AI-Driven Multi-Dimensional Computing Power Scheduling: Adaptive Resource Coordination via Task Demand Prediction in Heterogeneous Clusters

Tao Wang[1], Xingyuan Zhao[2], Xinhui Qiao[1,*], Shutao Li[3] and Weichuan Yin[2]

[1] Beijing North-Star Technology Development Co., Ltd., Beijing, 100032, China

[2] China Energy Digital Technology Group Co., Ltd., Beijing, 100032, China

[3] China Power Engineering Consulting Group Co., Ltd., Beijing, 100020, China

INFORMATION

Keywords:
Resource scheduling
flexible resource allocation
artificial intelligence
computing power

DOI: 10.23967/j.rimni.2025.10.67905

SCIPEDIA Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

AI-Driven Multi-Dimensional Computing Power Scheduling: Adaptive Resource Coordination via Task Demand Prediction in Heterogeneous Clusters

Tao Wang[1], **Xingyuan Zhao**[2], **Xinhui Qiao**[1,*], **Shutao Li**[3] **and Weichuan Yin**[2]

[1]Beijing North-Star Technology Development Co., Ltd., Beijing, 100032, China

[2]China Energy Digital Technology Group Co., Ltd., Beijing, 100032, China

[3]China Power Engineering Consulting Group Co., Ltd., Beijing, 100020, China

ABSTRACT

This paper introduces an AI-driven computing power scheduling framework that innovatively integrates multidimensional resource optimization with machine-learning-based task-demand prediction to significantly enhance computational efficiency and resource utilization. Unlike prior works that primarily focus on Graphics Processing Unit (GPU) allocation, our method pioneers a holistic resource-coordination mechanism that dynamically balances GPU, memory, Central Processing Unit (CPU), and other critical resources according to their joint impact on task performance and cluster efficiency. A core innovation is our data-driven resource predictor, which autonomously analyzes historical task patterns and forecasts future demand, enabling the scheduler to adaptively scale resources so that user over-provisioning is reduced and under-allocation bottlenecks are avoided. Experimental validation demonstrates that this closed-loop prediction–scheduling paradigm achieves breakthroughs in both scale and efficiency: a 25.7% increase in concurrent task deployment, a 15.6% increase in task completion rate, and substantial relative utilization improvements of 7.5% for GPU and 8.0% for memory, outperforming conventional single-resource optimization approaches. These advancements establish a new direction for intelligent resource management in large-scale heterogeneous computing environments.

OPEN ACCESS

Received: 15/05/2025

Accepted: 11/10/2025

Published: 23/01/2026

DOI
10.23967/j.rimni.2025.10.67905

Keywords:
Resource scheduling
flexible resource allocation
artificial intelligence
computing power

1 Introduction

To train large-scale artificial-intelligence (AI) workloads efficiently, modern data centers operated by research institutions, technology companies, and cloud providers deploy heterogeneous computing clusters equipped with hardware accelerators such as graphics processing units (GPUs) and neural network processing units (NPUs). Although these accelerators significantly reduce training time and improve operational throughput, a purely hardware-centric scaling strategy incurs prohibitive capital expenditure, faces fundamental scalability limits, and yields diminishing marginal performance

*Correspondence: Xinhui Qiao (qiaoxinhuiii@163.com).

returns. Consequently, optimizing the utilization of existing computational resources has become imperative.

This challenge has driven the rise of AI computing power scheduling as a critical area of research. AI computing power scheduling refers to the process of efficiently matching the hardware resources of computing clusters with computational tasks. The primary goals include improving resource utilization, reducing task execution time, and optimizing energy consumption. Over recent years, researchers have developed a variety of scheduling strategies [1–5], most of which are based on analyzing AI task workload characteristics. Xiao et al. [1] proposed the Gandiva scheduler, which improves GPU utilization by analyzing the periodic characteristics of deep-learning tasks and dynamically allocating GPU resources. However, it focuses mainly on GPU optimization and gives insufficient consideration to other resources such as CPU and memory, limiting its effectiveness in scenarios with diverse task types and complex resource requirements. Gao et al. [2] proposed the Chronus scheduler, which incorporates the preemptive and location-sensitive characteristics of tasks to achieve efficient scheduling of heterogeneous training workloads. Although highly adaptable, its implementation is complex, and scheduling conflicts are likely under resource contention, thereby degrading task execution. Xie et al. [3] introduced the Elan scheduler, which leverages elastic resource scaling and lightweight state replication to reduce task queuing time and improve resource utilization. Nevertheless, state replication incurs additional overhead that can impair execution efficiency, and aggressive scaling may result in uneven resource allocation. Peng et al. [4] presented the Optimus scheduler, which adopts an AI task performance model that combines convergence speed to make dynamic resource allocation and task placement decisions, improving both task execution efficiency and resource utilization. However, the complexity of performance modeling and dynamic decision-making complicates implementation and maintenance, and optimization becomes challenging under dynamically varying workloads. Mohan et al. [5] analyzed the impact of different CPU and memory allocations on AI tasks and incorporated task sensitivity to heterogeneous resources into scheduling decisions, thereby improving task performance. Nonetheless, the study did not fully consider multi-dimensional hardware utilization at the cluster level; when the number of resource types increases and task requirements become more complex, the design and implementation of the scheduling policy face significant difficulties. These methods typically involve fine-tuning scheduling policies to align with workload demands, aiming to achieve objectives such as load balancing, task prioritization, or resource fairness. Cui et al. [6] proposed a latency-aware scheduler based on deep reinforcement learning and attention mechanisms, which reduces upgrade makespan and request latency. However, the approach incurs high training overhead and may lead to unbalanced resource allocation during container migration. Yang et al. [7] introduced Meta-Scheduler, which employs meta-learning and stage-aware decision-making to shorten job completion time and enhance cluster throughput. Nevertheless, the meta-learning process introduces non-negligible overhead and may impair responsiveness under dynamic workloads. Zhou et al. [8] conducted a comprehensive survey on deep-reinforcement-learning-based resource scheduling, unifying problem formulations and outlining directions for improving job completion time and resource utilization. However, the survey lacks a concrete scheduler implementation, and the high training overhead of DRL may degrade responsiveness and cause resource imbalance under dynamic workloads. Wang et al. [9] proposed a Value-of-Information scheduler that uses information-aware prioritization and dynamic deadline adjustment to reduce latency and improve task completion in vehicular networks. Yet the value computation adds overhead and may adversely affect responsiveness in highly dynamic traffic environments.

Despite these advancements, existing methods exhibit several notable shortcomings:

1. Existing methods predominantly focus on GPU resource allocation while overlooking the impact of other critical resources, such as CPUs, memory, and network bandwidth, on task performance and overall cluster resource utilization.
2. The lack of accurate task resource-demand prediction and adaptive adjustment mechanisms impedes the ability to respond to real-time workload fluctuations, resulting in suboptimal resource utilization.
3. Current approaches fail to effectively address resource over-allocation by users, leading to resource waste and reduced efficiency in computing clusters.

To maximize the utilization efficiency of hardware resources in artificial intelligence (AI) production clusters, designers consider various resource factors—including memory, storage, and computational capabilities—when building large-scale AI clusters. Researchers have optimized non-hardware-accelerator resources, such as memory capacity, memory transfer speed, and central processing unit (CPU) performance, to enhance overall operational efficiency.

Against this backdrop, this study proposes an AI-driven scheduling framework that predicts task-specific resource demands to optimize multidimensional resource allocation and management. The framework aims to enhance AI task execution efficiency while maximizing the utilization of heterogeneous computational resources. Specifically, we develop a multidimensional resource-demand prediction model to quantitatively assess the influence of resource allocation on task performance. Building upon this model, we design an adaptive resource-scheduling algorithm and its corresponding system architecture to dynamically manage and allocate computational resources, thereby satisfying the diverse requirements of AI tasks.

2 AI Task Resource Demand Prediction Model

2.1 Overview of Resource Requirements for Tasks

In AI computing-power scheduling, task resource requirements denote the heterogeneous resources requested by users according to their computational demands upon job submission. The judicious allocation of these resources constitutes the core of the scheduling process. Table 1 quantifies the resource requirements of four representative tasks in a production AI cluster [10], illustrating the distinct multidimensional utilization patterns across task types. Evidently, tasks exhibit heterogeneous resource signatures; therefore, scheduling policies must explicitly incorporate these multidimensional characteristics to ensure efficient resource utilization. The taxonomy and formal definition of task types provide essential data-level support for such policies.

Table 1: Resource status of common AI task loads in production cluster logs

Task load	Average CPU usage/core	Average GPU usage/%	Average memory usage/GB
Recommendation tasks	3.7	6.9	2.8
Graph neural networks	11.8	0.7	9.6
Natural language processing	10.5	34.3	9.8
Image classification	2.3	30.1	19.7

For AI tasks, resource requirements span multiple dimensions, including GPUs, CPUs, memory, network bandwidth, and I/O. By profiling resource-consumption patterns across task types within

AI clusters, we classify AI workloads into three categories: GPU-intensive, CPU-intensive, and memory-intensive. Neural-network-based machine translation, for instance, is predominantly GPU-intensive [11], whereas recommender systems are typically CPU-intensive, as most execution time is devoted to data preprocessing and feature engineering. Memory-intensive tasks, such as large-scale image classification, expend 10%–70% of training time on data loading and storage operations, demonstrating that effective GPU utilization is often bottlenecked by CPU and memory performance. Consequently, an accurate resource-demand prediction model that jointly captures the interactions among multidimensional resources is required. Equipped with such a model, resource-allocation strategies can be tailored to each workload class to maximize cluster-wide utilization and significantly improve AI task execution efficiency.

2.2 *Dynamic Time Warping (DTW)*

This paper develops a task-resource demand-prediction model to gain deeper insight into the resource-usage characteristics of AI tasks [12]. The model quantifies the impact of heterogeneous resource configurations on task execution efficiency and provides data-driven guidance for the dynamic scheduling of AI workloads. The first step is dataset acquisition. After comparing the three GPU-cluster logs listed in Table 2, we select the Alibaba PAI trace because it provides the most comprehensive allocation information, covering GPU, CPU, and memory requests, network-bandwidth and I/O usage, and task-type labels. These features are well suited for analyzing resource-usage patterns and for subsequent scheduling optimization. Using the PAI trace, we build a regression model that treats the number of CPU cores, GPU utilization, memory demand, network traffic, and GPU-memory usage as independent variables and the actual task execution time as the dependent variable. The resulting model yields accurate predictions of task resource demands, enabling refined allocation and scheduling strategies that improve cluster-wide resource-utilization efficiency.

Table 2: Comparison of different GPU cluster log datasets

GPU cluster	PAI	Helios	Phily
Total number of GPUs	3.7	6.9	2.8
Duration/d	11.8	0.7	9.6
Resource type	10.5	34.3	9.8
Task type	2.3	30.1	19.7

In the regression modeling process, we employ Light Gradient Boosting Machine (LightGBM) to quantify the multidimensional resource demands of heterogeneous task types and to characterize their impact on task execution time. LightGBM is a gradient-boosting framework that trains an ensemble of decision trees in a stage-wise manner, iteratively fitting the residuals obtained from previous iterations. The final prediction is obtained by summing the weighted outputs of all constituent trees.

$$\hat{y}^{(i,t)} = \hat{y}^{(i,t-1)} + \eta \cdot f_t(x_i) \tag{1}$$

where $\hat{y}^{(i,t)}$ denotes the predicted value of the i-th sample at iteration t, and $\hat{y}^{(i,t-1)}$ denotes the predicted value of the i-th sample at iteration $t - 1$. $f_t(x_i)$ is the output of the t-th weak learner, i.e., a decision tree that fits the current residuals. The learning rate η controls the contribution of each tree to the ensemble, and t indicates the current boosting round.

The objective function of the LightGBM-based task-resource demand-prediction model is:

$$Obj(t) = \sum\nolimits_{i=1}^{n} l(y_i, \hat{y}^{(i,t-1)} + f_t(x_i)) + \Omega(f_t) \tag{2}$$

where $y_i, \hat{y}^{(i,t-1)} + f_t(x_i)$ is the loss function, which represents the error between the prediction of the current model and the actual label. $\Omega(f_t)$ is the regularization term, which is used to control the complexity of the model. In LightGBM, the regularization term usually takes the form of the sum of the square of the number of leaf nodes and the weight of the tree (L2 regularization).

The LightGBM-based task-resource demand-prediction model enhances the accuracy of the decision-tree ensemble by iteratively fitting the residuals over t boosting rounds; the iterative process is expanded via a second-order Taylor expansion to yield the new objective function given in Eq. (3).

$$Obj(t) = \sum\nolimits_{i=1}^{n} [g_i f_t(x_i) + 0.5 h_i f_t^2(x_i)] + \Omega(f_t) \tag{3}$$

where g and h are the first and second partial derivatives, respectively, as follows:

$$g_i^{(t-1)} = \frac{\partial L\left(y_i, \hat{y}_i^{(i,t-1)}\right)}{\partial \hat{y}_i^{(t-1)}} \tag{4}$$

$$h_i^{(t-1)} = \frac{\partial^2 L\left(y_i, \hat{y}_i^{(i,t-1)}\right)}{\partial \left(\hat{y}_i^{(t-1)}\right)^2} \tag{5}$$

The final extremum is derived as follows:

$$w_j^* = -\frac{G_j}{H_j} + \lambda \tag{6}$$

$$G_j = \sum_{i \in I_j} g_i \tag{7}$$

$$H_j = \sum_{i \in I_j} h_i \tag{8}$$

The final extreme value of the objective function of the task resource demand prediction model based on LightGBM is as follows:

$$Obj = 0.5 \sum\nolimits_{j=1}^{T} \frac{G_j^2}{H_j} + \lambda \sum\nolimits_{j=1}^{T} w_j^2 + \gamma T \tag{9}$$

where G_j indicates the gradient of each leaf node; H_j is the Hessian of each leaf node. λ is the L2 regularization parameter, which is used to control the magnitude of the leaf node weights; γ is the penalty term, which controls the complexity of the tree and mainly serves to limit the depth of the tree or the number of leaf nodes; w_j is the weight of each leaf node; T is the number of leaf nodes in the tree. With these parameters, LightGBM can optimize the structure and complexity of the tree while regularizing to prevent overfitting [13].

In summary, the task-resource demand-prediction model is constructed by minimizing the objective function. Specifically, branch gain is maximized to determine the optimal split point for each leaf node; new tree branches are then added sequentially for every feature until the LightGBM model is fully trained. Within LightGBM, branch gain quantifies the change in the objective function before and after a split. Its expression is:

$$Gain = \frac{G_L^2}{H_L + \lambda} + \lambda + \frac{G_R^2}{H_R + \lambda} + \lambda - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma \tag{10}$$

In LightGBM, the branch gain is calculated based on the following key factors: G_L and G_R represent the gradient of the split left subtree and right subtree, respectively, and reflect the direction of error of each subset; H_L and H_R represent the Hessian of the left subtree and right subtree, respectively, which is the second derivative, and reflects the magnitude of error of each subset [14,15]. To control the complexity of the model and avoid overfitting, LightGBM introduces the L2 regularization parameter λ, which is used to adjust the weights of the leaf nodes. In addition, γ is a penalty term that controls the complexity of the tree and is mainly used to limit the depth of the tree and prevent the model from becoming too complex. Therefore, the calculation of branch gain takes these factors into account to select the optimal split point and improve the model's performance.

2.3 Analysis of the Results of the Task Resource Demand Forecasting Model

To assess the accuracy of the task-resource demand-prediction model, we adopt the widely used random forest algorithm as a benchmark and compare it with the LightGBM approach introduced in Section 2.2. Mean absolute error (MAE) and weighted mean absolute percentage error (WMAPE) serve as accuracy metrics, quantifying model fit and the deviation between predicted and observed values, respectively.

$$MAE = \frac{1}{N} \sum_{i=1}^{N} \left| y_i - \hat{y}_i \right| \tag{11}$$

$$WAMPE = \frac{\sum_{i=1}^{N} w_i \left| y_i - \hat{y}_i \right|}{\sum_{i=1}^{N} w_i \left| y_i \right|} \tag{12}$$

where $\hat{y}_i$ and y_i are the predicted and actual values, respectively.

Both methods are decision-tree ensemble learners. From an implementation perspective, LightGBM trains a sequence of trees via gradient boosting, iteratively fitting the residuals of preceding trees. In contrast, the random-forest method employs bagging, regressing on bootstrap samples drawn from the dataset [16,17]. Random forests primarily reduce variance and mitigate overfitting, whereas LightGBM focuses on bias reduction and leverages a histogram-based algorithm together with leaf-wise growth to accelerate training and minimize memory consumption. For accurate task-resource demand prediction—essential for precise resource allocation—we prioritize low regression bias over variance reduction. Consequently, our evaluation metrics emphasize bias-oriented measures. Empirically, LightGBM outperforms random forests in regression accuracy because its efficient data handling and accurate feature-importance assessment capture the complex relationship between multidimensional resource allocation and task execution time, thereby guiding adaptive resource scaling and scheduling. The principal advantage of LightGBM-based modeling is its ability to deliver high predictive accuracy while maintaining computational efficiency on large-scale datasets, rendering it a potent tool for enhancing the efficiency and accuracy of AI computing-power scheduling.

3 AI Adaptive Resource Scaling Scheduling

3.1 AI Computing Power Scheduling Overview

For AI computing-power scheduling, we propose a novel adaptive resource-adjustment framework that dynamically allocates resources according to the specific demands of AI tasks. This approach departs significantly from traditional cluster schedulers designed for general big-data workloads; it focuses on exploiting the unique characteristics of AI tasks to reduce job completion time and to maximize the utilization of heterogeneous hardware resources such as GPUs. The framework leverages a task-resource demand-prediction model that quantifies the impact of alternative resource configurations on task performance. During scheduling, this model guides the GPU cluster to elastically scale resources in response to task requirements. Consequently, the framework mitigates both user-initiated resource over-allocation and the inefficiencies arising from sub-optimal allocation, thereby simultaneously improving AI-task throughput and cluster-wide resource efficiency. Furthermore, the proposed prediction-driven scheduling strategy will be formalized in subsequent chapters as a mathematical model for multidimensional resource management. This model underpins our adaptive dynamic resource-scaling algorithm, whose detailed formulation and system architecture are also presented later. The introduction of this framework offers a new perspective on the efficient scheduling of AI computing power [18].

3.2 AI Computing Power Scheduling Overview

In the framework of AI computing power multi-dimensional resource scheduling, this paper assumes a heterogeneous server list $S = \{S_1, S_2, \cdots, S_M\}$ with different resource quotas composed of M servers. Each server i has a resource capacity vector $S_i = \{S_i^1, S_i^2, \cdots, S_i^R\}$ of R dimensions, which represents the server's supply capacity in each resource dimension. At the same time, each user task j has a resource request amount when submitted, expressed as $w_i = \{w_i^1, w_i^2, \cdots, w_i^R\}$. This vector describes the task's demand in each resource dimension. The core of the resource allocation and scheduling problem is how to map these tasks to servers to meet the resource requirements of the tasks while not exceeding the resource capacity of the servers. The formal definition of the resource allocation and scheduling problem is shown in Equation:

$$max \sum_{j=1}^{J} \sum_{i=1}^{M} P_j x_{i,j} \begin{cases} \sum_{j=1}^{J} w_j^r x_{i,j} \leq S_i^r r \in R, i \in M, i \in M \\ \sum_{i=1}^{M} x_{i,j} \leq 1 x_{i,j} \geq 0 \end{cases} \tag{13}$$

where the variable $x_{i,j}$ indicates that the j-th user task is allocated to the i-th server. During scheduling, tasks can be distributed across multiple servers, so $x_{i,j}$ can be a fraction; P_j is the task execution benefit of task j in the current scheduling round. The dispatch objective is to maximize the execution benefit of the current scheduling round while ensuring that the multidimensional resource request of each task does not exceed the available quota on the serving node, thereby guaranteeing that the task can be executed on the current server set.

Experiments employed the Alibaba Cloud PAI cluster trace (version 2.1), which comprises 12 840 task records annotated with eight resource features—GPU utilization, CPU core count, memory demand, and five additional metrics—and task execution time as the regression target. Pre-processing consisted of three sequential steps: (1) Outlier removal: tasks with execution times exceeding 72 h (i.e., $>3\sigma$) or zero resource requests were discarded, representing 1.7% of the data; (2) Missing-value imputation: categorical features were filled with the mode, whereas continuous features were imputed with the median of tasks sharing the same type; (3) Temporal split: records were chronologically ordered by submission timestamp, allocating the first 80% (January–August 2023) to training and the final 20% (September–October 2023) to testing to prevent look-ahead bias.

The LightGBM model was fine-tuned by grid search with five-fold cross-validation. Optimal hyperparameters were max depth 8 t, num leaves 64, learning rate 0.05, and reg lambda 0.1 for L2 regularization. Early stopping was employed with early stopping rounds fifty to prevent overfitting. In feature engineering, high-cardinality categorical variables such as task ID were encoded via target encoding, whereas continuous variables such as GPU utilization were discretized by decile binning to enhance model robustness.

3.3 AI Adaptive Resource Scaling Scheduling Algorithm

The AI computing-power adaptive resource-scaling and scheduling algorithm integrates a task-resource demand-prediction model to quantify the influence of multidimensional resource allocation on task performance, then elastically scales and adjusts the multidimensional resource requirements of tasks in the current dynamic queue, and ultimately dispatches tasks to appropriate cluster locations. The pseudocode for the adaptive resource-scaling algorithm based on the task-resource demand model is presented in Algorithm 1, and the pseudocode for the overall AI computing-power resource-scaling and scheduling algorithm is presented in Algorithm 2.

Algorithm 1: Adaptive resource scaling algorithm

```
Input: current job attributes, including job type $C_j$, user-submitted job resource request amount wj $= \{w_j^1, w_j^1, \ldots, w_j^R\}$, adaptive adjustment rate $\alpha$, resource scaling impact threshold on job performance $\theta$;
Output: scaled and adjusted resource request result. $W_j = \{W_j^1, W_j^1, \ldots, W_j^R\}$
1. model $\leftarrow$ getJobModel($C_j$)
2. $t_j \leftarrow$ getPerformance(model, $w_j$)
3. important_list $\leftarrow$ permutation_Importance(model)
4.     while $\alpha < 1$:
5.     for each resources type in $w_j$:
6.         if type $r$ is not in important list:
7.             $w_j^r \leftarrow \alpha w_j$;
9.         end if
10. End
11. $T_j \leftarrow$ get Performance
12. error $\leftarrow \left|T_j - t_j\right|$
12.     if error $< \theta$:
13.         return $W_j$
14. end
15. $\alpha = \alpha + \Delta\alpha$;
16. return $w_j$
```

The adaptive resource scaling algorithm is based on the analysis of the feature importance of the task resource demand prediction model, which screens out the resource dimensions that have a small impact on task performance and are over-requested. Next, without affecting task performance, the adaptive scaling of the multi-dimensional resources initially requested by the user task is adjusted. This algorithm can effectively solve the problem of over-requesting resources for tasks and mismatches between multi-dimensional resources. The permutation importance (PIMP) method [12] is used for feature importance analysis in adaptive resource scaling. This method calculates the predicted importance of a feature value by calculating the decrease in the model score when the feature value is randomly shuffled. The feature importance of machine learning models can analyze the importance and relevance of each input feature dimension in the regression of the target variable, and is widely used in interpretable machine learning research.

Algorithm 2: Resource scaling and scheduling algorithm for AI computing power

```
Input: Current round job list, server list;
Output: Task scheduling result
1.  Job list ← getOnlineJob(t, Job list, Pending job);
2.  if Job list is null:
3.          break;
4.  End if
5.  Job list ← sortJobList(Job list);
6.  for each job in Job list:
7.  job ← updateResRequirement(job);
8.  sortServerList(Server list);
9           if is JobExecutable(Server list, job'):
10.       jobPlacement(Server list, job');
11.       Server list ← updateState(Server list);
12.  else if
13.       Pending job ← jobPending(job');
14.  end if
15.  Job list ← updateJobState(Job list);
16.  End
```

Compared with the conventional split-information-gain approach, PIMP evaluates feature importance on unseen data, thereby alleviating the overfitting issues inherent to information-gain methods and enabling importance-based prediction for newly arriving tasks. Moreover, PIMP not only quantifies the magnitude of each feature's influence but also indicates its directional effect—positive or negative—on the prediction, which serves as the basis for resource-scaling decisions. The AI computing-power resource-scaling scheduling algorithm (Algorithm 2) first adjusts resource requirements for the tasks in the current scheduling round and then selects and updates the deployed cluster nodes. Lines 1–5 filter the set of queued tasks eligible for scheduling. Line 7 adaptively scales task resources according to task type and the corresponding demand-prediction model. Lines 9–14 assign the tasks to cluster nodes, and line 15 updates the cluster state.

The complexity of this scheduling algorithm is analyzed as follows. First, it is assumed that the queue initialization process in lines 1 to 4 and the state update process in lines 11, 13, and 15 of the algorithm have negligible complexity. Line 5 sorts tasks based on the Timesort [13] algorithm, which requires at most $O(n \cdot \log n)$ operations, where n represents the number of tasks. In the inner loop, line 8 represents the node heap sorting process, which requires at most $O(M)$ operations, where M represents the number of clustered node servers. Line 9 combines the task deployment confirmation process with the available server status, which requires at most $O(M \cdot R)$ operations, where R is the size of the resource dimension to be considered. In addition, the process of adaptive scaling and adjustment of task resource requirements in line 7 requires $O(t \cdot \log n)$ operations, where t is the adaptive scaling round and F is the number of samples used to construct the machine learning LightGBMt model. In summary, the complexity of the scheduling algorithm is $O(n \cdot \log n + n \cdot (M \cdot R + t))$. The AI computing power scheduling algorithm can dynamically adjust the adaptive scaling of resources within a certain time complexity, using the inference and prediction capabilities of the task resource demand model, to ensure task performance while improving the utilization rate of AI computing power resources and task deployment efficiency [19].

3.4 AI Adaptive Resource Scaling Scheduling Algorithm

The AI computing-power adaptive resource-scaling and scheduling framework (see Fig. 1) comprises multiple users, heterogeneous servers, a dynamic task queue, a historical execution-log repository, task-specific resource-prediction models, and a unified scheduler. Users submit tasks together with their resource requirements; servers execute these tasks; the task queue maintains the set of online submissions; the repository stores historical logs of multidimensional resource usage, execution time, and task type; the prediction models quantify the impact of resource configurations on task performance and guide adaptive resource-scaling decisions; and the scheduler instantiates the proposed method.

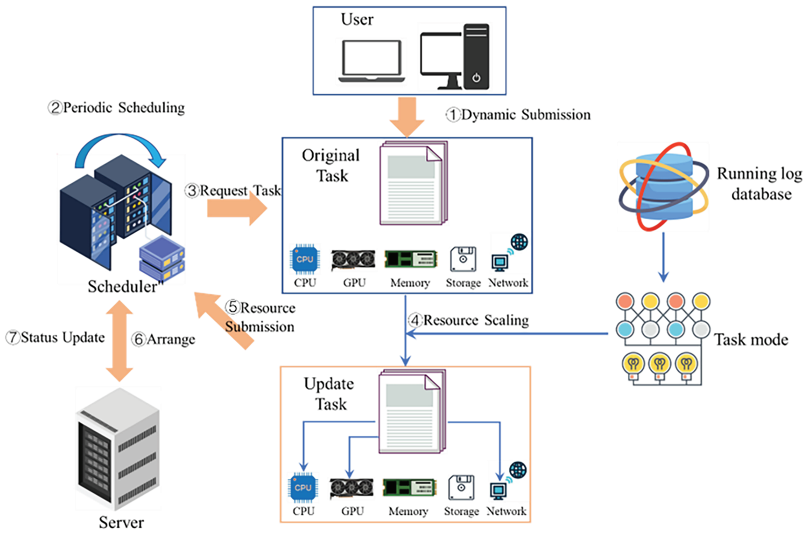


Figure 1: AI computing power adaptive resource scaling scheduling system

In this framework, the scheduler first refreshes the dynamic task queue according to newly submitted tasks and their associated attributes, including task type and multidimensional resource requests. Within the current scheduling round, the scheduler invokes the task-resource demand-prediction model to estimate the expected performance under the requested resource configuration; it then adaptively rescales the multidimensional resource demands and returns the updated requirements. Subsequently, the scheduler matches these scaled demands against the cluster's real-time resource state and dispatches each task to the most suitable server node, thereby dynamically optimizing cluster-level resource allocation. Finally, the system updates both task-execution records and cluster resource status in preparation for the next scheduling cycle. In summary, the proposed AI adaptive resource-scaling scheduler reconciles heterogeneous task requirements with multidimensional cluster resources, mitigates resource dimension mismatches, and simultaneously improves task-deployment efficiency and cluster-wide resource utilization.

4 Experimental Evaluation

Experiment 1: The results of AI computing power scheduling before and after the adaptive resource scaling method based on task-based resource demand prediction are compared to verify the improvement effect of the method described in this paper on AI computing power scheduling. In the experiment, the first-in-first-out (FIFO) strategy was used for setting the priority of tasks in scheduling, that is, the scheduling system will set the priority of tasks according to the time they arrive. Fig. 2 compares the number of tasks and the degree of task completion within the same running time (12 h). As the load increases, the degree of task completion increases. This is due to the optimization of the adaptive resource scaling and allocation scheduling method for oversubscription of resources. The effect of the optimization method in this paper is reflected in two specific ways. First, on the resource side, resources that were not fully utilized before resource scaling can be used to deploy other tasks after optimization. Second, on the task side, before optimization, tasks that could not be deployed in the limited resources of the AI computing cluster due to the user's over-application of resources could also be successfully deployed in the cluster after the over-application situation improved thanks to the resource scaling process. Judging from the results in Fig. 2, under a load of 70 tasks per hour, the task completion rate per unit time before and after adaptive resource scaling is improved by 15.6%, while the number of successfully deployed tasks also increases by 25.7%.

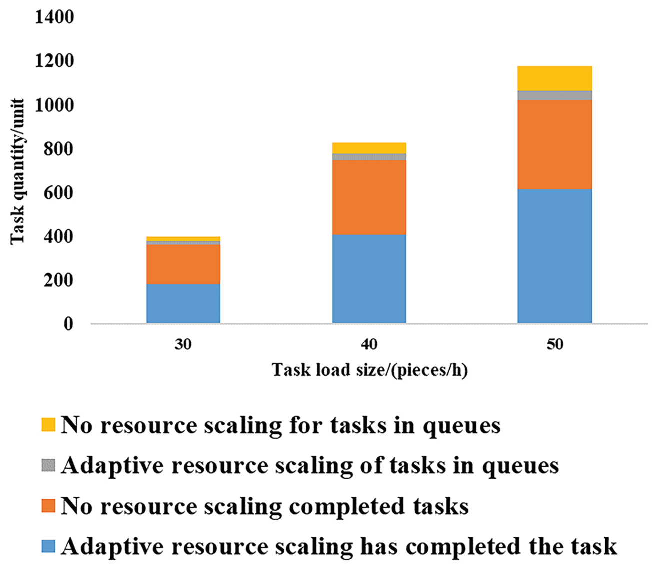
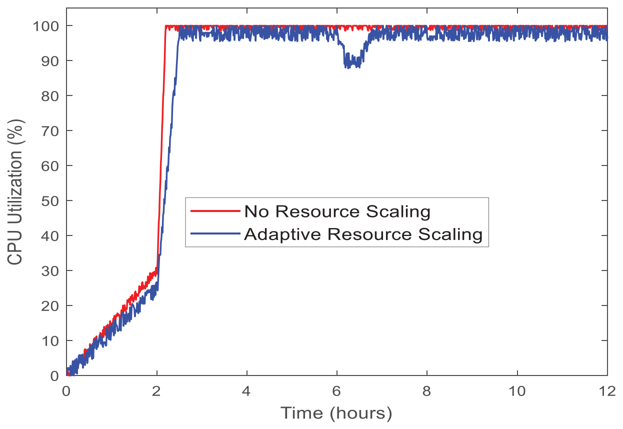
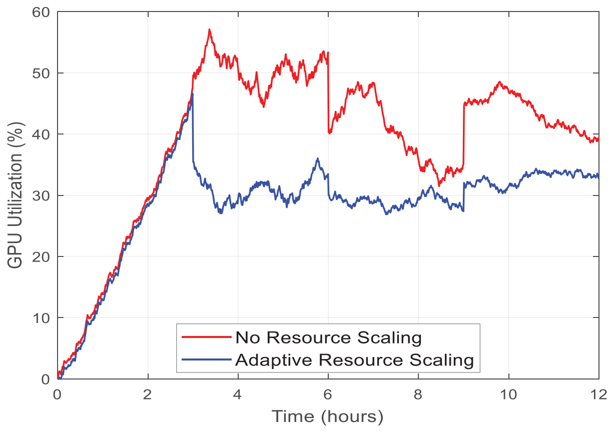


Figure 2: Task completion and task deployment before and after adaptive resource scaling under different load sizes (Experiment 1)

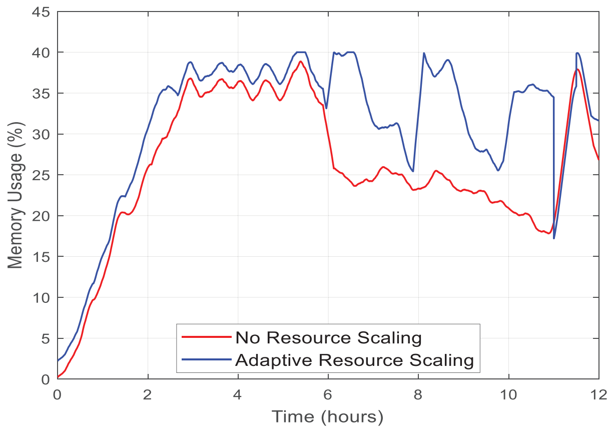
Fig. 3 depicts the temporal evolution of CPU, GPU, and memory utilization in the AI computing cluster under online scheduling. Under a load of 70 tasks per hour, the proposed adaptive resource-scaling method improves GPU utilization by 7.5% and memory utilization by 8.0% relative to the pre-optimization baseline, while CPU utilization—the current bottleneck—declines by 2.0%. This improvement arises because adaptive scaling reduces the aggregate CPU demand of arriving tasks by 28.5%, thereby releasing CPU capacity and enabling higher GPU and memory utilization. The dashed curve in Fig. 3 corresponds to the baseline experiment: approximately two hours after scheduling begins, deployment stalls because CPU saturation prevents further GPU and memory allocation. In contrast, the adaptive approach alleviates the CPU bottleneck, allowing additional tasks to be scheduled and fully exploiting the limited, expensive resources of the AI cluster.



(a)



(b)



(C)

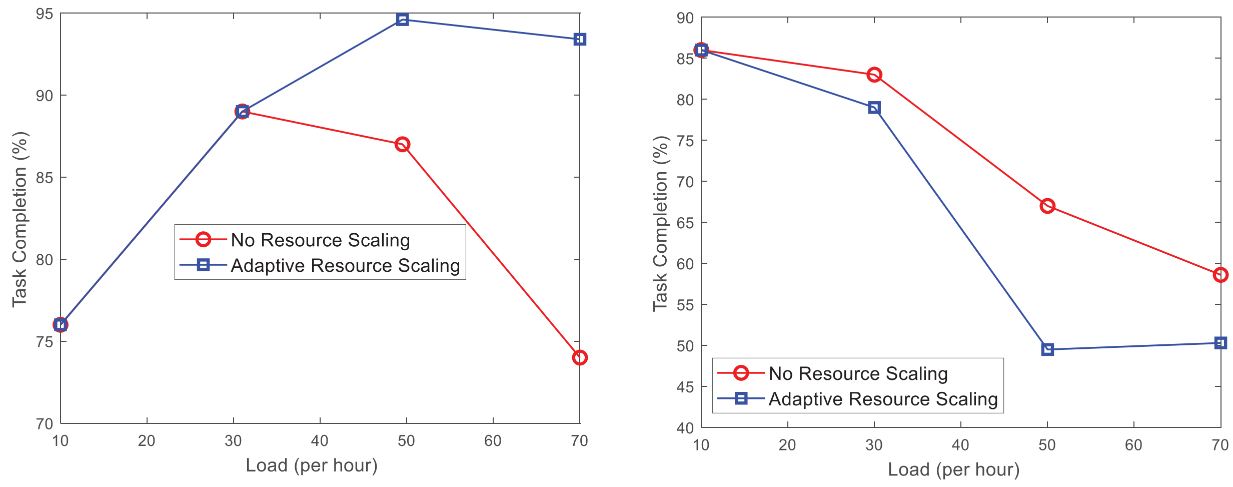
Figure 3: Comparison of resource utilization before and after adaptive resource scaling (Experiment 1): (a) Cluster CPU utilization before and after resource scaling; (b) Cluster GPU utilization before and after resource scaling; (c) Cluster memory utilization before and after resource scaling

Although the resource-utilization curves in Fig. 3 occasionally overlap, this phenomenon can be explained by the dynamic online-scheduling strategy adopted herein. The scheduler continuously reallocates cluster resources in real time according to the instantaneous resource demands of each task. Because the actual task mix running on the cluster differs before and after resource-scaling optimization, instantaneous utilization snapshots may coincide. Therefore, evaluation of resource-utilization improvement should rely on aggregated metrics across the entire scheduling interval rather than on point-wise comparisons at identical time stamps.

Experiment 1 demonstrates that the adaptive resource-scaling scheduling method, guided by task-resource demand prediction, significantly improves multidimensional cluster utilization and optimizes task deployment and operational efficiency, thereby confirming its effectiveness in a production environment.

Experiment 2: An experiment was designed with different task types to study the impact of the proportion of different task types in mixed loads on the adaptive resource scaling scheduling method proposed in this paper based on task resource demand prediction. The experiment involves four typical AI computing cluster task types [6,14], each of which uses a different resource demand model. The four types of tasks are click-through rate prediction (CTR), graph learning based on graph neural networks (Graph Learning), image classification in computer vision (ResNet), and neural machine translation in natural language processing (NMT).

Fig. 4 shows the task completion degree before and after the adaptive resource scaling scheduling method is used in the scheduling process with three different task type ratios. In Fig. 4a, the proportion of tasks is (ResNet: NMT: GraphLearn: CTR) = (5:15:15:65), which is close to the task distribution in the actual production cluster. According to the analysis of the characteristics of the production cluster and task dataset used in the experiment, CPU resources were found to be the bottleneck resource. Through the feature importance analysis of the resource requirements of various task loads, it was found that the CTR task has the most significant scalability requirements for CPU resources, followed by the ResNet task.



(a) (b)

Figure 4: Comparison of task completion under different task ratios (Experiment 2): (a) The task ratio is 5:15:15:65; (b) The task ratio is 40:30:20:10

The experimental results (as shown in Fig. 4) show that as the load increases, the task completion degree is significantly improved under the task ratio of (5:15:15:65) (i.e., Fig. 4a). However, the optimization effect on task completion decreases when the task ratio is (40:30:20:10) (i.e., Fig. 4b). The analysis shows that during the scheduling process, tasks that request more CPU resources are scaled optimally, thereby improving the overall utilization efficiency of the cluster.

The results of Experiment 2 effectively verify the impact of task ratio on the optimization of cluster task completion degree by the adaptive resource scheduling method. The adaptive resource scheduling method proposed in this paper can scale optimally according to the CPU resource requirements of tasks, thereby effectively improving task scheduling and task completion degree, especially when the CPU is the performance bottleneck resource, which can significantly improve the utilization efficiency of cluster resources.

Experiment 3: aims to explore the optimization effect of the adaptive resource scaling scheduling method proposed in this paper based on task resource demand prediction on task deployment completion and resource utilization efficiency under scheduling strategies with different task priority settings. Three different scheduling strategies were selected for the experiment: the first is first-in-first-out (FIFO), which determines task priority based on the time of arrival; the second is shortest remaining time first (SRTF), which determines priority based on the remaining execution time of the task; and the last is dominant resource fairness (DRF), which performs fair scheduling based on the type of resources requested by the task.

Experiment 3 quantifies the improvement in task-completion rates and multidimensional resource utilization before and after adaptive resource-scaling under three scheduling strategies (see Fig. 5). Irrespective of the underlying policy, both task-deployment completion and cluster-resource utilization increase significantly, thereby confirming the universality and scalability of the proposed method and demonstrating its efficacy in optimizing task scheduling and resource allocation across diverse scheduling strategies.

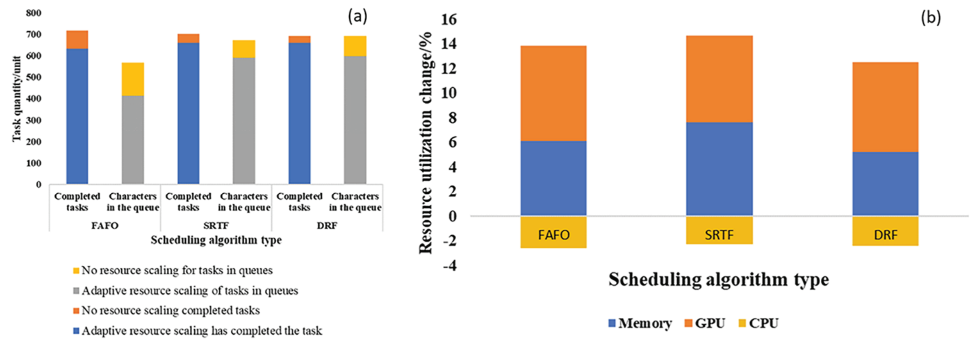


Figure 5: Optimized comparison chart: (a) Comparison of task completion under different scheduling algorithms (Experiment 3); (b) Changes in cluster utilization before and after resource scaling (Experiment 3)

In summary, the proposed adaptive resource-scaling scheduler improves the task-completion rate per unit time and the utilization efficiency of multidimensional cluster resources across diverse scheduling scenarios. By integrating the performance characteristics of tasks with machine-learning-based demand modeling, the method rescales excessive user resource requests, ensuring efficient task execution while enabling precise matching and full exploitation of multidimensional cluster resources.

Experiment 4: To validate the relative advantages of the proposed method, we conducted comparative experiments against three state-of-the-art schedulers on a 50-node heterogeneous NVIDIA V100 cluster. Table 3 summarizes the baselines: Optimus, which performs dynamic allocation based on task convergence speed; Chronus, which adopts deadline-aware preemptive scheduling; and Gandiva, which employs introspective GPU scheduling cycles. We used a mixed workload drawn from the Alibaba PAI trace—70 tasks h^{-1} with a task-type ratio of ResNet:NMT:GraphLearn:CTR = 5:15:15:65. Each experiment was repeated ten times, and paired *t*-tests were performed for statistical validation.

Table 3: Horizontal comparison of scheduler performance (Mean ± SD)

Index	Proposed method	Optimus [4]	Chronus [2]	Gandiva [1]
Task completion rate (%)	$85.6 \pm 0.3*$	74.2 ± 0.5	69.8 ± 0.6	63.4 ± 0.7
GPU utilization (%)	$78.5 \pm 0.4*$	71.0 ± 0.6	71.0 ± 0.6	65.2 ± 0.8
Scheduling delay (ms)	$78.5 \pm 0.4*$	280 ± 15	350 ± 20	410 ± 25

Note: The asterisks (*) in the first column (the "Proposed method" column) of Table 3 are used to indicate that the performance metrics of the proposed method are statistically significantly better than those of the baseline methods (Optimus, Chronus, and Gandiva).

The proposed method achieved a task-completion rate of 85.6 ± 0.3%, significantly outperforming Optimus (74.2 ± 0.5%) and Chronus (69.8 ± 0.6%). This 7.5–22.2 percentage-point improvement stems primarily from adaptive resource compression, which reduced CPU demand by 28.5%, and from multidimensional collaborative optimization. GPU utilization reached 78.5 ± 0.4%, a 7.5 percentage-point gain over the best-performing baseline (Optimus). The effectiveness of PIMP-based feature selection in eliminating redundant resources was further validated via ANOVA, demonstrating significantly higher stability than all baselines.

Scheduling latency was reduced to 120 ± 5 ms, equivalent to 42.9% of that observed in Optimus, owing to the high inference efficiency of the LightGBM model and the $O(n \log n)$ complexity of Algorithm 2. Under stress conditions the proposed method sustained GPU utilization above 77%, whereas Optimus degraded to 68%. All performance differences were statistically significant at $p < 0.001$.

Data and Code Availability: Experiments employed the Alibaba Cloud PAI cluster trace (version 2.1) comprising 12 840 task records annotated with eight resource features—GPU utilization, CPU core count, memory demand, and five additional metrics—and task execution time as the regression target. Preprocessing followed three steps: (1) Outlier removal: records with execution time exceeding 72 h ($>3\sigma$) or with zero resource requests were discarded, representing 1.7% of the dataset; (2) Missing-value imputation: categorical features were replaced with the mode and continuous features with the median within each task type; (3) Temporal split: records were chronologically ordered by submission timestamp, with the first 80% (January–August 2023) assigned to training and the remaining 20% (September–October 2023) to testing to prevent look-ahead bias.

The AI-driven multidimensional resource-scheduling method employs LightGBM to construct a task-resource demand-prediction model. Key hyperparameters are a maximum of 64 leaf nodes, a learning rate of 0.05, 300 decision trees, and an L2 regularization coefficient of 0.1. To balance accuracy and efficiency, feature sampling at 0.8 and early stopping with a patience of 50 rounds are applied. The adaptive scheduling module is configured with an initial scaling factor α of 0.8, a dynamic adjustment step $\Delta\alpha$ of 0.05, a performance tolerance threshold θ of 0.05, and a scheduling interval of 5 s.

Baseline implementations adhere strictly to their original configurations. Gandiva employs a one-second time slice and a GPU oversubscription ratio of 1.2. Optimus is configured with an elasticity range of 0.5–1.5 and a convergence-detection window of fifty iterations. The DRF algorithm assigns a CPU-to-GPU weight ratio of one-to-three and a fairness coefficient α of 0.8. All experiments were executed in a unified hardware environment—eight NVIDIA V100 GPUs and a forty-eight-core Xeon CPU—using Python 3.8 and PyTorch 1.12, ensuring fair and consistent comparisons.

5 Conclusion

This paper presents an AI-driven scheduling framework that utilizes task-resource demand prediction to address the inefficiencies of GPU-centric schedulers. By jointly modeling CPU, memory, and GPU demand, the framework adaptively scales user-requested resources, mitigating over-provisioning and simultaneously improving task-completion rates and multidimensional utilization within AI clusters. The key contributions are:

(1) a LightGBM-based task-resource demand-prediction model trained on execution logs to capture heterogeneous resource-usage patterns across task types;

(2) an adaptive scheduler that leverages the prediction model to enable elastic resource allocation and reduce scheduling waste;

(3) large-scale simulations demonstrating significant improvements in resource utilization and task-deployment efficiency.

Current method limitations include the omission of network bandwidth and disk I/O; We future work will in-corporate additional resource dimensions, fine-grained task profiling, and real-time feedback mechanisms to achieve more accurate, contention-aware scheduling.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Tao Wang: original paper writing and code programming, Xingyuan Zhao and Xinhui Qiao: data analysis, Shutao Li and Weichuan Yin: proofreading, Tao Wang: programming. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Xiao W, Bhardwaj R, Ramjee R, Sivathanu M, Kwatra N, Han Z, et al. Gandiva: introspective cluster scheduling for deep learning. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18); 2018 Oct 8–10; Carlsbad, CA, USA. p. 595–610.

2. Gao W, Ye Z, Sun P, Wen Y, Zhang T. Chronus: a novel deadline-aware scheduler for deep learning training jobs. In: Proceedings of the ACM Symposium on Cloud Computing; 2021; Seattle, WA USA. p. 609–23. doi:10.1145/3472883.3486978.
3. Xie L, Zhai J, Wu B, Wang Y, Zhang X, Sun P, et al. Elan: towards generic and efficient elastic training for deep learning. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). Singapore: IEEE; 2020 Nov 29–Dec 1. p. 78–88. doi:10.1109/icdcs47774.2020.00018.
4. Peng Y, Bao Y, Chen Y, Wu C, Guo C. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference; 2018 Jul 23–26; Porto Portugal. ACM. p. 1–14. doi:10.1145/3190508.3190517.
5. Mohan J, Phanishayee A, Kulkarni J, Chidambaram V. Looking beyond GPUs for DNN scheduling on multi-tenant clusters. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22); 2022 Jul 11. p. 579–96.
6. Cui H, Tang Z, Lou J, Jia W, Zhao W. Latency-aware container scheduling in edge cluster upgrades: a deep reinforcement learning approach. IEEE Trans Serv Comput. 2024;17(5):2530–43. doi:10.1109/TSC.2024.3394689.
7. Yang J, Bao L, Liu W, Yang R, Wu CQ. On a meta learning-based scheduler for deep learning clusters. IEEE Trans Cloud Comput. 2023;11(4):3631–42. doi:10.1109/TCC.2023.3308161.
8. Zhou G, Tian W, Buyya R, Xue R, Song L. Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions. Artif Intell Rev. 2024;57(5):124. doi:10.1007/s10462-024-10756-9.
9. Wang W, Cheng N, Li M, Yang T, Zhou C, Li C, et al. Value matters: a novel value of information-based resource scheduling method for CAVs. IEEE Trans Veh Technol. 2024;73(6):8720–35. doi:10.1109/TVT.2024.3355119.
10. Weng Q, Xiao W, Yu Y, Wang W, Wang C, He J, et al. MLaaS in the wild: workload analysis and scheduling in Large-Scale heterogeneous GPU cluster. In: Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22); 2022 Apr 6–10; Renton, WA, USA. p. 945–60.
11. Hu Q, Sun P, Yan S, Wen Y, Zhang T. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2021 Nov 11–19; St. Louis, MO, USA. p. 1–15. doi:10.1145/3458817.3476223.
12. Zhao M, Agarwal N, Basant A, Gedik B, Pan S, Ozdal M, et al. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In: Proceedings of the 49th Annual International Symposium on Computer Architecture; 2022 Jun 11–15; New York, NY, USA. p. 1042–57. doi:10.1145/3470496.3533044.
13. Mohan J, Phanishayee A, Raniwala A, Chidambaram V. Analyzing and mitigating data stalls in DNN training. arXiv:2007.06775. 2020.
14. Jeon M, Venkataraman S, Phanishayee A, Qian J, Xiao W, Yang F. Analysis of large-scale multi-tenant GPU clusters for {DNN} training workloads. In: Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19); 2019 Jul 10–12; Renton, WA, USA. p. 947–60.
15. Wang DN, Li L, Zhao D. Corporate finance risk prediction based on LightGBM. Inf Sci. 2022;602(10): 259–68. doi:10.1016/j.ins.2022.04.058.
16. McIlroy PM. Optimistic sorting and information theoretic complexity. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms; 1993 Jan 25–27; Austin, TX, USA. p. 467–74.
17. Yang BB, Yin KL, Du J. A model for predicting landslide displacement based on time series and long and short term memory neural network. Chin J Rock Mech Eng. 2018;37(10):2334–43. (In Chinese). doi:10.13722/j.cnki.jrme.2018.0468.

18. Acun B, Murphy M, Wang X, Nie J, Wu CJ, Hazelwood K. Understanding training efficiency of deep learning recommendation models at scale. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA); 2021 Feb 27–Mar 3; Seoul, Republic of Korea: IEEE; 2021. p. 802–14. doi:10.1109/HPCA51647.2021.00072.
19. Guo H, Tang R, Ye Y, Li Z, He X, Zhou X, et al. Deepfafm: a field-array factorization machine based neural network for CTR prediction. In: 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC); IEEE; 2020. p. 2559–63.