LobRA: Multi-tenant Fine-tuning over Heterogeneous Data

Sheng Lin[*][†]
Peking University
linsh@stu.pku.edu.cn

Fangcheng Fu[*][‡]
Shanghai Jiao Tong University
ccchengff@gmail.com

Haoyang Li[†]
Peking University
lihaoyang@stu.pku.edu.cn

Hao Ge[†]
Peking University
gehao@stu.pku.edu.cn

Xuanyu Wang[†]
Peking University
wxyz0001@pku.edu.cn

Jiawen Niu[†]
Peking University
niujiawen705@stu.pku.edu.cn

Yaofeng Tu
ZTE Corporation
tu.yaofeng@zte.com.cn

Bin Cui[†][§]
Peking University
bin.cui@pku.edu.cn

ABSTRACT

With the breakthrough of Transformer-based pre-trained models, the demand for fine-tuning (FT) to adapt the base pre-trained models to downstream applications continues to grow, so it is essential for service providers to reduce the cost of processing FT requests. Low-rank adaption (LoRA) is a widely used FT technique that only trains small-scale adapters and keeps the base model unaltered, conveying the possibility of processing multiple FT tasks by jointly training different LoRA adapters with a shared base model.

Nevertheless, through in-depth analysis, we reveal the efficiency of joint FT is dampened by two heterogeneity issues in the training data — the sequence length variation and skewness. To tackle these issues, we develop LobRA, a brand new framework that supports processing multiple FT tasks by jointly training LoRA adapters. Two innovative designs are introduced. Firstly, LobRA deploys the FT replicas (i.e., model replicas for FT) with heterogeneous resource usages and parallel configurations, matching the diverse workloads caused by the sequence length variation. Secondly, for each training step, LobRA takes account of the sequence length skewness and dispatches the training data among the heterogeneous FT replicas to achieve workload balance. We conduct experiments to assess the performance of LobRA, validating that it significantly reduces the GPU seconds required for joint FT by 45.03%-60.67%.

PVLDB Reference Format:
Sheng Lin, Fangcheng Fu, Haoyang Li, Hao Ge, Xuanyu Wang, Jiawen Niu, Yaofeng Tu, and Bin Cui. LobRA: Multi-tenant Fine-tuning over Heterogeneous Data. PVLDB, 18(8): 2616-2625, 2025.
doi:10.14778/3742728.3742752

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/PKU-DAIR/Hetu.

[*]Equal contribution.
[†]School of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
[‡]School of Artificial Intelligence, Shanghai Jiao Tong University
[§]Institute of Computational Social Science, Peking University (Qingdao)

Proceedings of the VLDB Endowment, Vol. 18, No. 8 ISSN 2150-8097.
doi:10.14778/3742728.3742752

1 INTRODUCTION

Transformer-based [60] pre-trained models, represented by Large Language Models (LLMs) [47, 48, 59, 66], have fueled an ever-increasing demand for their deployment in various applications such as chatbot assistants [51, 70], machine translation [17, 81], summarization [35, 69], database tuning [31, 78, 79], and more [12, 32, 68, 71]. Since pre-trained models are mostly trained with general data, it is a common practice to leverage a fine-tuning (FT) process [14, 50, 55], which further trains the pre-trained model with domain-specific data to adapt the model to the target applications.

Driven by the advancement of LLMs, many technology companies are enhancing their business strategies through the adoption of the Model as a Service (MaaS) paradigm [19]. To this end, providing FT services becomes essential. For example, a technology company may own a closed-source pre-trained model and allow users to upload their private or domain-specific datasets for FT [49]. For another example, cloud service providers would also offer FT services using popular open-source pre-trained models [10, 56]. As a result, given the diverse downstream applications, there would be many FT requests on the basis of the same pre-trained model, and consequently, it is of great value to reduce the cost associated with the FT requests over the same base model.

Meanwhile, FT services exhibit specific characteristics that are divergent from other kinds of services like model inference. For one thing, each user can submit multiple FT requests simultaneously, in order to build various domain-specific models with the same base model or to evaluate different dataset mixtures to see which gives the best performance [15, 61]. For another, compared to the inference scenario, FT requests arrive far less frequently (e.g., [2] reported an average of around 8.5 FT tasks per hour and some of them arrive simultaneously), and it takes significantly longer to process an FT request (tens of minutes to hours in practice), so both the request arrival rate and departure rate are much lower. Consequently, there would be a batch of FT requests that co-exist for long durations, and the batch does not change frequently. Given these characteristics, this work focuses on *how to reduce the cost of jointly executing a batch of FT tasks over the same base model*.

Owing to the astonishing model sizes and the scarcity of hardware accelerators (typically, GPUs), low-rank adaptation (LoRA) [21, 37] has become one of the most widely used and effective FT techniques. In essence, LoRA only trains small-scale adapters that consist of much fewer parameters than the base model, significantly reducing the FT cost. Since LoRA keeps the base model unaltered, it

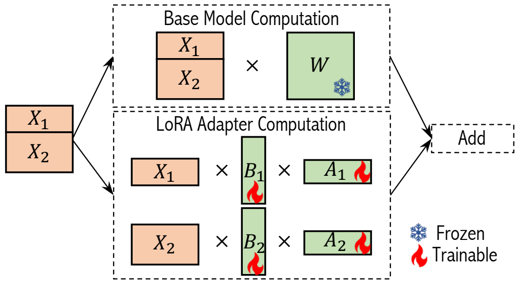


Figure 1: An illustration of the fusion of different LoRA adapters.

conveys the opportunity to share the same base model across multiple FT tasks rather than maintaining individual model replica(s) for each one. As shown in Figure 1, in each training step, we can fuse the input data from different tasks so that the computation of the base model can be fused into a batched operation whilst the computation of multiple LoRA adapters can be supported by customized operations. In fact, co-serving multiple LoRA adapters has been widely investigated for LLM inference [7, 52, 65, 80]. Inspired by this, this work focuses on how to efficiently carry out multiple FT tasks as a joint FT task by fusing multiple LoRA adapters.

Nevertheless, we experienced unsatisfactory efficiency when we naïvely leveraged the batch fusion idea to support joint FT. After an in-depth investigation, we found that this is because of two heterogeneity issues of the training data for joint FT.

The first is *the sequence length variation among tasks*. Specifically, since Transformer models take sequences as input, the training data (sequences) inevitably vary in length. As shown in Figure 2, the sequence length distributions are substantially divergent across FT tasks. This is reasonable as the training data of a few tasks (e.g., summarization) are usually much longer than the others (e.g., question answering). Meanwhile, since the memory consumption of FT is linear w.r.t. the lengths [8, 9, 73], it requires different numbers of GPUs to support the processing of sequences with different levels of lengths. Nevertheless, existing works overlook the discrepancy in resource demands, and straightforwardly deploy the FT replicas (i.e., model replicas for FT) in the same way for all training data, which conforms to the number of GPUs required for the highest sequence length. This would lead to efficiency degradation as the communication cost becomes higher for most training data.

To cope with this problem, this work proposes the idea of *heterogeneous FT replicas*, which deploys the FT replicas with varying resource usages and parallel configurations. Thus, we accelerate the processing of short sequences with FT replicas with low model parallel degrees, whilst avoiding out-of-memory errors by dispatching the long sequences to FT replicas with high model parallel degrees.

However, *the skewness in sequence lengths*, the second heterogeneity issue of FT data, raises another hurdle. As shown in Figure 2 again, most of the training data (sequences) are relatively short, which is a natural characteristic of human texts [11]. If we simply dispatch the training data to FT replicas according to their lengths, the workloads would be extremely imbalanced among the replicas. That is to say, FT replicas with low model parallel degrees must process much more training data, leading to heavier workloads compared to those with high parallel degrees. Since FT replicas must synchronize the parameters of LoRA adapters for every training step, it inevitably results in idle periods for some FT replicas.

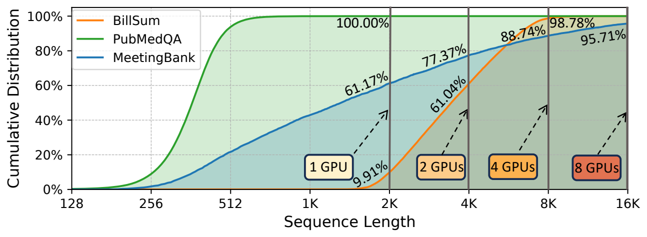


Figure 2: Cumulative distributions of sequence lengths of three FT datasets. The "n GPU(s)" indicates that we need n A100-40GB GPU(s) to process data with the corresponding sequence length without out-of-memory errors when fine-tuning the Llama2-7B model.

We address this issue by developing a *workload-balanced data dispatching* technique. This is inspired by the fact that short sequences can also be processed by FT replicas with high model parallel degrees. Thus, we adjust the data dispatching so that the workloads of FT replicas with low model parallel degrees can be migrated to those with high parallel degrees, achieving workload balance.

Putting them together, this work presents LobRA, a multi-tenant FT framework by manufacturing multiple LoRA adapters concurrently. To tackle the data heterogeneity issues, LobRA innovatively introduces the deployment of heterogeneous FT replicas and workload-balanced data dispatching to accelerate the joint FT process. In summary, this work makes the following contributions.

- We first anatomize different design choices of joint FT over heterogeneous training data, revealing the necessity of supporting heterogeneous model deployment and workload-balanced data dispatching. Based on the anatomy, we formulate a joint optimization problem to co-optimize these two factors.
- Subsequently, we propose a two-stage decomposition of the joint optimization problem for practical joint FT. On instantiation, the first-stage problem determines the heterogeneous model deployment plan that is optimal in expectation. Then, for each training step of the joint FT process, we derive the workload-balanced data dispatching plan by solving the second-stage problem.
- We evaluate the performance of LobRA by fine-tuning LLMs with up to 70B parameters and more than 10 tasks over 64 GPUs. Empirical results demonstrate that LobRA effectively reduces the GPU seconds required for joint FT by 45.03%-60.67%.

2 BACKGROUND AND RELATED WORKS

2.1 Fine-tuning over Variable-length Data

Fine-tuning based on LoRA. Fine-tuning (FT) is an essential process to adapt the pre-trained model to the target domain [14, 50, 55]. Low-rank adaption (LoRA) and its variants [21, 37] only fine-tune small-scale adapters rather than the base model itself, reducing resource demands substantially. For a model weight matrix $W \in \mathbb{R}^{in \times out}$, LoRA trains two low-rank matrices $A \in \mathbb{R}^{r \times out}, B \in \mathbb{R}^{in \times r}$ ($r \ll in, out$) and computes $XW + XBA$. By doing so, it approximates the change to the weight matrix as $\Delta W \approx BA$.

As LoRA does not alter the base model, previous works have proposed the co-serving of multiple LoRA adapters for inference [7, 52, 65, 80], and Miao et al. [39] further considered the scenario of inference and FT simultaneously. However, processing multiple FT tasks is not their primary focus. Ye et al. [67] and Zheng et al.

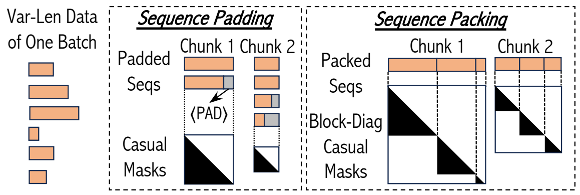


Figure 3: Illustration of applying sequence padding and packing to variable-length data of one batch. Sequence padding uses the special token ⟨PAD⟩ to ensure sequences within the same chunk are the same length. Sequence packing concatenates sequences together and uses the block-diagonal casual masks to avoid cross-contamination.

[76] considered fusing multiple LoRA adapters for joint FT. Nevertheless, they carry out the joint FT task by naïvely batching the input data, overlooking the data heterogeneity mentioned in §1. There are also many works that compose LoRA adapters for better performance [22, 57, 63, 64, 77], which is orthogonal to our work.

Processing Variable-length Data. In each training step, given a batch (a.k.a. mini-batch) of data, they are with diverse lengths due to the variable-length nature of sequences. Meanwhile, as GPU memory is limited, it is usually infeasible to process the batch at once. Thus, it is common to re-organize the batch into smaller chunks (a.k.a. micro-batches), process each sequentially, and accumulate model gradients computed from all chunks for model update.

Figure 3 shows two commonly used data re-organization techniques. Sequence padding sorts the data according to their lengths and takes those with similar lengths when crafting each chunk. Within each chunk, it adds special tokens (i.e. ⟨PAD⟩) to ensure a unique length. Sequence packing concatenates sequences for each chunk, eliminating the need for padding tokens. Meanwhile, it adjusts the casual mask to be block-diagonal to avoid cross-contamination among the sequences that are packed together [29].

In theory, packing provisions better training efficiency. However, Bai et al. [5] conducted experiments on FT tasks and observed that padding and packing exhibit comparable training efficiency, whilst training with padding usually achieves better model quality, which is because packing introduces biases to the contributions of different data. Thus, padding and packing have their pros and cons, and the choice between them is an open question. In this work, we assume padding is employed. Nevertheless, it is noteworthy that the proposed designs can also be applied when packing is employed.

Given the variable-length phenomenon, there are also several works investigating more variants of packing and padding [13, 30, 53]. However, none of these works have considered processing data of different lengths with different resource usages and parallel configurations. As a result, they are orthogonal to our work.

2.2 Parallel Configurations in Model Training

Model Parallel. There are two prevalent forms of model parallel, namely tensor parallel (TP) [28, 54] and pipeline parallel (PP) [20, 23, 41, 42]. Since TP and PP have different pros and cons, it is common to combine them for better efficiency, which is also known as hybrid model parallel [25, 40, 43, 58, 62, 72, 75]. In essence, model parallel distributes the model across GPUs to reduce the memory occupied by the model itself, sparing the space for intermediate results in training. Since the memory consumed by intermediate results is linear w.r.t. the summed lengths in each chunk [8, 9, 73], if we wish to support longer sequences, we usually need to increase the model parallel degree, yet at the price of larger communication overhead.

Table 1: Frequently used notation throughout this work.

N	The number of available GPUs.
R	The number of sequence length ranges, i.e., the training data of each batch are divided into R buckets.
S	The number of candidate parallel configurations.
$\mathcal{S}_i$	The i-th candidate parallel configuration.
n_i	The number of GPUs needed by $\mathcal{S}_i$ to deploy one FT replica.
p_i	The number of FT replicas deployed with $\mathcal{S}_i$.
r_i	The number of sequence length ranges that $\mathcal{S}_i$ supports, i.e., FT replicas with $\mathcal{S}_i$ support processing sequences in the first r_i ranges without out-of-memory errors.
$d_{i,j}$	The number of training data (sequences) in the j-th range that are assigned to the FT replicas with $\mathcal{S}_i$.

Model Replication (Data Parallel). Besides model parallel, a model can be replicated into multiple replicas. The input data are usually evenly dispatched to these replicas for concurrent processing (a.k.a. data parallel [6, 18, 26, 34, 44, 74]). To ensure the consistency of model parameters, it is necessary to synchronize the model gradients (or parameters) among the replicas for every training step, so balancing the workload across replicas is important.

Cost Model of Fine-tuning Replicas. In this work, we call each model replica a *fine-tuning (FT) replica*. Each replica is associated with a parallel configuration that describes how it is parallelized.

Given a training task, previous works generally develop cost modeling for running time and memory consumption, so that they can deduce how to parallelize the model [24, 25, 40, 62, 75]. However, they only support the same parallel configuration for all model replicas and do not consider fine-tuning with variable-length data.

Fortunately, we can borrow ideas from previous works to build the cost model for variable-length data, which involves profiling the time cost of essential modules and estimating the running time according to the critical path of the training workflow. In addition, since the memory consumption is linear w.r.t. the summed lengths in each chunk [8, 9, 73], we can easily profile the maximum supported sequence length for each kind of parallel configuration.

Throughout this work, we divide the sequence length into R non-overlapping ranges so that the variable-length data can be sorted into R buckets. We assume there are S candidate parallel configurations $\{\mathcal{S}_i\}_{i=1}^{S}$, where the i-th candidate requires n_i GPUs to deploy one FT replica and supports processing sequences in the first r_i ranges ($r_i \le R$) without out-of-memory errors. Subsequently, we denote $T(\{d_{\cdot,j}\}_{j=1}^{r_\cdot}; \mathcal{S}_\cdot)$ as the time cost[1] of an FT replica associated with $\mathcal{S}_\cdot$, where $d_{\cdot,j}$ represents how many sequences in the j-th range are dispatched to this FT replica. Due to the space constraint, we leave the details of our cost model in Appendix D [4].

3 DESIGN ANATOMY

This section anatomizes several design choices for joint FT, aiming to minimize the total GPU seconds needed to run one training step per task. To help readers better understand, we provide an

[1]Note that our work is applicable as long as the time cost function is linear w.r.t. $d_{\cdot,j}$.

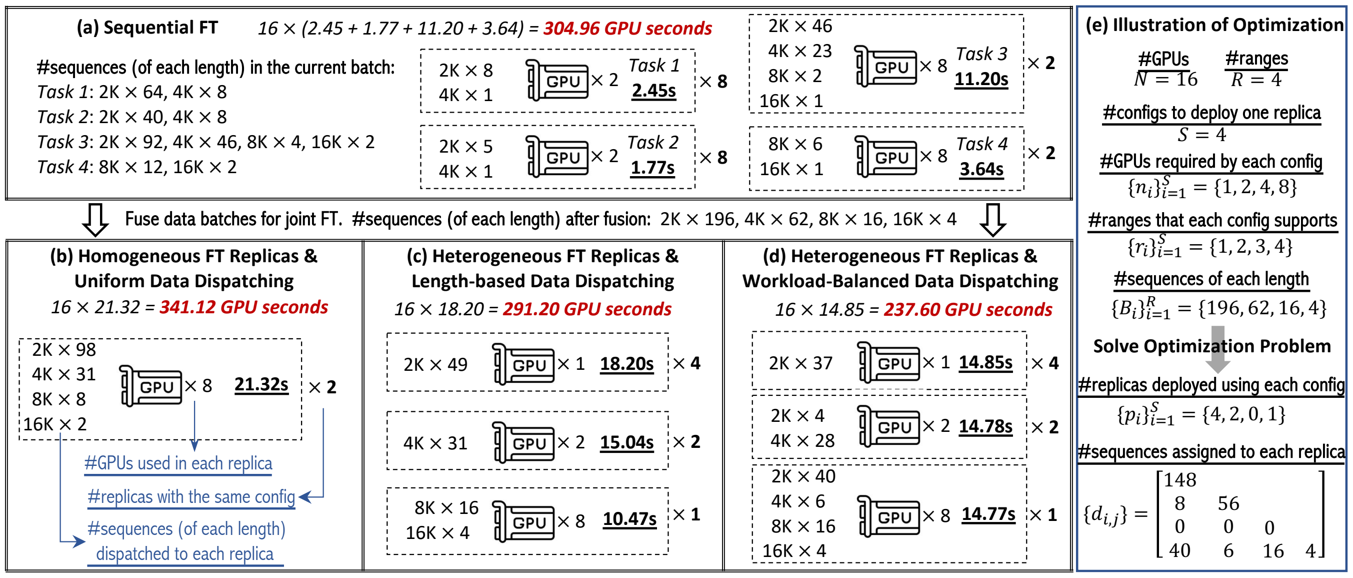

(a) Sequential FT
16 × (2.45 + 1.77 + 11.20 + 3.64) = 304.96 GPU seconds
#sequences (of each length) in the current batch:
Task 1: 2K × 64, 4K × 8
Task 2: 2K × 40, 4K × 8
Task 3: 2K × 92, 4K × 46, 8K × 4, 16K × 2
Task 4: 8K × 12, 16K × 2
2K × 8
4K × 1
GPU × 2
Task 1
2.45s
× 8
2K × 5
4K × 1
GPU × 2
Task 2
1.77s
× 8
2K × 46
4K × 23
8K × 2
16K × 1
GPU × 8
Task 3
11.20s
× 2
8K × 6
16K × 1
GPU × 8
Task 4
3.64s
× 2
Fuse data batches for joint FT. #sequences (of each length) after fusion: 2K × 196, 4K × 62, 8K × 16, 16K × 4
(b) Homogeneous FT Replicas & Uniform Data Dispatching
16 × 21.32 = 341.12 GPU seconds
2K × 98
4K × 31
8K × 8
16K × 2
GPU × 8
21.32s
× 2
#GPUs used in each replica
#replicas with the same config
#sequences (of each length) dispatched to each replica
(c) Heterogeneous FT Replicas & Length-based Data Dispatching
16 × 18.20 = 291.20 GPU seconds
2K × 49
GPU × 1
18.20s
× 4
4K × 31
GPU × 2
15.04s
× 2
8K × 16
16K × 4
GPU × 8
10.47s
× 1
(d) Heterogeneous FT Replicas & Workload-Balanced Data Dispatching
16 × 14.85 = 237.60 GPU seconds
2K × 37
GPU × 1
14.85s
× 4
2K × 4
4K × 28
GPU × 2
14.78s
× 2
2K × 40
4K × 6
8K × 16
16K × 4
GPU × 8
14.77s
× 1
(e) Illustration of Optimization
#GPUs N = 16
#ranges R = 4
#configs to deploy one replica S = 4
#GPUs required by each config $\{n_i\}_{i=1}^S = \{1, 2, 4, 8\}$
#ranges that each config supports $\{r_i\}_{i=1}^S = \{1, 2, 3, 4\}$
#sequences of each length $\{B_i\}_{i=1}^R = \{196, 62, 16, 4\}$
Solve Optimization Problem
#replicas deployed using each config $\{p_i\}_{i=1}^S = \{4, 2, 0, 1\}$
#sequences assigned to each replica
$\{d_{i,j}\} = \begin{bmatrix} 148 & & & \\ 8 & 56 & & \\ 0 & 0 & 0 & \\ 40 & 6 & 16 & 4 \end{bmatrix}$

Figure 4: An example of 4 FT tasks with four different approaches, where (a) denotes fine-tuning the 4 tasks sequentially, whilst (b)-(d) present three different designs discussed in §3. We focus on the total GPU seconds required to run one training step for each task. (e) illustrates the inputs and decision variables of Equation (1) based on (d).

example of four FT tasks in Figure 4. Figure 4(a) depicts a sequential FT process that executes the four FT tasks one by one, whilst Figure 4(b)-(d) illustrates three different design choices of joint FT.

Naïve Design: Homogeneous FT Replicas and Uniform Data Dispatching. As shown in Figure 4(b), an intuitive solution for joint FT is to instantiate the FT replicas with the same parallel configuration, fuse the data batches from all tasks together, and evenly dispatch them to these homogeneous FT replicas. By doing so, it is obvious that the training workloads are balanced across the replicas. However, the end-to-end performance of such an approach is unsatisfactory due to *the sequence length variation among tasks*.

Specifically, unlike well-structured tabular datasets, sequences inherently have diverse lengths. Moreover, the sequence lengths of different tasks are significantly divergent. Such a variation results in different memory consumption, thereby calling for different trade-offs between memory reduction and training efficiency. As a result, for tasks with long sequences, since their memory consumption is high, it is common to increase the model parallel degree (which requires more GPUs for one replica) to avoid out-of-memory errors, whilst incurring higher communication costs. In contrast, for tasks with short sequences, we can leverage a lower model parallel degree (which requires fewer GPUs for one replica) for better efficiency.

Since the naïve approach uses the same parallel configuration for all replicas, the configuration must be able to accommodate the memory consumption of the *longest* sequences to avoid out-of-memory errors. For instance, each FT replica in Figure 4(b) occupies 8 GPUs, whilst only 2 GPUs are necessary if we consider the first two FT tasks individually in Figure 4(a). Thus, their processing time would be significantly prolonged, making this approach ineffective.

Better Design: Heterogeneous FT Replicas and Length-based Data Dispatching. Based on the discussion above, a better design is to leverage heterogeneous FT replicas for variable-length sequences. As exemplified in Figure 4(c), we can instantiate the replicas with non-unique parallel configurations, classify the training data into buckets according to their lengths, and dispatch each bucket to the most suitable replica(s). By this means, from the perspective of each sequence, it can be processed by the most efficient configuration. Nevertheless, such a solution suffers from the workload imbalance problem caused by *the skewness in sequence lengths*.

To elaborate, in real-world corpora, most sequences are relatively short, and there are very few sequences that are significantly longer than others. Figure 2 presents the sequence length distributions of several FT datasets — more than half of the sequences are shorter than 2K, whilst only a few are longer than 8K. In fact, this is reasonable in human texts and similar observations have also been reported on extremely large-scale pre-training corpora [11, 16].

Back to the discussion about Figure 4(c), given the skewness issue, the replicas with low model parallel degrees would receive a large portion of training data, whilst the other replicas would receive very few. As synchronization is needed for model update, replicas must idly wait for the slowest one(s). Worse still, a higher model parallel degree requires more GPUs for each replica, so there is a huge waste of computational resources. For the example in Figure 4(c), the 8 GPUs in the third replica sit idle for approximately 42% of the time during the joint FT process (10.47 vs. 18.20 seconds).

Optimized Design: Heterogeneous FT Replicas and Workload-Balanced Data Dispatching. Inspired by this, we propose to adopt the combination of heterogeneous FT replicas and workload-balanced data dispatching. The rationale is that replicas with high model parallel degrees can also process short sequences, so we can dispatch short sequences to more kinds of replicas. As depicted in Figure 4(d), by doing so, we can strike a good balance among the heterogeneous FT replicas and improve end-to-end efficiency.

Given a batch of training data, let B_j be the number of sequences fallen into the j-th bucket ($j \in [1, R]$). There are two key factors.

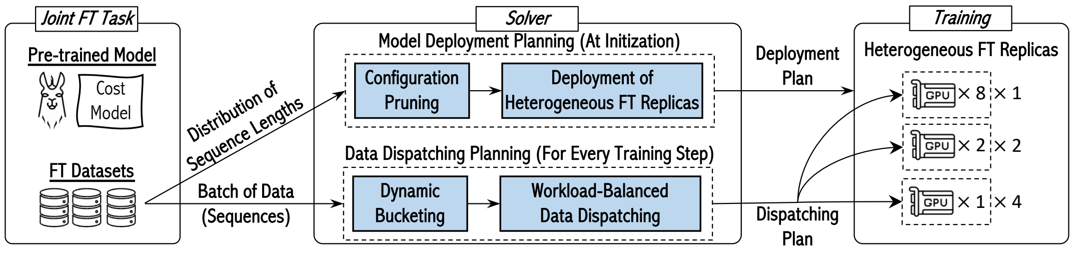

Joint FT Task
Pre-trained Model
Cost Model
FT Datasets
Distribution of Sequence Lengths
Batch of Data (Sequences)
Solver
Model Deployment Planning (At Initization)
Configuration Pruning
Deployment of Heterogeneous FT Replicas
Data Dispatching Planning (For Every Training Step)
Dynamic Bucketing
Workload-Balanced Data Dispatching
Deployment Plan
Dispatching Plan
Training
Heterogeneous FT Replicas
GPU × 8 × 1
GPU × 2 × 2
GPU × 1 × 4

Figure 5: Overview of LobRA. To start the joint FT task, given the base model and the sequence length distribution of the FT datasets, LobRA determines the deployment plan of FT replicas that minimizes the running time in expectation. During the FT process, for each training step, LobRA analyzes how the corresponding batch of data should be dispatched among the FT replicas in order to achieve workload balance.

Firstly, to deploy the heterogeneous FT replicas, we need to select the suitable parallel configurations and the number of replicas for each selected configuration. Secondly, for each batch, we need to dispatch the training data to minimize the running time of the slowest replica. We formulate the following optimization problem:

$$\begin{aligned} \underset{p_i, d_{i,j} \in \mathbb{N}_0 \text{ for } i\in[1,S], j\in[1,r_i]}{\arg\min} \quad & \max_{i\in[1,S]} T\left(\left\{\lceil d_{i,j}/p_i \rceil\right\}_{j=1}^{r_i}; \mathcal{S}_i\right) \\ \text{s.t.} \quad & \sum_{i\in\{i|r_i\geq j\}} d_{i,j} = B_j \text{ for } \forall j \in [1,R] \\ & d_{i,j} \leq B_j \times p_i \text{ for } \forall i \in [1,S], j\in[1,r_i] \\ & \sum_{i=1}^{S} p_i \times n_i \leq N \end{aligned} \tag{1}$$

where p_i represents the number of FT replicas that are deployed with $\mathcal{S}_i$ (i.e., the i-th parallel configuration, and $p_i = 0$ indicates $\mathcal{S}_i$ is not selected for deployment), and $d_{i,j}$ represents how many sequences in the j-th bucket are assigned to the p_i replica(s) with $\mathcal{S}_i$ for processing. $T(\cdot;\cdot)$ denotes the running time of the p_i replica(s) with $\mathcal{S}_i$ given the dedicated sequences. The first constraint ensures all sequences are processed. The second constraint requires no sequences will be assigned if a configuration is not selected (i.e., $d_{i,j} = 0$ as long as $p_i = 0$). The third constraint ensures the FT replicas can be instantiated using the available GPUs.

We illustrate an example of the optimization problem with Figure 4(d). There are $N = 16$ GPUs and we divide the sequences into $R = 4$ buckets based on their lengths. To deploy one replica, there are $S = 4$ candidate configurations, requiring $\{n_i\}_{i=1}^{S} = \{1,2,4,8\}$ GPU(s) and supporting sequences in the first $\{r_i\}_{i=1}^{S} = \{1,2,3,4\}$ bucket(s), respectively. In the current fused batch, the numbers of sequences fallen into the buckets are $\{B_j\}_{j=1}^{R} = \{196,62,16,4\}$. Then, by solving the optimization problem, the numbers of deployed replicas with the configurations are $\{p_i\}_{i=1}^{S} = \{4,2,0,1\}$, and the data dispatching is shown in Figure 4(e).

4 LOBRA

4.1 Overview

Although Equation (1) co-optimizes the model deployment and data dispatching, solving it for every training step is impractical. For one thing, across different steps, the best model deployment plan may vary, which implies that we would need to reconfigure the model partitioning. Given the substantial model sizes, this is prohibitively time-consuming. For another, solving the problem takes longer than the training of one step (as evaluated in §5.3), making it infeasible to solve it for every step. To tackle these obstacles, we propose a two-stage decomposition as depicted in Figure 5.

- The first stage (§4.2) produces the deployment plan of heterogeneous FT replicas. It is only done once at the initialization of the joint FT task, eliminating the need for reconfiguration of model deployment as well as the expensive solving cost for every step.
- Given the deployed FT replicas, the second stage (§4.3) only deduces the optimal data dispatching, which is fast and can be invoked for every step. This enables us to dynamically adapt the data dispatching to the randomly drawn batches during training.

4.2 Deployment of Heterogeneous FT Replicas

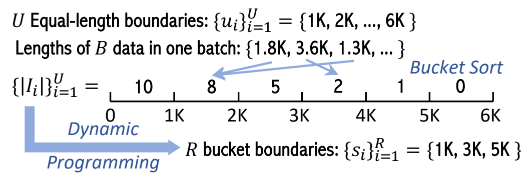
Problem Formulation. To instantiate the joint FT task, it is essential to determine how to deploy the heterogeneous FT replicas. However, according to Equation (1), the optimal deployment plan is relevant to how the sequences are distributed across the buckets (i.e., $\{B_j\}_{j=1}^{R}$). To address the discrepancy, we manage to utilize the overall distribution of sequence lengths of the FT datasets. In particular, we re-write Equation (1) as follows:

$$\begin{aligned} \underset{p_i, d_{i,j} \in \mathbb{N}_0 \text{ for } i\in[1,S], j\in[1,r_i]}{\arg\min} \quad & \max_{i\in[1,S]} T\left(\left\{\lceil d_{i,j}/p_i \rceil\right\}_{j=1}^{r_i}; \mathcal{S}_i\right) \\ \text{s.t.} \quad & \sum_{i\in\{i|r_i\geq j\}} d_{i,j} \geq B \times f_j \text{ for } \forall j \in [1,R] \\ & d_{i,j} \leq B_j \times p_i \text{ for } \forall i \in [1,S], j\in[1,r_i] \\ & \sum_{i=1}^{S} p_i \times n_i \leq N \end{aligned} \tag{2}$$

where B denotes the batch size of the joint FT task, and f_j denotes the percentage of sequences fallen into the j-th bucket. Solving Equation (2) returns the model deployment plan (p_i) and data dispatching plan ($d_{i,j}$) that is optimal in expectation. Then, LobRA instantiates the FT replicas based on the model deployment plan (whilst the data dispatching plan will be omitted).

Note that since f_j's are not integers, we have inequality constraints on $d_{i,j}$ in Equation (2). Formally speaking, the solutions to Equation (2) may have $\sum_j \sum_i d_{i,j} = B + R$, whilst the solutions to Equation (1) follow $\sum_j \sum_i d_{i,j} = B$, implying a difference. Nevertheless, after solving Equation (2), only p_i's are used for model deployment, whilst $d_{i,j}$'s are omitted. Combining with the fact that $B \gg R$, the difference between Equation (1) and Equation (2) is minor, and it does not affect the effectiveness of our work.

Configuration Pruning. Due to the term $d_{i,j}/p_i$, Equation (2) is a mixed-integer non-linear programming (MINLP) problem, which is



U Equal-length boundaries: $\{u_i\}_{i=1}^{U} = \{1K, 2K, ..., 6K\}$
Lengths of B data in one batch: $\{1.8K, 3.6K, 1.3K, ...\}$
Bucket Sort
$\{|I_i|\}_{i=1}^{U} =$
10
8
5
2
1
0
0
1K
2K
3K
4K
5K
6K
Dynamic
Programming
R bucket boundaries: $\{s_i\}_{i=1}^{R} = \{1K, 3K, 5K\}$

Figure 6: Illustration of dynamic bucketing.

a notorious combinatorial optimization problem. Although libraries like SCIP [3] support solving MINLP problems, it is very time-consuming, especially when there are many decision variables (i.e., $d_{i,j}$ and p_i). Specifically, the number of decision variables of Equation (2) is $S + \sum_{i=1}^{S} r_i \le S + S \times R$. Thus, the solving cost of Equation (2) is highly related to S, as R is not large in practice (e.g., we set $R = 16$ in our experiments). To speed up the solving process, we introduce two heuristics to filter the candidate configurations. Due to the space constraint, we briefly introduce the rationale below and refer interested readers to Appendix A [4] for more details.

- Firstly, we only consider a small set of configurations (rather than covering all possible ones) for problem-solving. In essence, if two configurations consume the same number of GPUs (i.e., same n) and one of them is consistently less efficient than the other, then it will never be selected for deployment. Besides, since the model architecture of each pre-trained model is fixed, we conduct offline benchmarking and propose the candidates in advance, without affecting the online problem-solving.
- Secondly, although there is no closed-form solution for MINLP problems, it is possible to heuristically estimate the lower bound of training efficiency given a deployment plan (i.e., $\{p_i\}_{i=1}^{S}$). The reason is that, given the aforementioned benchmarking, we can estimate how the running time of two FT replicas would change after we migrate some training data between them. Thus, given a deployment plan, we treat the length-based dispatching as a starting point and estimate the lower bound of workload-balanced dispatching. Then we filter out deployment plans that are predicted to be inefficient, which shrinks the solution space.

Based on these two heuristics, Equation (2) can be solved efficiently and accurately. For one thing, when the number of GPUs is relatively small (e.g., 16-32 GPUs), the heuristics do not affect the achieved solutions (i.e., the achieved solutions are the same as those achieved without pruning), yet accelerate the solving process substantially. For another, when there are more GPUs, solving without pruning fails to finish within an hour, whilst solving with pruning only takes several minutes. Since Equation (2) is only solved at initialization, the solving time is worthwhile given the speedup in the joint FT process. In Appendix A and Appendix B.2 [4], we have provided more details and empirical results.

4.3 Workload-Balanced Data Dispatching

Problem Formulation. After the heterogeneous FT replicas are deployed, each training step of the joint FT process involves randomly drawing a batch of training data, and feeding them to the FT replicas. Due to the randomness in batch sampling, the number of sequences that fall into each bucket may be divergent across different steps. Denote p_i^* as the optimal solution achieved by Equation (2), which describes the model deployment plan of the heterogeneous FT replicas, we formulate the following optimization problem to minimize the time cost of each training step.

$$\begin{aligned} \underset{d_{i,j}\in\mathbb{N}_0 \text{ for } i\in[1,S], j\in[1,r_i]}{\arg\min} \quad & \max_{i\in[1,S]} T\left(\{\lceil d_{i,j}/p_i^*\rceil\}_{j=1}^{r_i}; \mathcal{S}_i\right) \\ \text{s.t.} \quad & \sum_{i\in\{i|r_i\ge j\}} d_{i,j} = B_j \text{ for } \forall j\in[1,R] \\ & d_{i,j} \le B_j \times p_i^* \text{ for } \forall i\in[1,S], j\in[1,r_i] \end{aligned} \tag{3}$$

Since $\{p_i^*\}_{i=1}^{S}$ are constants rather than decision variables, the optimization problem in Equation (3) is an integer linear programming (ILP) problem, which can be efficiently solved via existing libraries like PuLP [1] and SCIP [3]. The number of decision variables in Equation (3) is $\sum_{i=1}^{S} r_i \le S \times R$. However, since we can remove the configurations that are not selected for deployment (i.e., whose p_i^* is zero), there are very few decision variables in practice. (For instance, there are only 3-5 selected configurations in our evaluation.) Thus, as we will evaluate in §5.3, the solving process is fast and can be fully overlapped by the training of previous step(s).

Dynamic Bucketing. Till now, we assume that the boundaries of buckets, denoted as $\{s_i\}_{i=1}^{R}$, are pre-defined and fixed throughout the joint FT process. However, due to the randomness in batch sampling, the optimal boundaries vary across different training steps — as each sequence must be padded to the closest boundary, using fixed boundaries would lead to undesirable padding. To address this problem, we develop a dynamic bucketing approach to facilitate the adaption to the sequence length distribution during training.

As depicted in Figure 6, our approach starts from U pre-defined boundaries $\{u_i\}_{i=1}^{U}$, which partition the range of sequence length into U intervals (in practice, we consider equal-length division $\{256, 512, \cdots\}$). Given a batch of B sequences, our approach bucket sorts them by lengths and determines R boundaries ($R \le U$) to form R buckets that minimize the padding via dynamic programming.

Denote $\mathcal{I}_i$ as the set of indices of sequences fallen into the i-th interval, and $\text{State}_{i,j}$ as the minimized padding when bucketing the first i intervals into j buckets. We have the following initial state and state transition expressions for dynamic programming:

$$\begin{aligned} &\text{State}_{0,j} = 0,\ \forall j \in [0,R], \qquad \text{State}_{i,0} = +\infty,\ \forall i \in [1,U], \\ &\text{State}_{i+1,j+1} = \min_{i'\in[0,i]} \left\{ \text{State}_{i',j} + \sum_{i''=i'+1}^{i} \left(|\mathcal{I}_{i''}| \times (u_{i+1} - u_{i''})\right) \right\} \end{aligned} \tag{4}$$

By computing $\text{State}_{U,R}$, we achieve the optimal boundaries for a given batch of data[2]. The time complexity of the dynamic programming is merely $O(B + RU^2)$, introducing negligible overhead.[3]

Note that the bucketing approach is also applied when solving the deployment of FT replicas in Equation (2). In particular, at the initialization of the joint FT task, we randomly sample a large number ($100 \times B$ by default) of training data and perform the bucketing to determine the boundaries for solving Equation (2).

5 EXPERIMENTS

5.1 Implementation and Experimental Setup

We implement LobRA on top of Hetu [33, 38], a distributed deep learning framework for large-scale models. We utilize SCIP [3] to

[2]The total number of padding tokens is $\text{State}_{U,R} + \sum_{i=1}^{U} \sum_{k\in\mathcal{I}_i} (u_i - s_k)$, where the second term denotes the padding needed inside each interval, which is a constant.
[3]We default R as 16 (sensitivity experiment provided in Appendix B.2 [4]) and ignore empty intervals in our implementation, so the term RU^2 is small in practice.

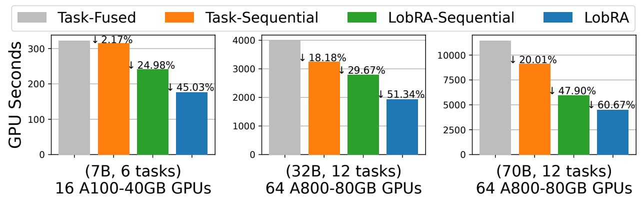


Figure 7: End-to-end evaluation.

Table 2: Parallel configurations used by Task-Fused and LobRA in end-to-end evaluation. (Those used by Task-Sequential and LobRA-Sequential are provided in Appendix B.3 [4].) Each $\langle \alpha, \beta \rangle \times \gamma$ indicates there are γ FT replica(s) with a TP degree of α and a PP degree of β.

	Task-Fused	LobRA
7B	⟨8,1⟩×2	⟨1,1⟩×6, ⟨2,1⟩×1, ⟨8,1⟩×1
32B	⟨8,1⟩×8	⟨1,6⟩×4, ⟨2,2⟩×4, ⟨4,1⟩×2, ⟨8,1⟩×2
70B	⟨16,1⟩×4	⟨2,4⟩×4, ⟨4,2⟩×1, ⟨8,1⟩×1, ⟨16,1⟩×1

solve the MINLP and ILP problems. Our framework incorporates libraries like FlashAttention [8, 9] for efficient computation of LLMs and NCCL [46] for communication. Even when training with homogeneous FT replicas, LobRA achieves state-of-the-art FT efficiency. More details are provided in Appendix C [4].

Although LobRA focuses on co-optimizing a given batch of FT tasks, it also supports dynamic batches. In practice, when the batch of FT tasks changes (e.g., some tasks exit earlier than others or new FT requests arrive), we simply re-generate a new model deployment plan with the updated sequence length distribution. If the new plan differs from the current one, we save checkpoints for LoRA adapters and restart the joint task to meet the new plan. (We do not need to save checkpoints for the base model.) The overhead of deployment adjustment is consistently less than 3 minutes in practice, which is worthwhile as the FT tasks would take hours to finish.

Experimental Environments. We use two environments for evaluation. The first consists of 2 servers equipped with 8 A100-40GB GPUs (16 GPUs in total). The GPUs within the same server are connected via 600GB/s NVLink and the servers are connected via 100GB/s InfiniBand. The second consists of 8 servers equipped with 8 A800-80G GPUs (64 GPUs in total). The intra- and inter-server communication bandwidths are 400GB/s and 200GB/s, respectively.

Competitors. The primary goal of our evaluation is to evaluate the effectiveness of heterogeneous FT replicas and workload-balanced data dispatching. Since none of the existing works have supported such designs for joint FT, we consider two baselines, termed Task-Fused and Task-Sequential, that employ homogeneous FT replicas and uniform dispatching. The first naïvely fuses the FT tasks (i.e., Figure 4(b)) whilst the second executes the FT tasks sequentially (i.e., Figure 4(a)). We tune their deployment plan to achieve the best efficiency. To further evaluate the effectiveness of batch co-optimization, we also consider a variant of LobRA, termed LobRA-Sequential, which also runs the FT tasks sequentially but optimizes each FT task by employing heterogeneous FT replicas and workload-balanced data dispatching.

Workloads. We consider three popular LLMs, which are Llama2-7B, Qwen2.5-32B, and Llama2-70B. We fine-tune the 7B model with the first environment and the other two models with the second environment. We consider 12 FT datasets, regarding each as one FT

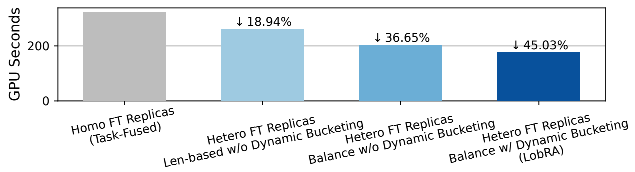


Figure 8: Ablation Studies (7B model, 16 A100-40GB GPUs).

task. The detailed descriptions of the FT datasets and the batch size settings are provided in Appendix B.1 [4]. By default, we consider 6 tasks for the 7B model and 12 tasks for the other two models. We use the Adam optimizer [27, 36] for all experiments.

Protocols. Since our goal is to improve the efficiency of joint FT, we focus on the GPU seconds required to train one step for all involved tasks. For all experiments, we report the mean of 100 training steps, and the standard deviation is within 10%.

5.2 End-to-End Evaluation

We first assess the end-to-end joint FT efficiency of LobRA. The results are shown in Figure 7, and we provide the parallel configurations used for deployment in Table 2.

In general, LobRA outperforms Task-Fused significantly for all three experiments, reducing 45.03%-60.67% GPU seconds. LobRA achieves greater improvement when fine-tuning larger models over 64 GPUs. This is reasonable since LobRA is able to explore a larger space of deployment plans — LobRA deploys 11 and 7 heterogeneous FT replicas for the 32B and 70B models, respectively, whilst Task-Fused can only deploy 8 and 4 homogeneous FT replicas. This allows more short sequences to get accelerated. Moreover, owing to the substantial size of the 70B model, Task-Fused must utilize a TP degree of 16 to accommodate the high memory consumption, which is extremely inefficient due to the slow communication across servers. In contrast, most of the FT replicas in LobRA do not need to span across servers, delivering much better efficiency. Thus, LobRA achieves the highest performance gain on the 70B model.

Task-Sequential performs better than Task-Fused. As analyzed in §3, Task-Fused must employ a high model parallel degree for all training data. Task-Sequential, however, can employ a lower model parallel degree for datasets that contain only short sequences (e.g., question-answering datasets), so the total GPU seconds are shorter. The smaller performance gap in the 7B model is primarily because the smaller GPU memory capacity (40GB) restricts Task-Sequential from choosing more efficient parallel configurations for most tasks. Nevertheless, Task-Sequential is still less efficient than LobRA. Although LobRA-Sequential outperforms Task-Sequential, LobRA still achieves 1.32-1.44× of speedup compared to LobRA-Sequential. This is not surprising for two reasons. Firstly, the data heterogeneity is milder within certain tasks, so the gain of using heterogeneous FT replicas is lower. Secondly, the batch size for each FT task is usually small, making it difficult to achieve workload balance by routing the data across the replicas. In practice, we find that some tasks would even experience an efficiency drop when employing LobRA-Sequential (detailed in Appendix B.2 [4]). Last but not least, it is noteworthy that Task-Sequential and LobRA-Sequential necessitate running the FT tasks individually, either requiring extra GPUs or unfairly forcing some tasks to queue for a long time. Consequently, LobRA is more efficient and suitable for processing FT requests.

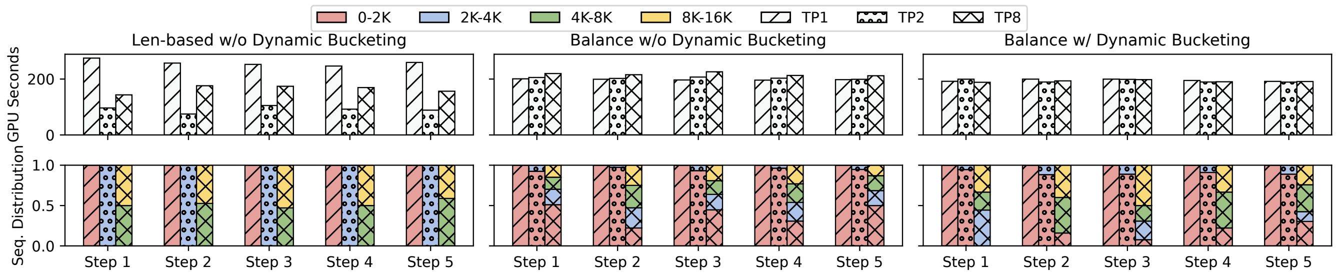


Figure 9: Case studies (7B model, 16 A100-40GB GPUs). Each bar represents one kind of FT replica(s). Top: The per-step time of each kind of FT replica(s). Bottom: The organization of dispatched data in terms of their sequence lengths for each kind of FT replica(s).

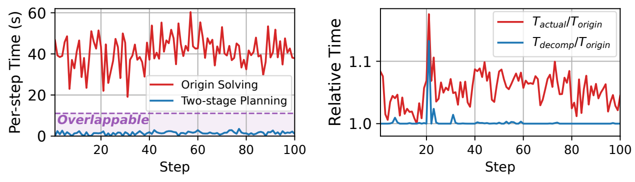


Figure 10: Left: Time cost of solving the original problem (Equation (1)) vs. the two-stage planning (dynamic bucketing + solving Equation (3)). The horizontal dashed line indicates the average per-step time. Right: Comparison of estimated running time for solving the original problem (T_{origin}), the two-stage decomposition (T_{decomp}), and the actual running time (T_{actual}).

5.3 Effectiveness of the Proposed Techniques

Ablation and Case Studies. We assess the effectiveness of heterogeneous model deployment and workload-balanced data dispatching, respectively. The results are shown in Figure 8 and Figure 9.

When heterogeneous FT replicas are deployed and the data are directly dispatched via their lengths (i.e., Figure 4(c)), we can reduce the GPU seconds by 18.94% compared to the naïvely fused approach. As shown in Table 2, the naïve approach needs to employ a high model parallel degree (resulting in 2 replicas with ⟨TP=8, PP=1⟩) to support long sequences, yet it is unsuitable for short sequences. In contrast, with the heterogeneous model deployment, we can deploy 7 replicas with low model parallel degrees (6 replicas with ⟨TP=1, PP=1⟩ and 1 replica with ⟨TP=2, PP=1⟩).

By further enabling the workload-balanced data dispatching and dynamic bucketing, the reduction in GPU seconds improves to 36.65% and 45.03%. Figure 9 visualizes the data dispatching and workload balance. When the data are simply assigned via their lengths (left-most of Figure 9), we observe severe imbalance across the replicas due to the skewness issue. Workload-balanced data dispatching (middle of Figure 9) strikes a good balance among the replicas by routing the data. With dynamic bucketing (right-most of Figure 9), the running time can be further reduced, particularly for replicas with a higher model parallel degree, since dynamic bucketing is more effective in reducing the padding of longer sequences.

Effectiveness of Planning. As discussed in §4.1, we employ a two-stage decomposition to Equation (1). One reason is that its solving is time-consuming. To examine this, we measure its solving time and compare it with the per-step running time. As shown on the left side of Figure 10, solving Equation (1) is slower than one training step, even if we have applied the configuration pruning heuristics introduced in §4.2 (the effectiveness of the two heuristics is evaluated in Appendix B.2 [4]). On the contrary, with the two-stage decomposition, we only need to perform the dynamic bucketing and solve Equation (3) for each step, which is extremely efficient and can be fully overlapped by the training of previous step(s).

In addition, we assess how the two-stage decomposition impacts the efficiency of the achieved solutions. To do so, we record the estimated running time given by solving the original problem, the estimated running time after the two-stage decomposition, and the actual running time in practice, denoted as T_{origin}, T_{decomp}, T_{actual}, respectively. The right side of Figure 10 presents T_{decomp}/T_{origin} and T_{actual}/T_{origin} across 100 steps. Overall, T_{decomp} and T_{origin} are extremely close in most steps. In occasional steps, the sampled batch does not contain long sequences, so solving the original problem produces a better deployment plan, leading to several spikes. However, the performance gap is still small (within 15%). These results verify the robustness and effectiveness of our two-stage decomposition. Besides, T_{actual} is also close to T_{decomp} (within 10%) across all steps, demonstrating the accuracy of our cost model.

More Experiments. We have conducted more experiments, including the scalability w.r.t. the number of GPUs and tasks, the sensitivity w.r.t. the sequence bucketing, and the effectiveness of our configuration pruning. Due to the space constraint, we leave more results and details of our experiments in Appendix B [4].

6 CONCLUSION

This work studies the processing of multiple FT requests by jointly training multiple LoRA adapters with the same base model. We conducted an anatomy on the efficiency of joint FT, showing that two data heterogeneity issues, i.e., the sequence length variation and skewness, pose significant challenges. Then, we developed a brand new joint-FT framework, namely LobRA, with two innovative designs. The first is the deployment of heterogeneous FT replicas, which addresses the sequence length variation and accelerate the processing of short sequences. The second is workload-balanced data dispatching, which eliminates the idle period of some FT replicas caused by skewness. Empirical results show that LobRA greatly reduces the GPU seconds required for joint FT by 45.03%-60.67%.

ACKNOWLEDGMENTS

This work is supported by National Science and Technology Major Project (2022ZD0116315), National Natural Science Foundation of China (U23B2048, 62402011), Beijing Municipal Science and Technology Project (Z231100010323002), ZTE-PKU joint program, and High-performance Computing Platform of Peking University. Fangcheng Fu and Bin Cui are the corresponding authors.

REFERENCES

[1] 2009. Optimization with PuLP. https://coin-or.github.io/pulp/.

[2] 2023. The Ymir Proejct: Dataset and Workload. https://sites.google.com/view/ymir-project#h.dw77b5uw44tb.

[3] 2024. SCIP: Solving Constraint Integer Programs. https://www.scipopt.org/.

[4] 2025. Full Version (with Appendix) of LobRA. https://github.com/ccchengff/LobRA/blob/main/LobRA_Full_Version_with_Appendix.pdf.

[5] Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. 2024. LongAlign: A Recipe for Long Context Alignment of Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024 (EMNLP Findings, 2024)*. 1376–1395.

[6] Weibo Cai, Shulin Yang, Gang Sun, Qiming Zhang, and Hongfang Yu. 2023. Adaptive load balancing for parameter servers in distributed machine learning over heterogeneous networks. *ZTE Communications* 21, 1 (2023), 72.

[7] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2024. Punica: Multi-Tenant LoRA Serving. In *Proceedings of Machine Learning and Systems 2024 (MLSys 2024)*.

[8] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations 2024 (ICLR 2024)*.

[9] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Annual Conference on Neural Information Processing Systems 2022 (NeurIPS 2022)*.

[10] DataBricks. 2025. Documents for Foundation Model Fine-tuning. https://docs.databricks.com/aws/en/large-language-models/foundation-model-training.

[11] Harm de Vries. 2023. In the long (context) run. https://www.harmdevries.com/post/context-length/.

[12] Letian Deng and Yanru Zhao. 2023. Deep learning-based semantic feature extraction: A literature review and future directions. *ZTE communications* 21, 2 (2023), 11.

[13] Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. 2024. Fewer Truncations Improve Language Modeling. In *International Conference on Machine Learning 2024 (ICML 2024)*.

[14] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. 2020. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. *CoRR* abs/2002.06305 (2020).

[15] Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. How Abilities in Large Language Models are Affected by Supervised Fine-tuning Data Composition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*. 177–198.

[16] Fei Du, Xin-Jian Ma, Jing-Ru Yang, Yi Liu, Chao-Ran Luo, Xue-Bin Wang, Hai-Ou Jiang, and Xiang Jing. 2024. A Survey of LLM Datasets: From Autoregressive Model to AI Chatbot. *J. Comput. Sci. Technol.* (2024).

[17] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding Back-Translation at Scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*. 489–500.

[18] Shaoduo Gan, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Jiawei Jiang, Binhang Yuan, Sen Yang, Ji Liu, and Ce Zhang. 2021. BAGUA: Scaling up Distributed Learning with System Relaxations. *Proc. VLDB Endow.* 15, 4 (2021), 804–813.

[19] Wensheng Gan, Shicheng Wan, and Philip S. Yu. 2023. Model-as-a-Service (MaaS): A Survey. *CoRR* abs/2311.05804 (2023).

[20] Lei Guan, Dong-Sheng Li, Jiye Liang, Wen-Jian Wang, Ke-shi Ge, and Xicheng Lu. 2024. Advances of Pipeline Model Parallelism for Deep Learning Training: An Overview. *J. Comput. Sci. Technol.* (2024).

[21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations 2022 (ICLR 2022)*.

[22] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition. In *First Conference on Language Modeling (COLM 2024)*.

[23] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Annual Conference on Neural Information Processing Systems 2019 (NeurIPS 2019)*. 103–112.

[24] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. 2023. Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP 2023)*. 382–395.

[25] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. In *Proceedings of Machine Learning and Systems 2019 (MLSys 2019)*.

[26] Youhe Jiang, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, and Bin Cui. 2023. OSDP: Optimal Sharded Data Parallel for Distributed Deep Learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI 2023)*. 2142–2150.

[27] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations 2015 (ICLR 2015)*.

[28] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing Activation Recomputation in Large Transformer Models. In *Proceedings of Machine Learning and Systems 2023 (MLSys 2023)*.

[29] Mario Michael Krell, Matej Kosec, Sergio P Perez, and Andrew Fitzgibbon. 2021. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *CoRR* abs/2107.02027 (2021).

[30] Achintya Kundu, Rhui Dih Lee, Laura Wynter, Raghu Kiran Ganti, and Mayank Mishra. 2024. Enhancing Training Efficiency Using Packing with Flash Attention. *CoRR* abs/2407.09105 (2024).

[31] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proc. VLDB Endow.* 17, 8 (2024), 1939–1952.

[32] Daiyi Li, Yaofeng Tu, Xiangsheng Zhou, Yangming Zhang, and Zongmin Ma. 2022. End-to-end chinese entity recognition based on bert-bilstm-att-crf. *ZTE Communications* 20, S1 (2022), 27.

[33] Haoyang Li, Fangcheng Fu, Hao Ge, Sheng Lin, Xuanyu Wang, Jiawen Niu, Xupeng Miao, and Bin Cui. 2025. Hetu v2: A General and Scalable Deep Learning System with Hierarchical and Heterogeneous Single Program Multiple Data Annotations. *CoRR* abs/2504.20490 (2025).

[34] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (2020), 3005–3018.

[35] Yang Liu and Mirella Lapata. 2019. Text Summarization with Pretrained Encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019)*. 3728–3738.

[36] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations 2019 (ICLR 2019)*.

[37] Yuren Mao, Yuhang Ge, Yijiang Fan, Wenyi Xu, Yu Mi, Zhonghao Hu, and Yunjun Gao. 2024. A Survey on LoRA of Large Language Models. *CoRR* abs/2407.11046 (2024).

[38] Xupeng Miao, Xiaonan Nie, Hailin Zhang, Tong Zhao, and Bin Cui. 2023. Hetu: a highly efficient automatic parallel distributed deep learning system. *Sci. China Inf. Sci.* 66 (2023).

[39] Xupeng Miao, Gabriele Oliaro, Xinhao Cheng, Mengdi Wu, Colin Unger, and Zhihao Jia. 2024. FlexLLM: A System for Co-Serving Large Language Model Inference and Parameter-Efficient Finetuning. *CoRR* abs/2402.18789 (2024).

[40] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism. *Proc. VLDB Endow.* 16, 3 (2022), 470–479.

[41] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP 2019)*. 1–15.

[42] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-Efficient Pipeline-Parallel DNN Training. In *International Conference on Machine Learning 2021 (ICML 2021)*, Vol. 139. 7937–7947.

[43] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *International Conference for High Performance Computing, Networking 2021 (SC 2021)*. 58.

[44] Xiaonan Nie, Yi Liu, Fangcheng Fu, Jinbao Xue, Dian Jiao, Xupeng Miao, Yangyu Tao, and Bin Cui. 2023. Angel-PTM: A Scalable and Economical Large-scale Pre-training System in Tencent. *Proc. VLDB Endow.* 16, 12 (2023), 3781–3794.

[45] NVIDIA. 2024. NeMo. https://github.com/NVIDIA/NeMo.

[46] NVIDIA. 2024. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.

[47] OpenAI. 2022. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt.

[48] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).

[49] OpenAI. 2024. OpenAI Platform: Fine-tuning. https://platform.openai.com/docs/guides/fine-tuning/.

[50] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Iqbal khan, and Arsalan Shahid. 2024. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *CoRR* abs/2408.13296 (2024).

[51] Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric Michael Smith, Y-Lan Boureau, and Jason

Weston. 2020. Recipes for building an open-domain chatbot. *CoRR* abs/2004.13637 (2020).

[52] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2024. SLoRA: Scalable Serving of Thousands of LoRA Adapters. In *Proceedings of Machine Learning and Systems 2024 (MLSys 2024)*.

[53] Weijia Shi, Sewon Min, Maria Lomeli, Chunting Zhou, Margaret Li, Xi Victoria Lin, Noah A. Smith, Luke Zettlemoyer, Wen-tau Yih, and Mike Lewis. 2024. In-Context Pretraining: Language Modeling Beyond Document Boundaries. In *International Conference on Learning Representations 2024 (ICLR 2024)*.

[54] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019).

[55] Arjun Singh, Nikhil Pandey, Anup Shirgaonkar, Pavan Manoj, and Vijay Aski. 2024. A Study of Optimizations for Fine-tuning Large Language Models. *CoRR* abs/2406.02290 (2024).

[56] Snowflake. 2025. Fine-tuning (Snowflake Cortex). https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-finetuning.

[57] Zijian Song, Wenhan Zhang, Lifang Deng, Jiandong Zhang, Kaigui Bian, and Bin Cui. 2024. MultiLoRA: Multi-Directional Low Rank Adaptation for Multi-Domain Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM 2024)*. 2148–2157.

[58] Jakub Tarnawski, Deepak Narayanan, and Amar Phanishayee. 2021. Piper: Multidimensional Planner for DNN Parallelization. In *Annual Conference on Neural Information Processing Systems 2021 (NeurIPS 2021)*. 24829–24840.

[59] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023).

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Annual Conference on Neural Information Processing Systems 2017 (NeurIPS 2017)*.

[61] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. How Far Can Camels Go? Exploring the State of Instruction Tuning on Open Resources. In *Annual Conference on Neural Information Processing Systems 2023 (NeurIPS 2023)*.

[62] Yujie Wang, Youhe Jiang, Xupeng Miao, Fangcheng Fu, Shenhan Zhu, Xiaonan Nie, Yaofeng Tu, and Bin Cui. 2024. Improving Automatic Parallel Training via Balanced Memory Workload Optimization. *IEEE Trans. Knowl. Data Eng.* 36, 8 (2024), 3906–3920.

[63] Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. MultiLoRA: Democratizing LoRA for Better Multi-Task Learning. *CoRR* abs/2311.11501 (2023).

[64] Xun Wu, Shaohan Huang, and Furu Wei. 2024. Mixture of LoRA Experts. In *The Twelfth International Conference on Learning Representations 2024 (ICLR 2024)*.

[65] Yifei Xia, Fangcheng Fu, Wentao Zhang Jiawei Jiang, and Bin Cui. 2024. Efficient Multi-task LLM Quantization and Serving for Multiple LoRA Adapters. In *Annual Conference on Neural Information Processing Systems 2024 (NeurIPS 2024)*.

[66] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 Technical Report. *CoRR* abs/2407.10671 (2024).

[67] Zhengmao Ye, Dengchun Li, Zetao Hu, Tingfeng Lan, Jian Sha, Sicong Zhang, Lei Duan, Jie Zuo, Hui Lu, Yuanchun Zhou, and Mingjie Tang. 2023. mLoRA: Fine-Tuning LoRA Adapters via Highly-Efficient Pipeline Parallelism in Multiple GPUs. *CoRR* abs/2312.02515 (2023).

[68] Huangzhao Zhang, Kechi Zhang, Zhuo Li, Jia Li, Jia Li, Yongmin Li, Yunfei Zhao, Yuqi Zhu, Fang Liu, Ge Li, and Zhi Jin. 2024. Deep learning for code generation: a survey. *Sci. China Inf. Sci.* 67 (2024).

[69] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. In *International conference on machine learning (ICML 2020)*, Vol. 119. 11328–11339.

[70] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. 2020. DIALOGPT : Large-Scale Generative Pre-training for Conversational Response Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (ACL 2020 Demo)*. 270–278.

[71] Zhongping Zhang, Yin Jia, Yuehan Hou, and Xinlu Yu. 2024. Explicit Behavior Interaction with Heterogeneous Graph for Multi-behavior Recommendation. *Data Sci. Eng.* 9 (2024).

[72] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. 2022. MiCS: Near-linear Scaling for Training Gigantic Model on Public Cloud. *Proc. VLDB Endow.* 16, 1 (2022), 37–50.

[73] Pinxue Zhao, Hailin Zhang, Fangcheng Fu, Xiaonan Nie, Qibin Liu, Fang Yang, Yuanbo Peng, Dian Jiao, Shuaipeng Li, Jinbao Xue, Yangyu Tao, and Bin Cui. 2025. Efficiently Training 7B LLM with 1 Million Sequence Length on 8 GPUs. In *Proceedings of the 2025 ACM International Conference on Management of Data (SIGMOD 2025)*.

[74] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.* 16, 12 (2023), 3848–3860.

[75] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2022)*. 559–578.

[76] 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 *Proceedings of the 53rd International Conference on Parallel Processing, (ICPP 2024)*. 357–366.

[77] Ming Zhong, Yelong Shen, Shuohang Wang, Yadong Lu, Yizhu Jiao, Siru Ouyang, Donghan Yu, Jiawei Han, and Weizhu Chen. 2024. Multi-LoRA Composition for Image Generation. *CoRR* abs/2402.16843 (2024).

[78] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-Bot: Database Diagnosis System using Large Language Models. *Proc. VLDB Endow.* 17, 10 (2024), 2514–2527.

[79] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Sci. Eng.* 9 (2024).

[80] Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. 2022. PetS: A Unified Framework for Parameter-Efficient Transformers Serving. In *2022 USENIX Annual Technical Conference (ATC 2022)*. 489–504.

[81] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2024. Multilingual Machine Translation with Large Language Models: Empirical Results and Analysis. In *Findings of the Association for Computational Linguistics: NAACL (NAACL Findings 2024)*. 2765–2781.

A DETAILS OF CONFIGURATION PRUNING

As introduced in §4.2, in order to accelerate the problem-solving of model deployment (i.e., Equation (2)), we propose the configuration pruning heuristics. Due to the space constraint, we only present the rationale of the two heuristics. In this section, we provide details about how to propose candidate parallel configurations and estimate the efficiency lower bound of a deployment plan.

Configuration Proposal. The first is to propose a small set of parallel configurations (rather than covering all possible ones) for problem-solving, which is based on the fact that many parallel configurations are consistently less efficient than others. To elaborate, we conduct an empirical study and find that a partial order relation on the efficiency exists among the configurations, as described in Observation 1.

Observation 1. *Given a sequence length s_0 and two parallel configurations $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$, if $\mathcal{S}_\alpha$ has higher throughput (i.e., processed tokens per GPU per second) on sequence length s_0, then for any sequence length $s < s_0$, if the batch size b satisfies $b \times s = s_0$, $\mathcal{S}_\alpha$ has higher throughput than $\mathcal{S}_\beta$.*

To validate the observation presented above, we conduct a series of experiments on different configurations. As shown in Table 3, for each configuration, we evaluate its throughput when faced with different sequence lengths. Taking $s_0 = 8K$ and *num_gpus* = 8 as an example, configuration ⟨TP=2, PP=4⟩ achieves higher throughput than other configurations with the same sequence length, and its throughput is still higher for shorter sequences, which matches Observation 1.

Based on this, when faced with a large number of parallel configurations, we propose candidate configurations by selecting those with the highest throughput for each (*num_gpus*, *seq_len*) pair, which can be depicted by the following SQL:

`SELECT` *config*, `MAX`(*thruput*) `FROM` *thruput_table* `GROUP BY` *num_gpus, seq_len*

By doing so, a configuration will not be selected if it is consistently outperformed by the others, so the achieved solutions will not be affected (as evaluated in Appendix B.2). In addition, it is obvious that the number of candidate configurations is limited to at most $O(R \log N)$, where R and N denote the number of buckets and GPUs, respectively. Consequently, we can effectively reduce the number of candidate configurations, whilst guaranteeing correctness.

Table 3: Throughput (i.e., processed tokens per GPU per second) of each candidate parallel configuration with sequence lengths and different numbers of GPUs. A cell is left empty if there are not enough GPUs to deploy with the corresponding parallel configuration. The symbol "✗" indicates the parallel configuration does not support processing the corresponding sequence length due to out-of-memory errors. The symbol "-" indicates the throughput remains the same after model replication.

seq_len	2K				4K			8K		16K
num_gpus	1	2	4	8	2	4	8	4	8	8
⟨TP=1, PP=1⟩	***5.11***	-	-	-	✗	✗	✗	✗	✗	✗
⟨TP=2, PP=1⟩		4.30	-	-	***4.12***	-	-	✗	✗	✗
⟨TP=1, PP=2⟩		4.88	-	-	✗	✗	✗	✗	✗	✗
⟨TP=4, PP=1⟩		✗	3.63	-		3.50	-	***3.25***	-	✗
⟨TP=2, PP=2⟩		✗	4.15	-		3.98	-	✗	✗	✗
⟨TP=1, PP=4⟩		✗	5.03	-		***4.78***	-	✗	✗	✗
⟨TP=8, PP=1⟩				2.79			2.71		2.56	***2.33***
⟨TP=4, PP=2⟩				3.48			3.34		3.12	✗
⟨TP=2, PP=4⟩				4.27			4.10		***3.79***	✗
⟨TP=1, PP=8⟩				4.45			4.25		✗	✗

Lower Bound Filtering. Secondly, we take lower bound estimation as a guideline to filter out inefficient deployment plans, leveraging length-based data dispatching and the rationale of data migration (or movement). In fact, considering the negative correlation between sequence length and throughput of a parallel configuration (i.e., lower throughput for longer sequences, which holds in practice), length-based data dispatching serves as a greedy approach — with length-based data dispatching, data is always assigned to the most efficient configuration that suffices the memory consumption requirement. However, as discussed earlier regarding the sequence length skewness in §3, workload balancing necessitates re-dispatching data, which may result in some short sequences being assigned to less efficient configurations compared to length-based data dispatching. Thus, length-based data dispatching can serve as an effective method for estimating the lower bound of running time. Specifically, it is formalized as the following theorem.

Theorem 1. *Consider a deployment plan with n heterogeneous FT replicas, each with a different maximum supportable sequence length. The corresponding number of GPUs is given by $\{N_1, N_2, \cdots, N_n\}$, and the running times when applying the length-based data dispatching are denoted by $\{t_1, t_2, \cdots, t_n\}$. After re-dispatching the training data according to the workload balancing principle, the running times adjust to $\{t'_1, t'_2, \cdots, t'_n\}$. Then, we have the following inequality:*

$$N\hat{t} \geq \sum_{i=1}^{n} N_i t_i \tag{5}$$

where N is the total number of GPUs and $\hat{t}$ denotes the maximum running time across the n replicas after workload-balancing, denoted as $\hat{t} = \max_{1 \le i \le n}\{t_i'\}$.

The proof of this theorem relies on the following definitions, properties and assumptions:

Definition 1. *The Average Throughput Bound (ATB) for a parallel configuration with N GPUs is defined as follows:*
Under the constraint that the number of chunks (i.e. micro-batches) is sufficiently large, for a given sequence length s and maximum supportable sequence length M, let the batch size b satisfies $b \times s \le M < (b+1) \times s$, and the running time for an input size of $b \times s$ is t, then the ATB for s is defined as

$$ATB_s = \frac{bs}{Nt} \tag{6}$$

Property 1. *Based on Observation 1, for a given sequence length s_0, if configuration $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$ satisfies $ATB_{s_0,\alpha} \ge ATB_{s_0,\beta}$, then for any sequence length $s \le \min\{M_\alpha, M_\beta\}$, it holds that $ATB_{s,\alpha} \ge ATB_{s,\beta}$. This property is denoted as $ATB_\alpha \ge ATB_\beta$.*

Property 2. *Suppose n replicas are arranged in descending order of ATB. Then, the maximum supportable length of each replica increases monotonically. Consequently, during data migration for workload balancing, the i-th replica can only receive data from the first $i-1$ replicas.*

Assumption 1. *The relationship between the ATB of configuration $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$ can reflect the corresponding average throughput per GPU relationship during runtime, under the constraint that the number of chunks is sufficiently large. Let the average throughput per GPU during the runtime of configuration $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$ be $\overline{T}_\alpha$ and $\overline{T}_\beta$, if $ATB_\alpha \ge ATB_\beta$, then $\overline{T}_\alpha \ge \overline{T}_\beta$.*

Proof. The proof for Theorem 1 proceeds by mathematical induction on the number of replicas, denoted by n. In the proof, the replicas are ordered in descending order of *ATB*.

(1) **Base Case.** When $n = 2$, let the number of GPUs for the two replicas be N_1 and N_2, and the number of input tokens for each replica before workload balancing be M_1 and M_2, respectively. During workload balancing, M tokens are moved from replica 1 to replica 2. By Assumption 1, we have

$$\begin{aligned}(N_1t_1' + N_2t_2') - (N_1t_1 + N_2t_2) &= \left(\frac{N_1(M_1 - M)}{T_1} + \frac{N_2(M_2 + M)}{T_2}\right) - \left(\frac{N_1M_1}{T_1} + \frac{N_2M_2}{T_2}\right) \\ &= M \times \left(\frac{N_2}{T_2} - \frac{N_1}{T_1}\right) \\ &= M \times \left(\frac{1}{\overline{T}_2} - \frac{1}{\overline{T}_1}\right) \\ &\ge 0\end{aligned} \tag{7}$$

Thus, we obtain the inequality $N_1t_1 + N_2t_2 \le N_1t_1' + N_2t_2' \le (N_1 + N_2)\hat{t} = N\hat{t}$.

(2) **Inductive Hypothesis.** Assume that for $n \le k$, the inequality $N\hat{t} \ge \sum_{i=1}^{n} N_it_i$ holds.

(3) **Inductive Step.** When $n = k+1$, let i be the smallest index such that for all $1 \le j < i$, no data are moved to other replicas during data migration for workload balancing. We discuss this in two cases:
   (a) If $i > 1$, by Property 2, the running time of the first $i-1$ replicas remains unchanged during workload balancing, so $t_j = t_j'$ for all $1 \le j < i$. For the remaining $n - i + 1$ replicas, by the inductive hypothesis, we have $\sum_{j=i}^{k+1} N_jt_j \le \sum_{j=i}^{k+1} N_jt_j'$. Therefore, we have

$$\sum_{j=1}^{k+1} N_jt_j \le \sum_{j=1}^{i-1} N_jt_j + \sum_{r=i}^{k+1} N_rt_r' = \sum_{j=1}^{k+1} N_jt_j' \tag{8}$$

   (b) If $i = 1$, during workload balancing, replica 1 migrates sequences to other replicas. Let M denote the set of data moved from replica 1. The migration process can be broken down into two steps:
      (i) Migrate all of M to replica 2.
      (ii) Replica 2 then re-dispatch M to other replicas to achieve workload balancing.

   After the first step, the running time of replica 1 and 2 are t_1' and $\tilde{t}_2$, respectively (replica 1 does not receive data from other replicas, so it has achieved workload balancing). By the base case of the induction, we have $N_1t_1 + N_2t_2 \le N_1t_1' + N_2\tilde{t}_2$. By the inductive hypothesis, for the remaining k replicas excluding replica 1, we have $N_2\tilde{t}_2 + \sum_{i=3}^{k+1} N_it_i \le \sum_{i=2}^{k+1} N_it_i'$. Therefore, we obtain

$$\sum_{i=1}^{k+1} N_it_i \le N_1t_1' + (N_2\tilde{t}_2 + \sum_{j=3}^{k+1} N_jt_j) \le N_1t_1' + \sum_{j=2}^{k+1} N_jt_j' = \sum_{i=1}^{k+1} N_it_i' \tag{9}$$

By induction, the conclusion holds. □

In the scenario where two replicas share the same maximum supportable sequence length, we can group these replicas together, and perform workload balancing within the group. This allows each group to be treated as a single replica in the proof process, thereby reducing the problem to the case where the maximum supportable sequence lengths of all replicas are distinct.

Theorem 1 suggests an efficient and effective method for estimating the lower bound of a deployment plan, expressed as $\frac{\sum_{i=1}^{n} N_i t_i}{N}$. The estimation leverages length-based data dispatching, which relies solely on sequence lengths and does not account for the workload relationships among different replicas.

However, Assumption 1 does not always hold in real-world scenarios where the number of chunks is variable and may be insufficient. As a result, in this work, we only treat the lower bound as a relative metric to filter out deployment plans whose estimated lower bounds exceed the current minimum by more than a pre-defined threshold (15% by default). This approach is demonstrated to be effective in practice.

Putting them together. In conclusion, we summarize the configuration pruning process as the following steps:

(1) Propose configuration candidates leveraging the offline benchmarking results.
(2) Construct possible deployment plans using the parallel configuration candidates, formulated as an integer partition problem, and solved via dynamic programming.
(3) Estimate the lower bound for each deployment plan and filter out those that are estimated to be inefficient.

Eventually, for each remaining deployment plan, we determine its performance individually by solving the problem defined in Equation 2. It is noteworthy that, if a deployment plan is given, then the problem turns into an ILP problem, which is much more efficient to solve. Thus, we enumerate all remaining deployment plans, solve the problem for each one (in parallel), and select the most efficient one as the final deployment plan.

In Appendix B.2, we will provide experiments to show that the solving of the model deployment plan can be accelerated significantly, whilst maintaining the same solution.

B MORE EXPERIMENTAL DETAILS AND RESULTS

B.1 Summary of Evaluated FT Tasks

We select 12 popular open-source FT datasets and treat each as one FT task, spanning instruction tuning, question answering, and summarization tasks. These datasets encompass diverse fields, including mathematics, coding, medicine, and routine task handling. The summary is presented in Table 4.

Table 4: Summary of FT datasets used in our experiments.

Dataset Name	Avg. Sequence Length	Skewness	Kurtosis	Type of Task	Batch Size
databricks-dolly-15k[4]	207	7.11	95.43	Instruction Tuning	256
python_code_instructions[5]	269	10.01	121.55	Code Instruction Tuning	128
Evol-Instruct[6]	702	6.59	80.28	Code Instruction Tuning	128
CommitPackFt[7]	663	0.79	1.68	Code Instruction Tuning	128
MathInstruct[8]	252	3.03	12.72	Math Instruction Tuning	128
MetaMathQA[9]	236	2.56	14.56	Math Question Answering	128
NuminaMath-CoT[10]	543	1.52	3.51	Math Question Answering	256
PubMedQA[11]	371	0.73	3.29	Medical Question Answering	64
XSum[12]	526	7.49	371.80	Summarization	128
BillSum[13]	3903	0.85	0.30	Summarization	32
cnn_dailymail[14]	947	0.89	0.64	Summarization	256
MeetingBank[15]	3622	4.35	26.50	Summarization	64

For the instruction tuning tasks, the databricks-dolly-15k dataset is a corpus of approximately 15,000 records generated by humans, covering various instruction categories, such as information extraction. The python_code_instructions dataset contains more than 18,600 problem descriptions and code in Python language, formatted in Alpaca style. The Evol-Instruct dataset augments the code instruction tuning dataset CodeAlpaca_20K via a total of 10 augmentation strategies, covering more than 11,000 samples. The CommitPackFT dataset leverages the natural structure of Git commits for instruction tuning, comprising about 491,000 pairs of code changes and human instructions and covering 4 terabytes of Git commits across 350 programming languages. The MathInstruct dataset is compiled from 13 math rationale datasets and focuses on the hybrid use of rationales like chain-of-thought, containing over 262,000 mathematical problem-solution pairs.

For the question-answering tasks, the MetaMathQA dataset augments the training sets of the GSM8K and MATH datasets by rewriting the questions from multiple perspectives, comprising roughly 395,000 mathematical question-answering samples. The NuminaMath-CoT dataset consists of approximately 860,000 mathematical problems, where each solution is formatted in a chain-of-thought manner. The PubMedQA dataset is a dataset for biomedical research question answering, and it has 211,300 artificially generated QA instances.

For the summarization tasks, the XSum dataset consists of approximately 204,000 text-summary pairs from the BBC news dataset. The BillSum dataset is a corpus of summarization of US Congressional and California state bills, compromising 189,00 text-summary pairs. The cnn_dailymail dataset contains over 300,000 unique news articles written by journalists at CNN and the Daily Mail and abstracts. The MeetingBank dataset contains 6,892 segment-level summarization instances from council meetings.

B.2 More Experimental Results

Below we present more experimental results to thoroughly evaluate the effectiveness of our work.

Scalability. We evaluate the scalability w.r.t. the number of GPUs and tasks, respectively. The results are provided in Figure 11. We first compare the GPU seconds of 4-task joint FT over varying numbers of GPUs. With only 16 GPUs, both LobRA and Task-Fused can only deploy one FT replica (with ⟨TP=16,PP=1⟩), so they have the same performance. However, with more GPUs, LobRA can leverage heterogeneous FT replicas, reducing the GPU seconds needed for each training step. On the contrary, Task-Fused's efficiency goes down slightly when

[4]https://huggingface.co/datasets/databricks/databricks-dolly-15k
[5]https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca
[6]https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K
[7]https://huggingface.co/datasets/chargoddard/commitpack-ft-instruct
[8]https://huggingface.co/datasets/TIGER-Lab/MathInstruct
[9]https://huggingface.co/datasets/meta-math/MetaMathQA
[10]https://huggingface.co/datasets/AI-MO/NuminaMath-CoT
[11]https://huggingface.co/datasets/qiaojin/PubMedQA
[12]https://huggingface.co/datasets/EdinburghNLP/xsum
[13]https://huggingface.co/datasets/FiscalNote/billsum
[14]https://huggingface.co/datasets/abisee/cnn_dailymail
[15]https://huggingface.co/datasets/huuuyeah/meetingbank

there are more GPUs, owing to the increased overhead of synchronization among FT replicas. Then we measure how the GPU seconds change w.r.t. the number of tasks using the 70B model over 64 GPUs. It can be observed that both LobRA and Task-Fused exhibit a near-linear increment in the GPU seconds when there are more tasks, which is reasonable as the FT workloads become higher. Yet LobRA consistently outperforms Task-Fused with all numbers of tasks.

Sensitivity. To process training data with different levels of lengths using divergent parallel configurations, our work introduces a hyper-parameter R, which indicates that we divide the training data into R buckets according to their lengths. To assess the sensitivity of our work w.r.t. R, we enumerate the value of R from 4 to 32 and record the training time as well as how many padding tokens are added. The results are shown in Figure 12. It can be observed that, as R increases, it consistently reduces the padding tokens, which is reasonable. However, after R goes beyond 12, the training time remains stable despite the reduction in padding tokens, which is because more buckets would lead to more overhead. In short, the efficiency of LobRA is robust to the choice of R.

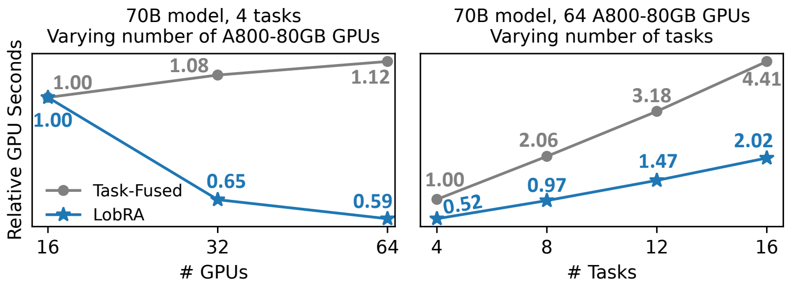


Figure 11: Scalability w.r.t. numbers of GPUs and tasks (70B model).

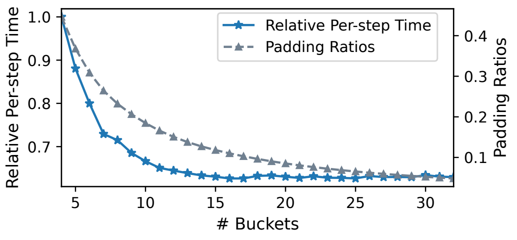


Figure 12: Impact of number of buckets (i.e., R) in dynamic bucketing to the per-step time and padding ratios (7B model, 16 A100-40GB GPUs). The per-step time is scaled by that with 4 buckets.

Effectiveness of Configuration Planning. To assess the scalability of configuration planning (i.e., solving Equation (2)) as well as the effectiveness of the two configuration pruning techniques, we conduct experiments to measure the time cost of solving Equation (2) for the 70B model using 16 to 256 GPUs. The datasets are consistent with those used in the GPU scalability evaluation, detailed in Appendix B.3. The results are presented in Table 5. Specifically, the parallel configurations solved by all approaches are the same (as long as there is no timeout), validating the accuracy of our configuration pruning methods. Furthermore, although the first pruning technique (i.e., configuration proposal) accelerates the solving process, it still encounters timeout issues when the number of GPUs exceeds 48. In contrast, enabling both pruning techniques (i.e., configuration proposal and lower bound filtering) efficiently delivers results within several minutes, even in the case of 256 GPUs, demonstrating its scalability. Although there is still an overhead of solving the model deployment at initialization, it is worthwhile given the speedup in the joint FT process. In addition, for FT, it is rare to use 256 GPUs or more, so our work fits real-world FT scenarios well.

Table 5: Time cost (in seconds) of configuration planning via different approaches (70B model). The symbol "✗" indicates that the problem solving fails to finish within 3600 seconds (1 hour). The achieved deployment plan is consistent across all approaches.

# GPUs	w/o Configuration Proposal w/o Lower Bound Filtering	w/ Configuration Proposal w/o Lower Bound Filtering	w/ Configuration Proposal w/ Lower Bound Filtering	Deployment Plan
16	0.067	0.034	0.014	⟨16,1⟩×1
24	0.842	0.540	0.378	⟨2,4⟩×1, ⟨16,1⟩×1
32	7.63	3.29	1.71	⟨2,4⟩×1, ⟨4,2⟩×1, ⟨16,1⟩×1
40	✗	264.24	28.78	⟨2,4⟩×1, ⟨4,2⟩×1, ⟨8,1⟩×1, ⟨16,1⟩×1
48	✗	2957.01	48.84	⟨2,4⟩×2, ⟨4,2⟩×1, ⟨8,1⟩×1, ⟨16,1⟩×1
64	✗	✗	148.81	⟨2,4⟩×3, ⟨4,2⟩×1, ⟨8,1⟩×2, ⟨16,1⟩×1
128	✗	✗	239.84	⟨2,4⟩×5, ⟨8,1⟩×9, ⟨16,1⟩×1
256	✗	✗	428.08	⟨2,4⟩×3, ⟨4,1⟩×16, ⟨4,2⟩×6, ⟨8,1⟩×13, ⟨16,1⟩×1

Comparison between Task-Sequential and LobRA-Sequential. To evaluate the effectiveness of our designs in single-task FT, we compare the training time of each task of Task-Sequential and LobRA-Sequential. It can be seen that although LobRA-Sequential achieves over 60% speedup on 1 task (which even surpasses the end-to-end speedup achieved by LobRA), the speedup on the other tasks is lower, and it even suffers from performance degradation on 2 tasks. This phenomenon is reasonable. For one thing, employing heterogeneous FT replicas is less beneficial for certain tasks whose data heterogeneity is not severe. For another, unlike pre-training, the batch size for each FT task is usually small in order to avoid over-fitting, as a result of which, it is difficult to achieve workload balance by adjusting the data dispatching across replicas.

Table 6: Comparison of GPU seconds for each FT task of Task-Sequential and LobRA-Sequential (70B model).

Dataset Name	Task-Sequential (T_1)	LobRA-Sequential (T_2)	$(T_1 - T_2)/T_1$
MathInstruct	209.5808	162.7456	22.35%
python_code_instructions	539.232	296.3648	45.04%
databricks-dolly-15k	404.4928	264.7744	34.54%
BillSum	1231.0016	800.7936	34.95%
CommitPackFt	471.6864	278.7072	40.91%
NuminaMath-CoT	697.8368	471.9232	32.37%
PubMedQA	129.9712	157.6768	-21.32%
MetaMathQA	168.1856	117.7536	29.99%
Evol-Instruct	1250.3296	476.224	61.91%
cnn_dailymail	1074.112	1227.7312	-14.30%
XSum	786.4	417.632	46.89%
MeetingBank	2186.4192	1249.5936	42.85%

B.3 Details of Experiment Configurations

In this section, we provide more details about the datasets and parallel configurations used in our experiments.

For the end-to-end evaluation in §5.2, we consider all 12 tasks listed in Table 4 for the 32B and 70B models. For the 7B model, we focus on a subset of 6 tasks: databricks-dolly-15k, Evol-Instruct, XSum, CommitPackFT, MeetingBank, and python_code_instructions. Furthermore, the specific parallel configurations of various tasks in the Task-Sequential and LobRA-Sequential baselines are presented in Table 7 and Table 8, respectively. Besides, the experiments in §5.3 use the same parallel configurations as those in the end-to-end evaluation.

For the GPU and task scalability evaluation in Figure 11, we focus on a subset of 4 tasks: Evol-Instruct, CommitPackFT, BillSum, and PubMedQA. Table 10 and Table 9 present the parallel configurations of Task-Fused and LobRA under varying GPU counts and task numbers.

Table 7: Parallel configurations used by Task-Sequential for the end-to-end evaluation. (Those used by Task-Fused and LobRA are provided in Table 2.) Each $\langle\alpha, \beta\rangle \times \gamma$ indicates that there are γ FT replica(s) with a TP degree of α and a PP degree of β.

Dataset Name	Task-Sequential		
	7B	**32B**	**70B**
MathInstruct	-	⟨2,2⟩×16	⟨4,2⟩×8
python_code_instructions	⟨8,1⟩×2	⟨8,1⟩×8	⟨16,1⟩×4
databricks-dolly-15k	⟨4,1⟩×4	⟨4,1⟩×16	⟨8,1⟩×8
BillSum	-	⟨8,1⟩×8	⟨16,1⟩×4
CommitPackFt	⟨4,1⟩×4	⟨4,1⟩×16	⟨8,1⟩×8
NuminaMath-CoT	-	⟨4,1⟩×16	⟨8,1⟩×8
PubMedQA	-	⟨4,1⟩×16	⟨2,4⟩×8
MetaMathQA	-	⟨2,2⟩×16	⟨4,2⟩×8
Evol-Instruct	⟨8,1⟩×2	⟨8,1⟩×8	⟨16,1⟩×4
cnn_dailymail	-	⟨4,1⟩×16	⟨8,1⟩×8
XSum	⟨8,1⟩×2	⟨8,1⟩×8	⟨16,1⟩×4
MeetingBank	⟨8,1⟩×2	⟨8,1⟩×8	⟨16,1⟩×4

Table 8: Parallel configurations used by LobRA-Sequential for the end-to-end evaluation. (Those used by Task-Fused and LobRA are provided in Table 2.) Each $\langle \alpha, \beta \rangle \times \gamma$ indicates that there are γ FT replica(s) with a TP degree of α and a PP degree of β.

Dataset Name	LobRA-Sequential		
	7B	32B	70B
MathInstruct	-	⟨1,2⟩×2, ⟨2,1⟩×26, ⟨1,4⟩×1, ⟨2,2⟩×1	⟨2,4⟩×5, ⟨4,1⟩×2, ⟨4,2⟩×2
python_code_instructions	⟨2,1⟩×2, ⟨4,1⟩×1, ⟨8,1⟩×1	⟨1,2⟩×2, ⟨2,1⟩×2, ⟨4,1⟩×10, ⟨8,1⟩×2	⟨2,4⟩×1, ⟨4,1⟩×2, ⟨8,1⟩×2, ⟨16,1⟩×2
databricks-dolly-15k	⟨1,1⟩×4, ⟨1,4⟩×2, ⟨4,1⟩×1	⟨1,4⟩×2, ⟨2,1⟩×6, ⟨2,2⟩×9, ⟨4,1⟩×2	⟨2,4⟩×2, ⟨4,1⟩×4, ⟨4,2⟩×3, ⟨8,1⟩×1
BillSum	-	⟨2,2⟩×9, ⟨4,1⟩×3, ⟨8,1⟩×2	⟨8,1⟩×2, ⟨16,1⟩×3
CommitPackFt	⟨1,1⟩×10, ⟨2,1⟩×1, ⟨4,1⟩×1	⟨2,2⟩×12, ⟨4,1⟩×4	⟨2,4⟩×1, ⟨4,1⟩×6, ⟨8,1⟩×4
NuminaMath-CoT	-	⟨1,4⟩×4, ⟨2,1⟩×20, ⟨4,1⟩×2	⟨2,4⟩×3, ⟨4,1⟩×6, ⟨4,2⟩×1, ⟨8,1⟩×1
PubMedQA	-	⟨1,2⟩×10, ⟨2,1⟩×14, ⟨1,4⟩×4	⟨2,2⟩×10, ⟨4,1⟩×4, ⟨2,4⟩×1
MetaMathQA	-	⟨1,4⟩×14, ⟨2,1⟩×2, ⟨2,2⟩×1	⟨2,4⟩×6, ⟨4,1⟩×2, ⟨4,2⟩×1
Evol-Instruct	⟨1,1⟩×6, ⟨2,1⟩×1, ⟨8,1⟩×1	⟨2,1⟩×16, ⟨2,2⟩×1, ⟨4,1⟩×3, ⟨8,1⟩×2	⟨2,4⟩×1, ⟨4,1⟩×4, ⟨8,1⟩×3, ⟨16,1⟩×1
cnn_dailymail	-	⟨1,4⟩×1, ⟨2,1⟩×14, ⟨4,1⟩×8	⟨2,4⟩×6, ⟨4,2⟩×1, ⟨8,1⟩×1
XSum	⟨1,1⟩×6, ⟨2,1⟩×1, ⟨8,1⟩×1	⟨1,2⟩×8, ⟨4,1⟩×8, ⟨8,1⟩×2	⟨2,4⟩×2, ⟨4,1⟩×2, ⟨4,2⟩×1, ⟨16,1⟩×2
MeetingBank	⟨1,4⟩×2, ⟨8,1⟩×1	⟨1,2⟩×6, ⟨1,4⟩×3, ⟨8,1⟩×5	⟨4,2⟩×1, ⟨8,1⟩×1, ⟨16,1⟩×3

Table 9: Parallel configurations used by Task-Fused and LobRA in Task scalability evaluation.

# Tasks	Task-Fused	LobRA
4	⟨16,1⟩×4	⟨2,4⟩×3, ⟨4,2⟩×1, ⟨8,1⟩×2, ⟨16,1⟩×1
8	⟨16,1⟩×4	⟨2,4⟩×3, ⟨4,2⟩×2, ⟨8,1⟩×1, ⟨16,1⟩×1
12	⟨16,1⟩×4	⟨2,4⟩×3, ⟨4,2⟩×2, ⟨8,1⟩×1, ⟨16,1⟩×1
16	⟨16,1⟩×4	⟨2,4⟩×3, ⟨4,2⟩×2, ⟨8,1⟩×1, ⟨16,1⟩×1

Table 10: Parallel configurations used by Task-Fused and LobRA in GPU scalability evaluation.

# GPUs	Task-Fused	LobRA
16	⟨16,1⟩×1	⟨16,1⟩×1
32	⟨16,1⟩×2	⟨2,4⟩×1, ⟨4,2⟩×1, ⟨16,1⟩×1
64	⟨16,1⟩×4	⟨2,4⟩×3, ⟨4,2⟩×1, ⟨8,1⟩×2, ⟨16,1⟩×1

C SYSTEM COMPARISON WITH HOMOGENEOUS CONFIGURATIONS

As mentioned in §5.1, LobRA achieves comparable efficiency against NeMo [45], a popular LLM training framework developed by NVIDIA, when training with homogeneous FT replicas. Here we present the results of extensive experiments to compare the performance of LobRA and NeMo thoroughly. To be specific, we consider fine-tuning the 7B model over 16 A100-40GB GPUs with a global batch size of 64 and varying the sequence lengths in different experiments. For each experiment, LobRA and NeMo utilize the same homogeneous parallel configuration and uniform data dispatching for fair comparison. In addition, to eliminate the impact of dynamic bucketing, all training data (sequences) are truncated or padded to the same maximum sequence length. The number of chunks (i.e. micro-batches) for gradient accumulation is tuned to achieve the best efficiency whilst avoiding out-of-memory errors. The results, summarized in Table 11, demonstrate that LobRA delivers comparable performance against NeMo when training with the same homogeneous parallel configurations.

Table 11: System comparison of LobRA and NeMo with different homogeneous parallel configurations.

# GPUs	Config	Max Sequence Length	# Chunks (Micro Batches)	LobRA Time (s)	NeMo Time (s)
16	⟨TP=1,PP=1⟩×16	2048	4	1.778	1.533
16	⟨TP=1,PP=2⟩×8	2048	8	1.978	1.785
16	⟨TP=1,PP=4⟩×4	2048	16	2.131	1.939
16	⟨TP=1,PP=4⟩×4	4096	16	4.141	3.872
16	⟨TP=1,PP=8⟩×2	2048	32	2.308	2.134
16	⟨TP=1,PP=8⟩×2	4096	32	4.492	4.247
16	⟨TP=2,PP=1⟩×8	2048	8	2.414	2.127
16	⟨TP=2,PP=1⟩×8	4096	8	4.297	3.922
16	⟨TP=2,PP=2⟩×4	2048	16	2.611	2.432
16	⟨TP=2,PP=2⟩×4	4096	16	4.612	4.294
16	⟨TP=2,PP=4⟩×2	2048	32	2.718	2.616
16	⟨TP=2,PP=4⟩×2	4096	32	4.915	4.548
16	⟨TP=2,PP=8⟩×1	2048	64	3.040	2.967
16	⟨TP=2,PP=8⟩×1	4096	64	5.391	5.221
16	⟨TP=2,PP=8⟩×1	8192	64	10.611	9.956
16	⟨TP=4,PP=1⟩×4	2048	16	3.395	4.040
16	⟨TP=4,PP=1⟩×4	4096	16	5.608	5.198
16	⟨TP=4,PP=1⟩×4	8192	16	10.530	9.956
16	⟨TP=4,PP=2⟩×2	2048	32	3.626	4.447
16	⟨TP=4,PP=2⟩×2	4096	32	5.911	5.494
16	⟨TP=4,PP=2⟩×2	8192	32	11.143	10.634
16	⟨TP=4,PP=4⟩×1	2048	64	3.793	4.637
16	⟨TP=4,PP=4⟩×1	4096	64	6.255	5.939
16	⟨TP=4,PP=4⟩×1	8192	64	11.770	11.139
16	⟨TP=8,PP=1⟩×2	2048	32	5.691	8.494
16	⟨TP=8,PP=1⟩×2	4096	32	8.649	8.589
16	⟨TP=8,PP=1⟩×2	8192	32	14.769	13.770
16	⟨TP=8,PP=1⟩×2	16384	32	29.271	28.054
16	⟨TP=8,PP=2⟩×1	2048	64	5.887	8.693
16	⟨TP=8,PP=2⟩×1	4096	64	9.032	9.028
16	⟨TP=8,PP=2⟩×1	8192	64	15.464	14.669
16	⟨TP=8,PP=2⟩×1	16384	64	30.468	29.299

D EXPLANATION OF TIME COST MODELING

The results on the right side of Figure 10 demonstrate that our cost model is accurate. However, due to the space constraint, §2.2 does not present the details of our cost model. Here we explain its design in detail.

Following previous works [40, 75], our cost model is built on top of offline profiling and curve fitting. For an input batch of size (b, s), where b and s represent the batch size and sequence length respectively, the training time $t(b, s)$ is composed of forward and backward passes. The running time for each pass is proportional to b, and from the perspective of modules, the running time of the attention mechanism is proportional to the square of s, whilst for other modules, it is proportional to s. Therefore, we construct the cost model as a function that is quadratic with respect to s and proportional to b. This model is fitted using offline profiling data for various (b, s) pairs, enabling us to accurately estimate the forward and backward times for an input batch. Furthermore, since modern large models (e.g., LLMs) typically consist of identical layers, we simplify and expedite the offline process by profiling only a single layer.

In practice, due to the memory limit and/or the use of pipeline parallel, training usually consists of multiple chunks (i.e., micro-batches), so it is necessary to construct a time cost function for the overall training process. We introduce our time cost function design in two stages: with and without pipeline parallel.

For training without pipeline parallel, the time cost is solely composed of forward and backward computation. Given R sequence buckets and d_j sequences of length s_j in the j-th bucket, for parallel configuration $\mathcal{S}$ with a maximum supportable sequence length M, the time cost function can be formulated as:

$$\begin{aligned} T(\{d_j\}_{j=1}^{R}; \mathcal{S}) &= \sum_{j=1}^{R} \left(m_j \cdot t(b_j, s_j) + t(r_j, s_j)\right) \\ \text{where} \quad d_j &= m_j \cdot b_j + r_j \\ b_j &= \lfloor \frac{M}{s_j} \rfloor \end{aligned} \tag{10}$$

When pipeline parallel is employed, the training batch is evenly divided into multiple micro-batches, and the forward and backward passes across micro-batches are executed on different stages in a pipeline manner. This process is composed of three stages: warm-up, steady, and cool-down. The time cost includes additional overhead such as data transfer and synchronization between pipeline stages. Specifically, in 1F1B pipeline parallel with p stages, if a mini-batch of size (b, s) is evenly split into m micro-batches, the time cost can be formulated as:

$$T(b, s, m) = \underbrace{m \cdot t(\frac{b}{m}, s)}_{\text{Compute Time}} + \underbrace{(p-1) \cdot t(\frac{b}{m}, s)}_{\text{Bubble Time}} \tag{11}$$

where the former represents computation time and the latter represents pipeline bubble time.

However, dynamic bucketing introduces challenges due to variable-length input. As presented in Figure 13, the critical path of a variable-length pipeline does not have a fixed paradigm as fixed-length pipeline parallel where the length is consistent across micro-batches, making precise time cost estimation more complicated. Additionally, variable-length pipelines introduce additional bubbles caused by the imbalanced time costs of micro-batches, resulting in increased idle time.[16]

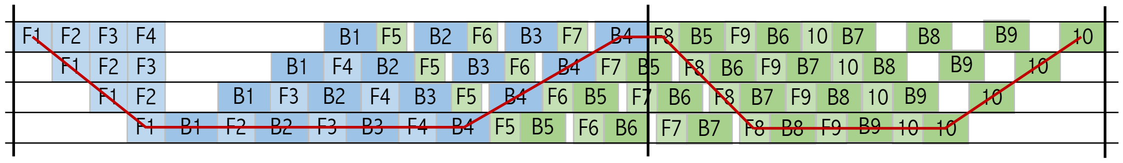


Figure 13: Illustration of the 1F1B pipeline parallel execution with variable-length inputs. Cells in light background color and dark background color represent forward and backward processes, respectively. The blue and green indicate two kinds of micro-batches that are different in length. The critical path of the execution is highlighted in red.

To reduce fragmented bubble time caused by variable-length inputs and to simplify time cost estimation, we propose sorting micro-batches in descending order of time cost and partitioning the critical path based on length for phased estimation. For the first and the most time-consuming micro-batches, the critical path includes the warm-up, steady, and cool-down stages of these micro-batches, which we define as the first phase. For the remaining micro-batches, the critical path is partitioned according to their length.

As shown in Figure 13, the time cost of the first phase is expressed as $m_1 \cdot t(b_1, s_1) + (p-1) \cdot t(b_1, s_1)$, accounting for both the compute time of micro-batches and the bubble time estimation of the entire pipeline. Since micro-batches with different lengths have interleaving

[16] In fact, reducing pipeline bubbles for the training of variable-length data is important and meaningful work. However, it is orthogonal to the goal of our work. And any optimization regarding the reduction in pipeline bubbles can be integrated with our work.

characteristics in the pipeline, the time cost of each remaining phase does not exceed the computation time of micro-batches with the same length, formulated as $m_i \cdot t(b_i, s_i)$.

Putting the above together, to effectively estimate the time cost for the variable-length case, we formulate the time cost function as follows:

$$\begin{aligned} T(\{d_j\}_{j=1}^{R}; \mathcal{S}) = & \underbrace{\sum_{i=0}^{R-1} (m_i \cdot t(b_i, s_i) + t(r_i, s_i))}_{\text{Compute Time}} \\ & \underbrace{+ (p-1) \times \max_{0 \le j \le R-1} \{t(b_j, s_j), t(r_j, s_j)\}}_{\text{Bubble Time}} \\ \text{where} \quad & n_i = m_i \cdot b_i + r_i \\ & b_i = \lfloor \frac{M}{s_i} \rfloor \end{aligned} \tag{12}$$

It is worth noting that our time cost function is compatible with the previously mentioned time cost function without pipeline parallel and with fixed-length pipeline parallel.

Last but not least, we wish to highlight that our work is applicable as long as the time cost function $T(\{d_{\cdot,j}\}_{j=1}^{r}; \mathcal{S})$ is linear w.r.t. $d_{\cdot,j}$. Either solving the ILP or MINLP problems mentioned in §4.2 and §4.3, $\{d_{\cdot,j}\}_{j=1}^{r_j}$ serve as the decision variables. Meanwhile, the objective is to minimize the maximum time of all FT replicas, presented as objective variable $\hat{t} \ge T\left(\left\{\left\lceil \frac{d_{i,j}}{p_i} \right\rceil\right\}_{j=1}^{r_i}; \mathcal{S}_i\right)$ that holds for all configurations $\{\mathcal{S}_i\}_{i=1}^{S}$. The linearity between the time cost function and $\{d_{\cdot,j}\}$ ensures that the constraint of the objective variable $\hat{t}$ remains linear, thereby preserving the ILP or MINLP properties of the problem formulation.