally, the total resource allocation over a given period (e.g., 24 hours) must be preserved to prevent job starvation.

3 GREEN Overview

GREEN is a GPU cluster scheduler designed to minimize the carbon footprint of ML job execution, while preserving time efficiency on par with existing ML job schedulers.

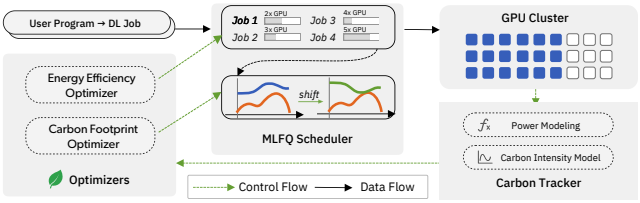


Figure 2: An overview of GREEN's system components and workflow

Workflow. As depicted in Figure 2, GREEN processes user-submitted jobs and allocates resources across them. Being model-agnostic, GREEN makes no assumptions of user implementation under specific frameworks. Similar to existing ML cluster schedulers [7, 20, 29] in the MLaaS setting [33], GREEN optimizes cluster-wide metrics without providing direct SLO or deadline guarantees for individual jobs.

Carbon Tracker (§4). Carbon tracker is the essential component for profiling. It makes GREEN carbon-aware by monitoring factors including per-job energy consumption throughout ML job execution as well as estimating the carbon footprint using the carbon-intensity model. These factors serve as important inputs for GREEN's carbon-related optimizers and scheduling algorithm described below.

Optimizers (§5). We propose two optimizers employed in GREEN: (1) the Energy Efficiency Optimizer, which allocates resources to jobs in a way that maximizes energy efficiency in the cluster, taking into account per-GPU energy-to-progress efficiency, and (2) the Carbon Efficiency Optimizer, which minimizes the cluster-wide carbon footprint by dynamically adjusting job priorities for peak load shifting.

MLFQ Scheduler (§6). The scheduling algorithm utilized by GREEN resembles a Multilevel Feedback Queue (MLFQ), allowing for different scheduling strategies based on varying job states. Two scheduling queues are managed by the aforementioned optimizers, in a synergistic manner: the upper queue emphasizes energy efficiency by scaling out jobs that achieve greater progress with lower energy consumption, while the lower queue maintains the scaling decision from the upper queue and minimizes the carbon footprint by shifting high-power-consuming jobs to greener hours with lower carbon intensity.

4 Carbon Tracker

GREEN is carbon-aware as it monitors per-job energy consumption throughout the execution of an ML job, and estimates carbon footprint using a carbon-intensity model.

4.1 Modeling Power Draw

The job's power draw P_{job} at time t can be estimated with:

$$P_{\texttt{job}}(t) = P_{gpu}(t) + P_{cpu\_model}(t) + P_{static}$$

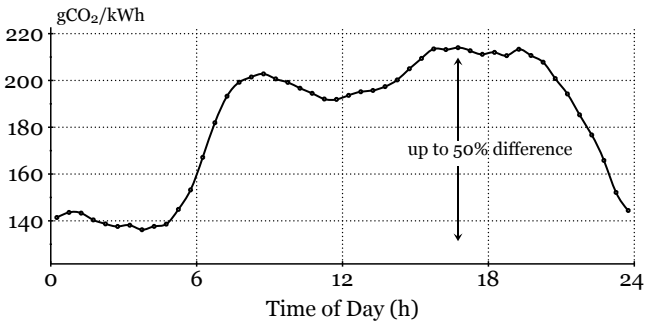


Figure 3: Carbon Intensity Curve (CIC): plotted with average carbon intensity values for each hour of the day in a 3-month period [18]. Y-axis is not zero-based.

We use NVML (NVIDIA Management Library) to read the power draw of GPUs $P_{\texttt{gpu}}$ in real-time. For CPU power consumption, we employ the model [22] to estimate power consumption $P_{\texttt{cpu}}$ in kW from CPU usage. Additionally, to factor in other devices like DRAM and motherboard, which are less influenced by their load, we use a static value $P_{\texttt{static}}$ to represent their power draw.

For the actual calculation, we discretized the equation into a step function with 30-second intervals to minimize measurement overhead. Notably, NVML power readings may not always be accurate [38], but our method remains adaptable to any measurement source, including external power meters for direct hardware power assessment.

4.2 Estimating Carbon Footprint

Carbon Footprint refers to the cumulative amount of carbon emissions generated since the start of a job. These emissions are measured in units of kilograms of carbon dioxide ($kgCO_2$).

Carbon Intensity. We introduce the Carbon Intensity Curve (CIC) as our carbon intensity model (Figure 3).

$$\texttt{FOOTPRINT}_{\texttt{job}}(T) = \int_0^T P_{\texttt{job}}(t) \cdot CIC(t) dt$$

The equation above defines the carbon footprint. By calculating the integral of $P_{job} \cdot CIC$ over time t, the result reflects both power consumption and carbon intensity over time.

Carbon intensity data is publicly accessible online in many regions [16, 31]. In our implementation, we make use of historical carbon intensity data, as it usually exhibits consistent daily pattern over a period of time. GREEN's carbon footprint optimizer can adapt to any pattern of CIC (§5.2), and recent work [28] proposes a software-based energy system to retrieve energy usage and carbon intensity in real-time.

5 GREEN Optimizers

This section introduces two optimizers in GREEN's scheduling algorithm: Energy Efficiency Optimizer (§5.1) allocates resources to jobs to maximize energy efficiency, taking into account the scalability of per-GPU energy-to-progress efficiency. Carbon Footprint Optimizer (§5.2) reduces the cluster-wide carbon footprint by dynamically adjusting job priorities to shift peak loads to greener hours.

5.1 Energy Efficiency Optimizer

The objective of GREEN's Energy Efficiency Optimizer is to determine the appropriate level of resources to allocate to a given job, maximizing energy efficiency in the cluster. This objective is motivated by the observation that a job's scalability measured in energy efficiency may differ from its performance scalability (e.g., training throughput), leading to suboptimal cluster-wide energy efficiency.

Definition. To quantify energy efficiency, we formulate it as the derivative of job progress $Progress_{\texttt{job}}$ with respect to total accumulated energy usage $\texttt{W}_{\texttt{job}}$, as shown in Equation 1.

$$\texttt{EFFICIENCY}_{\texttt{job}}(t) = \frac{\mathrm{d}Progress_{\texttt{job}}(t)}{\mathrm{d}\texttt{W}_{\texttt{job}}(t)} \tag{1}$$

This formulation enables the scheduling algorithm to leverage two key pieces of information:

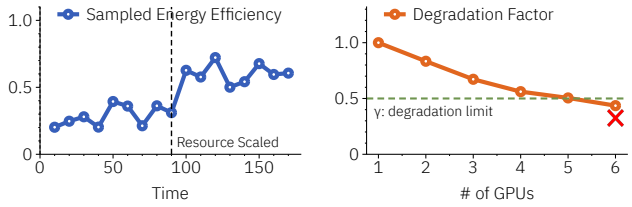
- Jobs with higher energy efficiency are able to make more progress with the same power consumption when compared to jobs with lower energy efficiency.
- By allocating more computing resources to a job, GREEN can observe changes in energy efficiency and learn about the job's scalability. Ideally, there would be no decrease in energy efficiency when scaling out a job.

The unit of energy efficiency is *not* important, as long as it can be considered as a unit of work done over a unit of power consumption. This is because our focus is on observing its relative change ratio during the job's execution, and the unit cancels out in the calculation.

Calculation of Energy Efficiency. In Equation 1, energy efficiency is defined as a function of time, and to obtain this function for an actual job run, GREEN needs to approximate it using runtime data. There are two built-in methods for GREEN to measure the progress of a job, depending on whether the progress of a job can be directly obtained during its runtime.

If job progress is available, GREEN calculates the efficiency function by sampling progress for a duration of time as defined in Equation 2.

$$\texttt{EFFICIENCY}_{\texttt{job}}(t) = \frac{Progress_{\texttt{job}}(t) - Progress_{\texttt{job}}(t - \Delta t)}{\texttt{W}_{\texttt{job}}(t) - \texttt{W}_{\texttt{job}}(t - \Delta t)} \tag{2}$$



(a) Energy Efficiency **(b)** Degradation Factor

Figure 4: An example of energy efficiency calculation. *Left*: Energy efficiency data sampled before and after resource scaling, with the unit being % of progress per kWh. *Right*: Degradation factor over the number of GPUs, obtained by averaging efficiency data per GPU.

The variable Δt represents the time elapsed since the most recent job scaling event. GREEN tracks job progress by recording the number of completed epochs and finding the ratio of completed epochs to the total epochs. This method works well because ML training repeats in rounds and each training epoch takes roughly the same amount of time.

If job progress cannot be directly obtained, such as when training large models with long epoch time, GREEN can approximate progress using the job's training throughput $Throughput_{\texttt{job}}$, i.e., the number of data samples trained over a duration of time, as shown in Equation 3.

$$\texttt{EFFICIENCY}_{\texttt{job}}(t) = \frac{\Sigma_{t'=t}^{t-\Delta t} Throughput_{\texttt{job}}(t')}{\texttt{W}_{\texttt{job}}(t) - \texttt{W}_{\texttt{job}}(t-\Delta t)} \quad (3)$$

When progress measurements are not available, the formula represents an absolute value of the progress made (i.e., throughput) during that duration of time.

Figure 4a shows an example of the energy efficiency collected before and after one scaling event.

We make note that retrieving job throughput is a common practice in today's cluster schedulers [20, 39], in practice, it often just requires a simple log print by the job.

Job Scaling with Energy Efficiency. Dynamic job scaling plays a crucial role in enhancing the performance of ML cluster JCT [20]. GREEN employs the energy efficiency metric, as defined earlier, to make scaling decisions.

For each job that is prepared to run, it commences with one GPU (or the minimum number of GPUs necessary for the job). We introduce an efficiency degradation factor, which represents the proportion of per-GPU energy efficiency decline relative to the initial job energy efficiency:

$$\texttt{DEGRADATION}_{\texttt{job}}(\texttt{t}) = \frac{\texttt{EFFICIENCY}_{\texttt{job}}(\texttt{t})}{\texttt{EFFICIENCY}_{\texttt{job}}(\texttt{t}_0)} \quad (4)$$

where $\texttt{EFFICIENCY\_job}(\texttt{t}_0)$ represents the job's energy efficiency with the minimum GPU required, which is profiled at the beginning time (t_0) of the job's execution.

In subsequent scheduling rounds, the job will receive an additional GPU if its efficiency degradation factor remains above a threshold γ. For instance, with γ at 0.9, it means that the job's energy efficiency must not decrease by more than 10% to be eligible for scaling out. The value of γ is dynamically adjusted according to both the cluster-wide occupancy rate and job characteristics to prevent over- or under-allocation of resources. Additionally, A maximum resource cap prevents well-scaled jobs from taking up all GPU resources. More details about GREEN's adaptive strategy can be found in §6.4.

Figure 4b demonstrates an example of the change in energy efficiency during scaling. The unit of energy efficiency cancels out, as we only use its relative change in percentage as a scheduling factor during job execution. This scaling approach reduces the cluster's carbon footprint by allocating more resources to jobs that efficiently utilize energy for progress.

Relation with Single-job Optimizations. The design of the Energy Efficiency Optimizer is orthogonal to previous optimizations on the job-level. GREEN optimizes the energy efficiency with resource allocation strategies, while earlier work such as [39] takes resource allocation as an input, and investigates job-level parallelization and configurations to optimize efficiency. These optimizations can be combined and utilized together to enhance overall energy efficiency.

5.2 Carbon Footprint Optimizer

The Carbon Footprint Optimizer in GREEN aims to reduce the cluster-wide carbon footprint through peak load shifting. This is achieved by dynamically adjusting job priorities in scheduling.

The job priority is calculated as a combination of two factors, namely the Carbon Factor and the Shifting Factor, with higher priority values indicating lower scheduling priorities.

$$\texttt{PRIORITY}_{\texttt{job}} = \left(\frac{\texttt{FOOTPRINT}_{\texttt{job}}}{\texttt{DEGRADATION}_{\texttt{job}}}\right) \cdot \texttt{SHIFTING}_{\texttt{job}} \quad (5)$$

An input t, denoting the system time, exists for all factors defined above and is omitted for readability.

Carbon Factor. The carbon factor is calculated by dividing a job's accumulated carbon footprint, denoted as $\texttt{FOOTPRINT}_{job}$ (§4), by its energy efficiency degradation factor represented by $\texttt{DEGRADATION}_{job}$ (Equation 4).

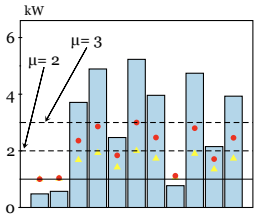
The Carbon Factor serves as the foundation for job priority and exhibits two essential properties:

(1) It tracks the carbon emissions produced during a job's execution, similar to the Least-Attained-Service (LAS) algorithm, while taking environmental impact into account. This factor helps achieve fair resource allocation among all jobs.

(2) It penalizes jobs with poor energy efficiency by dividing their carbon footprint by the degradation factor. This method enables energy-efficient jobs that are scaled out by the Energy Efficiency optimizer to complete earlier, thereby enhancing overall JCT. It also helps to avoid the unnecessary occupation of the cluster's computing resources.

$\texttt{SHIFTING}_{\texttt{job}}$	if $P_{job} \le \widetilde{P}$	if $P_{job} > \widetilde{P}$
if $CIC_t \le \bar{I}$	$P^*_{\texttt{job}}$	$1/P^*_{\texttt{job}}$
if $CIC_t > \bar{I}$	$1/P^*_{\texttt{job}}$	$P^*_{\texttt{job}}$

(a) Function for Shifting Factor



(b) $P^*_{\texttt{job}}$ Example

Figure 5: Left: Piecewise function for calculating Shifting Factor. Right: $P_{\texttt{job}}$ (bar) and $P^*_{\texttt{job}}$ when $\mu = 2$ (yellow dots) or $\mu = 3$ (red dots).

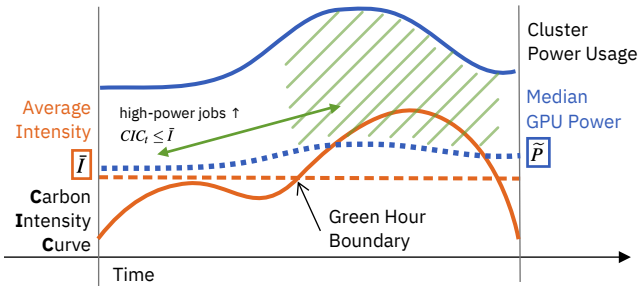


Figure 6: High-power jobs have higher priority when carbon intensity is below $\bar{I}$, and are assigned a small shift factor according to Table 5a to be scheduled earlier. The curve shown in figure is pre-shifting.

Shifting Factor. Temporal job shifting follows a heuristic that favors delaying high-power consumption jobs to "greener" hours with low carbon intensity and prioritizing low-power-consumption jobs during high-carbon-intensity periods.

The core idea behind the shift factor calculation is to adjust job priorities based on the current carbon intensity level relative to the daily values and the job's power consumption level relative to all unfinished jobs.

A piecewise function is employed, as outlined in Figure 5a, to achieve this using $\widetilde{P}$ and $\bar{I}$ as boundary values, dividing the shifting factor calculation into four quadrants:

1) *Average carbon intensity*, denoted by $\bar{I}$, is calculated using historical carbon intensity data, the hours with carbon intensity lower than $\bar{I}$ are considered to be *greener hours*.

2) *Median GPU power consumption*, denoted by $\widetilde{P}$, is obtained by finding the median GPU power draw over all profiled or running jobs. The jobs with GPU power draw higher than $\widetilde{P}$ are considered to be *high-power jobs*.

The piecewise function for calculating shifting factors rescales the job's power consumption $P_{\texttt{job}}$ to $[1, \mu]$, denoted as $P^*_{\texttt{job}}$, as defined below:

$$P^*_{\texttt{job}} = \frac{(P_{\texttt{job}} - P_{min})}{(P_{max} - P_{min})} \times (\mu - 1) + 1 \tag{6}$$

The parameter μ (≥ 1) controls the scale of $P^*_{\texttt{job}}$ (example shown in Figure 5b) and hence the degree of temporal shifting, with larger values causing more job shifting. When $\mu = 1$, temporal shifting is essentially disabled, as the shifting factor is always scaled to 1. Evaluations show that $\mu = 2$ effectively balances the two factors to achieve both time and carbon efficiency (see sensitivity experiments in §A.2).

Figure 6 depicts an example usage of the piecewise function: when the current carbon intensity is lower than $\bar{I}$, the high-power jobs are assigned with a small shifting factor according to Figure 5a, resulting in higher job priority.

The optimizer can adapt to any pattern of carbon intensity changes, as it dynamically calculates the boundary values $\bar{I}$ and $\widetilde{P}$ based on given CIC and job power consumption.

Impact on Cluster-wide and Individual JCT. Deferring high-power tasks does not necessarily lead to a delay in the cluster-wide job completion time (JCT). This is because the scheduler swaps the daily computation resources assigned to high-power and low-power jobs in the temporal dimension but preserves daily capacity for a job. Analysis in §6.4 detailed the discussion about how GREEN avoids starvation for both long and short jobs. Meanwhile, our evaluation in §8.2.2 also provides empirical evidence to support the statement.

6 Putting It Together: MLFQ Scheduling

GREEN's scheduling algorithm resembles a Multilevel Feedback Queue (MLFQ), which employs different scheduling strategies based on job states (Figure 7).

6.1 Upper Queue: Online Profiling and Scaling

The upper queue utilizes the Energy Efficiency (EE) Optimizer (§5.1) to determine the scale of computing resources allocated to each job, ensuring that jobs with higher energy efficiency receive a greater share of energy. The logic is shown in Algorithm 1 lines 3-11.

Upon submission to the cluster, new jobs are placed at the end of the upper queue and profiled for potential scaling.

When a job is initially submitted for execution, it starts with a single GPU (or the minimum number of GPUs necessary for the job). Repeatedly for each scheduling round, GREEN iterates through the queue in the first-come-first-serve (FCFS) order, and evaluates the energy efficiency degradation factor to determine a job's eligibility for further scaling (lines 4-7). When there are multiple jobs in this queue, the additional unit of computing resources is allocated to them in FCFS order until the cluster reaches its maximum capacity (lines 7-8).

Queue Transition. The job will not leave the upper queue as long as it is still eligible for scaling. Jobs transition to the lower queue once the EE optimizer has completed the scaling process. Transitioning between queues does not necessarily interrupt the job's execution, as it may remain eligible to run based on the priority assigned in the lower queue.

6.2 Lower Queue: Peak Load Shifting

The lower queue employs the Carbon Footprint (CF) Optimizer (§5.2) to conduct peak load shifting by relocating

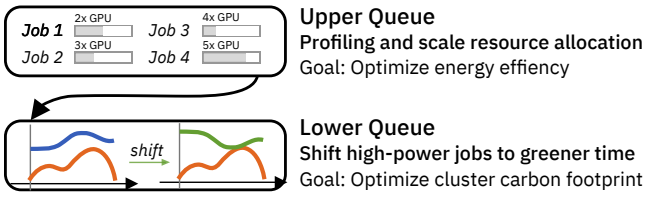


Figure 7: MLFQ enables two optimizers to work together. The upper queue determines the job's spatial placement (number of GPUs), while the lower queue decides the temporal aspect (time to run) to achieve different optimization goals.

energy-intensive jobs to greener hours with lower carbon intensity, thus reducing the cluster-wide carbon footprint. The core scheduling logic is shown in Algorithm 1 lines 12-17.

The lower queue determines job priorities using the CF optimizer, while inheriting resource scaling decisions from the upper queue. Partial execution from online profiling, is considered part of the running time, and the work completed during this phase is retained as part of the job progress.

In the lower queue, GREEN calculates each job's priority based on two factors (§5.2): the carbon factor, which considers job energy consumption while ensuring fairness in resource allocation, and the shifting factor, which moves high-energy-consuming jobs to hours with lower carbon intensity. GREEN sorts jobs by their priorities and schedules them for execution in descending order.

6.3 Scheduler Workflow

The scheduling workflow consists of both active and passive components.

Active scheduling, shown in Algorithm 1, operates in rounds to assign job priorities and preempt lower-priority jobs. It ranks jobs for immediate execution at the top (triggering preemption), followed by jobs in the FCFS upper queue, and then those in the lower queue ordered by the priorities calculated (line 12).

To limit excessive preemption, the interval between active scheduling, known as the time quantum, is set to 30 minutes. Our findings in §A.2 show that this interval length does not significantly influence overall outcomes, as long as it is not excessively short.

Passive scheduling comes into effect in the intervals between active scheduling rounds. During these periods, when a job finishes and releases resources, the scheduler deploys the passive strategy to avoid idling. This passive strategy starts the next job based on its rank in the latest active scheduling without recalculating priorities or triggering preemptions.

Job SLO and Preemptions. Similar to existing cluster schedulers [7, 20, 29] used in MLaaS environments [33], GREEN prioritizes global cluster efficiency rather than providing strict Service Level Objectives (SLOs) or deadline guarantees for individual jobs. GREEN assumes all jobs are preemptible, which may not align with workloads requiring strict

Algorithm 1: Carbon-efficient MLFQ Scheduling

Input:
- Jobs in Upper and Lower Queue: $\mathcal{J}_\mathcal{U}, \mathcal{J}_\mathcal{L} \in \mathcal{J}$
- Cluster Capacity and Upper Queue Cap: $\mathcal{C}$ and θ
- Current Timestamp: t

1 **begin**
2 Allocation $\mathcal{A} = \{\}$
3 **for** *Job* $J \in \mathcal{J}_\mathcal{U}$ *ordered by* $\texttt{DEGRADATION}_J$ **do**
4 **if** $\texttt{DEGRADATION}_J = \Delta\texttt{EFFICIENCY}_J \geq \gamma$ **then**
5 Scale out J with one more GPU
6 $\mathcal{A} = \mathcal{A} \cup J$
7 **if** $Capacity(\mathcal{A}) > \theta \times \mathcal{C}$ **then**
8 break
9 **else**
10 Remove J from $\mathcal{J}_\mathcal{U}$ and add to $\mathcal{J}_\mathcal{L}$
11 **end**
12 $\texttt{PRIO}_\texttt{J} = (\frac{\texttt{FOOTPRINT}_\texttt{J}}{\texttt{DEGRADATION}_\texttt{J}}) \cdot \texttt{SHIFTING}_\texttt{J}\ \forall J \in \mathcal{J}_\mathcal{L}$
13 **for** *Job* $J \in \mathcal{J}_\mathcal{L}$ *ordered by* $\texttt{PRIORITY}_J$ **do**
14 $\mathcal{A} = \mathcal{A} \cup J$
15 **if** $Capacity(\mathcal{A}) > \mathcal{C}$ **then**
16 break
17 **end**
18 Signal jobs $\forall J \in \mathcal{J}/\mathcal{A}$ for preemption
19 Allocate GPUs for $\forall J \in \mathcal{A}$
20 **end**

execution guarantees. In real-world deployments, GREEN's scheduling policy may be overridden to prevent preemptions for jobs with specific requirements.

6.4 Starvation Avoidance

Starvation Avoidance for Short Jobs. In the upper queue, where job scaling occurs, as a job progresses and could potentially be complete, it implicitly favors jobs with lower GPU consumption. This approximates the short-job-first (SJF) strategy, improving average JCT and avoiding starvation of shorter jobs. The scaling process in the upper queue also eliminates the need for a separate profiling stage. In the lower queue, the carbon footprint-based factor has an LAS-like behavior, which deprioritizes and preempts longer-running jobs over time, making room for shorter jobs. In §8.2.2, we quantitively evaluate the impact on cluster-wide and individual JCTs.

Starvation Avoidance for Long Jobs. To prevent starvation of longer jobs during peaks of new job submissions (when new jobs enter the upper queue and gain priority over existing long jobs in the lower queue.), a knob for the capacity cap is enforced in the upper queue. Default of 30%, this cap limits the total cluster capacity available to jobs in the upper queue. Upon reaching this cap, the scheduler prioritizes the jobs by their degradation factors, skips the remaining upper queue and considers the lower queue (Line 3-11 in Algorithm 1).

7 Implementation

We implement GREEN with Slurm [25]. Slurm is an open-source job cluster management system widely used to manage computing clusters. See Appendix B for our open-source plan.

Scheduler Implementation. GREEN's scheduler workflow is implemented as a Slurm scheduler plugin that supports custom policies to set job priorities in Slurm's job queue. The scheduler plugin communicates with an external daemon process, which stores CIC data, collects job energy statistics from profiler agents running on each compute node, and returns scheduling decisions to the scheduler plugin.

Energy-efficient Job Termination. We adopt the popular checkpoint-restart methods [20] for job scaling and preemption. However, the method may lose intermediate results since the last checkpoint, wasting computational resources. GREEN sends a `SIGTERM` signal with a 120-second grace period to ML jobs upon termination, allowing jobs to save a checkpoint after completing the current iteration and exit gracefully. The time for checkpointing and subsequent restart is recorded as part of the job's execution time.

8 Evaluation

We evaluated GREEN using time and energy metrics, comparing them against five baseline schedulers under a real-world ML workload and a trace-driven simulation workload.

8.1 Experimental Setup

Testbed. Our testbed cluster consists of 32 compute nodes. Each node is equipped with 2 Nvidia RTX 3090 GPUs, 20 CPU cores, and 64GB RAM, interconnected with the Mellanox ConnectX5 NIC. The nodes are managed by Docker, and all run on 64-bit Ubuntu 18.04 with CUDA v11.1.

Baseline Schedulers. We compare GREEN to 5 schedulers, representing the state-of-the-art in their respective categories, as introduced in §2. For schedulers that are not scale-adaptive, we preset each job's GPU allocation to the point where GREEN achieves the best energy efficiency.

1) *Tiresias* [7]. Tiresias adopts Least-Attained-Service (LAS) scheduling for ML clusters.

2) *GAIA's Carbon-Time* [9]. GAIA's Carbon-Time policy schedules jobs to start at times that minimize carbon emissions, allowing a maximum delay specified by parameter W. We set W to 12 hours, as suggested by the original paper.

3) *EcoVisor* [28]. EcoVisor enforces carbon-aware policies for jobs. We extend and implement its policy: jobs are queued in arrival order if the carbon intensity is higher than the threshold (set 10th percentile of the CIC in use).

4) *Pollux* [20]. Pollux is the state-of-the-art ML scheduler, jointly optimizing job hyperparameters and cluster resource allocation. Pollux is not model-agnostic and not energy-aware.

5) *Zeus* [39]. Zeus adjusts model hyperparameters and GPU power limits for individual jobs. We use Tiresias to allocate resources across jobs in our evaluation.

ML Workload. We collected 791 jobs from real users over a 24-hour period on a university-managed production cluster [32], containing self-contained job packages (code, data, configs). We replayed these jobs in our testbed following their original submission timeline. Our evaluation, with 791 jobs, is about 5 times larger than prior work (e.g., Pollux with 160 jobs [20]). The training duration of each job ranged from 1 minute to 37 hours (detailed in Table 2) and involved large ML models for tasks such as image classification (ResNet-50 [10], ShuffleNet [41]), speech recognition (DeepSpeech2 [2]), and natural language processing (BERT [5], GPT-2 [21]).

Pollux and Zeus are not model-agnostic and require job-specific configurations. Since user-submitted jobs may have incompatible implementations, we invested 100 man-hours to manually adjust them. Over 55% of the 791 jobs had unresolvable issues, such as unsupported learning rate scalers or gradient optimizers, while about 25% were excluded due to non-standard ML model implementations. Ultimately, we adjusted 150 jobs for non-model-agnostic evaluations (§8.2.3).

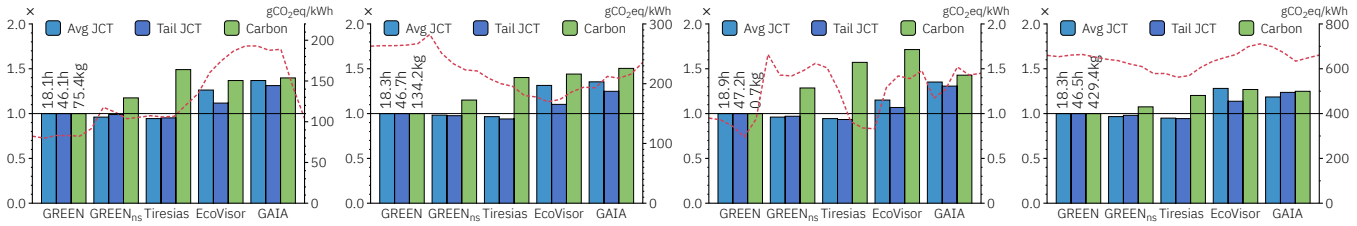
JCT Measurements. Typical ML cluster schedulers (as seen in baseline schedulers) operate in multi-tenant ML clusters where users submit jobs that run independently without deadline constraints. This workflow is also known as MLaaS [33]. The cluster scheduler allocates resources to optimize cluster-wide metrics like average JCT. In our testbed evaluation, we replayed jobs on their original timeline and calculated individual JCTs as the time from submission to completion — including queueing and preemption, with each job's running time determined by its user program.

Carbon Footprint Estimation. We tracked energy consumption to estimate emissions (details in §4), following a calculation method consistent with existing carbon-aware systems [8, 23, 28, 34]. In our evaluation, we used historical hourly carbon intensity data from Electricity Maps [16] for August 2023 across four regions: California, USA; the United Kingdom; Sweden; and Poland. These regions represent diverse combinations of carbon intensity and variability. Figure 8 presents the 24-hour carbon intensity patterns for each region in their respective subfigures with dotted red lines.

GREEN's scheduling algorithm (§5) dynamically computes boundary values for job shifting based on a given Carbon Intensity Curve (CIC). In our evaluation, we provide a 48-hour CIC to GREEN, representing typical carbon intensity patterns. For the year-long simulator experiments, a new 24-hour CIC is provided for every 24 hours to GREEN using consecutive historical data in 2023 to evaluate GREEN's adaptability to long-term, seasonal changes in carbon intensity.

8.2 Testbed Experiments

GREEN reduces the cluster-wide carbon footprint by up to 41.2%, with a trade-off of up to 3.6%-5.9% in average JCT and 5.1%-7.1% in tail JCT under a real ML workload when



(a) United Kingdom **(b)** California, USA **(c)** Sweden **(d)** Poland

Figure 8: Cluster-wide carbon footprint and JCT were evaluated across four regions with different carbon intensity profiles. The dotted red line shows the 24-hour carbon intensity pattern (right axis). The results are shown as ratios to GREEN (left axis).

compared to state-of-the-art cluster schedulers.

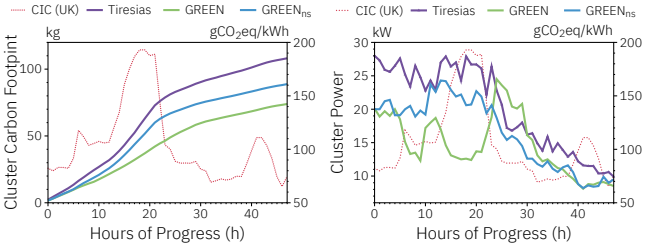
8.2.1 Cluster-wide Carbon Footprint

Figure 8 shows that GREEN reduces the cluster-wide carbon footprint by up to 41.2% (32.2% on average across 4 regions). When evaluating GREEN without job shifting over time (shown as GREEN_{ns}, meaning 'no shifting'), it achieves an average carbon reduction of 21% across the four regions.

Performance variations across regions. Regions with high variability in carbon intensity see more significant carbon footprint reductions due to job shifting during greener hours, such as the UK and Sweden (Figure 8a and 8c). As GREEN makes dynamic adjustments of the shifting threshold based on the carbon intensity profile, the average carbon intensity does not substantially affect the reduction percentage. However, we make a note that while the reduction percentage can be significant (over 30%), the absolute carbon footprint reduction may be minimal in areas like Sweden, where carbon intensity is comparatively low (less than 1% relative to other regions). In such cases, job shifting might introduce unnecessary overhead. These practical considerations are also noticed in recent literature [9, 30] and are further discussed in §9.

Carbon Footprint and Power Consumption. By monitoring energy efficiency during job scaling, GREEN avoids allocating additional resources to jobs with low energy efficiency. Figure 9a demonstrates the carbon footprint curve of GREEN, showing how the growth rate during high carbon intensity hours is lower than the other baselines. Figure 9b shows that GREEN reduces the cluster-wide power consumption. Under the UK CIC trace, GREEN achieves a 12% peak power reduction (from 28kW to 24.5kW) and a 25% decrease in total energy consumption (from 966kWh to 725kWh). This result shows another benefit of GREEN: physical clusters are typically designed and billed based on peak demand, and lower peak power usage reduces the required power capacity.

Comparison with Carbon-aware Baselines. GREEN outperforms GAIA and EcoVisor by 25.2% in average JCT across 4 regions. GAIA and EcoVisor delay job starts until the estimated carbon footprint is lower, often leaving resources idle during high carbon intensity periods. In contrast, GREEN dynamically calculates the job shifting threshold and prioritizes lower-power jobs during high-intensity periods, reducing



(a) Cluster-wide carbon footprint **(b)** Cluster-wide power draw

Figure 9: Cluster-wide carbon emission accumulation and power draw over the evaluation period (the same period shown in Figure 10). The dotted red line shows CIC on the right axis.

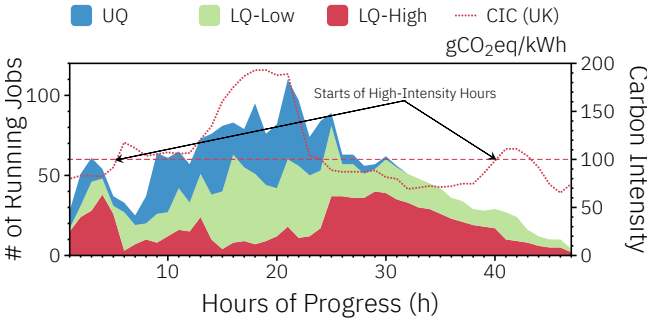


Figure 10: Job scheduling timeline showing the number of high- and lower-power jobs (left axis) responding to carbon intensity changes. UQ: Upper Queue; LQ-High/LQ-Low: Lower Queue's high/low-power jobs. The dotted red line represents the CIC and its boundary value (right axis).

power usage without leaving resources idle.

GAIA and EcoVisor exhibit higher carbon footprints because, in the multi-tenant ML cluster setting (targeted by GREEN, Tiresia, Pollux, and similar cluster schedulers), resource capacity is static: when jobs are delayed to a greener hour, idle resources still consume a baseline level of energy.

It is important to note that GAIA and EcoVisor are effective in their intended cloud environments, where resources are requested on demand by users, and idle resources do not incur additional costs or carbon emissions. Our evaluation extends their policies for multi-tenant ML cluster setting, where resource competition and constraints are factors.

Job Size (% of Total Jobs)	JCT Increase	
	Average	Tail
Extra Small Jobs (0-9 minutes, 22%)	−0.5%	−0.4%
Small Jobs (10-59 minutes, 30%)	1.7%	1.2%
Medium Jobs (1 - 10 hours, 30%)	4.4%	4.8%
Large Jobs (≥ 10 hours, 18%)	6.9%	5.8%

Table 2: GREEN's impact on individual JCTs compared to Tiresia across 4 job size categories.

8.2.2 JCT Performance

Figure 8 shows that GREEN maintains comparable JCT to baseline schedulers. Compared to the best-performing baseline, Tiresia, GREEN shows a 3.6%-5.9% increase in average JCT and a 5.1%-7.1% increase in tail JCT. With time-shifting disabled (GREEN_{ns}), the difference is smaller at 1.7%-1.8%, while still achieving a 21% reduction in carbon footprint.

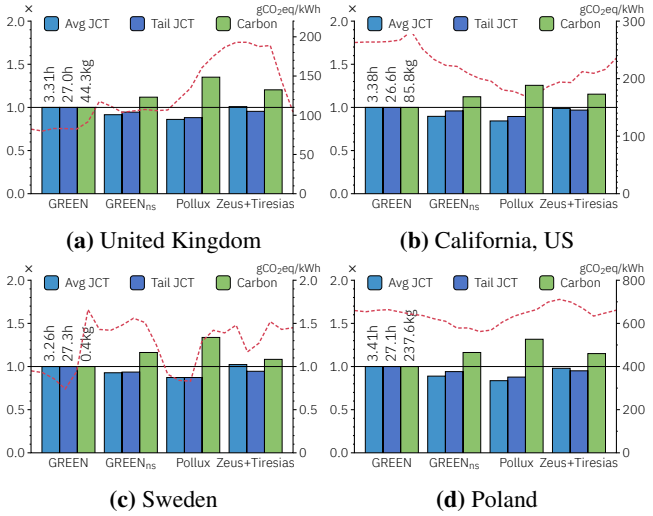
Cluster-wide Job Timeline. Figure 10 illustrates the job scheduling timeline in our testbed. During the transition between high and low carbon intensity hours, job priorities are swapped — lower-power jobs take precedence in high-intensity periods to reduce cluster power consumption. This adjustment is also reflected in Figure 9b, where power draws decrease during high carbon intensity hours. As shown in the figure, not all high-power jobs are stopped during high-intensity periods. They still run when (1) lower-power jobs do not fully utilize available resources or (2) high-power jobs have smaller carbon terms (§5) and gain higher priority.

Impact on Individual JCTs. Table 2 shows the evaluation of GREEN's impact on jobs' individual JCTs by comparing with Tiresia (the best JCT baseline) across 4 size categories based on total running time. GREEN had a maximum JCT increase of 6.9% for large jobs and a decrease of 0.5% for smaller jobs, demonstrating its effective use of MLFQ to prioritize shorter jobs while avoiding starvation of longer ones.

To understand the trade-off between overall JCT and carbon consumption, we conducted sensitivity analyses to evaluate how different extents of time shifting impact this trade-off. The results are provided in Appendix A for interested readers.

8.2.3 Non-Model-Agnostic Baselines

Figure 11 shows that GREEN reduces the carbon footprint by 23.9% on average across four regions compared to Pollux, with trade-offs of 17.2% in average JCT and 13.4% in tail JCT. Given that Pollux outperforms prior model-agnostic schedulers like Tiresia and Optimus by over 50% [20], this highlights GREEN 's ability to balance speed and carbon efficiency while remaining model-agnostic. GREEN also outperforms the combined Zeus (single-job energy optimization with GPU power limits) and Tiresia solution, achieving a 12.7% carbon footprint reduction with a similar average JCT.



(a) United Kingdom **(b)** California, US

(c) Sweden **(d)** Poland

Figure 11: Non-model-agnostic baselines evaluated with 150 compatible testbed jobs. Results are shown as ratios to GREEN (left axis), with CIC in dotted red lines (right axis).

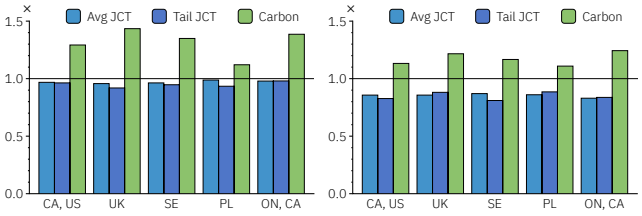
8.3 Trace-Driven Simulator Experiments

We conducted a year-long simulator experiment to evaluate GREEN's adaptability to seasonal changes in carbon intensity.

Simulator Construction. The simulator operates as a step-based discrete-time system with a 1-minute granularity, resulting in 518,400 steps for a full year. Each step performs three tasks: (1) calculate job progress and carbon footprint based on GPU time, progress factor (%/minute), current CIC, and power draw; (2) check the job trace for new job arrivals and adding them to the queue; (3) call the simulated scheduler to adjust jobs based on scheduling policies.

For workload simulation, we sample 400 job submissions from a 24-hour window in Alibaba's cluster workload trace [33] and repeat this process 365 times to construct a year-long trace with 146,000 jobs. The trace provides submission times and durations but lacks model-specific details. Following prior simulation methodologies in Pollux and Zeus, we categorize jobs by duration and map them to training jobs from our production workload (§8.1). To accurately model job execution, we pre-run jobs under varying configurations (e.g., GPU count, batch size) and record progress factors and power consumption. This enables the simulator to infer job progress and energy usage based on scheduling decisions. Additionally, we track checkpoint-resume overheads for each job category for preemptions to ensure simulation fidelity.

Performance Analysis. Figure 12 shows our simulation results. We evaluated GREEN across five regions, including Ontario, Canada, and compared it to the best-performing baselines, Tiresias and Pollux. The simulator showed improvement patterns across regions similar to our testbed results (showing better performance in high-variance locations). Compared to Tiresias, GREEN reduced the carbon footprint by 31.6%, with a 5.1% increase in average JCT and 7.5% in tail JCT.



(a) Tiresias **(b)** Pollux

Figure 12: Baselines evaluated in a year-long simulation across 5 regions. Results are shown as ratios to GREEN.

Job Size (% of Total Jobs)	JCT Increase	
	Average	Tail
Extra Small Jobs (0-9 minutes, 16%)	−0.1%	−0.3%
Small Jobs (10-59 minutes, 38%)	2.4%	4.7%
Medium Jobs (1 - 10 hours, 36%)	5.5%	8.0%
Large Jobs (≥ 10 hours, 8%)	5.9%	6.1%

Table 3: GREEN's impact on individual JCTs compared to Tiresia in the simulator, breaking down by job sizes.

Compared to Pollux, GREEN achieved a 21.1% reduction in carbon footprint with a 15% increase in both average and tail JCT. We also report individual JCT impact in Table 3. Overall, the results show that GREEN adapts to seasonal carbon intensity variations to deliver carbon and time efficiency.

9 Applicability and Limitations

We discuss the applicability of GREEN's core designs.

Carbon Tracking and Forecasting. While GREEN effectively reduces carbon emissions through optimized GPU scheduling, several broader factors influence ML clusters' overall carbon footprint. Notably, *embodied carbon*—the emissions associated with hardware manufacturing, transportation, and lifecycle—is not considered in GREEN's model. Additionally, GREEN does not account for *data center Power Usage Effectiveness (PUE)*, which captures energy overheads from cooling, power distribution, and auxiliary infrastructure. We note that scheduling has a limited impact on carbon emissions from these factors. Recent work has shown that carbon-aware scheduling should not account for embodied carbon emissions, as they represent a sunk cost [4]. Similarly, PUE optimization is typically addressed at the data center design level and largely independent of workload scheduling.

GREEN dynamically adjusts workload execution based on carbon intensity forecasts, leveraging publicly available carbon data to estimate emissions. However, inaccuracies in carbon intensity predictions can lead to suboptimal scheduling decisions, and incorporating more accurate forecasting methods [15] could improve scheduling effectiveness. Furthermore, integrating real-time carbon intensity data directly from data centers presents an opportunity to enhance carbon-aware scheduling, which remains an open research challenge.

Scheduling Mechanism. GREEN's scheduling mechanism supports diverse ML task types and GPU heterogeneity, as its carbon tracking methods can generalize across various ML training regimes. However, GREEN is not well-suited for inference workloads, which are latency-sensitive as they respond to user requests in real time. The preemption-based scheduling mechanisms used by GREEN may introduce unexpected interruptions for inference tasks.

We also identified the following cases where GREEN's scheduling approach may have limited effectiveness:

- Special carbon intensity patterns: In regions with consistently low carbon intensity (e.g., Sweden), significant emission reductions are harder to achieve. Similarly, when carbon intensity remains stable, temporal job shifting may not be effective and could cause unnecessary scheduling overhead from frequent preemptions.
- Long job starvation: While GREEN includes starvation mitigation mechanisms, long-running jobs in highly congested clusters may still experience delays due to its LAS-based scheduling. Like Tiresias [7], a starvation threshold that resets job priority can be used to prevent starvation.

Integration with ML Frameworks. System research in ML is progressing quickly, with significant improvements in frameworks and optimization methods [12, 17, 27, 40]. Previous studies [33] show that GPU usage can vary widely across ML workloads depending on the optimization techniques used, which also affect power consumption. GREEN is designed to be *non-intrusive*, meaning it can schedule ML tasks without requiring users to change their code. Therefore, it is *orthogonal* to job-level optimizations, allowing it to work with both current and future ML system improvements.

10 Conclusion

GREEN utilizes a carbon-intelligent algorithm to schedule, scale, and shift workloads to optimize cluster-wide environmental and efficiency performance. We evaluate GREEN under real ML workloads in a production cluster and show that it can achieve significant carbon footprint reductions while maintaining time efficiency comparable to existing work.

Acknowledgments

We thank our shepherd, Noman Bashir, and the anonymous reviewers from NSDI 2025, as well as reviewers from our previous submissions, for their valuable feedback. GREEN was developed and evaluated on the TACC cluster [37] at the Hong Kong University of Science and Technology. This work is supported in part by the Hong Kong RGC TRS T41-603/20R, GRF 16213621, NSFC 62062005, 62402407. Han Tian and Kai Chen are the corresponding authors.

References

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, pages 118–132. ACM, 2023.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 173–182. JMLR.org, 2016.

[3] Victor Avelar, Patrick Donovan, Paul Lin, Wendy Torell, and Maria A Torres Arango. The AI disruption: Challenges and guidance for data center design. *Schneider Electric*, 2023.

[4] Noman Bashir, Varun Gohil, Anagha Belavadi Subramanya, Mohammad Shahrad, David E. Irwin, Elsa Olivetti, and Christina Delimitrou. The sunk carbon fallacy: Rethinking carbon footprint metrics for effective carbon-aware scheduling. In *Proceedings of the 2024 ACM Symposium on Cloud Computing, SoCC 2024, Redmond, WA, USA, November 20-22, 2024*, pages 542–551. ACM, 2024.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[6] Diandian Gu, Yihao Zhao, Peng Sun, Xin Jin, and Xuanzhe Liu. Greenflow: A carbon-efficient scheduler for deep learning workloads. *IEEE Trans. Parallel Distributed Syst.*, 36(2):168–184, 2025.

[7] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, pages 485–500. USENIX Association, 2019.

[8] Walid A. Hanafy, Qianlin Liang, Noman Bashir, David E. Irwin, and Prashant J. Shenoy. Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency. In Michele Garetto, Andrea Marin, Florin Ciucu, Giulia Fanti, and Rhonda Righter, editors, *Abstracts of the 2024 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/PERFORMANCE 2024, Venice, Italy, June 10-14, 2024*, pages 49–50. ACM, 2024.

[9] Walid A. Hanafy, Qianlin Liang, Noman Bashir, Abel Souza, David E. Irwin, and Prashant J. Shenoy. Going green for less green: Optimizing the cost of reducing cloud carbon emissions. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, pages 479–496. ACM, 2024.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[11] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, pages 457–472. ACM, 2023.

[12] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast distributed computation over slow networks. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 273–288. USENIX Association, 2020.

[13] Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai

Chen. Understanding communication characteristics of distributed training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, APNet '24, page 1–8, New York, NY, USA, 2024. Association for Computing Machinery.

[14] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 289–304. USENIX Association, 2020.

[15] Diptyaroop Maji, Prashant J. Shenoy, and Ramesh K. Sitaraman. Carboncast: multi-day forecasting of grid carbon intensity. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2022, Boston, Massachusetts, November 9-10, 2022*, pages 198–207. ACM, 2022.

[16] Electricity Map. https://www.electricitymap.org/map, Accessed August 2024.

[17] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 1–15. ACM, 2019.

[18] National Grid ESO, Environmental Defense Fund Europe, University of Oxford Department of Computer Science, and WWF. Carbon intensity API. https://carbonintensity.org.uk, Accessed August 2024.

[19] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 3:1–3:14. ACM, 2018.

[20] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In Angela Demke Brown and Jay R. Lorch, editors, *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 2021.

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[22] Ana Radovanovic, Bokan Chen, Saurav Talukdar, Binz Roy, Alexandre Duarte, and Mahya Shahbazi. Power modeling for effective datacenter planning and compute management. *IEEE Transactions on Smart Grid*, 13(2):1611–1621, 2021.

[23] Ana Radovanovic, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. Carbon-aware computing for datacenters. *IEEE Transactions on Power Systems*, 38(2):1270–1280, 2022.

[24] TACC Github Repo. https://github.com/turingaicloud/.

[25] SchedMD. https://slurm.schedmd.com/documentation.html, Accessed August 2024.

[26] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

[27] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

[28] Abel Souza, Noman Bashir, Jorge Murillo, Walid A. Hanafy, Qianlin Liang, David E. Irwin, and Prashant J. Shenoy. Ecovisor: A virtual energy system for carbon-efficient applications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, pages 252–265. ACM, 2023.

[29] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 642–657. ACM, 2023.

[30] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David E. Irwin, and Prashant J. Shenoy. On the limitations of carbon-aware temporal and spatial workload shifting in the cloud. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*, pages 924–941. ACM, 2024.

[31] WattTime. https://www.watttime.org/, Accessed August 2024.

[32] TACC Website. https://tacc.ust.hk/.

[33] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu 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 2022, Renton, WA, USA, April 4-6, 2022*, pages 945–960. USENIX Association, 2022.

[34] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kain Kordian Gontarska, and Lauritz Thamsen. Let's wait awhile: how temporal workload shifting can reduce carbon emissions in the cloud. In *Middleware '21: 22nd International Middleware Conference, Québec City, Canada, December 6 - 10, 2021*, pages 260–272. ACM, 2021.

[35] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 595–610. USENIX Association, 2018.

[36] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 533–548. USENIX Association, 2020.

[37] Kaiqiang Xu, Decang Sun, Hao Wang, Zhenghang Ren, Xinchen Wan, Xudong Liao, Zilong Wang, Junxue Zhang, and Kai Chen. Design and operation of shared machine learning clusters on campus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 295–310. ACM, 2025.

[38] Zeyu Yang, Karel Adámek, and Wesley Armour. Part-time power measurements: nvidia-smi's lack of attention. *CoRR*, abs/2312.02741, 2023.

[39] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, pages 119–139. USENIX Association, 2023.

[40] Chaoliang Zeng, Xudong Liao, Xiaodian Cheng, Han Tian, Xinchen Wan, Hao Wang, and Kai Chen. Accelerating neural recommendation training with embedding scheduling. In Laurent Vanbever and Irene Zhang, editors, *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association, 2024.

[41] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6848–6856. Computer Vision Foundation / IEEE Computer Society, 2018.

A Deep Dive Experiments

In the deep dive experiments, we inspect the effectiveness of each GREEN's design component and also conduct a sensitivity analysis of the parameter used in the scheduling algorithm.

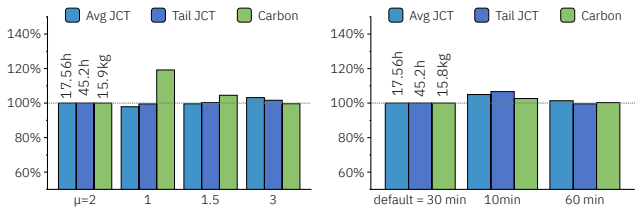
A.1 Understanding the Optimizers

Impact of Energy Efficiency Optimizer (§5.1). The optimizer allocates more resources to jobs with higher energy efficiency. We evaluate the impact of energy-efficiency-based scaling by fixing the number of GPUs to 1 and 2. Our results in Table 4 demonstrate that dynamic scaling decisions impact both energy consumption and execution speed, as evidenced by its worse performance in both carbon footprint and JCT.

Impact of Carbon Footprint Optimizer (§5.2). We replaced the carbon term with an LAS factor: the number of GPU hours consumed by the job. The results, as presented in Table 4, indicate that the cluster-wide carbon footprint increases by approximately 16%, as jobs with higher energy efficiency are no longer prioritized for scaling. We then disabled the temporal shifting by fixing the Shifting Factor to 1. While JCT performance improves slightly, the cluster-wide carbon footprint increases by 20%, as GREEN no longer moves high-power tasks to greener hours.

Scheduler	Job Completion Time		Carbon
	Average	Tail	
Energy Efficiency Optimizer			
GREEN	17.56h	45.20h	15.88kg
No scaling (1 GPU)	19.91h	48.89h	18.38kg
No scaling (2 GPU)	18.53h	46.57h	16.71kg
Carbon Footprint Optimizer			
GREEN	17.56h	45.20h	15.88kg
Carbon Factor = GPU hours	18.30h	47.15h	18.52kg
No Shifting	17.19h	44.99h	18.93kg

Table 4: Ablation experiments of GREEN optimizers.



(a) temporal shifting **(b)** scheduling interval

Figure 13: Sensitivity test of μ (§5.2). μ controls the level of temporal shifting. A larger μ value will shift more high-power jobs to greener hours. Sensitivity test of scheduling interval (§6.3). 30-minute or longer interval works well because carbon intensity changes slowly, allowing less frequent preemptions.

A.2 Sensitivity Analysis

Extent of Temporal Shifting (Figure 13a). In §5.2, we set $\mu\,(\geq 1)$ in controlling temporal shifting. Our tests indicate that setting $\mu = 2$ optimizes the balance between relevant factors. A too high μ skews scheduling, prioritizing high-power jobs unfairly due to an amplified shifting term.

Length of Scheduling Interval (Figure 13b). The interval in MLFQ scheduling influences preemption frequency (§6.3). Our default is 30 minutes, and sensitivity analysis reveals that its exact length has minimal impact on outcomes. This is because carbon intensity, despite significant absolute variations, changes slowly, allowing less frequent preemptions without compromising overall cluster optimization.

B Artifacts

GREEN is designed and evaluated as a scheduler policy on the TACC cluster [32] at the Hong Kong University of Science and Technology. We will release GREEN's source code in TACC's open-source repo [24], which includes the scheduler implementation as a Slurm plugin, a daemon program, and a profiler agent, along with scripts and traces for submitting jobs to the scheduler. More details on our cluster operation and analysis can be found in [13, 37].