Online Scheduling and Pricing for Multi-LoRA Fine-Tuning Tasks

Ying Zheng
Fudan University
China
zhengy18@fudan.edu.cn

Lei Jiao*
University of Oregon
USA
jiao@cs.uoregon.edu

Han Yang
Tsinghua University
China
h-yang23@mails.tsinghua.edu.cn

Lulu Chen
Fudan University
China
cellatly@gmail.com

Ying Liu
Tsinghua University
China
liuying23@mails.tsinghua.edu.cn

Yuxiao Wang
Fudan University
China
22210720244@m.fudan.edu.cn

Yuedong Xu*
Fudan University
China
ydxu@fudan.edu.cn

Xin Wang
Fudan University
China
xinw@fudan.edu.cn

Zongpeng Li
Tsinghua University
China
zongpeng@tsinghua.edu.cn

ABSTRACT

Fine-tuning pre-trained models with task-specific data can produce customized models effective for downstream tasks. However, operating large-scale such fine-tuning tasks in real time in the data center faces non-trivial challenges, including unpredictable task arrival and system environment dynamics, complex deadline-driven fine-tuning scheduling, and intertwined task pricing and cost management. In this paper, targeting the popular Low-Rank Adaptation (LoRA) fine-tuning technique, we present the design and study of a novel auction-based mechanism to jointly schedule and price LoRA tasks in an online manner. We first model the social welfare maximization problem as an integer program for the fine-tuning service provider, capturing all the aforementioned challenges. Then, to solve this NP-hard problem online, we equivalently reformulate this original problem into a schedule selection problem, where each schedule corresponds to a concrete pre-specified operation plan over time for a task. We can thus design a polynomial-time online approximation algorithm via the online primal-dual method to determine the schedule, and with the dual variables, also determine the pricing for each admitted task. We rigorously prove the competitiveness of our online approach against the offline optimum, and prove the economic properties of truthfulness and individual rationality regarding pricing. Finally, we conduct extensive experiments and have validated the substantial advantages of our approach compared to existing methods.

KEYWORDS

Fine-tuning, Scheduling, Pricing, Online algorithm, Auction

*Corresponding authors

ICPP '24, August 12–15, 2024, Gotland, Sweden

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673083

ACM Reference Format:
Ying Zheng, Lei Jiao, Han Yang, Lulu Chen, Ying Liu, Yuxiao Wang, Yuedong Xu, Xin Wang, and Zongpeng Li. 2024. Online Scheduling and Pricing for Multi-LoRA Fine-Tuning Tasks. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673083

1 INTRODUCTION

Fine-tuning refers to the process of using relatively small task-specific datasets to adapt a large model pre-trained on extensive datasets to downstream tasks. This approach retains the rich knowledge encapsulated within the pre-trained model while avoiding the design of new models and training from scratch, thus saving significant time and costs. For instance, one can fine-tune the pre-trained Large Language Model (LLM) Bert or GPT to machine translation and semantic question answering via corresponding datasets [23, 31]. Low-Rank Adaption (LoRA) [2, 6, 15, 29] is among the most widely employed fine-tuning methods, as exemplified by the more than 1100 LLaMA-related models fine-tuned by LoRA on Hugging Face [9]. LoRA injects an "adapter" composed of two low-rank matrices into each "transformer layer" [24] of the pre-trained model, where only parameters in the adapters need to be updated and other parameters stay unchanged. For example, for GPT3, compared to fine-tuning the entire model, LoRA reduces the number of trainable parameters from 175B to 37M and the GPU memory consumption from 1.2TB to 350GB, while achieving a similar accuracy [6].

It is yet non-trivial for the *cloud fine-tuning service* to operate and manage large-scale LoRA-based fine-tuning tasks from the users, due to the following unique and fundamental challenges.

First, as the fine-tuning tasks arrive unpredictably, it is difficult to continuously schedule them for execution in real time in a dynamic data center environment whose operational cost can also be constantly changing [5, 21, 27]. Each task needs to be controlled to potentially suspend and resume execution alternately, while ensuring sufficient fine-tuning with minimum incurred cost in the long term before a pre-specified deadline that may exist. Some tasks may require data pre-processing (e.g., labeling, and cleaning) which also needs to be coordinated and completed before the fine-tuning process starts, especially when the fine-tuning service

allows outsourcing such efforts [17]. LoRA tasks can actually share the parameters of the pre-trained model [28] and only fine-tune an individual adapter for each task, demanding careful inter-task management for high training throughput and resource utilization. All of these factors need to be comprehensively handled for the cloud data center which often has heterogeneous GPUs equipped.

Second, it is not straightforward to appropriately price the received fine-tuning tasks while ensuring the profitability and the agile adaptability to the ever-changing market demand and supply. The de facto fixed pricing, as adopted by some providers [10], often fail to meet these requirements. Auction can be a more effective approach, with the fine-tuning service as the auctioneer and each fine-tuning task as a bid. Yet, existing auction design cannot be directly used in this scenario, where the bids arrive sequentially and need to be processed irrevocably on the fly, unlike a typical auction setting where all bids often come simultaneously; further, the winning-bid selection is intrinsically intertwined with the corresponding task execution, e.g., a particular task may bid a high price, but it may be impossible to schedule it for the best performance with the minimum cost if it is admitted, and vice versa. This also lifts the difficulty for designing novel auction mechanisms with desired economic properties such as truthfulness (e.g., a bid has no motivation to lie about its bidding price) and individual rationality (e.g., a bid incurs no loss to itself even when it loses in the auction).

To the best of our knowledge, none of the existing work has addressed both of the aforementioned challenges for fine-tuning tasks. Titan [4] is a scheduler tailored for fine-tuning tasks; however, it targets the offline scenario and ignores the pricing, deadline, and data pre-processing issues. Other conventional deep learning task schedulers [11, 14, 16, 19, 20, 22, 25, 30, 32] focus on traditional metrics including time efficiency, training throughput efficiency, fairness, and hence are not suitable for solving our problem. Eris [18] prices and schedules deep learning tasks based on auction mechanisms, which could be the most similar to the problem we are investigating. Yet, it ignores the performance improvement brought by multi-LoRA pre-trained model sharing, the ever-changing operational cost, and the data pre-processing decisions in the marketplace.

In this paper, we demonstrate a rigorous algorithmic study of the auction-based, online joint scheduling and pricing mechanism tailored to the fine-tuning service. We make several contributions:

- We model and formulate a long-term optimization problem to maximize the *social welfare* of the entire system, i.e., the utilities of both the cloud fine-tuning service and the LoRA fine-tuning tasks. This problem is an integer program, provably intractable even in the offline setting. Our formulation grasps all the aforementioned challenges and is general, with only mild or almost no assumption on all the inputs such as the input dynamics and heterogeneities.
- We design a smart reformulation to equivalently transform the original problem into a schedule selection problem, where a schedule is a concrete pre-specified operation plan of a fine-tuning task over time. To solve this new problem, we further design an online primal-dual algorithm to dynamically conduct the admission control and decide the schedule for each task as it arrives at the service.
- We also design the pricing mechanism for each admitted task, *i.e.,* the payment that each winning bid needs to make to the fine-tuning service, using the dual variables that are carefully and continuously updated as tasks arrive and the relevant inputs obtained from the primal problem.
- We rigorously characterize and prove multiple theoretical guarantees, including the NP-hardness of the problem, the polynomial-time complexity of our algorithms, the competitive performance of our online algorithms against the offline optimum, and the auction properties of truthfulness and individual rationality of our pricing mechanism.
- We conduct extensive trace-driven experiments. Our proposed approach consistently outperforms baselines in various settings. Specifically, in the high workload scenario, our approach improves social welfare by 48.99%, 151.57%, and 184.94% compared to three baseline algorithms.

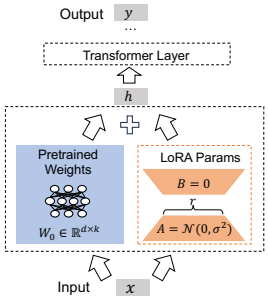

Figure 1: LoRA.

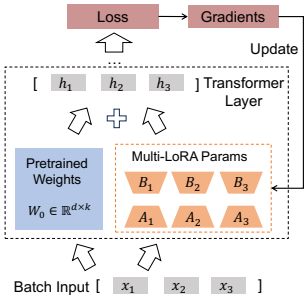

Figure 2: Multi-LoRA.

2 MODELING AND FORMULATION

2.1 System Settings and Models

Cloud System: We consider a service provider that operates a cloud data center or a GPU cluster of a set $[K] = \{1, 2, \cdots, K\}$ of GPU compute nodes for executing fine-tuning tasks submitted by the end users. Without loss of generality, we consider the entire system operating in slotted time $[T] = \{1, 2, \cdots, T\}$. Each compute node $k \in [K]$ in the cloud has its computation capacity C_{kp} in terms of the maximum number of data samples that can be processed per single time slot, and has its GPU memory capacity C_{km} in GB.

Fine-Tuning Tasks: A LoRA [6] fine-tuning task, as shown in Figure 1, uses two low-rank matrices to approximate the parameters that need to be updated for the dense layer when fine-tuning pre-trained neural networks. That is, for each "transformer layer" [24], let $W_0 \in \mathbb{R}^{d\times k}$ be the parameters on the pre-trained dense layer, and ΔW be the parameter update (*i.e.,* the adapter). Then, the resulting fine-tuned parameters can be represented as $W = W_0 + \Delta W$. LoRA approximates the update by $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. During fine-tuning, the forward result of the dense layer is calculated as $h = W_0 x + \Delta W x = W_0 x + BAx$, and the pre-trained weights W_0 can stay unchanged and only the gradient updates of the matrices A and B need to be computed during the backward propagation. Since $r \ll \min(d, k)$, LoRA significantly reduces the number of trainable parameters. As shown in Figure 2, if multiple tasks fine-tune the same pre-trained model, then these tasks can share the pre-trained model and just have their own separate adapters [28], in order to further reduce GPU memory usage and improve training efficiency.

We use $[I] = \{1, 2, ..., I\}$ to refer to the set of fine-tuning tasks. The fine-tuning task i can be represented as $\{a_i, d_i, \mathcal{D}_i, r_i, M_i, f_i, b_i\}$, where a_i is the arrival time of the task i; d_i is the deadline no later than which the task i needs to be finished; $\mathcal{D}_i$ refers to the training dataset used for fine-tuning; r_i is the GPU memory requirement for the task i; M_i is the total cumulative amount of computation required for sufficiently fine-tuning the task i; f_i indicates whether the task i needs data pre-processing; and b_i is the bidding price for the task i. Data pre-processing and bidding will be further elaborated next. When executing the fine-tuning task i on the compute node k, we denote by s_{ik} the amount of computation that can be done per single time slot, and denote by e_{ikt} the operational cost (e.g., energy consumption) at the time slot t. In this paper, we focus on the case where every task fine-tunes the same pre-trained model whose size is r_b, and up to one replica of this pre-trained model needs to be kept on each compute node, as in the LoRA weight-sharing situation. Different "zones" within the cloud data center can be set up for tasks fine-tuning different pre-trained models.

Auction-Based Pricing: The cloud fine-tuning service acts as the auctioneer and each fine-tuning task acts as a bid. Note that we consider each user (bidder) submitting only one task (bid); a user that submits multiple tasks can be essentially considered as multiple virtual users. For the task i, the bidding price b_i refers to the money that the corresponding user is willing to pay to the service for executing this task. For each bid, the service decides whether to choose the bid as a winning bid. If it is a winning bid, it means that the the service provider admits the task for execution and the user makes the payment p_i (note that p_i is determined by the service provider and may not equal b_i); if not, then the service provider declines the task and the user makes no payment. Thus, p_i is the service provider's pricing for the task i. We also define and analyze the desired economic properties of truthfulness and individual rationality for our auction later in this paper.

Data Pre-Processing: Each fine-tuning task contains its dataset, and real-world cloud fine-tuning services [17] often allow outsourcing the data pre-processing (e.g., labeling, and cleaning) to third-party *labor vendors*. Sometimes, such pre-processing is a must as the cloud fine-tuning service may have strict format requirements for the training data [8]. We consider a marketplace of multiple labor vendors indexed by $[N] = \{1, 2, ..., N\}$. As the fine-tuning task i is admitted and requests data pre-processing, the service will firstly check the price q_{in} that each labor vendor n charges and the processing delay h_{in} that each labor vendor n takes for pre-processing the task i's data, and then select and use one and only one labor vendor for this task. The service will pay the selected labor vendor correspondingly. The data pre-processing needs to be completed before the corresponding fine-tuning task starts to execute.

Control Decisions: As each fine-tuning task i arrives at the time slot a_i, the service provider responds immediately and makes the following control decisions online: (i) Whether or not to admit the task i for execution (i.e., select the bid i as a winning bid), denoted by u_i; (ii) Whether or not to execute the task i on the compute node k at the time slot t, $\forall t \geq a_i$, denoted by x_{ikt}; (iii) Whether or not to choose and use the labor vendor n for the task i's data pre-processing, denoted by z_{in}; (iv) Payment p_i that the user needs to make to the cloud fine-tuning service.

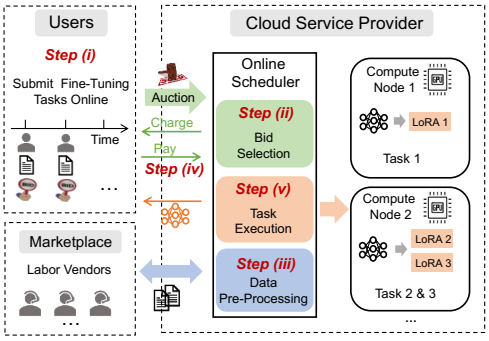

Figure 3: Fine-tuning service workflow.

System Workflow: Overall, the system workflow is shown in Figure 3, consisting of multiple steps: (i) Users submit fine-tuning tasks as bids, where different bids may arrive at different moments; (ii) As soon as each bid arrives, the fine-tuning service decides whether the bid is a winning bid; (iii) Jointly with the winning-bid determination, the service also decides the labor vendor for data pre-processing (if requested by the bid), the execution plan of the task, and the payment to charge; (iv) The user makes the payment to the service; (v) The service executes the data pre-processing and the fine-tuning process as planned, and returns the result to the user at completion. Depending on the agreement between the service and the users, "(iv)" may occur before or after "(v)".

2.2 Problem Formulation

Social Welfare: We define the *social welfare* as our optimization objective, which is the sum of each user's *utility* and the service provider's *utility*, following the conventions as in a typical auction. A user's utility is her true valuation of her fine-tuning task minus her payment to the fine-tuning service. Thus, all users' utility is

$$U_r = \sum_i U_i = \sum_i (b_i - p_i) u_i. \tag{1}$$

Note that b_i also represents the true valuation, as we will prove the *truthfulness* of our mechanism later. The service provider's utility is the payment received from the users minus the sum of the expense it pays to the data pre-processing labor vendors and its own operational cost of executing the fine-tuning tasks. That is

$$U_c = \sum_i p_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt}. \tag{2}$$

Based on these, we have the social welfare as

$$U = U_r + U_c = \sum_i b_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt}. \tag{3}$$

Note that the payment is canceled out, aligned with existing auction research; yet, in our algorithms, we still need to decide the payment for each winning bid as part of our auction outcome.

Problem Formulation: We formulate our multi-LoRA fine-tuning scheduling and pricing problem that is to be solved by the fine-tuning service provider as follows:

$$P: \max \sum_i b_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt} \tag{4}$$

$$\text{s.t. } f_i u_i \leq \sum_n z_{in} \leq 1, \forall i, \tag{4a}$$

$$\sum_k x_{ikt} \leq 1, \ \forall i, t, \tag{4b}$$

$$(a_i + f_i \sum_n h_{in} z_{in}) x_{ikt} \leq x_{ikt} t, \forall i, k, t \tag{4c}$$

$$x_{ikt} t \leq d_i, \forall i, k, t, \tag{4d}$$

$$\sum_t \sum_k s_{ik} x_{ikt} \geq M_i u_i, \forall i, \tag{4e}$$
$$\sum_i s_{ik} x_{ikt} \leq C_{kp}, \forall k, t, \tag{4f}$$
$$\sum_i r_i x_{ikt} + r_b \leq C_{km}, \forall k, t, \tag{4g}$$
$$u_i \in \{0,1\}, x_{ikt} \in \{0,1\}, z_{in} \in \{0,1\}, \forall i, k, t, n. \tag{4h}$$

We maximize the social welfare. Constraint (4a) ensures that for each task, up to one labor vendor is selected if the task is admitted and needs data pre-processing. Constraint (4b) ensures that each task at each time slot runs on no more than one compute node. Constraint (4c) ensures that each task is only executed after it arrives at the system and finishes its data pre-processing. Constraint (4d) ensures that each task is only executed before its deadline. Constraint (4e) guarantees enough computation cumulatively to complete the task. Constraints (4f) and (4g) enforce the computation capacity and the memory capacity on each compute node, respectively. Constraint (4h) specifies the domains of the control variables. Unless otherwise noted, the scopes for our indices are $i \in [I], k \in [K], t \in [T]$ and $n \in [N]$. Our problem is provably intractable; see Section 4.1 for details.

3 ALGORITHM DESIGN

3.1 Overview and Rationale

Our idea is to firstly reformulate the original problem P equivalently into a *schedule selection* problem P_1. Rather than dynamically determining when to execute a task on which compute note, we generate a series of static schedules for the task as it arrives, where each schedule is a concrete plan of executing the task on a particular compute node at a particular (and unnecessarily consecutive) set of time lots, and then select the best schedule for the task. The schedules of a task can cover all the possibilities of how to execute this task, while respecting the constraints. Each schedule of a task uniquely determines task admission, labor vendor selection, and task execution; and vice versa. We can thus solve the schedule selection problem and then recover the corresponding solution to the original problem. This method simplifies our formulation, and enables us to just focus on dynamically making one type of decision—schedule selection—instead of simultaneously making the multiple types of decisions as in the original problem.

Then, we note that, to design an online algorithm for the schedule selection problem P_1 with provably-guaranteed performance, we can derive its Lagrange dual problem D_1 and design an online primal-dual algorithm [1]. That is, as a task arrives, i,e., the constraints of the primal problem (5) and the dual problem (6) appear dynamically, we always carefully maintain a feasible solution for the primal problem and a feasible solution for the dual problem, so that the changes in the corresponding objective function values incurred by the two feasible solutions possess a certain "relationship". According to weak duality, the objective function value of the dual problem is always an upper bound of the optimal objective function value of the primal problem, and hence we can guarantee the theoretical performance of the online algorithm by firstly connecting our online solutions to the dual objective via the aforementioned relationship and then further to the (offline) primal optimum.

With such a primal-dual algorithm, we further design pricing to ensure the economic properties of truthfulness and individual rationality, as typically desired in auctions. Truthfulness ensures that every bid has no motivation to lie about its bidding price, and individual rationality ensures that every bid has no loss regardless of the auction outcome. To that end, we design the payment of a winning bid in our auction using the values of the dual variables. Dual variables can be considered as "shadow prices" [3] for the computation and the memory resources indicated in the constraints of the primal problem, where the shadow price represents the increase of the dual problem's objective value per unit increase in the amount of the resource. Intuitively, we set the payment based on the consumed resources, which is independent of its bidding price, thereby achieving the desired economic properties (while the winning-bid selection still depends on the bidding prices).

3.2 Problem Reformulation

Schedule Selection: We reformulate the problem (4) equivalently into a schedule selection problem (5). Here, we define a schedule l of the task i as an assignment of a set of concrete values to the decision variables $\{u_i, \{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}$ for the task i, satisfying Constraints (4a)-(4e). Then, the problem (4) is rewritten as

$$P_1: \max \sum_i \sum_{l \in \zeta_i} b_{il} x_{il} \tag{5}$$
$$\text{s.t.} \sum_l x_{il} \leq 1, \forall i, \tag{5a}$$
$$\sum_i \sum_{l: t \in l} s_{kt}(il) x_{il} \leq C_{kp}, \forall k, t, \tag{5b}$$
$$\sum_i \sum_{l: t \in l} r_{kt}(il) x_{il} + r_b \leq C_{km}, \forall k, t, \tag{5c}$$
$$x_{il} \in \{0,1\}, \forall i, l \in \zeta_i, \tag{5d}$$

where the binary variable x_{il} represents whether the task i is scheduled for execution using the schedule l, and ζ_i is the set of all the feasible schedules for the task i satisfying Constraints (4a)-(4e).

We introduce some additional notations. We use b_{il} to denote the increment of the objective value of the problem (4) when using the schedule l to execute the task i. Formally, $b_{il} = b_i u_i - \sum_n q_{in} z_{in} - \sum_k \sum_t e_{ikt} x_{ikt}$, where values of the variables are taken from the schedule l. Also, denote $s_{kt}(il)$ and $r_{kt}(il)$ as the computation and the memory consumption on the compute node k at the time slot t, incurred by using the schedule l to execute the task i, which are calculated as $s_{kt}(il) = s_{ik} x_{ikt}$ and $r_{kt}(il) = r_i x_{ikt}$ with $x_{ikt} \in l$, respectively. For the ease of representation, we also use $t \in l$ to indicate that the time slot t is one of the time slots as specified in schedule l, *i.e.*, $\sum_k x_{ikt} = 1$. We inevitably note that the problem (4) has a solution space of the size $2^{I+IKT+IN}$, while the the problem (5) has a solution space of the size $2^{I \cdot 2^{1+KT+N}}$; fortunately, by carefully designing our online algorithm, we can control the algorithm to run in polynomial time as shown and proved later.

Dual Problem: To design the online algorithm, we adopt the primal-dual idea. The domain of the decision variable x_{il} is then relaxed to $x_{il} \in [0,1]$. We thus write the Lagrange dual problem of the primal problem (5) as

$$D_1: \min \sum_i \mu_i + \sum_k \sum_t C_{kp} \lambda_{kt} + \sum_k \sum_t (C_{km} - r_b) \varphi_{kt} \tag{6}$$
$$\text{s.t.} \ \mu_i \geq b_{il} - \sum_k \sum_{t: t \in l} (s_{kt}(il) \lambda_{kt} + r_{kt}(il) \varphi_{kt}), \forall i, l \in \zeta_i, \tag{6a}$$
$$\mu_i \geq 0, \lambda_{kt} \geq 0, \varphi_{kt} \geq 0, \forall i, k, t, \tag{6b}$$

where μ_i, λ_{kt} and φ_{kt} are the dual variables associated with Constraints (5a), (5b), and (5c), respectively.

3.3 Online Scheduling

Online Primal-Dual Algorithm: We define the values of the dual variables λ_{kt} and φ_{kt} as

$$\lambda_{kt}^{(i)} = \lambda_{kt}^{(i-1)}(1 + \frac{s_{kt}(il)}{C_{kp}}) + \alpha(\frac{\bar{b}_{il}s_{kt}(il)}{C_{kp}}), \tag{7}$$

$$\varphi_{kt}^{(i)} = \varphi_{kt}^{(i-1)}(1 + \frac{r_{kt}(il)}{C_{km} - r_b}) + \beta(\frac{\bar{b}_{il}r_{kt}(il)}{C_{km} - r_b}), \tag{8}$$

where $\bar{b}_{il} = \frac{b_{il}}{\sum_k \sum_t s_{kt}(il) + r_{kt}(il)}$. Intuitively, $\bar{b}_{il}$ can be expressed as the social welfare improvement incurred by using per unit resource per single time slot. We use $\lambda_{kt}^{(i)}$ and $\varphi_{kt}^{(i)}$ to represent the value of λ_{kt} and φ_{kt} after the online algorithm handles the task i. Such $\lambda_{kt}^{(i)}$ and $\varphi_{kt}^{(i)}$ have the following properties: (i) They are initialized to zero and then increase as the resource consumption increases; (ii) If a schedule decision causes the cumulative usage of resources to exceed the capacity, i.e., to violate Constraint (4f) or (4g), then no more tasks will be scheduled on the compute node k at the time slot t, as shown in Lemma 2 in Appendix; (iii) They are carefully designed so that our online algorithms can achieve provably-good performance, as shown later in Theorem 5.

We select the schedule for the task i as

$$l_i = \arg\max_{l \in \zeta_i} \{F(il)\}, \tag{9}$$

where $F(il)$ is defined as

$$F(il) = b_{il} - \max_{(k,t) \in l} \{\lambda_{kt}^{(i-1)}\} \sum_k \sum_t s_{kt}(il) - \max_{(k,t) \in l} \{\varphi_{kt}^{(i-1)}\} \sum_k \sum_t r_{kt}(il). \tag{10}$$

Here, $(k, t) \in l$ refers to the compute node k and the time slot t where $x_{ikt} = 1$ in schedule l. Note that there may exist multiple pairs of (k, t) that make $x_{ikt} = 1$ in the schedule l, as a task may execute at multiple time slots. We set the dual variable μ_i as

$$\mu_i = \max\{0, F(il)\}. \tag{11}$$

This means that if the schedule l_i returned by (9) leads to a negative value of $F(il)$, then we just reject the task and set μ_i to zero; in contrast, if $\mu_i > 0$, then the service admits the task i and executes it according to the control decisions as specified in the schedule l_i.

Algorithm 1 is our online schedule selection algorithm. Line 1 initializes the dual variables λ_{kt} and φ_{kt} to zero. Upon the arrival of each task i, Line 3 checks whether the task i's dataset needs pre-processing, and if so, Line 4 collects the expense and the delay of each labor vendor. Line 5 invokes Algorithm 2, which will be described next, to find the schedule l_i consisting of all the control decisions for the task i. Lines 6-7 indicate that if $F(il) > 0$, then we update the dual variables λ_{kt} and φ_{kt}; otherwise, we reject the task i in Line 13. Line 8 checks whether there are sufficient resources to execute the task i. Formally, we check whether the condition $\sum_{i'=1}^{i} \sum_l x_{i'l} s_{kt}(i'l) \le C_{kp}$ and $\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(i'l) + r_b \le C_{km}, \forall k, t$ holds. If there are sufficient resources, then Line 9 admits the task i and executes it according to the schedule l_i; otherwise, it rejects the task i in Line 12. Line 10 pays q_{in} to the labor vendor n if it is selected. Line 11 charges the user i at the payment in (14).

Algorithm 1: Online Task Scheduling Algorithm

Input: $\{\{a_i, d_i, \mathcal{D}_i, r_i, M_i, f_i, b_i\}\}_i, r_b, C_{km}, C_{kp}$

1 Initialize $\lambda_{kt}^{(0)} = 0, \varphi_{kt}^{(0)} = 0, \forall k, t$;
2 **for** *task i* **do**
3 **if** $f_i > 0$ **then**
4 Collect $\{q_{in}, h_{in}\}_n$ of each labor vendor;
5 Invoke **Algorithm 2** to generate $l_i = \{\{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}$ and $F(il)$;
6 **if** $F(il) > 0$ **then**
7 Update λ_{kt} and φ_{kt} according to (7) and (8);
8 **if** *enough resources* **then**
9 Admit task i, i.e., set $u_i = 1$, and execute it using l_i;
10 Pay q_{in} to labor vendor n with $z_{in} = 1$;
11 Charge p_i from task i according to (14);
12 **else** Reject task i, i.e., set $u_i = 0$;
13 **else** Reject task i, i.e., set $u_i = 0$;

Optimal Schedule: The next issue is how to find the optimal schedule defined in (9) for a single task. From a high-level perspective, the essence of (9) is to determine the labor vendor (if the task i needs data-processing) and the concrete time slots to execute the task i, given the value of the dual variables λ_{kt} and φ_{kt} after our algorithm handles the task $i-1$ while satisfying Constraints (4a)-(4e). Therefore, for each pair of the expense p_{in} and the delay h_{in} of the labor vendor n, finding the optimal schedule can be formulated as the following optimization problem:

$$\min \sum_k \sum_t x_{ikt}(s_{ik}\hat{\lambda} + r_i\hat{\varphi} + e_{ikt}) \tag{12}$$
$$\text{s.t. } \sum_k \sum_t s_{ik} x_{ikt} \ge M_i, \tag{12a}$$
$$\sum_k x_{ikt} \le 1, \forall t, \tag{12b}$$
$$\hat{\lambda} \ge x_{ikt}\lambda_{kt}, \forall k, t, \tag{12c}$$
$$\hat{\varphi} \ge x_{ikt}\varphi_{kt}, \forall k, t, \tag{12d}$$
$$x_{ikt} \in \{0, 1\}, t \in [a_i + h_{in}, d_i], \hat{\lambda} \ge 0, \hat{\varphi} \ge 0, \forall k, t, \tag{12e}$$

where x_{ikt}, and t are integer decision variables. Note that the subscript i is given and fixed here, as we only target the task i. For each labor vendor n, we solve the problem (12) to obtain the task execution decision $\{x_{ikt}\}$, and then select the labor vendor that achieves the lowest objective value of the problem (12) to obtain the labor vendor selection decision $\{z_{in}\}$.

We use dynamic programming to find the optimal solution to the problem (12). When scheduling task i, for a labor vendor n with p_{in} and h_{in}, let $dp[t, w]$ be the minimum objective value of the problem (12) achieved by allocating a total amount of w computation for the task i until the time slot t, where $t \in [a_i + h_{in}, d_i], w \in [0, W_i]$, W_i is the computation amount required by task i. The update rule is

$$dp[t, w] = \min\left\{dp[t-1, w], \min_k \left\{dp[t-1, w - s_{ik}] + \Delta_{kt}\right\}\right\}, \tag{13}$$

where Δ_{kt} is the increment of the objective value of the problem (12) incurred by executing the task i on the compute node k at t.

Algorithm 2: Per-Task Schedule Selection Algorithm

Input: $\{a_i, d_i, \mathcal{D}_i, r_i, M_i, f_i, b_i\}$, $\{\lambda_{kt}^{(i-1)}\}$, $\{\varphi_{kt}^{(i-1)}\}$
Output: $l = \{\{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}, F(il)$

1 Initialize $\mu_i = 0, x_{ikt} = 0, z_{in} = 0, \forall k, t, n$;
2 **for** *labor vendor* $n \in [N]$ **do**
3 $\quad l_n = \text{findSchedule}(a_i, d_i, h_{in}, \{\lambda_{kt}^{(i-1)}\}, \{\varphi_{kt}^{(i-1)}\}, W_i)$;
4 $\quad$ Calculate $F(il_n)$ according to (10);
5 $n^* = \arg\max_n\{F(il_n)\}$, $z_{in^*} = 1$, $F(il) = \max_n\{F(il_n)\}$;
6 **Function** `findSchedule`$(a_i, d_i, h_{in}, \{\lambda_{kt}^{(i-1)}\}, \{\varphi_{kt}^{(i-1)}\}, W_i)$**:**
7 $\quad$ Initialize $dp[t, w] = \infty, \forall t \in [a_i + h_{in}, d_i], w \in [1, W_i]$;
$\quad\quad dp[t, 0] = 0, \forall t \in [a_i + h_{in}, d_i]$; $\hat{\lambda}_{kt} = 0$, $\hat{\varphi}_{kt} = 0$, $\forall k, t$;
8 $\quad$ **for** $w \in [1, W_i]$ **do**
9 $\quad\quad$ **for** $t \in [a_i + h_{in} + 1, d_i]$ **do**
10 $\quad\quad\quad$ **for** $k \in [K]$ **do**
11 $\quad\quad\quad\quad$ Calculate $\Delta_{kt} = s_{ik}\hat{\lambda}_{kt} + r_i\hat{\varphi}_{kt} + e_{ikt}$ based on $dp[t-1, w-s_{ik}]$ and $\lambda_{kt}^{(i-1)}, \varphi_{kt}^{(i-1)}$;
12 $\quad\quad\quad$ $dp[t, w] = \min\left\{dp[t-1, w], \min_k\left\{dp[t-1, w-s_{ik}] + \Delta_{kt}\right\}\right\}$
13 $\quad$ **return** *schedule that achieves* $dp[d_i, W_i]$
14 **return** l_{in^*} *and* $F(il)$;

Algorithm 2 is the dynamic programming process for finding the best schedule for a given task, and is invoked by Algorithm 1. Specifically, Line 1 is the initialization. Lines 2-4 find the optimal task execution plan for each feasible labor vendor selection. Line 5 indicates we select the labor vendor that achieves the maximum $F(il_n)$. Lines 6-13 describe the specific steps of dynamic programming. Recall that we use $dp[t, w]$ to represent the minimum objective value of the problem (12) achieved by a total amount of w computation for the task i until the time slot t. Line 7 initializes the values in the dynamic programming (DP) table. Lines 11 calculates Δ_{kt} for each labor vendor n. Line 12 describes the update rule of $dp[t, w]$. Line 13 returns the task execution plan that achieves $dp[d_i, W_i]$. Line 14 returns the optimal task execution plan across all labor vendors and the corresponding value of $F(il)$.

3.4 Online Pricing

If a task i is admitted and executed using a schedule l, i.e., $F(il) > 0$, then the user i needs to pay p_i to the service. We design the payment p_i as

$$p_i = \sum_n z_{in}p_{in} + \max_{(k,t)\in l}\{\lambda_{kt}^{(i-1)}\}\sum_k\sum_t s_{ik}x_{ikt} + \max_{(k,t)\in l}\{\varphi_{kt}^{(i-1)}\}\sum_k\sum_t r_i x_{ikt}, \tag{14}$$

where the values of x_{ikt} and z_{in} are taken from the schedule l. Recall that $(k, t) \in l$ refers those k and t with $x_{ikt} = 1$ in the schedule l. We treat $\max_{(k,t)\in l}\{\lambda_{kt}^{(i-1)}\}$ and $\max_{(k,t)\in l}\{\varphi_{kt}^{(i-1)}\}$ as the marginal price of computation and memory resources after handling the task $i-1$, respectively. The bidding price affects whether a bid wins in the auction or not, and if it wins, we set the corresponding payment for this bid as only based on its consumed resources, not depending on the bidding price any more. We note that the payment can also be written as $p_i = \sum_n z_{in}p_{in} + \max_{(k,t)\in l}\{\lambda_{kt}^{(i-1)}\}\sum_k\sum_t s_{kt}(il) + \max_{(k,t)\in l}\{\varphi_{kt}^{(i-1)}\}\sum_k\sum_t r_{kt}(il)$. As shown in Theorems 3 and 4, this payment design ensures the expected economic properties of truthfulness and individual rationality.

4 PERFORMANCE ANALYSIS

4.1 Intractability

Theorem 1. *Our problem* (4) *is NP-hard.*

Proof. Our problem contains the well-known 0-1 knapsack problem, and is thus NP-hard. To see this, we focus on the decision variables u_i, $\forall i$. Joining Constraints (4e) and (4f), we get $\sum_i M_i u_i \le T \cdot \sum_k C_{kp}$. That is, for the "item" i, we consider M_i as the "weight" and b_i in the objective function as the "value"; we also consider $T \cdot \sum_k C_{kp}$ as the capacity of the "knapsack" in terms of the total tolerable weight. Therefore, we have the 0-1 knapsack problem of selecting and placing items into the knapsack to maximize the total value while respecting the knapsack's capacity, after we ignore all other terms and constraints. □

4.2 Time Complexity

Theorem 2. *Our algorithms finish in polynomial time.*

Proof. Our approach consists of two algorithms, where Algorithm 1 invokes Algorithm 2. We consider the key steps for each algorithm. For the "findSchedule" function in Algorithm 2, the "for" loop in Line 8, Line 9, and Line 10 iterates at most W, T, and K times, respectively. Line 12 runs in $O(K)$. Then, the "findSchedule" function runs in $O(WTK)$. Line 3 in Algorithm 2 invokes the "findSchedule" function for each labor vendor n. Thus, overall, the Algorithm 2 runs in $O(NWTK)$. Algorithm 1 invokes Algorithm 2 for each task, and thus the time complexity of Algorithm 1 is $O(INWTK)$. □

4.3 Truthfulness and Individual Rationality

We formally define the utility of a bid, based on which we further formally define and prove the economic properties of truthfulness and individual rationality achieved by our proposed algorithms.

Definition 1. ***Utility****: The utility of a bid i is*

$$U_i(b_i) = \begin{cases} v_i - p_i, & \text{if } u_i = 1 \\ 0, & \text{if } u_i = 0 \end{cases} \tag{15}$$

where b_i is the bidding price; v_i is the true valuation of the bid i; p_i is the payment made to the auctioneer if the bid i is a winning bid; u_i represents whether the bid i is chosen as a winning bid by the auctioneer. Note that u_i is ultimately a function of b_i.

Definition 2. ***Truthfulness****: An auction is truthful if every bid maximizes its utility by bidding the true valuation, i.e., for all $b_i \neq v_i$, $U_i(v_i) \ge U_i(b_i)$, $\forall i$.*

Theorem 3. *Our online auction is truthful.*

Proof. Each task (or bid) i can be either admitted (i.e., chosen as a winning bid) or rejected (i.e., not chosen as a winning bid). We analyze these two cases respectively as follows.

When the task i is rejected with a bidding price v_i, i.e., all schedules make $F(il)$ as in (10) less than 0, we have $u_i = 0$ and $U_i(v_i) = 0$. Recall that $F(il)$ is essentially $b_i - p_i$, thus this rejected task i makes $v_i - p_i < 0$. Now, if the task i bids $b_i < v_i$, it is obvious that $F(il)$ remains less than 0 and we have $u_i = 0$ and $U_i(b_i) = 0$. If the task i bids $b_i > v_i$, then there could be a chance for this task to be admitted; yet, $U_i(b_i)$ decreases due to $v_i - p_i < 0$.

When the task i is admitted with a bidding price of v_i, i.e., the optimal schedule generated by our algorithms makes $F(il)$ greater than 0, then we have $v_i - p_i > 0$, $u_i = 1$, and $U_i(v_i) = v_i - p_i$. Now, if the task i bids $b_i < v_i$, $U_i(b_i)$ would not increase. Even worse, it may lead to the task i being rejected and reducing $U_i(b_i)$ to zero. If the task i bids $b_i > v_i$, $F(il)$ would still be greater than 0 and thus we continue to have $u_i = 1$, and $U_i(b_i) = v_i - p_i$.

In both cases as analyzed above, we have $U_i(v_i) \geq U_i(b_i), \forall i$. □

Definition 3. ***Individual Rationality**: An auction is individually rational if every bid always has non-negative utility regardless of the auction outcome, i.e., for any b_i, we always have $U_i(b_i) \geq 0, \forall i$.*

Theorem 4. *Our online auction is individually rational.*

Proof. Recall that $F(il)$ is essentially $b_i - p_i$. From Theorem 3 we know that all bids bid truthfully, and thus $F(il) = v_i - p_i$. Algorithm 1 ensures that for each admitted task i, $F(il) > 0$. Therefore, we have the bid i's utility as $U_i(v_i) = v_i - p_i > 0$. If the task i is rejected, then the utility is always set to zero. Therefore, a bid always has non-negative utility regardless of the auction outcome. □

4.4 Competitive Ratio

The competitive ratio characterizes the multiplicative gap between the objective function value evaluated with the online solutions and that evaluated with the offline optimal solutions. Online solutions are produced by online algorithms on the fly as time goes as the inputs are gradually observed, and the offline optimal solutions are computed by solving the problem optimally at hindsight assuming all the inputs over the entire time horizon are observed all at once.

Definition 4. ***Competitive Ratio**: Let OPT be the offline optimal objective value of the problem P. Let P^I be the objective value of P from our online approach after it handles all the I tasks. Our online approach has the competitive ratio γ if there exits a constant $\gamma \geq 1$ so that $P^I \geq \frac{1}{\gamma} OPT$ always holds regardless of the inputs to P.*

Theorem 5. *Our proposed online approach has the competitive ratio $\gamma = \rho(1 + \max\{\alpha, \beta\})$ for the problem P, i.e., the problem (4).*

Proof. We place all the related lemmas and the details of this proof in the Appendix. We actually follow the roadmap below:

$$P^I = P_1^I \geq \frac{1}{\rho}\tilde{P}_1^I \geq \frac{1}{\rho}\frac{1}{1+\max\{\alpha,\beta\}} D_1^I \geq \frac{1}{\rho(1+\max\{\alpha,\beta\})} OPT. \tag{16}$$

P^I, P_1^I, and D_1^I are the objective values of the problems P, P_1 and D_1, respectively, after handling all I tasks. We use $\tilde{P}_1^I$ to represent the objective value of a virtual almost-feasible problem, which assists this proof. The equality in (16) holds since the problem P is equivalent to the problem P_1. The first inequality in (16) is due to Lemma 3. The second inequality in (16) is due to Lemma 1. Lemma 4 indicates that the value of D^I is attained by a dual feasible solution. The last inequality in (16) holds due to weak duality. □

5 EXPERIMENTAL EVALUATIONS

5.1 Evaluation Settings

Cloud Service Settings: We consider the entire system operating in one day, which is 144 time slots with each time slot lasting for 10 minutes. We investigate a cloud data center of different scales from 50 to 200 compute nodes. Regarding the compute node capacity of nodes, we use two types of compute nodes, including the A100(80GB) and A40(48GB). In addition, we also considered the scenario where a mix of these two types of GPUs forms a heterogeneous computing environment. We implement the trace-driven fine-tuning task scheduling simulation system using Python on a server with Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz running Ubuntu 20.04.6 LTS.

Fine-Tuning Tasks: To obtain the values of experimental parameters, including r_i, r_b, s_{ik}, C_{kp} and C_{km}, we finetune GPT-2 model using LoRA on the NVIDIA A100(80GB) GPU and A40(48GB) GPU, respectively. We record the amount of computation (number of data samples) within a time slot that the GPU can process under different batch size values. For each time slot, the number of fine-tuning tasks arrived online is based on three public real-world traces, including MLaaS[26], Philly[12], and Helios[7], and our synthetic traces. The number of epochs of each task is generated randomly between 1 and 5. Note that in fine-tuning tasks, we usually only need a few epochs to obtain a satisfactory downstream model. Referring to publicly available datasets such as Samsum, we randomly generate the number of training data for each user based on a uniform distribution between [5,20]k.

Algorithms for Comparison: We implement and compare our approach, **pdFTSP** (Online **p**rimal-**d**ual based **F**ine-Tuning **T**ask **S**cheduling and **P**ricing), against the following alternatives: **(i) *Titan*[4]**: Titan schedules fine-tuning tasks based on solving an offline MILP problem, and thus only works in offline scenarios. To adapt it to our online scenario, we solve the MILP via Gurobi at the beginning of each time slot for the tasks arrived at the beginning of the time slot. Additionally, we allow Titan to select the labor vendor in the marketplace randomly; **(ii) *EFT (Earliest Finish Time)***: For each task, EFT chooses the labor vendor with the lowest delay for data pre-processing in the marketplace. EFT allocates the computation of the incoming task to the compute nodes at the time slots where the task can be finished as soon as possible; **(iii) *NTM (No Task Merging)***: For each task, NTM chooses the labor vendor in the marketplace randomly. In NTM, there is only one task can be executed on each compute node at each time. NTM also allocates the computation to the compute nodes so that the task can be finished as soon as possible.

5.2 Evaluation Results

Impact of System Scale: Figure 4 illustrates the impact of the number of compute nodes on normalized social welfare. As the number of computing nodes increases, more tasks can be processed,

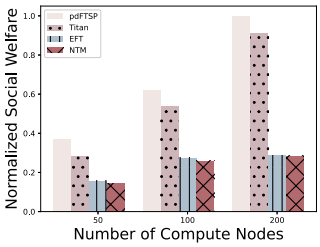

Figure 4: Impact of Data Center Scale

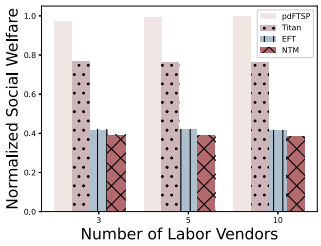

Figure 5: Impact of Number of Labor Vendors

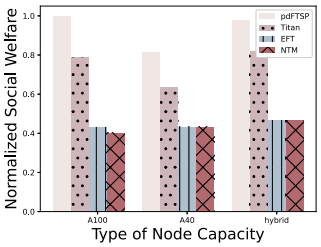

Figure 6: Impact of Per-Node Capacity

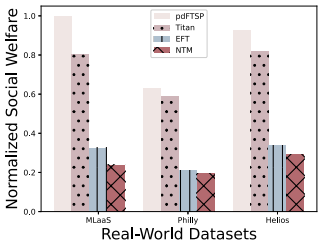

Figure 7: Impact of Real-World Task Traces

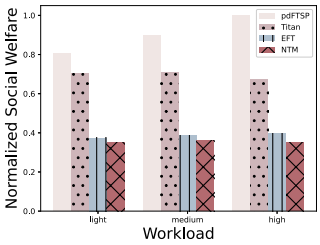

Figure 8: Impact of Task Dynamics

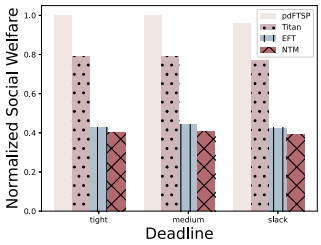

Figure 9: Impact of Task Deadlines

Figure 10: Truthfulness

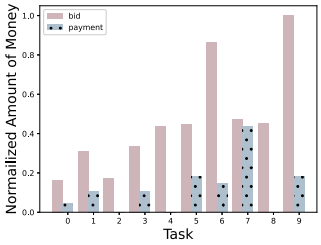

Figure 11: Individual Rationality

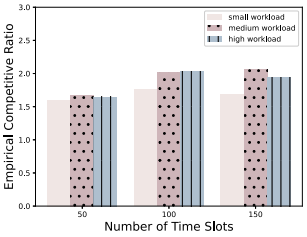

Figure 12: Competitiveness

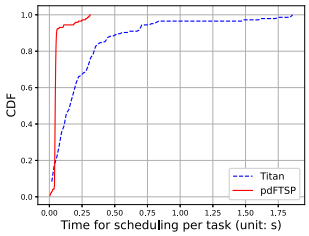

Figure 13: Running time

and thus social welfare grows accordingly. When the tested number of compute nodes is 50, the performance improvements over Titan, EFT, and NTM are 30.78%, 137.35%, and 155.84%, respectively. Additionally, in Figure 5, we vary the number of labor vendors in the marketplace. The social welfare value increases slightly as the number of labor vendors increases, since we have more choices for data pre-processing for the fine-tuning tasks. Figure 6 describes the social welfare values of four algorithms when we change the resource capacity of compute nodes. The resource capacities for these compute nodes in the first two sets of experiments are based on the NVIDIA A100(80GB)GPU and A40(48GB) GPU, respectively. Due to the stronger computation capacity of the A100, the first set of experiments achieve higher social welfare values than the second. In the third set of experiments of mixed GPUs, pdFTSP consistently achieves the best social welfare.

Impact of Task Dynamics and Deadlines: We simulate the task dynamics by using both real-world and synthetic traces, and the results are shown in Figures 7 and 8, respectively. We can observe that our proposed pdFTSP consistently outperforms three baseline algorithms when tested with three real-world traces. For the synthetic traces, we generate the number of arrived tasks at the beginning of each time slot following the Poisson process. The light, medium, and high workload correspond to the Poisson process with average value 30, 50 and 80, respectively. The proposed pdFTSP achieves more performance improvement as workload increases. Specifically, in high workload scenario, the improvement is 48.99%, 151.57%, and 184.94% when compared to Titan, EFT, and NTM, respectively. Figure 9 depicts the algorithms' performance with different methods for generating deadlines, and the pdFTSP still achieves the best performance.

Truthfulness and Individual Rationality: Figure 10 confirms the truthfulness of the proposed approach, where we consider a bid randomly drawn from our experiments. The true valuation is 15, and the optimal schedule returned by Algorithm 2 incurs a total expanse of 10. As shown, the bidding price only affects the auction outcome of whether a bid wins, while bidding the true valuation always yields the maximum utility. In Figure 11, we randomly sample 10 tasks, and illustrate the users' bids and their payments. We see that the bid is always higher than its payment, indicating the non-negative utility and thus the individual rationality.

Competitive Ratio: Figure 12 evaluates the empirical competitive ratio, which is the ratio of the social welfare achieved by offline optimum to that achieved by our online solution. We obtain the offline optimum via Gurobi solver. Results demonstrate that the proposed algorithm pdFTSP achieves empirical competitive ratios of no more than 3 in the various settings.

Algorithm Runtime: Figure 13 demonstrates the runtime of pdFTSP and Titan when scheduling a single fine-tuning task in the scenario of 100 compute nodes. Since Titan solves a MILP that includes multiple tasks at the beginning of each time slot, we average the Gurobi solver's runtime over the number of tasks. This figure shows that pdFTSP has a shorter algorithm runtime, and Titan's runtime becomes worse as the problem size grows.

6 RELATED WORK

To the best of our knowledge, Titan [4] is currently the only scheduler tailored for fine-tuning tasks in the GPU clusters. Titan formulates the fine-tuning task scheduling problem as a Mixed Integer Linear Program (MILP), and uses an MILP solver. However, it takes

relatively long time to find the solution as the problem scales up. Titan assumes the jobs' arrival information is known in prior, and thus cannot perform well in the online scenario. It also ignores the pricing issue and cannot achieve joint pricing and scheduling.

As for existing conventional deep learning job schedulers in data centers, AFS [11] leverages elastic resource sharing to reduce the average job completion time. However, the gain comes from the long processing time (*e.g.*, up to 2.8 days), which is not suitable for our LoRA-based fine-tuning tasks that can be completed within hours. Optimus [19] uses online fitting to estimate training speed, minimizing the job completion time by joint resource allocation and task placement. Themis [16] targets fairness scheduling based on the sharing incentive metric and uses an auction to allocate resources. These works, as well as other existing schedulers [14, 20, 22, 25, 30, 32], focus on traditional metrics such as time efficiency, training throughput, and fairness, ignoring the important joint pricing and scheduling problem and the challenges brought by task deadlines. Eris [18] prices and schedules deep learning tasks in edge networks based on auctions, which could be the most similar to our work. But it ignores the multi-LoRA paradigm, the ever-changing operational cost, and the data pre-processing decisions.

7 CONCLUSION

Scheduling and pricing fine-tuning tasks with large pre-trained models is an increasingly important issue that needs to be addressed by cloud services in the AI era. This paper presents our mathematical study toward this direction. We conduct auction-based social welfare optimization and propose online algorithms from the cloud service's perspective to schedule and price each fine-tuning task as it arrives. Our work features the provable optimization guarantees and economic properties, and the thorough numerical evaluations that validate our design. For future work, we intend to extend our study to serving fine-tuning tasks with paradigms beyond LoRA.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China (2022YFB2901300), by China Unicom Guangdong, by the Quan Cheng Laboratory (QCLZD202304 and SYS202201), by the U.S. National Science Foundation (CNS-2047719 and CNS-2225949), by the National Natural Science Foundation of China (62072117), and by the Shanghai Natural Science Foundation (22ZR1407000).

REFERENCES

[1] Niv Buchbinder, Joseph Seffi Naor, et al. 2009. The Design of Competitive Online Algorithms via a Primal-Dual Approach. *Foundations and Trends ® in Theoretical Computer Science* 3, 2–3 (2009), 93–263.
[2] Tim Dettmers, Artidoro Pagnoni, et al. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *NeurIPS*.
[3] Tomas Gal. 1986. Shadow Prices and Sensitivity Analysis in Linear Programming under Degeneracy: State-of-the-Art-Survey. *Operations-Research-Spektrum* 8, 2 (1986), 59–71.
[4] Wei Gao, Peng Sun, et al. 2022. Titan: A Scheduler for Foundation Model Fine-tuning Workloads. In *ACM SoCC*.
[5] Albert Greenberg, James Hamilton, et al. 2008. The Cost of a Cloud: Research Problems in Data Center Networks. *ACM SIGCOMM Computer Communication Review* 39, 1, 68–73.
[6] Edward J Hu, Yelong Shen, et al. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685* (2021).
[7] Qinghao Hu, Peng Sun, et al. 2021. Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters. In *ACM/IEEE SC*.
[8] Hugging Face Fine-Tunig Data Format. [n. d.]. https://huggingface.co/docs/autotrain/llm_finetuning.
[9] Hugging Face Models. [n. d.]. https://huggingface.co/models.
[10] Hugging Face Pricing. [n. d.]. https://huggingface.co/pricing#spaces.
[11] Changho Hwang, Taehyun Kim, et al. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *USENIX NSDI*.
[12] Myeongjae Jeon, Shivaram Venkataraman, et al. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *USENIX ATC*.
[13] Esmaeil Keyvanshokooh, Cong Shi, et al. 2021. Online Advance Scheduling with Overtime: A Primal-Dual Approach. *Manufacturing & Service Operations Management* 23, 1 (2021), 246–266.
[14] Xiaotong Li, Ruiting Zhou, et al. 2019. Online Placement and Scaling of Geo-distributed Machine Learning Jobs via Volume-discounting Brokerage. *IEEE Transactions on Parallel and Distributed Systems* 31, 4 (2019), 948–966.
[15] Shih-Yang Liu, Chien-Yi Wang, et al. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. *arXiv preprint arXiv:2402.09353* (2024).
[16] Kshiteej Mahajan, Arjun Balasubramanian, et al. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *USENIX NSDI*.
[17] Microsoft Azure Labeling Service. [n. d.]. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-outsource-data-labeling?view=azureml-api-2.
[18] Jinlong Pang, Ziyi Han, et al. 2023. Eris: An Online Auction for Scheduling Unbiased Distributed Learning Over Edge Networks. *IEEE Transactions on Mobile Computing* 23, 6 (2023), 7196–7209.
[19] Yanghua Peng, Yixin Bao, et al. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *ACM EuroSys*.
[20] Aurick Qiao, Sang Keun Choe, et al. 2021. Pollux: Co-Adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *USENIX OSDI*.
[21] Lei Rao, Xue Liu, et al. 2010. Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment. In *IEEE INFOCOM*.
[22] Suraiya Tairin, Haiying Shen, et al. 2023. Embracing Uncertainty for Equity in Resource Allocation in ML Training. In *ICPP*.
[23] Hugo Touvron, Louis Martin, Kevin Stone, et al. 2023. LLAMA 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).
[24] Ashish Vaswani, Noam Shazeer, et al. 2017. Attention is All You Need. In *NIPS*.
[25] Ne Wang, Ruiting Zhou, et al. 2022. Preemptive Scheduling for Distributed Machine Learning Jobs in Edge-Cloud Networks. *IEEE Journal on Selected Areas in Communications* 40, 8 (2022), 2411–2425.
[26] Qizhen Weng, Wencong Xiao, et al. 2022. MLaaS in the wild: Workload analysis and scheduling in Large-Scaleheterogeneous GPU clusters. In *USENIX NSDI*.
[27] Zichuan Xu and Weifa Liang. 2015. Operational Cost Minimization of Distributed Data Centers through the Provision of Fair Request Rate Allocations while Meeting Different User SLAs. *Elsevier Computer Networks* 83 (2015), 59–75.
[28] Zhengmao Ye, Dengchun Li, et al. 2023. ASPEN: High-Throughput LoRA Fine-Tuning of Large Language Models with a Single GPU. *arXiv preprint arXiv:2312.02515* (2023).
[29] Qingru Zhang, Minshuo Chen, et al. 2022. Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In *ICLR*.
[30] Qin Zhang, Ruiting Zhou, et al. 2020. Online Scheduling of Heterogeneous Distributed Machine Learning Jobs. In *ACM MOBIHOC*.
[31] Renrui Zhang, Jiaming Han, et al. 2023. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention. *arXiv preprint arXiv:2303.16199* (2023).
[32] Ruiting Zhou, Jinlong Pang, et al. 2023. Online Scheduling Algorithm for Heterogeneous Distributed Machine Learning Jobs. *IEEE Transactions on Cloud Computing* 11, 2 (2023), 1514–1529.

A APPENDIX

Lemma 1. *(Relationship between Almost-Feasible problem and dual problem.)* $\tilde{P}_1^I \geq \frac{1}{1+\max\{\alpha,\beta\}} D_1^I$.

Proof. We first define two types of conditions for each task i to assist in the proof: **(i) Almost-Feasible Condition**: $F(il) > 0$; **(ii) Feasible Condition**: $(F(il) > 0) \wedge (\sum_{i'=1}^{i} \sum_l x_{i'l} s_{kt}(i'l) \leq C_{kp}, \forall k, t) \wedge (\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(i'l) + r_b \leq C_{km}, \forall k, t)$. In Algorithm 1, line 6 checks Almost-Feasible Condition , while line 6 and line 8 together check Feasible Condition. We refer to the solution generated by the Almost-Feasible Condition as the almost-feasible primal solution. Likewise, the solution generated by the Feasible Condition is called the feasible primal solution. An almost-feasible primal solution can be easily transformed into a feasible

primal solution by simply not executing tasks that satisfy line 6 but not line 8 in Algorithm 1. Let $\tilde{P}_1^i$ denote the objective function value of problem P_1 after processing task i achieved by the almost-feasible primal solution. Denote D_1^i as the objective function value of dual problem D_1 after processing task i. In this lemma, we will specify the relationship between $\tilde{P}_1^I$ and D_1^I. Let $S_a \subset [I]$ be the set of tasks that satisfy the Almost-Feasible Condition, i.e., $F(il) > 0, \forall i \in S_a$. Let $S_r = [I] \backslash S_a$ be the set of tasks that the Almost-Feasible Condition directly rejects. For task $i \in S_r$, $\tilde{P}_1^i - \tilde{P}_1^{i-1} = 0$, $D_1^i - D_1^{i-1} = 0$. For task $i \in S_a$, suppose that the optimal schedule generated by the online approach (Algorithm 2) is l, then $\tilde{P}_1^i - \tilde{P}_1^{i-1} = b_{il}$. We also have $D_1^i - D_1^{i-1} = \mu_i + \sum_k \sum_t C_{kp}(\lambda_{kt}^{(i)} - \lambda_{kt}^{(i-1)}) + \sum_k \sum_t (C_{km} - r_b)(\varphi_{kt}^{(i)} - \varphi_{kt}^{(i-1)})$. Equation (11) indicates that for the selected optimal schedule l and $F(il) > 0$, $\mu_i = F(il) \leq b_{il} - \sum_k \sum_{t:t \in l}(s_{kt}(il)\lambda_{kt}^{(i-1)} + r_{kt}(il)\varphi_{kt}^{(i-1)})$. Additionally, from equation (7) and (8) we have $\lambda_{kt}^{(i)} - \lambda_{kt}^{(i-1)} = \lambda_{kt}^{(i-1)} \frac{s_{kt}(il)}{C_{kp}} + \alpha(\frac{\bar{b}_{il} s_{kt}(il)}{C_{kp}})$, $\varphi_{kt}^{(i)} - \varphi_{kt}^{(i-1)} = \varphi_{kt}^{(i-1)} \frac{r_{kt}(il)}{C_{km} - r_b} + \beta(\frac{\bar{b}_{il} r_{kt}(il)}{C_{km} - r_b})$. Next, we substitute μ_i, $\lambda_{kt}^{(i)} - \lambda_{kt}^{(i-1)}$ and $\varphi_{kt}^{(i)} - \varphi_{kt}^{(i-1)}$ in $D_1^i - D_1^{i-1}$ with the above results. Then we get $D_1^i - D_1^{i-1} \leq b_{il} + \sum_k \sum_t \alpha \cdot \bar{b}_{il} \cdot s_{kt}(il) + \sum_k \sum_t \beta \cdot \bar{b}_{il} \cdot r_{kt}(il)$. Recall that $\bar{b}_{il} = \frac{b_{il}}{\sum_k \sum_t s_{kt}(il) + r_{kt}(il)}$. Thus $D_1^i - D_1^{i-1} \leq b_{il} + \max\{\alpha, \beta\} b_{il} = (1 + \max\{\alpha, \beta\}) b_{il} = (1 + \max\{\alpha, \beta\})(\tilde{P}_1^i - \tilde{P}_1^{i-1})$. The initial values $\tilde{P}_1^0 = D_1^0 = 0$. Therefore, $\tilde{P}_1^I = \tilde{P}_1^0 + \sum_{i \in S_a}(\tilde{P}_1^i - \tilde{P}_1^{i-1}) = \sum_{i \in S_a}(\tilde{P}_1^i - \tilde{P}_1^{i-1}) \geq \sum_{i \in S_a} \frac{1}{(1 + \max\{\alpha, \beta\})}(D_1^i - D_1^{i-1}) = \frac{1}{(1 + \max\{\alpha, \beta\})}(D_1^I - D_1^0) = \frac{1}{(1 + \max\{\alpha, \beta\})} D_1^I$.

□

Lemma 2. *(Capacity control.) Assume $\bar{b}_{il} \geq 1, \forall i, l$, let $\alpha = \max_i\{\frac{b_i}{M_i}\}$ and $\beta = \max_i\{\frac{b_i}{r_i}\}$, then for any task i and its corresponding optimal schedule l generated by Algorithm 2, if there exists a pair of (k, t) resulting in $\sum_{i'=1}^{i} \sum_l x_{i'l} s_{kt}(i'l) \geq C_{kp}$ or $\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(i'l) + r_b \geq C_{km}$, then no future task would be scheduled to execute on compute node k at time t.*

Proof. Since $\bar{b}_{il} = \frac{b_{il}}{\sum_k \sum_t s_{kt}(il) + r_{kt}(il)}$, then the assumption $\bar{b}_{il} \geq 1$ is equivalent to say that there is a lower bound on the minimum wage, which is commonly used as in [13], since we can scale the units of b_{il}, $s_{kt}(il)$ and $r_{kt}(il)$. According to (7), we have $\lambda_{kt}^{(i)} + \alpha = \lambda_{kt}^{(i-1)}(1 + \frac{s_{kt}(il)}{C_{kp}}) + \alpha(\frac{\bar{b}_{il} s_{kt}(il)}{C_{kp}}) + \alpha = \lambda_{kt}^{(i-1)}(1 + \frac{s_{kt}(il)}{C_{kp}}) + \alpha(1 + \frac{\bar{b}_{il} s_{kt}(il)}{C_{kp}})$. Since $\bar{b}_{il} \geq 1$, we can obtain $\lambda_{kt}^{(i)} + \alpha \geq (\lambda_{kt}^{(i-1)} + \alpha)(1 + \frac{s_{kt}(il)}{C_{kp}}) \geq (\lambda_{kt}^{(i-1)} + \alpha) 2^{\frac{s_{kt}(il)}{C_{kp}}}$, where the last inequality is due to $1 + x \geq 2^x$ for $x \in [0, 1]$. Using the above formula recursively until we reach the initial status, then we get $\lambda_{kt}^{(i)} + \alpha \geq (\lambda_{kt}^{(0)} + \alpha) 2^{\frac{\sum_{i'=1}^{i} \sum_l x_{i'l} s_{kt}(il)}{C_{kp}}}$. Likewise, according to (8), we have $\varphi_{kt}^{(i)} + \beta \geq (\varphi_{kt}^{(0)} + \beta) 2^{\frac{\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(il)}{C_{km} - r_b}}$. If a task i with schedule l makes the allocation results exceed the resource capacity, *i.e.*, $\sum_{i'=1}^{i} \sum_l x_{i'l} s_{kt}(i'l) \geq C_{kp}$ or $\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(i'l) + r_b \geq C_{km}$, then we have either $\frac{\sum_{i'=1}^{i'} \sum_l x_{i'l} s_{kt}(i'l')}{C_{kp}} \geq 1$ or $\frac{\sum_{i'=1}^{i'} \sum_l x_{i'l} r_{kt}(i'l')}{C_{km} - r_b} \geq 1$. Thus we have either $\lambda_{kt}^{(i)} + \alpha \geq (\lambda_{kt}^{(0)} + \alpha) \cdot 2$ or $\varphi_{kt}^{(i)} + \beta \geq (\varphi_{kt}^{(0)} + \beta) \cdot 2$. Recall that the initial values $\lambda_{kt}^{(0)} = \varphi_{kt}^{(0)} = 0$. Then we have either $\lambda_{kt}^{(i)} \geq \alpha$ or $\varphi_{kt}^{(i)} \geq \beta$. Recall that $\alpha = \max_i\{\frac{b_i}{M_i}\}$ and $\beta = \max_i\{\frac{b_i}{r_i}\}$. When $\lambda_{kt}^{(i)} \geq \alpha$, for any future task $\hat{i}$, if a schedule $\hat{l}$ contains executing the task $\hat{i}$ on compute node k at time slot t, then the resource price $\max_{(k,t) \in l}\{\lambda_{kt}^{(i)}\} \geq \lambda_{kt}^{(i)} = \alpha$. Thus we have $F(\hat{i}\hat{l}) = b_{\hat{i}\hat{l}} - M_{\hat{i}} \max_{(k,t) \in l}\{\lambda_{kt}^{(i)}\} \leq b_{\hat{i}} - M_{\hat{i}} \lambda_{kt}^{(i)} = b_{\hat{i}} - M_{\hat{i}} \alpha < 0$. Therefore, we have $F(\hat{i}\hat{l}) < 0$, and hence, the future task $\hat{i}$ would not be allocated to compute node k at time slot t. Likewise, $\varphi_{kt}^{(i)} \geq \beta$ and $\beta = \max_i\{\frac{b_i}{r_i}\}$ also incurs the future task $\hat{i}$ would not be allocated to compute node k at time slot t. □

Lemma 3. *(Relationship between primal problem and Almost-Feasible problem.) $P_1^I \geq \frac{1}{\rho} \tilde{P}_1^I$, where $\rho = 1 + \max\{\frac{\bar{b}_{il,max}}{\bar{b}_{il,min}} \frac{s_{ik,max}}{s_{ik,min}}, \frac{\bar{b}_{il,max}}{\bar{b}_{il,min}} \frac{r_{i,max}}{r_{i,min}}\}$.*

Proof. Let S_a and S_c be the set of accepted tasks filtered by Almost-Feasible Condition and Feasible Condition, respectively. Then we have $S_c \subset S_a$. Note that we can convert the almost-feasible solution to the feasible solution by simply not executing the tasks in $S_a \backslash S_c$. We use $\tilde{P}_1^I$ and P_1^I to represent the objective value of problem P_1 after processing all I incurred by almost-feasible solution and feasible solution, respectively. Then

$\frac{\tilde{P}_1^I}{P_1^I} = \frac{\sum_{i \in S_a} \sum_l x_{il} b_{il}}{\sum_{i \in S_c} \sum_l x_{il} b_{il}} = \frac{\sum_{i \in S_c} \sum_l x_{il} b_{il} + \sum_{i \in S_a \backslash S_c} \sum_l x_{il} b_{il}}{\sum_{i \in S_c} \sum_l x_{il} b_{il}} = 1 + \frac{\sum_{i \in S_a \backslash S_c} \sum_l x_{il} b_{il}}{\sum_{i \in S_c} \sum_l x_{il} b_{il}}$

$= 1 + \frac{\sum_{i \in S_a \backslash S_c} \sum_l x_{il} \bar{b}_{il} \sum_k \sum_t (s_{kt}(il) + r_{kt}(il))}{\sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} \sum_k \sum_{t:t \in l} (s_{kt}(il) + r_{kt}(il))}$

$= 1 + \frac{\sum_k \sum_t \sum_{i \in S_a \backslash S_c} \sum_l x_{il} \bar{b}_{il} (s_{kt}(il) + r_{kt}(il))}{\sum_k \sum_t \sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} (s_{kt}(il) + r_{kt}(il))}$. From Lemma 2 we know that for any given k and t, the online algorithm ensures there is at most only one task that would exceed the resource capacity. Let $H(k, t)$ represent the last task executed on compute node k at time t. Then, $\frac{\tilde{P}_1^I}{P_1^I} \leq 1 + \frac{\sum_k \sum_t \sum_l x_{il} \bar{b}_{H(k,t)l} (s_{kt}(il) + r_{kt}(il))}{\sum_k \sum_t \sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} (s_{kt}(il) + r_{kt}(il))} \leq 1 + \max_{(k,t)}\{\max\{\frac{\sum_l x_{il} \bar{b}_{H(k,t)l} s_{kt}(il)}{\sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} s_{kt}(il)}, \frac{\sum_l x_{il} \bar{b}_{H(k,t)l} r_{kt}(il)}{\sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} r_{kt}(il)}\}\} \leq$

$1 + \max\{\frac{\bar{b}_{il,max}}{\bar{b}_{il,min}} \frac{s_{ik,max}}{s_{ik,min}}, \frac{\bar{b}_{il,max}}{\bar{b}_{il,min}} \frac{r_{i,max}}{r_{i,min}}\}$, where $\bar{b}_{il,max} = \max_{(i,l)}\{\bar{b}_{il}\}$ and $\bar{b}_{il,min} = \min_{(i,l)}\{\bar{b}_{il}\}$ are the maximum and minimum social welfare increment incurred by utilizing per unit of resource per time slot; $s_{ik,max} = \max_{(i,k)}\{s_{ik}\}$ and $s_{ik,min} = \min_{(i,k)}\{s_{ik}\}$; $r_{i,max} = \max_i\{r_i\}$ and $r_{i,min} = \min_i\{r_i\}$. Note that when the resource requirement of a task is far less than capacity, we have $\frac{s_{kt}(il)}{\sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} s_{kt}(il)} \to 0$, and $\frac{r_{kt}(il)}{\sum_{i \in S_c} \sum_l x_{il} \bar{b}_{il} r_{kt}(il)} \to 0$, then $\rho \to 1$. □

Lemma 4. *(Dual Feasibility Guarantee.) D_1^I is the objective value achieved by a dual feasible solution.*

Proof. Suppose that $(\{\hat{\mu}_i\}_i, \{\hat{\lambda}_{kt}\}_{k,t}, \{\hat{\varphi}_{kt}\}_{k,t})$ is a solution to problem D_1. We next determine the specific values of the above variables while satisfying all the constraints of problem D_1. Let $\hat{\lambda}_{kt} = \lambda_{kt}^{(I)}, \hat{\varphi}_{kt} = \varphi_{kt}^{(I)}, \forall k, t$. For each job i, since we select the schedule that achieves the maximum $F(il)$, combining with λ_{kt} and φ_{kt} are monotonically increasing functions, i.e., $\lambda_{kt}^i \leq \lambda_{kt}^I, \varphi_{kt}^i \leq \varphi_{kt}^I, \forall i, k, t$, then the constraint (6a) is satisfied. Therefore, D_1^I is the objective value achieved by the designed dual feasible solution $\tilde{\mu}_i = \mu_i, \hat{\lambda}_{kt} = \lambda_{kt}^{(I)}, \hat{\varphi}_{kt} = \varphi_{kt}^{(I)}, \forall i, k, t$. □