Learning with Adaptive Resource Allocation

Jing Wang [1 2] Miao Yu [1 2] Peng Zhao [1 2] Zhi-Hua Zhou [1 2]

Abstract

The study of machine learning under limited resources has gathered increasing attention, considering improving the learning efficiency and effectiveness with budgeted resources. However, previous efforts mainly focus on *single* learning task, and a common resource-limited scenario is less explored: to handle *multiple* time-constrained learning tasks concurrently with budgeted computational resources. In this paper, we point out that this is a very challenging task because it demands the learner to be concerned about not only the progress of the learning tasks but also the coordinative allocation of computational resources. We present the *Learning with Adaptive Resource Allocation* (LARA) approach, which comprises an efficient online estimator for learning progress prediction, an adaptive search method for computational resource allocation, and a balancing strategy for alleviating prediction-allocation compounding errors. Empirical studies validate the effectiveness of our proposed approach.

1. Introduction

The impact of limited computational resources on machine learning (ML) is of increasing attention due to its significant influence on the efficiency and effectiveness of the model training process in various real-world scenarios. This is particularly noticeable in situations involving large models, such as large language models (Achiam et al., 2023), which always demand the use of thousands of GPUs for several months to achieve effective training, as well as the situation of tiny training devices such as certain IoT devices or microcontrollers (Lin et al., 2023), which are incapable of supporting even regular sized models. All these situations highlight the diverse challenges posed by computational resource limitations in different contexts of learning tasks.

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China [2]School of Artificial Intelligence, Nanjing University, China. Correspondence to: Zhi-Hua Zhou <zhouzh@lamda.nju.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

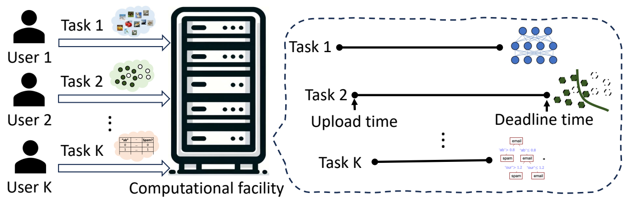


Figure 1. Multiple users upload diverse learning tasks to a shared computational facility, all intended to be completed before user-defined deadlines. Due to computational resource limitation, we aim to complete as many tasks as possible within these constraints.

In order to improve training efficiency and effectiveness with limited computational resources, various significant strides have been made in prior research, which include: improving resource usage efficiency (Dean et al., 2012; Li et al., 2014), reducing model size (Frankle & Carbin, 2019; Mirzadeh et al., 2020), designing memory-efficient optimization algorithms (Anil et al., 2019; Hu et al., 2022), etc. Notably, previous efforts mainly focus on the computational resource issue of *single* learning task. However, a common resource limitation scenario has received less attention: *multiple* time-constrained learning tasks competing for budgeted computational resources, as illustrated in Figure 1.

Consider a practical example of a quantitative finance startup, where there can be multiple analysts or traders who aim to train many different learning models to help forecast the market or manage risks, and these model training tasks require a large amount of computational resources. Given their limited budgets, these startups often face challenges in acquiring advanced computational resources necessary for concurrently handling everyone's computational needs. This limitation typically results in a first-come, first-served basis in resource allocation. Such an approach can be highly inefficient, particularly when earlier uploaded tasks use resources poorly, which not only leads to computational power wastage but also causes delays in executing other critical tasks. In the dynamic and fast-paced domain of finance, such delays can incur significant costs, such as missed opportunities or delayed responses to emerging risks.

A similar challenge arises when a user purchases or leases cloud computational services. As the user always needs to execute multiple tasks concurrently, each corresponding to different model structures or hyperparameter tunings, these computational resources are often incapable of supporting parallel training for these learning tasks. Consequently, the user has to adopt a sequential approach for model training.

To this end, Zhou (2023) advocated that the concept of *time-sharing* should be introduced to machine learning. On one hand, this will enable us to take into account the computational resource constraints in real-world issues mentioned above and recognize that the performance of machine learning depends not only on the amount of data received but also on the computational resources available to process it. On the other hand, current intelligent supercomputing facilities generally operate in an *exclusive* manner, allocating a fixed amount of resources to each user, which can either be insufficient or excessive, leading to inefficiencies. Such a working style can be improved if *time-sharing* concept is taken into account. For this purpose, Zhou (2023) formally proposed the ***Computational Resource Efficient Learning*** (CoRE-Learning) paradigm where the key is to consider the influence of *resource scheduling* during learning process, aiming to complete as many tasks on time as possible within resource limits. In this paper, we present the first practical CoRE-Learning approach. We identify that the key challenge of CoRE-Learning lies in the coupling of learning progress and resource allocation: the learning progress of each task guides resource allocation, while resource allocation affects the learning progress of each task.

To tackle the challenge of CoRE-Learning problem, we introduce the *Learning with Adaptive Resource Allocation* (LARA) approach, which periodically predicts each task's resource requirement and allocates resources accordingly. LARA consists of three components: (i) *Resource prediction*: this component adopts an efficient online estimator to fit the learning progress curve from historical data and predicts the resources needed for each task to meet success criteria; (ii) *Resource allocation*: based on each task's resource prediction, this component models the resource allocation problem and employs an adaptive searching method guaranteed to find the optimal solution; (iii) *Prediction-allocation balancing*: this component employs an exploration strategy to reduce compounding error caused by the prediction inaccuracies and their cascading effects. Experimental results validate the effectiveness of our LARA approach, demonstrating its capacity to significantly improve resource utilization efficiency in the context of CoRE-Learning.

Notation. In this paper, we let $\{a_k\}_{k=1}^K \triangleq \{a_1, ..., a_K\}$; $[T] \triangleq \{1, 2, ..., T\}$; $[a, b] \triangleq \{a+1, ..., b\}$; $\mathbb{I}(\cdot)$ is indicator function; $|A|$ denotes the cardinality of set A.

2. Problem Formulation and Key Challenge

In this section, we provide the problem formulation of CoRE-Learning (Zhou, 2023) and discuss its key challenge.

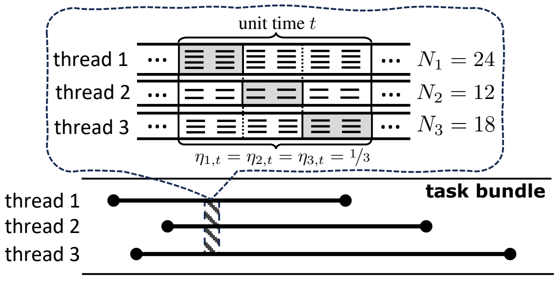


Figure 2. An example of uniform allocation within unit time t.

2.1. Problem Formulation

According to Zhou (2023), we consider a task bundle $\{\mathcal{T}_k\}_{k=1}^K$ during the time period $[T]$. The k-th thread $\mathcal{T}_k$ represents a machine learning training task uploaded by the user, which is defined by: the task beginning time b_k and the user-defined deadline time d_k such that $1 \le b_k < d_k \le T$, signifying the time constraint of k-th thread; the data budget N_k represents the upper limit of the data amount that k-th thread can learn within any unit time $t \in [T]$, reflecting the computational resource capability.

At unit time t, the learner observes a set of active threads $A_t = \{k \mid b_k \le t \le d_k, k \in [K]\}$. The budget computational resources cannot simultaneously support all threads $k \in A_t$ to meet their maximum data capabilities N_k, so the learner needs to select the data throughput $\{\eta_{k,t}\}_{k \in A_t}$ such that k-th thread can learn $\eta_{k,t} N_k$ data within unit time t. Figure 2 provides an example of allocation for three threads. The computational budgets bring constraints: $\forall k \in A_t, \eta_{k,t} \ge 0$; $\sum_{k \in A_t} \eta_{k,t} \le 1$. Then, the learner observes the new training loss value $\ell_k(s)$ of k-th thread, where s represents the amount of accumulated usage data. The success criteria for k-th thread is that its training loss reaches a user-defined threshold within the time constraint $[b_k - 1, d_k]$: $\ell_k(\sum_{t=b_k}^{d_k} \eta_{k,t} N_k) \le \epsilon_k$. The learner's goal is to maximize the number of tasks that meet their success criterion within the time constraints.

2.2. Key Challenge

We can model the CoRE-learning problem for task bundle $\{\mathcal{T}_k\}_{k=1}^K$ as solving the following optimization problem

$$\begin{aligned} \max_{\{\eta_{k,t}\}_{k \in [K], t \in [T]}} \; & \sum_{k \in [K]} \mathbb{I}\left[\ell_k\left(\sum_{t=b_k}^{d_k} \eta_{k,t} N_k\right) \le \epsilon_k\right] \\ \text{s.t.} \quad & \forall t \in [T], \sum_{k \in A_t} \eta_{k,t} \le 1, \\ & \forall k \in [K], t \in [T], \eta_{k,t} \ge 0. \end{aligned} \tag{2.1}$$

The key challenge of Problem (2.1) arises from the *coupling of learning progress and resource allocation*. Specifically, the loss function $\ell_k(\cdot)$ is unknown, which requires estimating ℓ_k based on early data and solving the allocation problem with the extrapolated function $\widehat{\ell}_k(\cdot)$. The predictive error in $\widehat{\ell}_k(\cdot)$ directly impacts the effectiveness of resource allocation. Additionally, even with a known $\ell_k(\cdot)$, solving Problem (2.1) remains difficult. The non-convex and non-continuous nature of the objective function and the extensive number of parameters make classic methods like dynamic programming computationally intensive and time-consuming, potentially affecting the learning process. One may also consider using reinforcement learning techniques to deal with the learning-allocation coupling issue, similar to the exploration-exploitation dilemma. However, these techniques are not directly applicable to CoRE-Learning due to the need for efficient, high-frequency re-estimation and reallocation to minimize the impact on learning tasks.

3. Our Approach

In this section, we present *Learning with Adaptive Resource Allocation* (LARA), an innovative approach developed to manage budgeted computational resources in the context of CoRE-Learning. LARA comprises three components: resource prediction, resource allocation, and prediction-allocation balancing. At each unit time, LARA predicts the data amount required for each thread to succeed and then allocates resources based on these predictions. The balancing component is vital in minimizing compounded errors from prediction and allocation during training. The following sections provide detailed discussions: resource prediction in Section 3.1, resource allocation in Section 3.2, and prediction-allocation balancing in Section 3.3.

3.1. Resource Prediction

In this part, we introduce our resource prediction method. Briefly, this process involves analyzing the previously observed training loss values and the corresponding cumulative training data volume. By applying regression to fit the training loss curve, LARA extrapolates the fitted loss curve to predict when it is likely to fall below the success threshold.

Many existing studies indicate that the training loss curves of various models generally follow the form of a negative power function (Hestness et al., 2017; Kaplan et al., 2020). Leveraging this insight, our approach utilizes a negative power function to fit the training loss curve. Specifically, for the k-th thread $\mathcal{T}_k$, we model this relationship between the loss and the training data amount as

$$\ell_k(s) = a_k s^{-b_k} + \xi_{k,s}, \tag{3.1}$$

where $\ell_k(s)$ denotes the training loss value of the cumulative data amount s for the k-th thread, a_k and b_k are unknown positive constants, and $\xi_{k,s}$ represents noise. To facilitate a more efficient regression, we transform it into a linear form by taking $\ln(\cdot)$ on both sides of Eq. (3.1):

$$r_{k,s} = X_s^\top \theta_k + \xi'_{k,s}, \tag{3.2}$$

where $r_{k,s} \triangleq \ln \ell_k(s)$, $X_s \triangleq [\ln s; 1]$, $\theta_k \triangleq [-b_k; \ln a_k]$ and $\xi'_{k,s}$ is the surrogate noise.

It is critical to note that LARA needs to use early samples from the training loss function to estimate the true trajectory of the loss function curve. Then, we extrapolate this curve to predict the future data amount required for reducing the loss function below a specific success threshold. This extrapolation process requires paying more attention on minimizing future errors in the function estimation. To enhance accuracy, recent data is given priority, as it more accurately reflects upcoming trends. Consequently, we adopt a weighted regularized least squares method (Guo et al., 1993), which gives increased weight to recent data points while reducing the influence of older ones. For the k-th thread $\mathcal{T}_k$, after observing n data pairs $\{s_i, \ell_k(s_i)\}_{i=1}^n$, the estimator $\widehat{\theta}_{k,n}$ is obtained by solving the following problem:

$$\min_\theta \sum_{i=1}^n \gamma^{n-i}(X_{s_i}^\top \theta - r_{k,s_i})^2, \tag{3.3}$$

where $\gamma \in (0,1)$ is the discounted factor, measuring the degree of discounting to early data. Problem (3.3) admits a closed-form solution

$$\widehat{\theta}_{k,n} = V_n^{-1}\left(\sum_{i=1}^n \gamma^{n-i} r_{k,s_i} X_{s_i}\right), \tag{3.4}$$

where $V_n \triangleq \sum_{i=1}^n \gamma^{n-i} X_{s_i} X_{s_i}^\top$ is the covariance matrix. Moreover, we can reformulate the solution (3.4) into an *online* update format (Haykin, 2002, Chapter 10.3)

$$\begin{aligned} \widehat{\theta}_{k,n} &= \widehat{\theta}_{k,n-1} + V_n^{-1} X_{s_n}(r_{k,s_n} - X_{s_n}^\top \widehat{\theta}_{k,n-1}) \\ V_n^{-1} &= \frac{1}{\gamma}\left(V_{n-1}^{-1} - \frac{V_{n-1}^{-1} X_{s_n} X_{s_n}^\top V_{n-1}^{-1}}{\gamma + X_{s_n}^\top V_{n-1}^{-1} X_{s_n}}\right). \end{aligned} \tag{3.5}$$

This new format does not require matrix inversion and is *one-pass*, in the sense that it processes each data only once, hence eliminating the need to store historical data and significantly enhancing the efficiency of the estimation.

Upon estimating the unknown parameters by Eq. (3.5) as $\widehat{\theta}_{k,n} = [-\widehat{b}_{k,n}, \ln \widehat{a}_{k,n}]$, we derive the estimated negative power function $\widehat{\ell}_k(s) = \widehat{a}_{k,n} s^{-\widehat{b}_{k,n}}$. By applying the success criterion $\widehat{\ell}_k(s) \le \epsilon_k$, we can compute the data volume needed for the k-th thread to achieve its success criterion as $s = \exp\left({}^{1}\!/\!{}_{\widehat{b}_{k,n}} \ln {}^{\widehat{a}_{k,n}}\!/\!{}_{\epsilon_k}\right)$. By subtracting the already used data amount s_n, we obtain the remaining data requirement

$$\widehat{S}_{k,n} = \left\lceil \exp\left(\frac{1}{\widehat{b}_{k,n}} \ln \frac{\widehat{a}_{k,n}}{\epsilon_k}\right) - s_n \right\rceil. \tag{3.6}$$

Remark 1. Recent studies on curve extrapolation and scaling laws for various models have yielded notable advancements. For instance, Domhan et al. (2015) employ an ensemble of diverse structural models, leveraging their combined strengths to approximate the loss curve more accurately. Alabdulmohsin et al. (2022) suggest using an extrapolation loss instead of the traditional interpolation loss for better prediction accuracy. Additionally, Shen & Meinshausen (2023) introduce engression as an advanced extrapolation technique. These approaches have demonstrated improved performance in curve extrapolation tasks. However, these advanced techniques typically require *substantial time and computational resources* for managing multiple models or optimizing the extrapolation or engression losses. Such extensive resource requirements are infeasible in the context of CoRE-Learning, where an efficient estimation algorithm is essential for frequent updates.

3.2. Resource Allocation

At time τ, let A_τ denote the current active thread set and $K_\tau \triangleq |A_\tau|$. We reorder and re-label the threads in A_τ such that their deadlines satisfying $d_0 \le d_1 \le d_2 \le \ldots \le d_{K_\tau}$ where $d_0 \triangleq \tau - 1$. Let S_k denotes the data amount still required for k-th thread to succeed, which can be estimated by estimator (3.6), and the success criteria becomes $\sum_{t=d_0+1}^{d_k} \eta_{k,t} N_k \ge S_k$. The learner's goal at time τ is to maximize the number of successful threads within the active thread set A_τ, which can be formulated as

$$\begin{aligned}
&\max_{\{\eta_{k,t}\}_{k\in A_\tau, t\in[d_0, d_{K_\tau}]}} \sum_{k\in A_\tau} \mathbb{I}\left(\sum_{t=d_0+1}^{d_k} \eta_{k,t} N_k \ge S_k\right) \\
&\text{s.t.} \quad \forall t \in [d_0, d_{K_\tau}], \sum_{k\in A_\tau} \eta_{k,t} \le 1, \\
&\qquad \forall k \in A_\tau, t \in [d_0, d_{K_\tau}], \eta_{k,t} \ge 0.
\end{aligned} \tag{3.7}$$

The indicator function introduces non-convexity to Problem (3.7). A typical solution might be convex relaxation followed by gradient descent. Here instead, we identify that the nature of non-continuous allows for both dimensionality reduction and discretization of the solution space, while guaranteeing that the optimal solution is encompassed within this space. This advantage allows us to develop efficient search methods to effectively locate optimal solutions.

Parameter Reduction. The following lemma presents an optimal solution representation that reduces the number of parameters from $\sum_{k\in A_\tau}(d_k - d_0)$ to merely K_τ.

Lemma 1. *At time τ, there exists a set of time-invariant parameters $\{\widetilde{\eta}_k\}_{k\in A_\tau}$, where $\forall k \in A_\tau, 0 \le \widetilde{\eta}_k \le 1$, such that the optimal solution $\{\eta^*_{k,t}\}_{k\in A_\tau, t\in[d_0, d_{K_\tau}]}$ to the allocation Problem (3.7) can be characterized by $\{\widetilde{\eta}_k\}_{k\in A_\tau}$. Specifically, for each time interval $[d_{j-1}, d_j], j \in [K_\tau]$,*

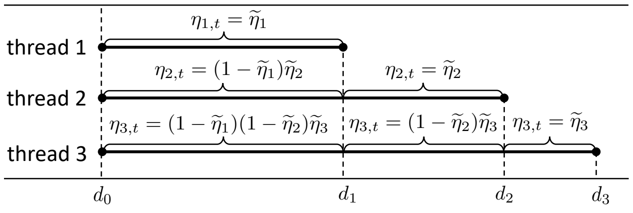

thread 1
thread 2
thread 3
$\eta_{1,t} = \widetilde{\eta}_1$
$\eta_{2,t} = (1-\widetilde{\eta}_1)\widetilde{\eta}_2$
$\eta_{2,t} = \widetilde{\eta}_2$
$\eta_{3,t} = (1-\widetilde{\eta}_1)(1-\widetilde{\eta}_2)\widetilde{\eta}_3$
$\eta_{3,t} = (1-\widetilde{\eta}_2)\widetilde{\eta}_3$
$\eta_{3,t} = \widetilde{\eta}_3$
d_0
d_1
d_2
d_3

Figure 3. An example of parameter reduction for three threads.

$$\eta_{k,t} = \begin{cases} \widetilde{\eta}_j, & k = j, \\ \prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)\widetilde{\eta}_k, & j < k \le K_\tau. \end{cases} \tag{3.8}$$

Figure 3 provides an example to illustrate Eq. (3.8). Lemma 1 shows that we only needs to determine the set $\{\widetilde{\eta}_k\}_{k=1}^{K_\tau}$ to find the optimal solution of Problem (3.7).

Adaptive Binary Tree Search. After parameter reduction, the cumulative data amount $\sum_{t=d_0+1}^{d_k} \eta_{k,t} N_k$ allocated to thread k can be expressed as $D_k(\widetilde{\eta}_k) \cdot N_k$, where we define $D_1(\widetilde{\eta}_1) \triangleq \Delta_1 \widetilde{\eta}_1$, and $\forall k \ge 2$,

$$D_k(\widetilde{\eta}_k) \triangleq \left(\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)\Delta_j + \Delta_k\right)\widetilde{\eta}_k, \tag{3.9}$$

where $\Delta_j \triangleq d_j - d_{j-1}$ is the time interval width. A critical insight is that for each thread, resource allocation has only two outcomes: success (i.e. $D_k(\widetilde{\eta}_k) \cdot N_k \ge S_k$) or failure. Therefore, $\widetilde{\eta}_k$ can take only two possible values: a value that ensures the thread's success or 0. Starting with $\widetilde{\eta}_1$, the decision is either to allocate no resources ($\widetilde{\eta}_1 = 0$) or just enough to meet its requirement ($\widetilde{\eta}_1 = S_1/N_1\Delta_1$). Once $\widetilde{\eta}_1$ is set, a similar decision is made for $\widetilde{\eta}_2$. This process is then iteratively applied to each subsequent thread, such that when $\widetilde{\eta}_1, \ldots, \widetilde{\eta}_{k-1}$ are determined, we can set $\widetilde{\eta}_k$ accordingly. Based on this allocation method, we can construct a complete binary tree of K_τ layers, such that at each node in the k-th layer, we make the following allocation

$$\widetilde{\eta}_k = \begin{cases} 0, & \text{if } \alpha_k > 1, \\ \alpha_k, & \text{otherwise}, \end{cases} \tag{3.10}$$

where $\alpha_k \triangleq S_k/N_k(\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)\Delta_j + \Delta_k)$. The binary tree with 2^{K_τ} leaf nodes represents all possible allocation methods for $\{\widetilde{\eta}_k\}_{k=1}^{K_\tau}$, and our goal is to find the optimal path within this tree. However, a full tree traversal would have an impractical time complexity of $\mathcal{O}(2^{K_\tau})$. To this end, more efficient search methods are required. We find that prioritizing threads based on earlier deadline times d_k or a lower ratio of required resources to work S_k/N_k can have a better effect. With this strategy in mind, we have developed an *adaptive search algorithm* for the binary tree. This algorithm allocates resources preferentially to tasks with earlier deadlines and smaller resource requirements.

Algorithm 1 Adaptive Binary Tree Search

```
Input: active thread set $A$, current time $\tau$, required resource $S_k$, candidate thread set $C = \emptyset$
 1: Set $d_0 = \tau - 1, \forall k \in A$, set $\widetilde{\eta}_k = 0$
 2: Sort and re-label threads in $A$ such that $d_0 \leq ... \leq d_{|A|}$
 3: for $k = 1, 2, ..., |A|$ do
 4:     Add $k$ into $C$
 5:     if $\frac{S_k/N_k}{d_k - d_0 - \sum_{i \in C} S_i/N_i + S_k/N_k} > 1$ then
 6:         Choose $i = \arg\max_{i \in C} \frac{S_i}{N_i}$, remove $i$ from $C$
 7:     end if
 8: end for
 9: for $k \in C$ do
10:     Set $\widetilde{\eta}_k = \frac{S_k/N_k}{d_k - d_0 - \sum_{i \in C, i \leq k} S_i/N_i + S_k/N_k}$
11: end for
12: return $\{\widetilde{\eta}_k\}_{k \in A}$
```

Our algorithm maintains a set C to keep track of candidate threads where each $k \in C$ has a non-zero $\widetilde{\eta}_k$, then the resource allocation rate α_k in the allocation condition (3.10) can be recalculated as $\alpha_k = \frac{S_k/N_k}{d_k - d_0 - \sum_{i \in C, i \leq k} S_i/N_i + S_k/N_k}$. Initially, C is empty. Starting from $k = 1$ and progressing to K_τ, the algorithm first adds thread k to C and if resource allocation to the k-th thread breaches the constraints ($\alpha_k > 1$), it then identifies and removes the most resource-demanding thread from C, determined by $i = \arg\max_{i \in C} {}^{S_i}/_{N_i}$. Once all K_τ threads have been considered, the algorithm assigns $\widetilde{\eta}_k = 0$ for $k \notin C$ and $\widetilde{\eta}_k = \alpha_k$ for all $k \in C$. The overall algorithm is summarized in Algorithm 1.

Algorithm 1 significantly reduces the time complexity of binary tree search from $\mathcal{O}(2^{K_\tau})$ to $\mathcal{O}(K_\tau)$. Additionally, the subsequent theorem demonstrates that Algorithm 1 effectively attains the optimal solution for the Problem (3.7).

Theorem 1 (Optimality). *For the allocation Problem (3.7), the results $\{\widetilde{\eta}_k\}_{k=1}^{K_\tau}$ generated by Algorithm 1 can be used to construct the optimal resource allocation $\{\eta^*_{k,t}\}_{k \in A_\tau, t \in [d_0, d_{K_\tau}]}$, as outlined in Eq. (3.8).*

3.3. Prediction-Allocation Balancing

In previous sections, we introduced a method to predict the resource requirements for each task to complete each task based on their learning progress. This prediction is vital for resource allocation, which in turn influences the learning progress and future predictions. Initially, our dataset of training loss observations for each thread is limited, leading to significant uncertainty in predictions. Early allocation based on these imprecise predictions could introduce compounding errors, potentially misdirecting resources from feasible tasks to less feasible ones. We note that it is important to address this issue early in the process of regular prediction and allocation. Otherwise, such compounded errors might accumulate over time, reducing the algorithm's overall effectiveness. This challenge requires a *prediction-allocation balancing*, where resources are allocated widely enough to ensure accurate predictions, while also being strategically focused on tasks with the greatest likelihood of success. To achieve this, we design our balancing strategy based on the idea of Explore-Then-Exploit (ETE) strategy from bandits theory (Lattimore & Szepesvári, 2020).

Algorithm 2 Learning with Adaptive Resource Allocation

```
Input: Task bundle $\{\mathcal{T}_k\}_{k=1}^K$, exploration period $H_k$
 1: Set exploration set $E_t = \emptyset$, active thread set $A_t = \emptyset$
 2: for $t = 1, 2, ..., T$ do
 3:     Update active thread set $A_t$ and exploration set $E_t$
 4:     Calculate resource estimation $\widehat{S}_k$ by Eq. (3.6)
 5:     if $E_t \neq \emptyset$ then
 6:         for all $k \in E_t$ do
 7:             Allocate $\eta_{k,t} = 1/|E_t|$ to thread $k$
 8:         end for
 9:     else
10:         Run Algo. 1 with $A_t$, $t$, $\widehat{S}_k$ and return $\{\widetilde{\eta}_k\}_{k \in A_t}$
11:         Set $\eta_{1,t} = \widetilde{\eta}_1$ and $\forall k \geq 2, \eta_{k,t} = \prod_{i=1}^{k-1}(1 - \widetilde{\eta}_i)\widetilde{\eta}_k$
12:         Rescale to make sure $\sum_{k \in A_t} \eta_k = 1$
13:     end if
14:     Learn $\eta_{k,t} N_k$ data for thread $k \in A_t$ and receive new loss value $\ell_k$ for $k \in A_t$
15:     Update regression $\widehat{\theta}_k$ by Eq. (3.5)
16: end for
```

Explore-then-Exploit. At each time τ, we maintain an exploration set $E_\tau = \{k \mid \sum_{t=1}^{\tau} \eta_{k,t} N_k \leq H_k, k \in A_t\}$, where H_k is a predefined exploration threshold. This threshold indicates that if a thread's cumulative allocated data amount is below H_k, it requires more data for accurate evaluation. If $E_\tau \neq \emptyset$, implying there are still exists under-explored active threads, we then allocate the resources of time unit τ uniformly among all threads in E_τ, such that $\forall k \in E_k, \eta_{k,\tau} = {}^1/_{|E_\tau|}$. This allocation is vital for gathering sufficient data for precise training loss estimation, thus significantly reducing errors in resource prediction. It is important to clarify that this approach differs from pre-training a model for loss curve fitting, as the exploration period is a critical part of each thread's time allocation. Setting the exploration threshold H_k appropriately is crucial: too long a period can use up excessive time, affecting subsequent resource allocation, while too short a period may cause significant prediction errors, exacerbating compounded errors. The complete LARA algorithm with the Explore-then-Exploit strategy, is detailed in Algorithm 2.

Remark 2. The ETE strategy relies on a exploration threshold, which still depends on an empirical adjustment. This suggests the need for more adaptive algorithms, such as the Upper Confidence Bound (UCB). We could adopt UCB here by adding a bonus to resource prediction, for example, at time τ, set corrected resource prediction as $\widetilde{S}_{k,n} = \widehat{S}_{k,n} - \beta\sqrt{\ln(\tau-b_k)/\sum_{t=b_k}^{\tau}\eta_{k,t}}$, where $\beta > 0$ is a UCB control constant, $\ln(\tau - b_k)$ represents the potential data budget, and $\sum_{t=b_k}^{\tau}\eta_{k,t}$ is the actual data received. Less exploration leads to a larger bonus and a smaller $\widetilde{S}_{k,n}$, encouraging more resource allocation to the thread. However, we have conducted preliminary experiments and found that UCB-type strategy may not outperform the simple ETE. This UCB-type strategy may often allocate fewer resources than required. This is because UCB adopts an optimistic way to encourage exploration: $\widetilde{S}_{k,n} = \widehat{S}_{k,n} -$ optimism. So the allocated resource $\widetilde{S}_{k,n}$ may be fewer than the actual resource required by the task. In the future, we might consider integrating other adaptive strategies such as Thompson sampling to improve the exploration phase of our approach.

4. Experiments

In this section, we evaluate the empirical performance of our proposed LARA approach. We begin with an experiment involving a pure task bundle, where five different models are trained concurrently on the same dataset to demonstrate our approach's efficiency and effectiveness. Next, we conduct an experiment with a mixed task bundle, where different models are trained concurrently on different datasets. We then perform a scalability experiment to assess how our algorithm performs as the number of threads increases. Finally, we conduct experiments to evaluate the performance of specific components of our algorithm. In each experiment, each thread represents a model training task with a specific beginning time b_k, deadline d_k, and predetermined success threshold for the loss ϵ_k. Each time unit $t \in [T]$ is set to one second. At the start of each time unit, the learner observes the training loss of the previous time unit and allocates the resources for the current time unit as needed.

4.1. Pure Task Bundle Experiment

In this part, we consider a pure task bundle, where each task trains a different model but uses the same dataset.

Setting. We focus on image classification of CIFAR-10 dataset (Krizhevsky et al., 2009). We train five distinct models concurrently, each featuring unique hyperparameters and structures. The training loss for all models is calculated as the average cross-entropy loss over the past training batches. All five models begin training concurrently, and the beginning time for each thread is set to 0-th second. The specific settings for each of the five models, including model type, data budget (N_k), deadline time (d_k), and success threshold (ϵ_k), are outlined in Table 1. N_k is the maximum data processable per second calculated by the average time to process unit data during the exploration, and unit data is defined by a batch with 64 data points. For the deadline d_k and success criteria ϵ_k, we initially designed a series of tasks that could be completed in short priority order. We then adjusted certain tasks, creating scenarios with both short and challenging tasks (small d_k, large ϵ_k) as well as long and simple ones. For the subsequent experiment over mixed task bundle, we employed the same generation strategy.

Model	ViT	LSTM	CNN	ResNet18	ResNet34
N_k	163	220	375	97	55
d_k	500	570	590	610	630
ϵ_k	1.35	1.18	0.15	0.35	0.45

Table 1. The setting of pure task bundle.

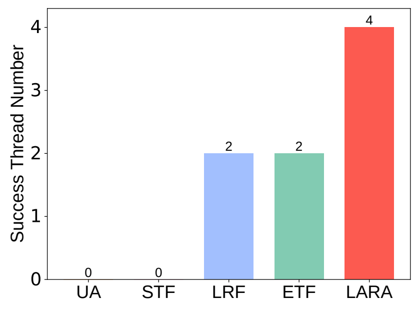
We evaluate the performance of our proposed LARA approach against several classic resource allocation strategies: (a) Uniform Allocation (UA), which distributes computational resources equally across all active threads regardless of their specific needs or deadlines; (b) Shortest Thread First (STF), giving priority to threads with the nearest deadlines and allocating all resources to them first; (c) Least Resources First (LRF), allocating resources primarily to threads requiring the smallest amount of training data S_k, thereby aiming to quickly complete tasks with lower demands; and (d) Easiest Thread First (ETF), focusing on the ratio of required data to time constraints S_k/d_k-b_k, thus prioritizing threads that are relatively easier to complete within their available time. Both Least Resources First and Easiest Thread First use the same resource prediction method as ours, as specified in (3.6). For exploration we set all exploration threshold $H_k = 8000$, this takes a total of about 335 seconds to explore all threads.

Allocation Result. Figure 4(a) illustrates the results of successfully completed thread number using various resource allocation strategies. Both UA and STF methods allocate excessive resources to the challenging thread 1, leading to none of the threads being completed. LRF and ETF strategies, which employ our provided resource prediction method, manage to avoid dedicating resources to some of the more challenging threads. However, their inherently greedy allocation methods result in the completion of only two threads. In contrast, the LARA algorithm, by strategically giving up on the difficult-to-complete thread 1 early, efficiently allocates resources and successfully completes the remaining four threads.

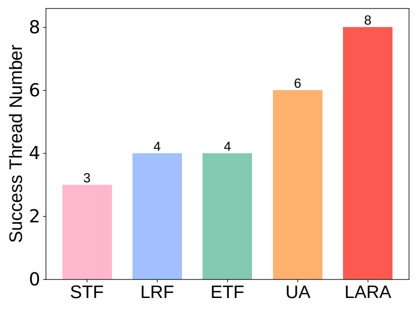
4.2. Mixed Task Bundle Experiment

In this part, we consider a mixed task bundle, where each task trains a different model on a different dataset.

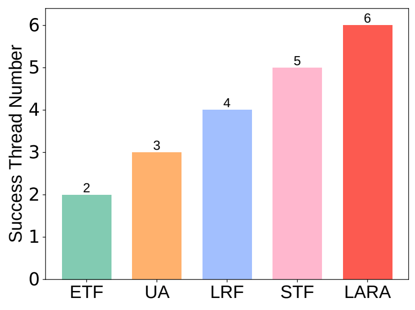
Setting. We consider 10 threads across four different types of learning tasks: computer vision (CV) with CIFAR-10, natural language processing (NLP) with IMDB (Maas et al.,



(a) Pure task bundle



(b) Mixed task bundle with ϵ_k^1



(c) Mixed task bundle with ϵ_k^2

Figure 4. Effectiveness experiments of resource allocation for pure and mixed task bundles.

2011), reinforcement learning (RL) with Montezuma's Revenge[1], and audio processing with Yesno[2], each using different models. All ten models begin training concurrently, with the starting time for each thread set to the 0-th second. The specific settings, including model type, data budget (N_k), deadline time (d_k), and success threshold (ϵ_k), are outlined in Table 2. We provide two sets of success thresholds, ϵ_k^1 and ϵ_k^2, to characterize task bundles of varying difficulties.

Num	Task	Model	d_k	ϵ_k^1	ϵ_k^2	N_k
1	CV	ViT	530	1.1	0.99	176
2	CV	ViT	540	1.35	1.215	176
3	RL	DAgger+CNN	545	0.0011	0.0011	69
4	Audio	Transformer	585	0.06	0.09	293
5	NLP	Attention+LSTM	625	7×10^{-4}	6.3×10^{-4}	139
6	CV	ResNet18	655	0.55	0.46	107
7	CV	ResNet18	685	0.55	0.48	107
8	NLP	Attention+LSTM	690	9×10^{-4}	4.5×10^{-4}	139
9	Audio	Transformer	700	0.08	0.108	293
10	RL	DAgger+CNN	710	0.0012	0.0006	69

Table 2. The setting of mixed task bundle.

Allocation Result. Figures 4(b) and 4(c) illustrate the comparative results of successfully completed threads using various resource allocation strategies. The task bundle with ϵ_k^1 contains difficult short tasks (threads 1 and 2) and easier longer tasks. Figure 4(b) shows that this scenario is more suited for UA, which fails only in the short threads but completes the six easier long threads. STF allocates excessive resources to challenging threads 1 and 2, leading to only two tasks in the task bundle being completed. LRF and ETF, employing our resource prediction method, manage to avoid dedicating resources to some of the more challenging threads. However, their inherently greedy allocation methods result in the completion of only four threads. In contrast, the LARA algorithm, by strategically giving up on the difficult-to-complete threads 1 and 2 early, efficiently allocates resources and successfully completes the remaining eight threads. For the task bundle with ϵ_k^2, threads 4 and 9 are made easier while the rest become harder, which is more suitable for STF. Figure 4(c) shows that, under this setup, UA finishes only three tasks. STF finishes five tasks by completing the two easier tasks. LRF and ETF continue to perform poorly due to their greedy allocation strategy. However, our LARA algorithm still manages to complete six tasks. These scenarios show that our LARA strategy performs well in different environments, demonstrating its robustness and effectiveness.

[1]Details can be found at Montezuma's Revenge.

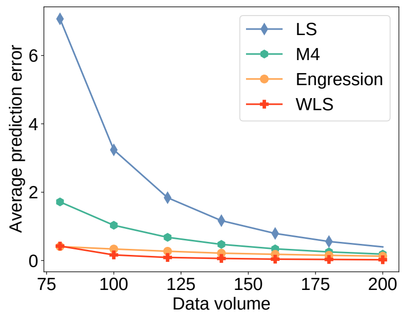
[2]Information about the Yesno dataset is available at Yesno.

4.3. Scalability Experiment

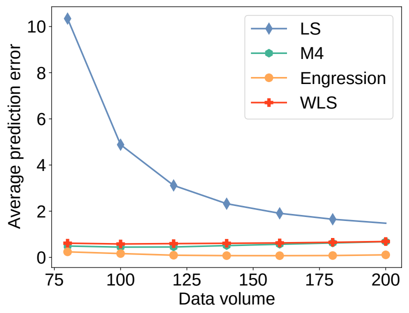
To more comprehensively validate our experiment, we aimed to investigate how increasing the number of training tasks influences the effectiveness of our approach.

Setting. In this part, we focus on image classification of MNIST dataset (Deng, 2012). To generate task bundles, we start by randomly selecting a model from those listed in Table 1 and randomly setting its success threshold ϵ within a certain range. We run each model to completion with the full budgeted computational resources, recording the time taken as its deadline time. This deadline is then used as the beginning time of the next thread. After all threads generated, we let all beginning time be 0-second, ensuring that a series of learning tasks is created in a way that allows them to be completely solvable using the Shortest Thread First algorithm. To add complexity, we modify $1/3$ of these tasks by reducing their success thresholds to $1/10$ of the original values, thereby increasing the difficulty of the entire task bundle. In this experiment, we create task bundles with 10, 25, 50, 75, and 100 threads to analyze how the number of threads affects the performance of the scheduling algorithm.

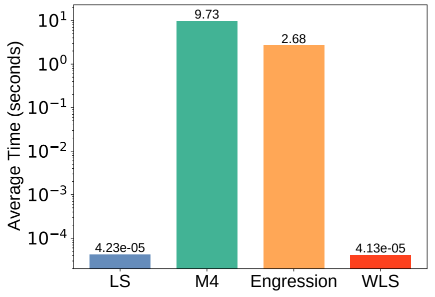
Result. Figure 6(a) demonstrates the changes in the number of successful threads for different resource allocation strategies as the number of threads increases. It is evident that LARA consistently outperforms other strategies with an increasing thread count. Uniform Allocation and Shortest Thread First, lacking thorough evaluation of threads, show declining success rates as the number of threads grows. In contrast, Least Resources First and Easiest Thread First, utilizing our resource prediction method, perform better



(a) Prediction error over pure task bundle



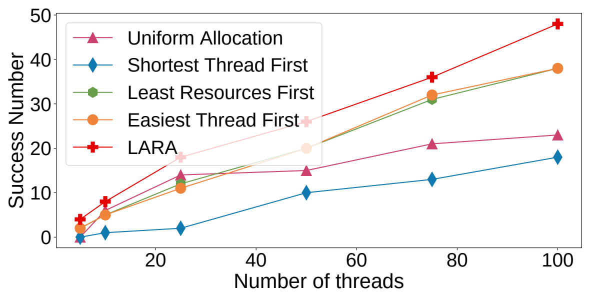
(b) Prediction error over mixed task bundle



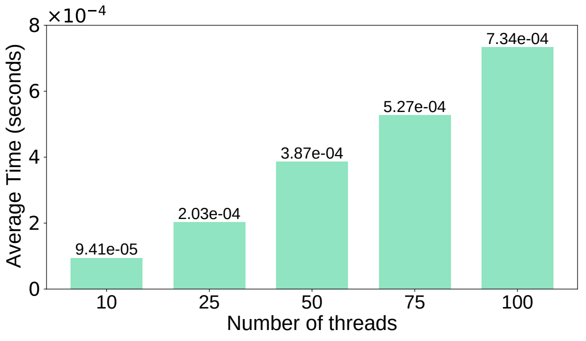
(c) Average time of single round prediction

Figure 5. Effectiveness and efficiency experiments of resource prediction.

than the former two strategies. This highlights the effectiveness of our resource prediction. However, their reliance on a greedy allocation strategy limits their effectiveness compared to LARA. Under the same resource prediction conditions, LARA continues to outperform them as the number of threads increases, demonstrating the superior performance of our allocation algorithm. Figure 6(b) shows the average time cost for a single round of resource prediction and allocation with our LARA algorithm as the number of threads increases. It is observed that the time taken for each update grows linearly with the number of threads. Notably, even with 100 threads, a single update period requires only 0.0007 seconds. Since LARA performs a periodical resource prediction and allocation every second, this time cost has a negligible impact on the model training process. Moreover, all model training tasks are executed on the GPU, while prediction and allocation tasks are handled on the CPU, further minimizing LARA's impact on model training.



(a) Scalability experiment of effectiveness



(b) Scalability experiment of efficiency

Figure 6. Scalability experiments.

4.4. Component Performance Evaluation

In this section, we evaluate the performance of different resource prediction methods on a pure task bundle in Table 1 and on a mixed task bundle in Table 2.

Setting. We compare the prediction performance of our proposed Weighted Least Squares (WLS) method with several extrapolation methods mentioned in Remark 1: (a) Least Squares (LS); (b) the $\mathcal{M}_4$ estimator (Alabdulmohsin et al., 2022); and (c) Engression (Shen & Meinshausen, 2023). We evaluate these four extrapolation methods based on two measures: (i) the average prediction error, calculated for each task using the formula $|\widehat{S} - S|/S$, where $\widehat{S}$ is the predicted resource and S is the true resource, then averaging these errors across the entire task bundle; and (ii) the average time cost of a single round of prediction. For the WLS, we set the discounted factor γ to 0.9. Both the $\mathcal{M}_4$ and Engression methods are applied with their default settings. To perform the extrapolations, we use varying amounts of data points (specifically, 60, 80, up to 200 data points) for each task.

Prediction Result. Figure 5(a) illustrates how the average prediction error changes as more data points are used for the pure task bundle. Surprisingly, the results reveal that the WLS method results in a lower prediction error compared to methods specifically designed for extrapolation. The LS method, which is originally for interpolation, shows the worst performance in this scenario. Figure 5(b) explores performance over the mixed task bundle. Here, WLS performs slightly worse than the two extrapolation-specific strategies but significantly better than LS. Among these methods, Engression outperforms the others for extrapolation. Additionally, Figures 5(c) demonstrate that WLS incurs significantly lower time costs compared to both the $\mathcal{M}_4$ estimator and Engression. This suggests that WLS offers a good balance between accuracy and efficiency, achieving relatively precise outcomes efficiently. We also conducted parameter sensitivity tests on some components of our algorithm. For more experimental details, please refer to Appendix B.

5. Related Topics and Discussion

In this section, we will discuss some topics related to combining machine learning with resource scheduling.

Distributed Machine Learning. Distributed Machine Learning (DML) (Dean et al., 2012; Li et al., 2014) aims to improve the training efficiency of a single large-scale model through distributed computing strategies, especially in scenarios with sufficient computational resources. During the learning process, DML promotes parallel processing by dividing data and model training tasks, and achieves load balancing through reasonable resource allocation methods to accelerate the training process. Although DML also considers the problem of combining machine learning and resource allocation, it diverges significantly from our study. In DML, multiple training tasks are usually coordinated, and the efficiency of each training task is ensured through resource allocation to maximize the efficiency and effect of the entire model training. In contrast, our focus is on training multiple independent learning tasks concurrently with budgeted computing resources. Here, each learning task competes for these budgeted resources, which requires careful evaluation of each learning task and making informed task-wise trade-offs in resource allocation.

Tiny/Edge Machine Learning. Tiny/Edge ML (Lin et al., 2023) focuses on processing learning tasks on devices with extremely limited resources. A line of studies (Wang et al., 2020a;b; Zhou et al., 2021) explored parallel training of multiple learning tasks on power-constrained edge devices, aiming to ensure the effectiveness of each learning task through power resource allocation. Although these works also study the scheduling of multiple learning tasks under resource limitations, they have yet to consider the key challenge caused by the coupling of learning and allocation, as raised in our paper. In their approach, each task's learning progress is known in advance, escaping the uncertainty inherent in the learning process and reducing the issue to a purely computational problem. Furthermore, this also exhibits many setup differences, including the absence of deadlines and varying objectives for scheduling, etc. In contrast, in our study, we cannot obtain any prior before the start of each task. This requires us to predict the learning progress of each task during the resource allocation process. The uncertainty caused by the prediction will significantly affect the allocation decisions, requiring us to balance the accuracy of prediction and the effectiveness of allocation.

Related Studies in Operation System. Recently, a line of operating system research focuses on designing cluster schedulers to improve the efficiency and effectiveness of parallel training of multiple learning tasks. Efforts devoted to maximizing the efficiency of CPU or GPU clusters by Zhang et al. (2017) and Peng et al. (2018) also involve integrating learning task management with resource allocation. However, our research diverges fundamentally in both its objectives and challenges. Unlike these studies, which often rely on pre-training estimations of learning progress, our approach requires the real-time estimation for learning progress of each task during the training phase. Our objective is not just to minimize time or maximize performance; instead, we are faced with the problem of making strategic decisions about which tasks to prioritize. Sometimes, this means making the difficult choice to sacrifice the completion of certain tasks to ensure the successful execution of others. This trade-off between the completion of individual tasks and the overall success of the learning objectives is a unique aspect of our research, distinguishing it fundamentally from existing studies on combining learning and scheduling.

6. Conclusion and Future Work

In this paper, we consider the Computational Resource Efficient Learning (CoRE-Learning) problem which aims to manage multiple time-constrained learning tasks with budgeted computational resources. We identify that the key challenge of this problem lies in the coupling of learning progress and resource allocation. To address this issue, we propose the LARA approach which periodically predicts the learning progress of each learning task, and allocates resources accordingly. LARA consists of three essential components: an efficient online estimator to predict the resource requirements of each thread, an adaptive searching method for resource allocation guaranteed to find the optimal solution and an exploration strategy for prediction-allocation balancing. Through a series of experiments, we validate the effectiveness of our approach in handling multiple learning tasks with limited computational resources.

Currently, our LARA approach is in a preliminary stage, with several directions open for future research in this field. Methodologically, we use WLS for resource prediction, which is essentially an interpolation strategy for extrapolating loss curves. Therefore, future work could explore efficient and extrapolation-specific strategies. For prediction-allocation balancing, we employ the ETE approach, which depends on predefined exploration thresholds. Future research could investigate more adaptive exploration methods, such as ideas based on Thompson sampling. Theoretically, while the optimality has been proved for our resource allocation method, it remains open on how to provide generalization guarantees for the overall process. Future work could introduce curve extrapolation or scaling law theories to provide the theoretical basis and establish the learnability guarantee in the context of CoRE-Learning framework.

Acknowledgements

This research was supported by National Science and Technology Major Project (2022ZD0114800) and Collaborative Innovation Center of Novel Software Technology and Industrialization.

Impact Statement

This paper presents a preliminary study on multiple learning tasks training concurrently with limited computational resources. High-performance computational resources are often expensive and inaccessible, creating resource bottlenecks. Our research aims to enhance resource utilization efficiency and fairness while reducing energy waste. This line of research benefits budget-constrained users and supports better intelligent supercomputing resource management.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. GPT-4 technical report. *ArXiv preprint*, arxiv:2303.08774, 2023.

Alabdulmohsin, I. M., Neyshabur, B., and Zhai, X. Revisiting neural scaling laws in language and vision. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pp. 22300–22312, 2022.

Anil, R., Gupta, V., Koren, T., and Singer, Y. Memory efficient adaptive optimization. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pp. 9746–9755, 2019.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A. W., Tucker, P. A., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25 (NIPS)*, pp. 1232–1240, 2012.

Deng, L. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3460–3468, 2015.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Guo, L., Ljung, L., and Priouret, P. Performance analysis of the forgetting factor RLS algorithm. *International Journal of Adaptive Control and Signal Processing*, 7(6): 525–537, 1993.

Haykin, S. S. *Adaptive Filter Theory*. Pearson Education India, 2002.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the 29th (IEEE) Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Hestness, J., Narang, S., Ardalani, N., Diamos, G. F., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *ArXiv preprint*, arXiv:1712.00409, 2017.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *ArXiv preprint*, arxiv:2001.08361, 2020.

Krizhevsky, A., Hinton, G., et al. *Learning Multiple Layers of Features from Tiny Images*. Toronto, ON, Canada, 2009.

Lattimore, T. and Szepesvári, C. *Bandit Algorithms*. Cambridge University Press, 2020.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 583–598, 2014.

Lin, J., Zhu, L., Chen, W.-M., Wang, W.-C., and Han, S. Tiny machine learning: Progress and futures. *IEEE Circuits and Systems Magazine*, 23(3):8–34, 2023.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the*

Association for Computational Linguistics: Human Language Technologies (ACL-HLT), pp. 142–150, 2011.

Mirzadeh, S., Farajtabar, M., Li, A., Levine, N., Matsukawa, A., and Ghasemzadeh, H. Improved knowledge distillation via teacher assistant. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 5191–5198, 2020.

Peng, Y., Bao, Y., Chen, Y., Wu, C., and Guo, C. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the 13th European Conference on Computer Systems (EuroSys)*, pp. 1–14, 2018.

Shen, X. and Meinshausen, N. Engression: Extrapolation for nonlinear regression? *ArXiv preprint*, arxiv:2307.00835, 2023.

Wang, S., Wang, R., Hao, Q., Wu, Y., and Poor, H. V. Learning centric power allocation for edge intelligence. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020a.

Wang, S., Wu, Y., Xia, M., Wang, R., and Poor, H. V. Machine intelligence at the edge with learning centric power allocation. *IEEE Transactions on Wireless Communications*, 19(11):7293–7308, 2020b.

Zhang, H., Stafman, L., Or, A., and Freedman, M. J. Slaq: Quality-driven scheduling for distributed machine learning. In *Proceedings of the 8th Symposium on Cloud Computing (SoCC)*, pp. 390–404, 2017.

Zhou, L., Hong, Y., Wang, S., Han, R., Li, D., Wang, R., and Hao, Q. Learning centric wireless resource allocation for edge computing: Algorithm and experiment. *IEEE Transactions on Vehicular Technology*, 70(1):1035–1040, 2021.

Zhou, Z.-H. Learnability with time-sharing computational resource concerns. *ArXiv preprint*, arXiv:2305.02217, 2023.

A. Proofs

In this section, we provide the analysis for our resource allocation strategy. In Appendix A.1, we analyze our parameter reduction method and prove that optimal solutions to Problem 3.7 can still be found after reduction, which is captured in Lemma 1. In Appendix A.2, we provide a proof for the optimality of our adaptive binary tree search method for Problem 3.7.

A.1. Proof of Lemma 1

Proof. For Problem 3.7, we let set C, comprising K_C threads, represent those successfully completed under the optimal allocation solution denoted by $\{\eta^*_{k,t}\}_{k\in C, t\in[d_0,d_{K_C}]}$. The threads in C are sorted and labeled such that the deadline time satisfies that $d_0 < d_1 < d_2 < ... < d_{K_C}$, where d_0 represents the current unit time. Then the optimal allocation should satisfy the following three optimal conditions,

$$\begin{aligned}
&\text{(i)} \quad \forall k \in C, D_k \triangleq \sum_{t=d_0+1}^{d_k} \eta^*_{k,t} \geq \frac{S_k}{N_k};\\
&\text{(ii)} \quad \forall t \in [d_0, d_{K_C}], \sum_{k\in C} \eta^*_{k,t} \leq 1;\\
&\text{(iii)} \quad \forall k \in C, \forall t \in [d_0, d_{K_C}], \eta^*_{k,t} \geq 0.
\end{aligned}$$

At the same time, it can also be found that the allocation method that satisfies these three conditions is the optimal allocation method. We can first modify the optimal allocation $\{\eta^*_{k,t}\}_{k\in C, t\in[d_0,d_{K_C}]}$ such that in any interval $[d_{i-1}, d_i]$ satisfying $i \in C, d_{i-1} \neq d_i$, $\eta^*_{k,t} = \eta^*_{k,[d_{i-1},d_i]}$ is time-invariant. For the interval $[d_{i-1}, d_i]$, we let

$$\eta^*_{k,[d_{i-1},d_i]} = \frac{\sum_{t=d_{i-1}+1}^{d_i} \eta^*_{k,t}}{d_i - d_{i-1}}, \tag{A.1}$$

we can find that the allocation (A.1) still satisfies the optimal condition:

$$\begin{aligned}
&\text{(i)} \quad \forall k \in C, \quad \sum_{i=1}^{k}(d_i - d_{i-1})\eta^*_{k,[d_{i-1},d_i]} = \sum_{i=1}^{k}\sum_{t=d_{i-1}+1}^{d_i} \eta^*_{k,t} = \sum_{t=d_0+1}^{d_k} \eta^*_{k,t} = D_k \geq \frac{S_k}{N_k},\\
&\text{(ii)} \quad \forall i \in C, \quad \sum_{k\in C} \eta^*_{k,[d_{i-1},d_i]} = \frac{\sum_{k\in C}\sum_{t=d_{i-1}+1}^{d_i} \eta^*_{k,t}}{d_i - d_{i-1}} = \frac{\sum_{t=d_{i-1}+1}^{d_i}\sum_{k\in C} \eta^*_{k,t}}{d_i - d_{i-1}} \leq \frac{d_i - d_{i-1}}{d_i - d_{i-1}} = 1,\\
&\text{(iii)} \quad \forall k \in C, \quad \forall i \in C, \quad \eta^*_{k,[d_{i-1},d_i]} = \frac{\sum_{t=d_{i-1}+1}^{d_i} \eta^*_{k,t}}{d_i - d_{i-1}} \geq 0.
\end{aligned}$$

Based on condition (ii), we know that $\sum_{j=1}^{k} \eta^*_{j,[d_{i-1},d_i]} \leq \sum_{j\in C} \eta^*_{j,[d_{i-1},d_i]} \leq 1$, then we have

$$\forall i \in C, \forall k \in C, \quad \eta^*_{k,[d_{i-1},d_i]} \leq 1 - \sum_{j=1}^{k-1} \eta^*_{j,[d_{i-1},d_i]},$$

then $\forall k \in C$, we have

$$\begin{aligned}
\sum_{i=1}^{k} \eta^*_{k,[d_{i-1},d_i]}(d_i - d_{i-1}) &= D_k\\
\sum_{i=1}^{k}\left(1 - \sum_{j=1}^{k-1} \eta^*_{j,[d_{i-1},d_i]}\right)(d_i - d_{i-1}) &\geq D_k\\
\sum_{i=1}^{k}(d_i - d_{i-1}) - \sum_{i=1}^{k}\sum_{j=1}^{k-1} \eta^*_{j,[d_{i-1},d_i]}(d_i - d_{i-1}) &\geq D_k
\end{aligned}$$

$$d_k - d_0 - \sum_{j=1}^{k-1}\sum_{i=1}^{k} \eta^*_{j,[d_{i-1},d_i]}(d_i - d_{i-1}) \geq D_k$$

$$d_k - d_0 - \sum_{j=1}^{k-1} D_j \geq D_k.$$

So we have $\forall k \in C$,

$$\frac{D_k}{d_k - d_0 - \sum_{j=1}^{k-1} D_j} \leq 1. \tag{A.2}$$

For the 1-st thread, we define $\widetilde{\eta}_{1,[d_0,d_1]} \triangleq \eta^*_{1,[d_0,d_1]}$, then we have

$$(d_1 - d_0)\widetilde{\eta}_{k,[d_0,d_1]} = D_1,$$

then for the interval $[d_0, d_1]$, we can define

$$\begin{aligned}
\widetilde{\eta}_{2,[d_0,d_1]} &\triangleq \frac{\eta^*_{2,[d_0,d_1]}}{(1-\widetilde{\eta}_{1,[d_0,d_1]})} \\
\widetilde{\eta}_{3,[d_0,d_1]} &\triangleq \frac{\eta^*_{3,[d_0,d_1]}}{(1-\widetilde{\eta}_{1,[d_0,d_1]})(1-\widetilde{\eta}_{2,[d_0,d_1]})} \\
&\dots \\
\widetilde{\eta}_{k,[d_0,d_1]} &\triangleq \frac{\eta^*_{k,[d_0,d_1]}}{\prod_{i=1}^{k-1}(1-\widetilde{\eta}_{i,[d_0,d_1]})} \\
&\dots
\end{aligned}$$

Similarly, for any interval $[d_{i-1}, d_i], i \in C$, we can define

$$\begin{aligned}
\widetilde{\eta}_{i,[d_{i-1},d_i]} &\triangleq \eta^*_{i,[d_{i-1},d_i]} \\
&\dots \\
\widetilde{\eta}_{k,[d_{i-1},d_i]} &\triangleq \frac{\eta^*_{k,[d_{i-1},d_i]}}{\prod_{j=i}^{k-1}(1-\widetilde{\eta}_{j,[d_{i-1},d_i]})} \\
&\dots
\end{aligned}$$

Based on the success criteria and definition of $\widetilde{\eta}_{i,[d_{i-1},d_i]}$, we have

$$\begin{aligned}
\widetilde{\eta}_{1,[d_0,d_1]}(d_1-d_0) &= D_1 \\
(1-\widetilde{\eta}_{1,[d_0,d_1]})\widetilde{\eta}_{2,[d_0,d_1]}(d_1-d_0) + \widetilde{\eta}_{2,[d_1,d_2]}(d_2-d_1) &= D_2 \\
&\dots \\
\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_{i,[d_{i-1},d_i]})\widetilde{\eta}_{k,[d_{j-1},d_j]}(d_j-d_{j-1}) + \widetilde{\eta}_{k,[d_{k-1},d_k]}(d_k-d_{k-1}) &= D_k \\
&\dots
\end{aligned}$$

Next, we try to prove that for any k, $\widetilde{\eta}_{k,[d_0,d_1]}, \dots, \widetilde{\eta}_{k,[d_{k-1},d_k]}$ can be replace by a fixed $\widetilde{\eta}_k$, which still satisfies these three optimal conditions. We let

$$\widetilde{\eta}_k = \frac{D_k}{\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)(d_j-d_{j-1}) + (d_k-d_{k-1})}, \tag{A.3}$$

And for any interval $[d_{i-1}, d_i], i \in C$, the allocation becomes

$$\eta^*_{i,[d_{i-1},d_i]} = \widetilde{\eta}_i$$
$$\ldots$$
$$\eta^*_{k,[d_{i-1},d_i]} = \prod_{j=i}^{k-1}(1-\widetilde{\eta}_j)\widetilde{\eta}_k$$
$$\ldots$$

then we satisfies condition (i) as follows,

$$\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)\widetilde{\eta}_k(d_j-d_{j-1}) + \widetilde{\eta}_k(d_k-d_{k-1}) = D_k.$$

Moving $\widetilde{\eta}_k$ on the left side of the equation to the right, we have

$$\begin{aligned}
\frac{D_k}{\widetilde{\eta}_k} =& \sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)(d_j-d_{j-1}) + (d_k-d_{k-1}) \\
=& \sum_{j=1}^{k-2}\prod_{i=j}^{k-2}(1-\widetilde{\eta}_i)(d_j-d_{j-1})(1-\widetilde{\eta}_{k-1}) + (1-\widetilde{\eta}_{k-1})(d_{k-1}-d_{k-2}) + (d_k-d_{k-1}) \\
=& (1-\widetilde{\eta}_{k-1})\left(\sum_{j=1}^{k-2}\prod_{i=j}^{k-2}(1-\widetilde{\eta}_i)(d_j-d_{j-1}) + (d_{k-1}-d_{k-2})\right) + (d_k-d_{k-1}) \\
=& \frac{D_{k-1}}{\widetilde{\eta}_{k-1}} - D_{k-1} + (d_k-d_{k-1}) \\
=& \sum_{i=1}^{k}(d_i-d_{i-1}) - \sum_{i=1}^{k-1}D_i \\
=& d_k - d_0 - \sum_{i=1}^{k-1}D_i.
\end{aligned}$$

Then we still have

$$\widetilde{\eta}_k = \frac{D_k}{d_k - d_0 - \sum_{i=1}^{k-1}D_i},$$

based on property (A.2), we have $\forall k \in C, \widetilde{\eta}_k \leq 1$. Then we have

$$\begin{aligned}
\forall i \in C, \sum_{k\in C}\eta^*_{k,[d_{i-1},d_i]} &= \widetilde{\eta}_i + \sum_{k=i+1}^{K_C}\prod_{j=i}^{k-1}(1-\widetilde{\eta}_j)\widetilde{\eta}_k \\
&\leq \widetilde{\eta}_i + \sum_{k=i+1}^{K_C-1}\prod_{j=i}^{k-1}(1-\widetilde{\eta}_j)\widetilde{\eta}_k + \prod_{j=i}^{K_C-1}(1-\widetilde{\eta}_j) \\
&\leq \widetilde{\eta}_i + \sum_{k=i+1}^{K_C-2}\prod_{j=i}^{k-1}(1-\widetilde{\eta}_j)\widetilde{\eta}_k + \prod_{j=i}^{K_C-2}(1-\widetilde{\eta}_j) \\
&\leq \widetilde{\eta}_i + 1 - \widetilde{\eta}_i \\
&= 1,
\end{aligned}$$

which satisfies the condition (ii). Building upon the conditions that $\forall k \in C, \widetilde{\eta}_k \leq 1$, the sequence of deadlines $d_0 < \ldots < d_{K_C}$, and the allocation method as described by equation (A.3), we can obtain that $\forall k \in C, \widetilde{\eta}_k \geq 0$. Based on this observation, $\eta^*_{k,[d_{i-1},d_i]}$ further satisfies the condition (iii), thus we complete the proof. $\square$

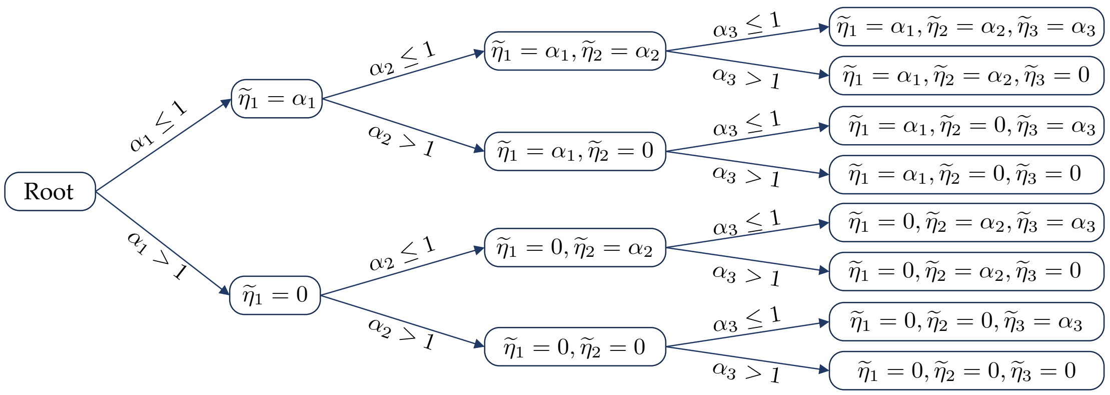

Root
$\alpha_1 \le 1$
$\alpha_1 > 1$
$\widetilde{\eta}_1 = \alpha_1$
$\widetilde{\eta}_1 = 0$
$\alpha_2 \le 1$
$\alpha_2 > 1$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = \alpha_2$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = 0$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = \alpha_2$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = 0$
$\alpha_3 \le 1$
$\alpha_3 > 1$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = \alpha_2, \widetilde{\eta}_3 = \alpha_3$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = \alpha_2, \widetilde{\eta}_3 = 0$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = 0, \widetilde{\eta}_3 = \alpha_3$
$\widetilde{\eta}_1 = \alpha_1, \widetilde{\eta}_2 = 0, \widetilde{\eta}_3 = 0$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = \alpha_2, \widetilde{\eta}_3 = \alpha_3$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = \alpha_2, \widetilde{\eta}_3 = 0$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = 0, \widetilde{\eta}_3 = \alpha_3$
$\widetilde{\eta}_1 = 0, \widetilde{\eta}_2 = 0, \widetilde{\eta}_3 = 0$

Figure 7. Example of Binary Search Tree with 3-layers.

A.2. Proof of Theorem 1

Proof. For the Problem 3.7, let set C, comprising K_C threads, denote those successfully completed under the optimal allocation solution denoted by $\{\eta^*_{k,t}\}_{k\in C, t\in[d_0, d_{K_C}]}$. According to Lemma 1, we know that there exists a set of $\{\widetilde{\eta}^*_k\}_{k\in A_\tau}$, where $\forall k \in A_\tau, 0 \le \widetilde{\eta}^*_k \le 1$, can formulate a solution with Eq. (3.8) also achieve set C, such that $\forall k \in C, D_k(\widetilde{\eta}^*_k)\cdot N_k \ge S_k$, where D_k is defined in Eq. (3.9), representing accumulated data throughput. Since $D_k(\widetilde{\eta}_k)$ is a linear monotonically increasing function with respect to $\widetilde{\eta}_k$, which means there exists set of $\{\widetilde{\eta}_k\}_{k\in A_\tau}$, where $\forall k \in C, 0 \le \widetilde{\eta}_k \le \widetilde{\eta}^*_k$ satisfying $D_k(\widetilde{\eta}_k)\cdot N_k = S_k$ and $\forall k \notin C, \widetilde{\eta}_k = 0$. This means $\{\widetilde{\eta}_k\}_{k\in A_\tau}$ will also achieve the optimal solution, since the indicator function in Problem 3.7 only need $D_k(\widetilde{\eta}_k)\cdot N_k = S_k$ to success. Now for $\widetilde{\eta}_k$ for each thread $k \in A_\tau$ only has two outcomes: success ($D_k(\widetilde{\eta}_k)\cdot N_k \ge S_k$) or failure ($\widetilde{\eta}_k = 0$), which can be formulated as

$$\widetilde{\eta}_k = \begin{cases} 0, & \text{if } \frac{S_k}{N_k\left(\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)(d_j - d_{j-1}) + 1\right)} > 1, \\ \frac{S_k}{N_k\left(\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)(d_j - d_{j-1}) + 1\right)}, & \text{otherwise.} \end{cases} \tag{A.4}$$

We can observe that, $\widetilde{\eta}_k$ is determined by the previous $\widetilde{\eta}_1, \ldots, \widetilde{\eta}_{k-1}$. Based on this allocation method, we can construct a complete binary tree of K_τ layers, Figure 7 provides a example for illstration, where $\alpha_k \triangleq S_k / N_k\left(\sum_{j=1}^{k-1}\prod_{i=j}^{k-1}(1-\widetilde{\eta}_i)(d_j - d_{j-1}) + 1\right)$. Now the optimal solution of Problem 3.7 is the path contains maximum number of $\widetilde{\eta}_k = \alpha_k, \alpha_k \le 1$ within this tree.

So now, we just need to prove that, the output of Algorithm 1, is the optimal path of this tree such that contains maximum number of $\widetilde{\eta}_k = \alpha_k, \alpha_k \le 1$. We present an exchange argument to prove the optimality of Algorithm 1 for finding the optimal path. Let A and B denote two sequences of threads, ordered by their duration, representing the choices made by our algorithm and a hypothetical alternative, respectively. In these sequences, threads are either chosen to be allocated just enough resources, or not included at all. The success count for A is $\#succ_A$, and we assume that B has a higher success count $\#succ_B > \#succ_A$.

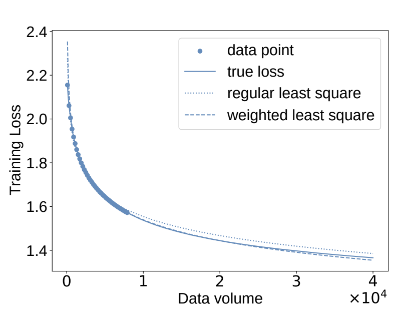
Defining A and B. Sequence A represents the threads chosen by our algorithm, while sequence B represents an alternate set of threads with a presumed higher success rate. Both A and B are ordered based on thread deadline time, with shorter-duration threads prioritized.

Identifying the First Divergence. By comparing sequences A and B, we identify the first point of divergence at thread k. Two scenarios are considered: (i) A includes thread k but B does not; (ii) B includes thread k but A does not.

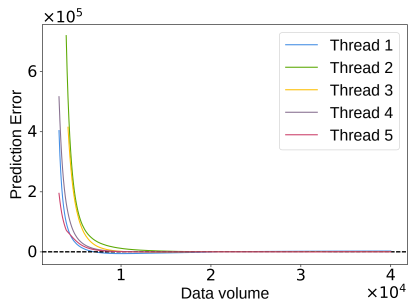
(i) If A chooses thread k and B does not, we examine the subsequent threads in sequence. If B selects a later thread $k+i$ that A omits, we can replace thread $k+i$ in B with k because A's choice reflects a more resource-efficient selection. We continue this process until we reach a thread $k+j$ where up to $k+j$, $\#succ_B$ equals $\#succ_A$. Since A makes more resource-efficient choices, we can align B's choices with A without reducing $\#succ_B$.

(ii) If B includes thread k, it implies that at thread k, A also considers k but ultimately replaces it with a later thread i where $D_i < D_k$. If B selects k but not i, we can directly replace k in B with i, maintaining $\#succ_B$. If B selects both k and i, and a thread j exists with $D_j < D_k$ not chosen by B, then we can replace k in B with j. If no such j exists and all threads in A with $D < D_k$ are included in B, it becomes impossible for B to have $\#succ_B > \#succ_A$.

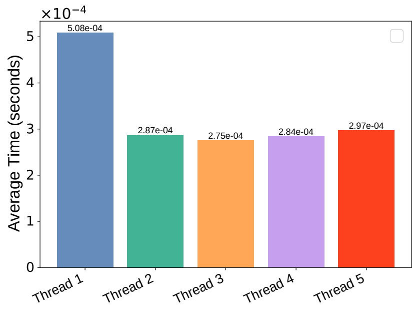
Conclusion. By evaluating each thread from 1 to K and adjusting B to match A, a contradiction arises if we assume $\#succ_B > \#succ_A$. Therefore, no better choice exists than the output of Algorithm 1, confirming its optimality. □



(a) Curve fitting performance of thread 1

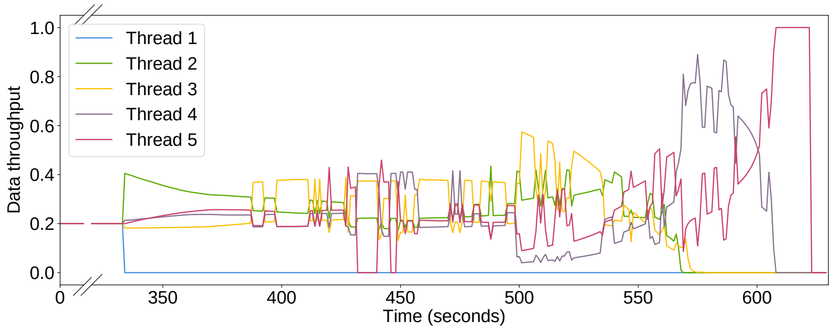


(b) Convergence of prediction error

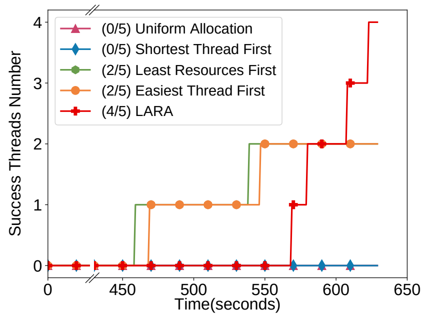


(c) Average time of single round prediction

Figure 8. Effectiveness and efficiency experiments in resource prediction.



(a) Adaptive changes in data throughput of LARA



(b) Algorithm comparation

Figure 9. Performance of computational resource allocation.

B. Experimental Supplement

In this section, we provide supplementary information for our experiments. Appendix B.1 outlines the specific settings for the model structures used in our experiments. Appendix B.2 presents detailed illustrations of the resource prediction and resource allocation process for the pure task bundle experiments. Additionally, we conduct parameter sensitivity tests in Appendix B.3 to explore the performance of the LARA algorithm under different exploration thresholds H_k and various discounted factors γ, respectively.

B.1. Implementation Details

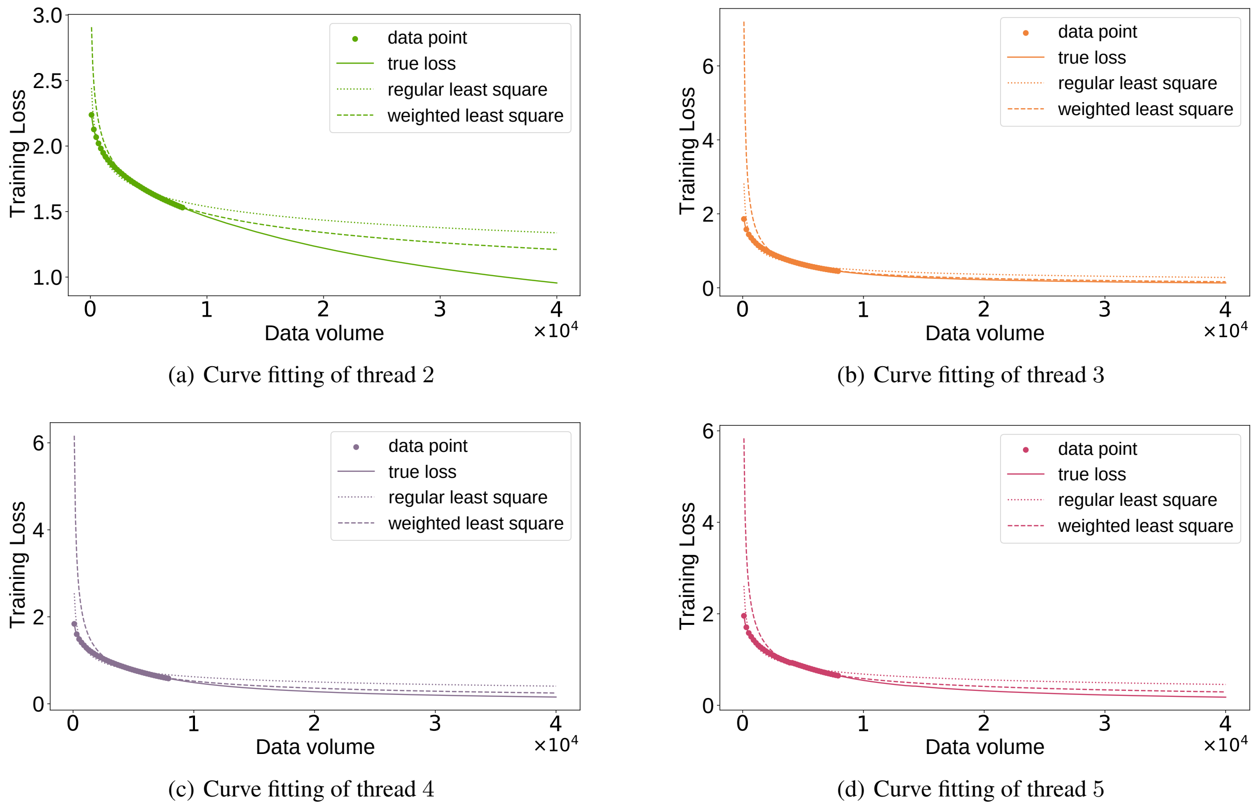
In this section, we will discuss the implementation of the model architectures detailed in Table 1 and Table 2. For Table 1, the first model is a Vision Transformer (ViT) (Dosovitskiy et al., 2021), implemented as a Simple ViT model.[3] The second model is a Long Short-Term Memory (LSTM) network, comprising two LSTM layers followed by three fully connected layers. The third model is a Convolutional Neural Network (CNN), including three convolutional layers, two pooling layers, and two fully connected layers. The fourth and fifth models are ResNet18 and ResNet34, respectively (He et al., 2016).

For Table 2, the implementations are as follows: Tasks 1 and 2 use ViT, and tasks 6 and 7 use ResNet18, both of which follow the implementations described in Table 1. Tasks 3 and 10 use a DAgger+CNN model, involving five iterations of convolutional and pooling layers, followed by flattening and three fully connected layers. Tasks 5 and 8 use an Attention+LSTM model, where the embedding results are fed into a self-attention layer, followed by LSTM layers. The final hidden state is passed through four fully connected layers. Tasks 4 and 9 use a Transformer model, which processes the input through a TransformerEncoder and then passes it through a fully connected layer.

B.2. Details of Pure Task Bundle Experiments

In this section, we present additional details for the pure task bundle experiments described in Section 4.1. First, we illustrate the loss function curve fitting performance of the weighted least squares method for each threads, as well as the associated time costs. Next, we provide a detailed view of the real-time resource allocation process over the running of the five-thread

[3]https://github.com/lucidrains/vit-pytorch?tab=readme-ov-file



(a) Curve fitting of thread 2

(b) Curve fitting of thread 3

(c) Curve fitting of thread 4

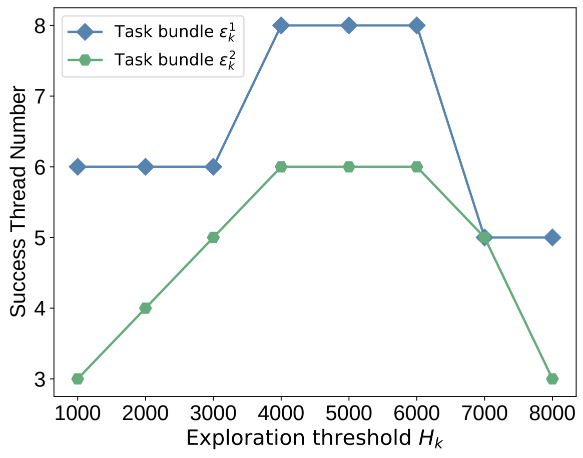
(d) Curve fitting of thread 5

Figure 10. Loss function curve fitting of weighted least square on different models.

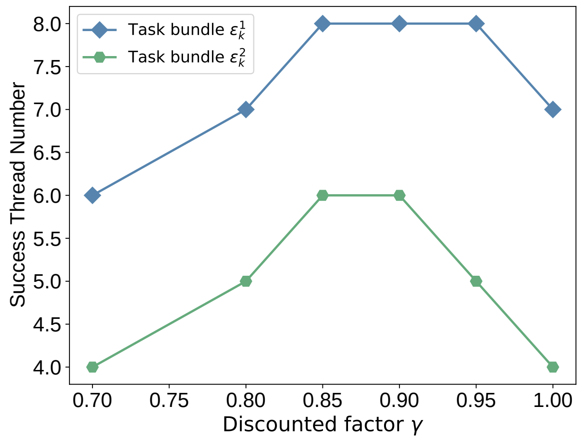
experiment and show how the completion status of each thread changes over time.

Prediction Result. Figure 8(a) shows the fitting and extrapolation of the loss function for thread 1, using the loss values observed during the exploration phase. The performance across the remaining four threads is shown in Figure 10. We compare our weighted least squares method, with a discounted factor $\gamma = 0.9$, against the regular least squares approach. The weighted least squares method provides a closer approximation to the actual loss function during the extrapolation phase and outperforms the regular method, despite slightly inferior performance during the interpolation phase. This indicates that weighted least squares is more suitable for predicting future trends of the loss function. Figure 8(b) presents the estimation error between the estimated and actual data amounts needed to meet the success criteria. The estimation error progressively diminishes and eventually converges to zero as the volume of data increases. Figure 8(c) shows the average time taken for resource prediction across different tasks. It demonstrates that the time cost of a single round prediction across different model training tasks is generally consistent, indicating minor impact from model differences. This time cost is negligible compared to the time unit (e.g., one second), and therefore does not impact the model training process.

Allocation Result. Figure 9(a) illustrates the dynamic changes in data throughput $\eta_{k,t}$ for LARA, highlighting the process of ongoing iterative adjustments in resource prediction and resource allocation. The first 335 seconds is the exploration period, such that all the threads have equal data throughput 0.2. Initially, LARA fully explores thread 1 during the exploration period and abandons it directly after the exploration period due to its low likelihood of completion. Subsequently, between 335 and 500 seconds, LARA continually refines its resource allocation strategy, and pauses the thread 5 (has the most ample time) multiple times to prioritize resources for other more urgent tasks around 450-second. Around 500-second, it prioritizes allocating more resources to thread 2 and thread 3 due to their more urgent deadlines. The remaining resources are then directed towards completing threads 4 and 5. Figure 9(b) illustrates the comparative results of successfully completed threads over time using various resource allocation strategies. Both Shortest Thread First and Uniform Allocation methods allocate excessive resources to the challenging thread 1, leading to none of the threads being completed. Least Resources First and Easiest Thread First strategies, employing our provided resource prediction method, manage to avoid dedicating resources to some of the more challenging threads. However, their inherently greedy allocation methods result in the



(a) Parameter sensitivity test of H_k



(b) Parameter sensitivity test of γ

Figure 11. Parameter sensitivity tests of exploration threshold H_k and discounted factor γ.

completion of only two threads. In contrast, the LARA algorithm, by strategically giving up on the difficult-to-complete thread 1 early, efficiently allocates resources and successfully completes the remaining four threads.

B.3. Parameter Sensitivity Tests

In this section we conduct the sensitivity experiments to study the LARA's performance under different exploration thresholds H_k and dicounted factors γ, repectively. Both experiments focus on the mixed task bundle over different sets of success criteria (ϵ_k^1 and ϵ_k^2), as outlined in Table 2.

Exploration threshold H_k. We test the success thread number of our LARA algorithm with different choices of the exploration threshold H_k with a fixed discounted factor $\gamma = 0.9$. Figure 11(a) shows that LARA's performance decreases when H_k is either too small or too large, and becomes relatively stable when lies in an appropriate region $4000 - 6000$. When H_k is too small, the precision of resource predictions is very low, leading to poor resource allocation. Conversely, a too large H_k results in excessive exploration time, leaving inadequate time for effective resource allocation.

Discounted factor γ. We evaluate the successful thread number of LARA employing weighted LS estimators with different discounted factors γ with a fixed $H_k = 6000$. Figure 11(b) shows that when γ is either too large (close to 1) or too small (close to 0.5), there is a significant increase in prediction error, leading to bad performance of LARA. Both results over task bundles with success criteria ϵ_k^1 and ϵ_k^2 indicate that when discounted factor γ in an appropriate region $0.85 - 0.95$, the performance of LARA approaches its optimum.