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 conducted 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

$$\begin{aligned}(N_1t'_1 + N_2t'_2) - (N_1t_1 + N_2t_2) &= (\frac{N_1(M_1-M)}{T_1} + \frac{N_2(M_2+M)}{T_2}) - (\frac{N_1M_1}{T_1} + \frac{N_2M_2}{T_2}) \\ &= M \times (\frac{N_2}{T_2} - \frac{N_1}{T_1}) \\ &= M \times (\frac{1}{\overline{T}_2} - \frac{1}{\overline{T}_1}) \\ &\ge 0\end{aligned} \tag{7}$$

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 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 10. 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 11. 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.

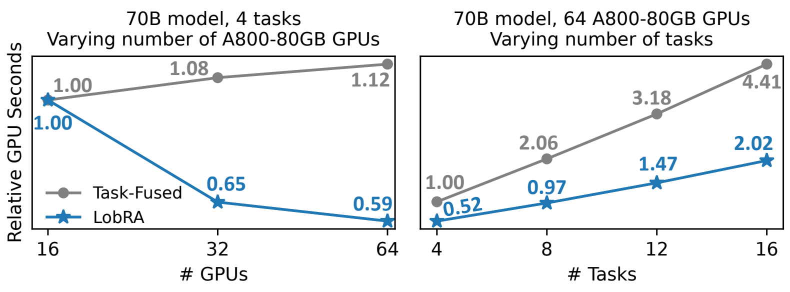


Figure 10: Scalability w.r.t. numbers of GPUs and tasks (70B model).

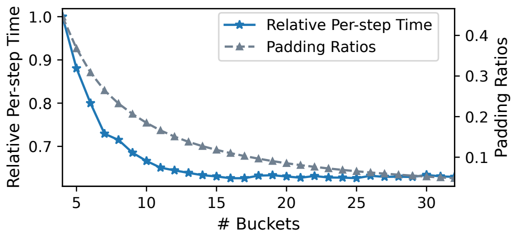


Figure 11: Impact of number of buckets (i.e., R) in dynamic bucketing to the per-step time and padding ratios (7B model, 16 A100-40GB GPUs). The per-step time is scaled by that with 4 buckets.

Effectiveness of Configuration Planning. To begin with, we evaluate the effectiveness of configuration planning by solving the deployment plan of the 70B model and 4-task joint FT. The 4-task datasets are consistent with those used in the GPU scalability evaluation, detailed in Appendix B.3. The results in Table 5 highlight the effectiveness of configuration pruning. Specifically, the parallel configurations solved via configuration pruning are consistent with those from solving the original MINLP problems. Furthermore, while the configuration proposal method accelerates solving the original MINLP problems, it encounters significant timeout issues when the number of GPUs exceeds 48. In contrast, configuration pruning efficiently delivers results within the time limit, 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 joint FT scenarios well.

Table 5: Time cost (in seconds) of configuration planning via different approaches. 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	MINLP w/o Config. Proposal	MINLP w/ Config. Proposal	Config. Pruning	Deployment Plan
32	7.63	3.29	1.71	⟨2,4⟩×1, ⟨4,2⟩×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

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 baseline are presented in Table 6. 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 10, we focus on a subset of 4 tasks: Evol-Instruct, CommitPackFT, BillSum, and PubMedQA. Table 8 and Table 7 present the parallel configurations of Task-Fused and LobRA under varying GPU counts and task numbers.

Table 6: 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	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 7: 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 8: 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 NVIDIA NeMo [34] 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 9, demonstrate that LobRA delivers comparable performance against NeMo when training with the same homogeneous parallel configurations.

Last but not least, when we try to compare LobRA (with homogeneous FT replicas) with mLoRA [55], which incorporates the joint FT of multiple LoRA adapters with pipeline parallel, we find that mLoRA suffers from extremely poor efficiency. A major reason is that mLoRA does not incorporate FlashAttention [5, 6], a de facto implementation for the attention mechanism in Transformer models. This leads to slow attention computation and makes the memory consumption quadratic w.r.t. the sequence length (which is also indicated in Equation (5) of the technical report of mLoRA [54]). Consequently, mLoRA encounters out-of-memory errors for the experiments in our end-to-end evaluation. Even if we restrict the maximum sequence length to be 512 (which is used in their technical report [54]), mLoRA is still much slower than LobRA (with homogeneous FT replicas) due to the inefficient kernels and memory management.

Considering the fact that the primary goal of our evaluation is to examine the effectiveness of the two key designs, i.e., the deployment of homogeneous FT replicas and the workload-balanced data dispatching, as well as the fact that none of the existing frameworks have supported such designs, we do not involve NeMo and mLoRA throughout the experiments in §5 and Appendix B.

Table 9: 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 9 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 [29, 61], 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, while 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:

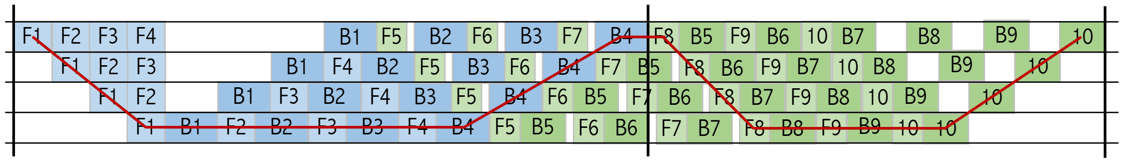
$$\begin{aligned} T(\{d_j\}_{j=1}^{R}; \mathcal{S}) &= \sum_{j=1}^{R} \left(m_j \cdot t(b_j, s_j) + t(r_j, s_j) \right) \\ \text{where} \quad d_j &= m_j \cdot b_j + r_j \\ b_j &= \lfloor \frac{M}{s_j} \rfloor \end{aligned} \tag{10}$$

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 12, 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]



F1 F2 F3 F4 B1 F5 B2 F6 B3 F7 B4 F8 B5 F9 B6 10 B7 B8 B9 10
F1 F2 F3 B1 F4 B2 F5 B3 F6 B4 F7 B5 F8 B6 F9 B7 10 B8 B9 10
F1 F2 B1 F3 B2 F4 B3 F5 B4 F6 B5 F7 B6 F8 B7 F9 B8 10 B9 10
F1 B1 F2 B2 F3 B3 F4 B4 F5 B5 F6 B6 F7 B7 F8 B8 F9 B9 10 10

Figure 12: 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 12, 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:

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

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.