DLoRA: Distributed Parameter-Efficient Fine-Tuning Solution for Large Language Model

immediate

Abstract

To enhance the performance of large language models (LLM) on downstream tasks, one solution is to fine-tune certain LLM parameters and make it better align with the characteristics of the training dataset. This process is commonly known as parameter-efficient fine-tuning (PEFT). Due to the scale of LLM, PEFT operations are usually executed in the public environment (e.g., cloud server). This necessitates the sharing of sensitive user data across public environments, thereby raising potential privacy concerns. To tackle these challenges, we propose a distributed PEFT framework called *DLoRA*. DLoRA enables scalable PEFT operations to be performed collaboratively between the cloud and user devices. Coupled with the proposed Kill and Revive algorithm, the evaluation results demonstrate that DLoRA can significantly reduce the computation and communication workload over the user devices while achieving superior accuracy and privacy protection.

1 Introduction

Large Language Models (LLMs) have recently incited substantial public interest. Their ability to grasp context and nuance enables them to handle natural language processing (NLP) tasks such as text generation (Brown et al., 2020; Zhuang et al., 2023), translation (Zhu et al., 2023; Hadi et al., 2023) and summarization (Zhang et al., 2023b) with remarkable proficiency. Because of the extensive number of parameters in LLMs and the substantial computational workload during their operations, LLMs are usually implemented on nodes with rich compute resources such as cloud servers (OpenAI and Microsoft; Badr, 2023). During operation, users send their data to the cloud server for LLM processing, after which the LLM results are transmitted back to the user devices.

Previous studies (Brown et al., 2020) has shown that LLMs can extend their learned knowledge to novel tasks not seen during the training phase, a phenomenon commonly referred to as *zero-shot* capability. However, fine-tuning still remains essential to enhance LLM performance on unseen user datasets and tasks. Due to its scale, a widely adopted strategy for fine-tuning LLMs involves adjusting a limited number of LLM parameters while keeping the remainder unchanged. This approach, termed *parameter-efficient-fine-tuning (PEFT)*, adds small modules of parameters to predefined positions of the pre-trained LLM and only fine-tunes these modules (Houlsby et al., 2019; Guo et al., 2020; Mao et al., 2021; Karimi Mahabadi et al., 2021a) over the downstream tasks to better adapt to the user data.

While PEFT presents an efficient approach for improving LLM performance, it also poses significant challenges for system deployment. To deploy PEFT, one potential solution involves transferring user input to the cloud, and the entire PEFT process is performed over the cloud servers. This scheme is referred to as *Cloud-only* solution (Figure 1 (a)). However, this approach comes with several drawbacks. On one hand, keeping private user data in a shared cloud environment raises immediate privacy concerns. On the other hand, in order to deliver a personalized LLM service, it is necessary to create and fine-tune a separate set of personal LLM parameters using the training dataset from each user. This can result in significant scalability challenges as the user group expands in size. By contrast, another option is to offload the LLM fine-tuning process completely to the user device, presented as *Edge-only* solution in Figure 1 (b), Unfortunately, this approach is often impractical due to the limited computational resources available on user devices.

To mitigate the aforementioned system problems, in this work we propose a distributed PEFT solution named *DLoRA* (Figure 1 (c)) for collaborative PEFT operations between a cloud server and

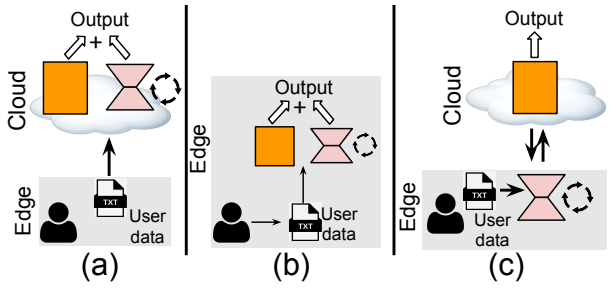

Figure 1: (a) Cloud-only solution. (b) Edge-only solution. (c) DLoRA scheme. The frozen and learnable parameters are shown in orange and red, respectively.

user device. DLoRA eliminates the need to deliver private user data for LLM fine-tuning in the cloud thereby ensuring the personal LLM parameters are stored completely within the user device, thereby minimizing the risk of privacy leakage. Additionally, DLoRA offloads partial computational workload for LLM fine-tuning to user devices, effectively mitigating the scalability issues.

Beyond that, our preliminary studies on conventional PEFT algorithms suggests that the majority of trainable parameters within LLM remain fairly constant throughout the fine-tuning process, with only a small subset of parameters undergoing active changes. This group of changing parameters varies with the training dataset and downstream tasks. Motivated by this observation, we introduced a *Kill and Revive (KR)* algorithm for DLoRA that dynamically identifies and fine-tunes the set of LLM parameters most responsive to the training data, resulting in a substantial reduction in computation and communication workloads on the user derives. Overall, our contribution can be summarized as follows:

- We introduce *DLoRA*, an PEFT framework capable of executing LLM fine-tuning seamlessly between cloud and edge devices. DLoRA ensures the user data and personal parameters to store on user devices throughout the PEFT operation, eliminating the risk of privacy leakage while enabling scalability.

- We introduce the *Kill and Revive (KR)* algorithm for DLoRA, which dynamically identifies and fine-tunes the subset of LLM parameters that are most sensitive to the training data. This approach results in a notable decrease in computational and communication burdens on user devices.

- We evaluate performed an assessment involving three LLM models across eight datasets. The evaluatiaon results indicate that the KR algorithm can deliver an average reduction of 82% in computational load and a 87.5% reduction in communication between the user device and the cloud, while still achieving comparable or even better results than the baseline solutions.

2 Background and Related Work

In this section, we introduce LLM computations in Section 2.1. We then describe the computational flow of PEFT operations in Section 2.2. Subsequently, we introduce two well-known privacy-preserving mechanisms, Federated Learning 2.3 and on-device machine learning, and compare our methods against them.

2.1 LLM Computation

To analyze the computations in LLM execution, we examine LLaMA (Touvron et al., 2023), a renowned LLM known for its superior performance across various NLP tasks (Cui et al., 2023; Roziere et al., 2023; Zhang et al., 2023a). As depicted in Figure 2 (a), LLaMA comprises three main components: an embedding layer, a stack of decoder blocks, and a linear layer. It processes text by transforming user-inputted tokens into numerical vectors via the embedding layer, which are then processed by the decoder layers. The output of the last decoder layer, which generates a probability distribution over the entire *vocabulary*, predicts the next token in the sequence. This token is concatenated with previous tokens for subsequent processing rounds in an auto-regressive manner, producing a full sequence or *completion* as shown in Figure 2 (b). The training process mirrors inference but with sentences compared directly to the ground truth to compute training loss and gradients to minimize loss across the LLM's weights.

2.2 Parameter Efficient Fine-Tuning

Fine-tuning is crucial for adapting LLMs to new tasks. However, it also introduces several challenges, including overfitting and high computational expenses (Houlsby et al., 2019; Guo et al., 2020; Mao et al., 2021; Karimi Mahabadi et al., 2021a; He et al., 2021; Zaken et al., 2021; Valipour et al., 2022). PEFT mitigates these issues by selectively updating a subset of parameters, exemplified by LoRA (Hu et al., 2021) and Adapter tech-

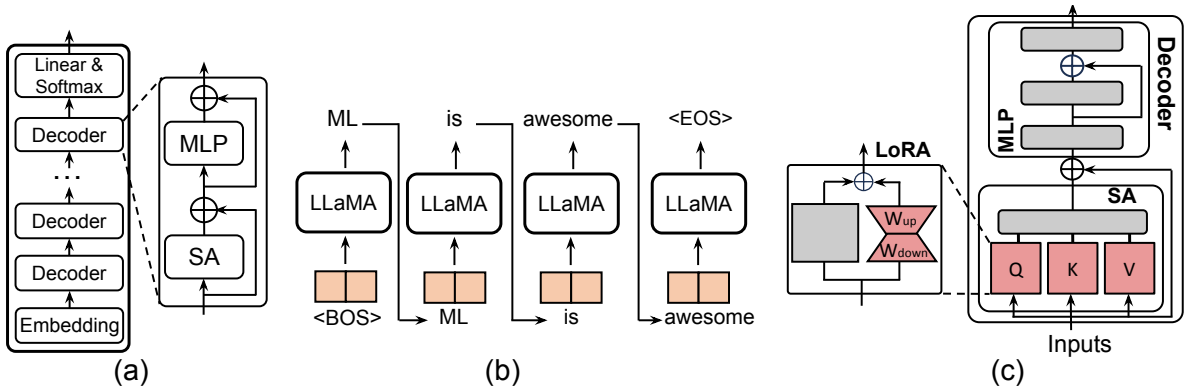

Figure 2: (a) LLaMA architecture. (b) LLaMA auto-regressive pattern. (c) LoRA operation. All the learnable components are highlighted in red, while the frozen components are highlighted in grey. LoRA is applied on all the query, key, and value blocks, we only show one of them for illustration simplicity.

niques (Rücklé et al., 2020; Pfeiffer et al., 2020; Karimi Mahabadi et al., 2021b; Wang et al., 2022; Zhang et al., 2023d; Gao et al., 2023; He et al., 2022). LoRA introduces trainable modules within the SA blocks of LLMs, enhancing the query, key, and value generation with equations:

$$Q = (W_Q^\top + \alpha W_{up,Q}^\top W_{down,Q}) h_{in} \quad (1)$$

$$K = (W_K^\top + \alpha W_{up,K}^\top W_{down,K}) h_{in} \quad (2)$$

$$V = (W_V^\top + \alpha W_{up,V}^\top W_{down,V}) h_{in} \quad (3)$$

where W_{up} and W_{down} are LoRA's weight matrices, h_{in} is the input, and α is a scalar hyperparameter. Adapters insert additional blocks within each decoder's MLP block, featuring a residual connection to mitigate overfitting and computational challenges.

For clarity, we term a block of tunable LLM parameters as a *PEFT module*. For example, the PEFT module for LoRA involves all the learnable parameters within a transformer block. A collection of PEFT modules within a LLM is called *PEFT module pool*. This work configures the PEFT module pool to include the LoRA parameters. However, our approach is compatible with other PEFT schemes (e.g., Adapter).

2.3 Privacy Preserving Solution

As fine-tuning downstream tasks is always associated with personalized data, privacy issues have been a problem that has raised attention; federated learning and on-device machine learning are two commonly used mechanisms.

Federated Learning (FL) (McMahan et al., 2017) has emerged as a groundbreaking machine learning approach, enabling powerful models to be created by leveraging decentralized data sources while respecting user privacy. FL aims to collect data from different users without revealing their data to develop a super-model. In contrast to FL, which conducts model training exclusively within edge devices, DLoRA introduces a collaborative distributed training framework between a single edge device and cloud servers.

Additionally, DLoRA can also integrate with FL, facilitating fine-tuning processes across multiple edge devices. In our DLoRA system, the intermediate results are the only data transferred from personal devices and the centralized cloud. The embedding layer results are located in the user device and can not be recovered unless the attacker steals the tokenizer and embedding layer weights.

Several studies (Rakin et al., 2022; Zhu et al., 2021) explore utilizing activation recovery to extract data. However, attacks targeting intermediate results can be mitigated by employing Fully Homomorphic Encryption (FHE), as proposed in (Zhang, 2022; Zhang et al., 2023c). By leveraging FHE, users can perform computations on cloud servers without exposing their intermediate results, ensuring the security of their data. This approach offers a reliable safeguard under this assumption.

Another approach is to deploy a machine learning model exclusively on a personalized device. A pretrained large-scale model is first compressed and then deployed to a computationally constrained device to prevent the exposure of personal data to the public cloud. While the compressed model may experience reduced accuracy and fine-tuning quality across general tasks, we also evaluate our DLoRA system against an on-device privacy-aware solution in Section 2.3.

Limitation The incorporation of FHE into DLoRA improves data security but also introduces

higher computational overhead. This increased demand for computation consumes significant resources and time on both user devices and cloud servers, potentially degrading the efficiency benefits gained from DLoRA.

3 Kill and Revive Mechanism

In this section, we describe *Kill and Revive* (KR) algorithm in detail, which aims to minimizing both computational and communication burdens on user devices during PEFT computations. We will begin by discussing the *Early-Kill* mechanism in Section 3.2, which is a simple yet effective approach to search for a minimal set of tunable parameters and eliminate redundant finetuning operations with negligible impact on accuracy. Following that, we will introduce the parameter revival mechanism in Section 3.3, which selects and reactivates a subset of previously frozen parameters, further enhancing the LLM accuracy.

3.1 Computation Pattern for DLoRA

To begin with, we first illustrate the computational pattern for a single round of DLoRA operation. As depicted in Figure 4, this procedure can be divided into two phases: forward propagation and backward propagation, which are highlighted in grey and blue in Figure 4, respectively. Initially, the user data is first processed by the embedding layer, with the outputs of the embedding layer sent to the cloud for forward propagation across rest layers. This approach inherently mitigates privacy risks by keeping user data local during the PEFT operation, with only the text embeddings being transmitted to the cloud server for subsequent processing. While recent research attempts have been made to reverse text embeddings to retrieve the original text (Pan et al., 2020; Morris et al., 2023), these efforts predominantly operate under the assumption that attackers have unlimited access to query the text embeddings model. However, in our scenario, this assumption is unrealistic because the text embedding model is implemented within user devices, and will deny all the external query attempts. Subsequently, the results from the frozen LLM blocks are sent back to the user devices for forward propagation across the PEFT modules, whose weights are stored on the user devices. This process continues until the final LLM output is generated. Consequently, the computation and communication overhead on user devices scales proportionally with the number of PEFT modules.

Algorithm 1 KR Algorithm (simplified version)

Inputs: LLM module pool F; Total number of layer L; Selection criteria ϵ. Total finetuning epoch E;
▷ Pre-tuning phase
Tune all PEFT modules within F for several iterations to collect statistics.
Record the changes on l_2 norms for each module, define the selection criteria.
▷ Main tuning phase
for $0 \leq e \leq E-1$ **do**
  for $l \in L$ **do**
    if l_2 norm change on $l-th$ PEFT module less than ϵ **then**
      Frozen the PEFT module at layer l.
    else
      Activate l-th PEFT module.
  Finetune all active PEFT modules.
  Recalculate ϵ.

Likewise, the backward propagation begins with comparing the LLM output with the ground truth output, which further producing the gradient for the last LLM layer. These gradients are subsequently employed to compute output gradients for earlier layers, progressing until a PEFT module is reached. The gradients are then transmitted back to the user device for backward propagation and weight updates. This process continues until all the weights within the PEFT module have been updated. Likewise, as in the forward propagation scenario, the computational and communication overhead during backward propagation also scales in proportion to the number of PEFT modules.

3.2 Early Kill Mechanism

Considering the computation pattern outlined in Section 3.1, an simple strategy for reducing training cost over the user device is to simply reduce the amount of the PEFT modules. To achieve this while preserving the accuracy, DLoRA dynamically identifies the most relevant and significant PEFT modules that contribute most to downstream task accuracy, and only finetune these modules. To investigate the significance of PEFT modules towards training accuracy, we conduct experiments to evaluate the importance of each PEFT module using multiple training datasets. Specifically, we configure the PEFT module pool to comprise all the learnable parameters outlined in LoRA (Sec-

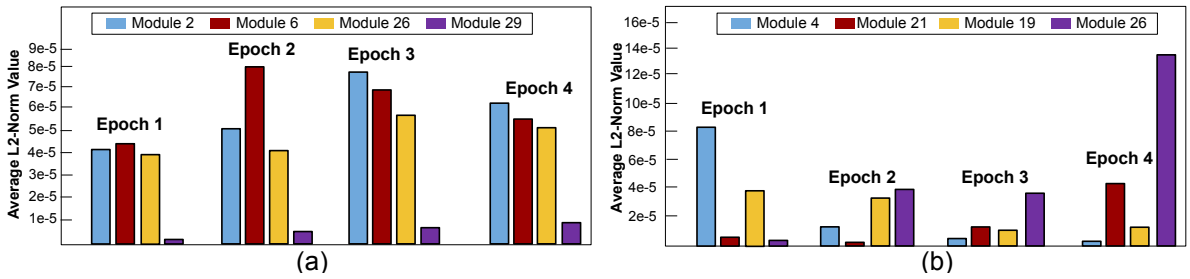

Figure 3: l_2-norm variation of selected PEFT modules across training iterations over multiple downstream tasks including (a) Arc-Challenge, (b) Social-QA.

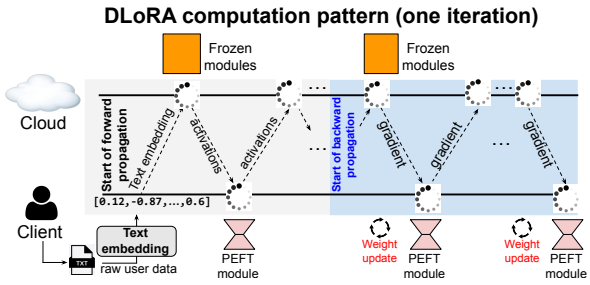

Figure 4: DLoRA computation pattern for one iteration.

tion 2.2), and make all the PEFT modules in the pool learnable. We then record the weights changes for different PEFT modules during the finetuning process. Figure 3 presents the variation on PEFT module magnitudes during the PEFT process over two downstream tasks. We notice a great variation on the PEFT module magnitudes across the finetuning iterations. For instance, in Figure 3 (a), PEFT module 6 (highlighted in red) exhibits an l_2-Norm value exceeding $16\times$ the average l_2-norm. This observation suggests that the parameters within this module undergo substantial changes during the PEFT process. We refer to the PEFT modules with substantial changes as *active PEFT modules*. On the contrary, PEFT module 29 (marked as purple) in Figure 3 (a) has a low l_2 norm value through the whole tuning process, which shows it exhibits no noticeable changes. We refer to blocks like that as *idle PEFT modules*. The Early Kill (EK) mechanism is designed to dynamically detect and freeze the idle PEFT modules to reduce computation and communication load on user devices. The EK mechanism comprises two phases, which are explained in detail below:

Pre-tuning Phase: In this phase, all the PEFT modules within the PEFT module pool are configured to be learnable. A short preliminary fine-tuning is performed and the changes on the weight magnitudes for each PEFT modules are recorded. This change naturally reflects the degree of activities of each PEFT module.

Main tuning phase: We rank all the PEFT modules base on the magnitude differences recorded during the Pre-tuning phase. PEFT modules with magnitude change smaller than a predefined threshold are identified as idle PEFT module, which are then frozen ('killed') to enhance compute efficiency of user device.

3.3 Parameter Revival Mechanism

The EK technique, detailed in Section 3.2, enables us to selectively update only the active PEFT modules, resulting in substantial reductions in computation and communication for user devices. However, our evaluation results indicate a noticeable LLM accuracy drop when EK mechanism is applied. To better understand the reason, we analyze the magnitude variation across all the PEFT modules in LLaMA-7B model (Touvron et al., 2023) over social-qa (Sap et al., 2019) dataset. As depicted in Figure 3 (b), the PEFT module 26 (marked as purple) changes greatly in the 4th epoch while its parameters stay steady in the first three epochs. The PEFT module 4 (marked as blue) changes greatly in the first epoch and becomes stable after that. This phenomenon indicates that the active PEFT modules in the previous epoch may no longer be active in the later epochs, while the idle PEFT modules can become active in the later epochs.

The fluctuations in the weight magnitude presented in Figure 3 (b) indicate the need for regularly reviving the PEFT modules that were killed previously, as they could have a significant impact on LLM accuracy in the later stage of finetuning process. Based on this observation, we propose a *Kill and Revival* (KR) Algorithm (Algorithm 1). Specifically, at the end of each epoch of fine-tuning, we rank all PEFT modules according to their magnitude changes within the epoch and kill the idle PEFT modules (Line 9 in Algorithm 1). In addition,

we also pick a subset of killed PEFT modules and reactivate them (described in algorithm 1 line 13). The selection criteria for PEFT modules revival are determined by their l_2 norm changes during the last epoch in which they were active.

While alternative criteria are feasible, we have noticed that employing the l_2 distance in the KR algorithm produces outstanding performance. We maintain a constant number of active PEFT modules to restrict the computation cost across the entire finetuning process. By adhering to this *computation budget*, we ensure that the computation and communication costs of the user device remain consistent throughout the PEFT operation.

4 Evaluation

In this section, we provide a comprehensive evaluation of the KR algorithm and DLoRA System. We begin by describing the evaluation setup in Section 4.1. Next, we assess the accuracy and system performance of the KR algorithm across multiple tasks and LLMs in Section 4.2. Subsequently, we perform multiple ablation studies in Section 4.3.

4.1 Experiment Setup

Datasets and models: We evaluate our KR algorithm on three popular LLMs, including OPT-6.7B (Zhang et al., 2022), BLOOM-7B (Scao et al., 2022), and LLaMA-7B (Touvron et al., 2023). Additionally, we conduct evaluations of the KR algorithm across a range of tasks, including Question and Answering tasks such as OpenBookQA (OBQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), Social IQa (SIQA) (Sap et al., 2019) and BoolQ (Clark, 2019), problem compilation and concluding tasks including Winograde (ai2, 2019) and HellaSwag (Zellers et al., 2019) and multi-choice science questions such as ARC-easy and ARC-challenge (Clark et al., 2018).

PEFT Settings: Experiments utilized the HUGGINGFACE library (Huggingface, 2016) with the AdamW optimizer (Pytorch, 2023) and a cosine learning rate scheduler. Evaluations of backbone model were conducted on an Nvidia A100-SMX4 with 40GB memory (NVIDIA, 2021), using CUDA version 11.6. Fine-tuning spanned five epochs across downstream tasks, maintaining 16 active PEFT modules ($B = 16$) per Algorithm 1. All data were processed in floating-point precision, with plans to investigate the impact of quantization.

DLoRA System Configuration: To measure the system performance of DLoRA in a practical environment, we have built a testbed with an Nvidia Jetson Xavier (NVIDIA, 2022) device and a cloud server to simulate the cloud and edge environment. We then measure the processing latency of the DLoRA system across various tasks.

Baselines: To fairly evaluate the accuracy performance of the KR algorithm, we compare it with a baseline algorithm termed the Fully-tune (FT) algorithm, which keeps all the PEFT modules active throughout the entire finetuning process. In comparison, KR algorithm applies the Early kill mechanism described in Section 3.2 with the parameter-revival mechanism discussed in Section 3.3. The purpose of this baseline is to evaluate the importance of the parameter revival mechanism over the accuracy performance. We configure the PEFT module pool for the KR algorithm to include all the LoRA parameters. To be noted here, our question-answering experiments are without any promotion, and we selected 25% of the dataset as training sets and left the rest 75% as testing sets.

4.2 Evaluation Results

Table 1 presents the accuracy results for FT and KR across different LLMs and datasets. KR often achieves similar or even superior performance compared to FT. For instance, in the SIQA task, KR consistently outperforms FT across all LLMs. KR matches or exceeds FT's performance for other datasets while notably reducing compute and communication workloads, as detailed in Figure 5.

The findings indicate that KR significantly enhances training efficiency, substantially reducing computing costs. Additionally, considerable computational savings are observed when applying KR with an Adapter scheme. This efficiency is due to KR's dynamic selection and management of a minimal set of active modules, significantly lowering computational load.

Moreover, Figure 5 shows the total communication during one epoch of the PEFT process for both FT and KR across various tasks. Unlike FT, KR attains an average reduction of 87.5% in communication, highlighting the communication efficiency of the KR algorithm. In summary, KR substantially reduces computation and communication while maintaining comparable accuracy levels to FT.

LLM	Methods	BoolQ	PIQA	SIQA	WinoGrande	OBQA	ARC-easy	ARC-challenge
BloomZ	FT	64.6%	71.0%	75.4% %	60.8%	72.2%	75.4%	45.9%
	DLoRA	64.6% (+0.0%)↑	73.7% (+2.7%)↑	73.2% (+2.2%)↑	65.0% (+4.2%)↑	72.1% (-0.1%)↓	75.1% (-0.3%)↓	44.6% (-1.3%)↓
LLaMA-7B	FT	70.7%	80.9%	75.6%	66.5%	70.0%	65.5%	47.9%
	DLoRA	74.3% (+3.6%)↑	79.7% (-1.2%)↓	77.0% (+1.4%)↑	65.7% (-1.8%)↓	73.8% (+3.8%)↑	71.6% (+6.1%)↑	46.7% (-1.2%)↓
OPT	FT	66.6%	74.4%	72.2%	50.4%	33.8%	46.0%	26.2%
	DLoRA	64.8% (-1.2%)↓	78.0% (+3.6%)↑	72.6% (+0.4%)↑	48.0% (-2.4%)↓	35.4% (+1.6%)↑	46.0% (+0.0%)↑	28.8% (+2.6%)↑

Table 1: Accuracy performance evaluation of FT and DLoRA across all the tasks over different LLMs. The changes on accuracies are also highlighted in green or red.

LLM	Budgets	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	OBQA	ARC-easy	ARC-challenge
Bloom-Z	16 (baseline)	64.6%	70.5%	73.2%	67.5%	65.0%	72.1%	75.1%	44.6%
	8	+7.7%↑	+4.3%↑	+0.8%↑	-4.5%↓	-1.4%↓	-3.3%↓	-1.5%↓	-6.2%↓
	4	-8.2%↓	+3.7%↑	-4.6%↓	-3.7%↓	+8.0%↑	+0.9%↑	-0.1%↓	-3.8%↓
LLaMA	16 (baseline)	74.3%	79.7%	77.0%	73.6%	65.7%	73.8%	71.6%	46.7%
	8	-6.0%↓	-0.4%↓	+1.1%↑	+2.5%↑	+5.6%↑	-2.6%↓	-6.0%↓	+0.5%↑
	4	-8.2%↓	+0.0%	+1.7%↑	+3.8%↑	-6.4%↓	-3.8%↓	-0.6%↓	+0.5%↑
OPT	16 (baseline)	64.8%	78.0%	72.6%	45.0%	48.0%	35.4%	46.0%	28.8%
	8	-1.6%↓	-2.0%↓	+0.0%	+0.6%↑	+3.6%↑	-1.4%↓	-0.8%↓	+1.0%↑
	4	-3.0%↓	-1.6%↓	-5.4%↓	-0.4%↓	+1.4%↑	-3.4%↓	-2.4%↓	-0.8%↓

Table 2: Accuracy performance under different compute budget. For each dataset, we use the accuracy of KR with a compute budget of 16 as the baseline and describe the changes in accuracy relative to it.

Dataset/Method	DLoRA	SparseGPT	Wanda
PIQA	79.7%	73.1%	73.0%
HellaSwag	73.6%	44.8%	43.6%
OBQA	73.8%	62.6%	63.6%
ARC-E	71.6%	30.2%	30.3%
ARC-C	46.7%	24.4%	25.0%
# learnable param.	1.1M	10.41B	10.69B
Peak memory cost	4.2MB	38.74GB	39.82 GB

Table 3: Accuracy performance and learnable parameters compared to different LLM pruning mechanisms. DLoRA outperforms SparseGPT and Wanda in terms of both accuracy and peak memory utilization.

4.3 Ablation Studies

Impact on computation budget To better understand the impact of the computation budget on the accuracy performance, we change the computation budget B from 16 to 8 and 4 and evaluate the accuracy performance. As shown in Table 2, there is a general trend of accuracy degradation as the computation budget decreases, primarily due to the reduction in the number of learnable parameters in the LLM. Interestingly, accuracy improves for specific tasks even when computation budgets are reduced. For example, training Bloom-Z with the PIQA and Boolq datasets outperforms the performance of KR with a computation budget of 16. This suggests that only a smaller subset of PEFT modules are responsive to the downstream tasks specified by the Bloom-Z dataset.

Comparison With LLM compression In addition to fine-tuning (FT), we explore alternative baselines using pruning techniques, specifically SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023), to enhance computational efficiency for PEFT operations on edge devices. These methods implement efficient post-training pruning, each proposing unique selection criteria to optimize sparse LLM accuracy. Our evaluation of Wanda and SparseGPT on a single A100 GPU, as shown in Table 3, reveals that DLoRA surpasses accuracy, peak memory usage, and computational costs.

Compare with FL We also compare our approach with a privacy-preserving distributed system offsite-tuning (Xiao et al., 2023) that utilizes model distillation to facilitate personalized fine-tuning of LLMs. As table 4 demonstrates, our Bloom-Z model outperforms this system in four of five datasets.

Table 4: Performance compared to privacy-preserving system

Method\Dataset	**OpenQA**	**PIQA**	**ARC-E**	**ARC-C**	**HellaSwag**
DLoRA	72.1%	73.7%	69.8%	45.8%	67.5%
Offsite-Tuning	29.6%	74.6%	66.8%	36.8%	48.3%

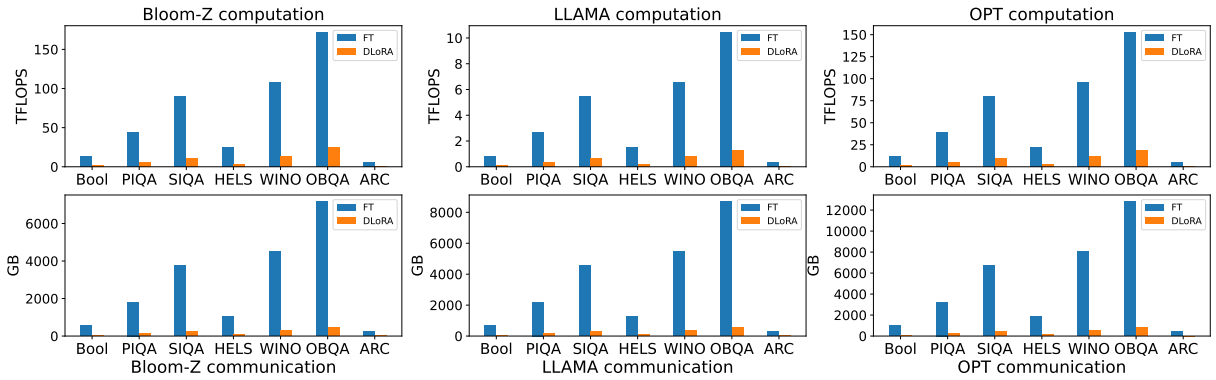

Figure 5: Computation costs of KR and FT over different LLMs, the measurement denotes the computation cost in TFLOPs. The communication costs for KR and FT are measured in Gigabytes (GB).

LLM	Methods	BoolQ	PIQA	OBQA	ARC-challenge
BloomZ	FT	66.0%	78.2%	65.4%	44.2%
	DLoRA	70.0% (+3.4%)↑	76.2% (-2.0%)	72.0% (+6.6%)	45.8% (+1.6%)
LLaMA-7B	FT	63.7%	71.0%	70.0%	45.3%
	DLoRA	67.2% (+3.5%)↑	79.4% (+8.4%)↑	73.1% (+3.1%)↑	44.0% (-1.3%)↓
OPT	FT	62.8%	76.8%	41.0%	26.0%
	DLoRA	67.4% (+4.6%)↑	79.4% (+2.6%)↑	43.8% (+2.8%)↑	26.0% (+0.0%)↑

Table 5: Accuracy performance evaluation with Adapter across all the tasks over different LLMs.

LLM	Compute Budget	BoolQ	PIQA	SIQA
Bloom-Z	Baseline	64.6%	70.5%	73.2%
	Q=8, B=16	+1.8 %↑	+5.1%↑	+1.6% ↑
	Q=8, B=8	+1.6% ↑	+4.6%↑	-0.6%↓
	Q=8, B=4	+0.2%↑	+3.6%↑	-1.8%↓

Table 6: Accuracies under different budget (B) and quantization bitwidth (Q). We present the accuracy variations relative to the baseline setting (B=16, Q=32).

System Latency Measurement Next, we measure the processing latency required to complete one epoch of PEFT for DLoRA over the actual edge device. The measurement reveals that with LoRA finetuned using FT, a single epoch of PEFT over Bloom-Z on OpenbookQA will consume 200.72s. This latency includes the processing time on both the cloud server and the user device. In contrast, with the DLoRA, a single epoch of PEFT only takes 182.59 seconds. This is attributed to the fact that DLoRA greatly reduces the computational workload on the user device, resulting in a decrease in overall processing latency. Table 3 shows that our DLoRA system outperforms other baseline algorithms in accuracy and peak memory usage.

DLoRA on Adapter We evaluate DLoRA performance with Serial Adapter (Hu et al., 2023) over different downstream tasks and LLMs. All the settings are kept the same as those described in Section 4.1. The evaluation on accuracies are presented in Table 5. We notice that DLoRA also outperforms FT over multiple tasks and LLMs, demonstrating the generalizability of DLoRA across various fine-tuning schemes.

Impact of Quantization Precision By default, the activation and gradient matrices exchanged between the cloud and the user device are encoded as a 32-bit floating-point number. To further reduce communication costs, we implement low-precision quantization on the transmitted data, mapping the original full-precision numbers to their nearest quantized values. To demonstrate the impact of quantization precision, we employ 8-bit precision to quantize all communication between the cloud and the user device during the PEFT, compared with floating-point precision; this will further lead to $4\times$ saving on communication overhead. The results across various datasets under different computer budgets are presented in Table 6. It is observed that the accuracies experience a modest decrease.

5 Conclusion

The paper introduces DLoRA, a novel distributed solution tailored for efficient PEFT operations spanning cloud and edge devices. DLoRA utilizes the Kill and Revive algorithm to enhance efficiency during the fine-tuning process, resulting in substantial reductions in computational and communication workloads. Experimental findings illustrate that compared to both cloud-only and edge-only solutions, our DLoRA system notably minimizes computation and communication while upholding accuracy and privacy.

References

2019. Winogrande: An adversarial winograd schema challenge at scale.

Mariam Badr. 2023. Unleashing the power of ai: the microsoft and openai partnership.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Christopher et al. Clark. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Yiming Cui, Ziqing Yang, and Xin Yao. 2023. Efficient and effective text encoding for chinese llama and alpaca. *arXiv preprint arXiv:2304.08177*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. 2023. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Muhammad Usman Hadi, R Qureshi, A Shah, M Irfan, A Zafar, MB Shaikh, N Akhtar, J Wu, and S Mirjalili. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *TechRxiv*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. 2022. SparseAdapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2184–2190, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Huggingface. 2016. Hugging face. In *https://huggingface.co/*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021a. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021b. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

John X Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M Rush. 2023. Text embeddings reveal (almost) as much as text. *arXiv preprint arXiv:2310.06816*.

NVIDIA. 2021. Nvidia a100 tensor core gpu. *https://www.nvidia.com/en-us/data-center/a100/*.

NVIDIA. 2022. Nvidia jetson xavier. *https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/*.

OpenAI and Microsoft. 2023. *https://openai.com/blog/openai-and-microsoft*.

Xudong Pan, Mi Zhang, Shouling Ji, and Min Yang. 2020. Privacy risks of general-purpose language models. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1314–1331. IEEE.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Pytorch. 2023. Adamw optimizer. In *https://pytorch.org*.

Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2022. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE symposium on security and privacy (SP)*, pages 1157–1174. IEEE.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 1(2):4.

Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Hang Zhang, Xin Li, and Lidong Bing. 2023a. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*.

Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023b. Summit: Iterative text summarization via chatgpt. *arXiv preprint arXiv:2305.14835*.

Jingyao Zhang, Mohsen Imani, and Elaheh Sadredini. 2023c. Bp-ntt: Fast and compact in-sram number theoretic transform with bit-parallel modular multiplication. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6.

Jingyao et al Zhang. 2022. Inhale: Enabling high-performance and energy-efficient in-sram cryptographic hash for iot. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA. Association for Computing Machinery.

Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. 2023d. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Lingpeng Kong, Jiajun Chen, Lei Li, and Shujian Huang. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*.

Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. 2021. Hermes attack: Steal {DNN} models with lossless inference accuracy. In *30th USENIX Security Symposium (USENIX Security 21)*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*.