PRIVATELORA FOR EFFICIENT PRIVACY PRESERVING LLM *

Yiming Wang, Yu Lin, Xiaodong Zeng, Guannan Zhang
Ant Group
Shanghai, China

ABSTRACT

End users face a choice between privacy and efficiency in current Large Language Model (LLM) service paradigms. In cloud-based paradigms, users are forced to compromise data locality for generation quality and processing speed. Conversely, edge device paradigms maintain data locality but fail to deliver satisfactory performance. In this work, we propose a novel LLM service paradigm that distributes privacy-sensitive computation on edge devices and shared computation in the cloud. Only activations are transmitted between the central cloud and edge devices to ensure data locality. Our core innovation, PrivateLoRA, addresses the challenging communication overhead by exploiting the low rank of residual activations, achieving over 95% communication reduction. Consequently, PrivateLoRA effectively maintains data locality and is extremely resource efficient. Under standard 5G networks, PrivateLoRA achieves throughput over 300% of device-only solutions for 7B models and over 80% of an A100 GPU for 33B models. PrivateLoRA also provides tuning performance comparable to LoRA for advanced personalization. Our approach democratizes access to state-of-the-art generative AI for edge devices, paving the way for more tailored LLM experiences for the general public. To our knowledge, our proposed framework is the first efficient and privacy-preserving LLM solution in the literature[2].

1 Introduction

In the rapidly evolving field of artificial intelligence, Large Language Models (LLMs) have emerged as a powerhouse[1, 2, 3, 4]. Previously unsolvable long-tail problems are effectively tackled by LLMs, such as programming[5, 6], instruction following[3, 7] and real world interaction[8, 9, 10, 11]. To fully harness the potential of large language models, it is crucial to concentrate on the privacy-preserving aspect of LLMs. A critical dimension of this focus is data locality, which implies that user data are not stored on remote cloud servers but are instead kept closer to the user's own environment. This approach not only reduces the risks associated with data breaches and unauthorized access but also aligns with growing global concerns over data sovereignty and user privacy.

However, like all other privacy preserving efforts, practicality of data locality is severely challenged by efficiency in the context of LLMs. For current cloud-only and device-only LLM service solutions, end users are forced to choose between data locality and model scale. Cloud-only centralized solution offers quick and quality generations from large scale models with its sufficient computing power. But the data locality of personal data and personalized parameters are compromised. Alternative decentralized solutions[12] like deploying smaller quantized[13] LLMs ($\leq$ 7B) on edge devices offer superior privacy but at the cost of performance. Firstly, smaller models often fall short in delivering the emergent abilities exhibited by their larger counterparts[14, 4]. Secondly, on edge devices like smartphones, pure device solutions offer markedly low throughput. For instance, a 3-bit quantized 7B model on a flagship smartphone achieves only 58.1 and 8.1 tokens per second for prefill and decode, respectively. The prefill throughout is only 0.5% of single A100 80G GPU, let alone comparing to clusters of high-end GPUs on Cloud. Other efforts such as Federated Learning and Split Learning also face the challenge in efficiency, and do not solve the problem of inference.

After comparing Cloud-based and Device-based solutions, we confront a fundamental question: Can their benefits be synergistically combined to simultaneously achieve data locality and model scale? Our solution is a heterogeneous

*Preprint. Work in progress.
[2]Demo and code is coming to github soon.

distributed system[15] that leverages edge devices' storage for private data and personalized parameters, while utilizes the Cloud for computational enhancement. Model parameters are split across the cloud and edge devices and only unreadable activations and gradients are transmitted to meet the requirement of data locality. The largest challenge to build such heterogeneous systems is to use much smaller connection bandwidth to transmit equal amount of activations in homogeneous setups[16, 17]. The central cloud and edge devices are presumably connected via Internet, the bandwidth of which is significantly smaller than dedicated connection found in homogeneous setup ($1Gbps$ vs $100Gbps$). The low connection bandwidth being the largest challenge, other challenges involve balanced workload distribution to tolerate significant internal performance gap. For example, A100 GPU and flagship chipsets from smartphones can have over 20 times FLOPS difference and 40 times memory bandwidth difference. The hardware performance gap can result in reduced throughput caused by blocking in pipeline.

To tackle the above challenges, we propose PrivateLoRA, a novel Parameter Efficient Fine-Tuning (PEFT)[18, 19, 20] method that exploits low rank of residual activations for communication and workload distribution. Given the fact that low rank transforms on residual activation are enough to adapt transformer models, we make the hypothesis that decomposing one integral low rank transform into three sequential transforms yields comparable adaptation performances. Therefore, PrivateLoRA integrates three sequential low-rank matrices (A, M, B) for weight adaptation (in Figure 1). Non-trainable A and B, deployed on the Cloud, serve as an encoder-decoder duo that condenses residual activations to reduce communication overhead. We term this practice to reduce communication Low Rank Residual Transmission. Trainable M on edge devices transforms residual activations to steer the transformer for personalized outputs. With less than 0.1% of total parameters on edge devices, FLOPS and memory requirement are also largely reduced, yielding closer workload to processing power ratio between the cloud and edge devices.

On the basis of PrivateLoRA, we propose a data locality preserving paradigm for LLM on a heterogeneous distributed system built on central cloud and edge devices. Raw data and personalized parameters M are kept on edge devices throughout training and inference. Only unreadable activations and gradients are transmitted. Thanks to Low Rank Residual Transmission, the communication overhead of activations are reduced by over 95% percent, yielding comparable throughput to cloud-only solutions.

As a result, while totally respecting data locality, PrivateLoRA effectively harnesses large-scale LLMs on the Cloud with limited resources on edge devices and provides good adaptation performance for personalization. PrivateLoRA has been tested across various scales and benchmarks, including GSM8K, MMLU, BoolQ, and HellaSwag. Despite the untrainable A, B and the triplet structure, PrivateLoRA maintains tuning performance on par with the original LoRA. Our throughput estimations reveal that PrivateLoRA surpasses device-only solutions in both inference and training. Utilizing average consumer-accessible network bandwidth and smartphones, PrivateLoRA achieves 175.5 and 26.5 tokens per second on 7B model for generation prefill and decoding, respectively, which are both over 300% of device-only solutions. Additionally, as model scales increase, PrivateLoRA's advantages become more pronounced, achieving over 74% of the throughput of an A100 80G GPU with a 33B model backend. Our work is also orthogonal to previous efforts on efficient transformer inference, thus can be employed together to further boost efficiency.

In summary, the contributions of our paper can be summarized as,

- We propose PrivateLoRA, a novel PEFT method that achieves communication reduction and balanced workload distribution in distributed scenario.
- On the basis of PrivateLoRA, we propose a new LLM service paradigm that heterogeneously distributes LLMs to protect data locality.
- Extensive empirical experiments are carried out to present a comprehensive study on the tuning performance, integrity and scalability of PrivateLoRA.
- Numerical estimations show that with PrivateLoRA, edge devices can achieve a throughput over 3 times of device-only solutions and comparable to running on GPU.

2 Related Work

2.1 PEFT

PEFT methods lowers hardware requirement of model fine-tuning by significantly reducing trainable parameters and consequently optimizer states cached in VRAM. By exploiting the local optimum of a pretrained model, a much smaller solution space brought by reduce trainable parameters helps PEFT methods achieve comparable tuning performance[19, 20]. PEFT can be classified into two categories: 1) reparameterization-based methods[21, 22] that retrain a portion of the parameters and 2) addition-based methods that train additional parameters[23, 24, 25]. Recent

works in PEFT focus on resource efficiency[25, 26, 23, 24]. LoRA[23] fits incremental weights by decomposing them into low-rank matrices. $(IA)^3$ tunes hidden states with learned multipliers.

2.2 Distributed Learning For Data Privacy

Federated Learning (FL) and Split Learning[15] are proposed to tackle the problem of data privacy in a distributed manner. FL allows multiple nodes to locally train a complete neural networks without explicit exchange of local data via specialized optimization algorithms and transmission protocols. FL has been widely applied in various domain such as computer vision, text typing (Google's G-board) and intrusion detection[27, 28]. Split Learning splits model vertically among different nodes so that only activations and gradients are transmitted to protect data locality. In the context of LLMs, efficiency of these methods are highly questionable as both compute and communication are astrological compared to conventional deep neural networks.

2.3 Running LLMs on Edge Devices

Efforts have also been made to run LLMs on edge devices. llama.cpp[3] ports LLaMA model in C/C++ so that models can be efficiently executed on limited hardwares such as laptops and Raspberry Pi. MLC Chat[12] showcases a model compilation solution for deploying language models on diverse hardware backends and applications. With MLC Chat, quantized 7B model can be run on smart phones. As for training LLMs on edge device, PockEngine[29] introduces sparse back-propagation, pruning the backward graph, and updating the model sparsely to save memory and reduce latency while maintaining model quality. PockEngine has demonstrated the capability to fine-tune LLaMav2-7B on NVIDIA Jetson AGX Orin with significant speed and memory efficiency compared to standard TensorFlow.

3 Method

3.1 Problem Formulations

3.1.1 Notations

Two heterogeneous runtimes with huge performance gap are in the scope, namely the cloud (C) and the edge device (D). Cloud has powerful hardware and a large-scale shared autoregressive transformer model $P_\Phi(y|x)$ that produces non-personalized outputs. Edge device has limited hardware and stores private data $Z = \{(x_i, y_i)\}$, where both x_i and y_i are sequences of tokens. For example, Z could be rounds of chat and x_i and y_i denotes messages of senders and receivers. Hardwares are parameterized with FLOPS F and memory bandwidth MB. Two runtimes are connected with Internet with asymmetric network bandwidths denoted as B_{D2C} and B_{C2D}. And we note such hybrid runtime $D\&C$. We obviously have $F_C \gg F_D$ and $MB_C \gg MB_D$. For a paired connection, B_{D2C} and B_{C2D} are usually bound by edge device's Internet Service Provider, which result in $B_{C2D} > B_{D2C}$ but B_{C2D} and B_{D2C} are in the same order of magnitude.

Model $P_\Phi(y|x)$, parameterized by Φ, is composed of an embedding layer, N transformer layers and an LM Head for token prediction. The dimension of hidden states h is noted as d. For simplicity of discussion, we assume M is decoder-only transformer but obviously PrivateLoRA also works with encoder architecture.

3.1.2 Objectives of PrivateLoRA

Low Rank Residual Transmission aims to solve the impossible triangle of performance, parameter-based personalization and data locality becomes for centralized or decentralized paradigm. We then give formal definition on the three objectives.

Performance indicates both task solving capability and processing speed. Since cloud-based solution fails in data locality, the performance to beat is benchmarked from device-only solutions. PrivateLoRA has to outperform device-only solutions from both aspects. PrivateLoRA is capable of leveraging large scale model on Cloud, therefore guarantees to outperform smaller models on edge devices in task solving capabilities.

Task solving capability is measured with conventional benchmark scores. Processing speed is measured by number of processed tokens per second, or tokens per second (TPS) for short. Transformer throughput largely depends on the sequence length and batch size, therefore throughput of generation prefill, generation decoding and batched training will all be considered.

[3] https://github.com/ggerganov/llama.cpp

Parameter-based Personalization achieves personalization by tuning knowledge or preferences into the models. For example, in order to make model mimic receiver's tone, we can tune additional parameters $\Delta\Phi$ with chat records Z while freezing Φ. The objective can be written as,

$$\arg\max_{\Delta\Phi} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log(P_{\Phi+\Delta\Phi}(y_t|x, y_{<t})) \tag{1}$$

Compared to In-Context Learning (ICL) based personalization, the major advantage of parameter-based personalization is the capability to learn unseen concepts. ICL-based personalization also increases sequence length and consequently computation and communication overhead, which may be critical to throughput on edge devices.

Data Locality requires that persistent storage of both raw data Z and its derivative is only allowed on edge devices. Raw data Z is used as input for generation and input-label pair for autoregressive training. Derivatives of data, especially personalized parameters, are necessary in training and inference. Between the two, persistent storage of personalized parameters is more critical for centralized cloud-based solutions. From the perspective of Cloud, storage of raw data is no longer required as long as the personalized parameters are obtained. For PrivateLoRA, this restriction also poses challenges on processing speed as it forces communication between edge devices and the cloud.

3.2 Low Rank Residual Transmission

Given the fact that low rank transform is enough for effective LLM adaptation, PrivateLoRA exploits such low rank for communication and workload distribution. Drawing inspiration from LoRA, PrivateLoRA adapts model weights by adding three sequential low rank matrices parallel to target weight (see Figure 1). Given weight matrix $W \in \mathbb{R}^{d\times k}$ in target linear module and activations $\mathbf{x}$ of dimension d, adaptation of PrivateLoRA can be written as,

$$\mathbf{x}W + \mathbf{x}\Delta W = \mathbf{x}W + \mathbf{x}AMB \tag{2}$$

where $A \in \mathbb{R}^{d\times r_{C2D}}$, $M \in \mathbb{R}^{r_{C2D}\times r_{D2C}}$ and $B \in \mathbb{R}^{r_{D2C}\times d}$. $r_{C2D}, r_{D2C} \ll d$, dimensions of M, are separately noted to account for the asymmetric bandwidth B_{C2D}, B_{D2C}.

Like all PEFT methods, the decoder stack of the model is frozen and only additional parameters are trainable. In the adapted model forward, residual activations are firstly down-projected by A to low rank for fast download, then transformed by M while maintaining its low dimension for upload transmission, and finally up-projected by B to merge residual activations into the base activation. By dividing one integral low rank transform ΔW into three sequential transforms, we could explicitly exploit the low rank for communication. Matrices A, B serve as encoder-decoder pair to condense residual activations for transmission.

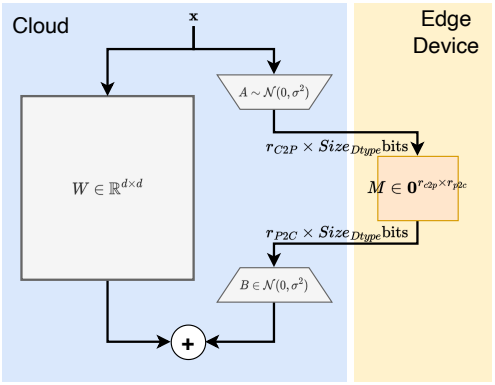
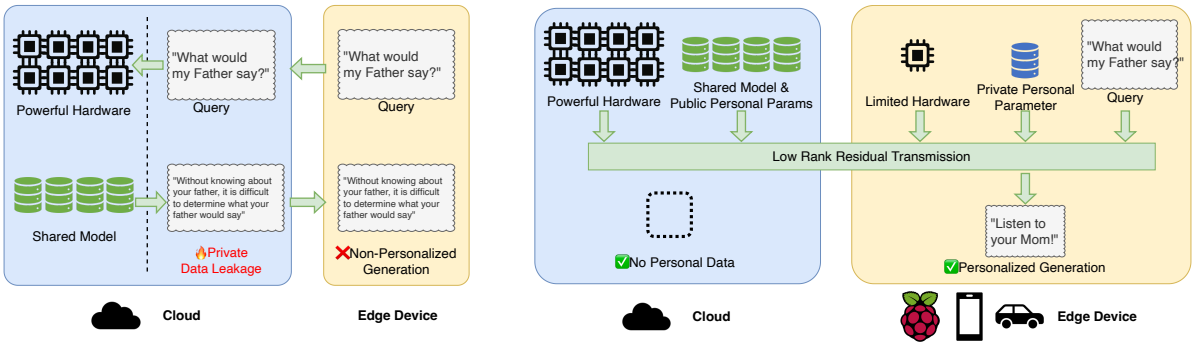


Figure 1: Weight adapted with PrivateLoRA. Adaption matrices A and B are kept frozen after random initialization, thus contain no user information and can be stored and deployed on Cloud. Weight M is trained with user data thus deployed on Phone. Thanks to PrivateLoRA, the communication base multiplier is significantly smaller than model's hidden size and irrelevant to model architecture and scales.

3.2.1 Communication Overhead

PrivateLoRA cuts down communication base from d to r_{D2C} or r_{C2D} and make the transmission base number irrelevant to model scale. Assuming the original LoRA is similarly deployed, transmission overhead of decoder stack would be proportional to the dimension of hidden states d. d is too big as transmission base and also scales up with model scale. For example, d equals 4096, 5120 and 6656 for LLaMA 2-7B, LLaMA 2-13B and LLaMA-30B, respectively. Transmitting hidden states of one token of 16-bit precision on all decoder layers of LLaMA 2-7B produces over 2Mb overhead, let alone multiple back-and-forth transmissions and much longer sequences in actual training and inference. Whereas for PrivateLoRA, assuming $r_{C2D} = r_{D2C} = 128$, the base multiplier is cut down from 4096 to 128, a reduction of 96.88%. Another thing worth mentioning is that r_{C2D}, r_{D2C} is invariant to model architecture, thus don't scale up with model scale. The implication is that when leveraging larger models as personalization backend, the impact of communication overhead is more negligible.



(a) Centralized cloud-based paradigm

(b) PrivateLoRA-powered paradigm

Figure 2: Inference pipeline comparison between (a) centralized Cloud-based paradigm and (b) PrivateLoRA-powered paradigm. PrivateLoRA has the advantages of data locality and personalized generation compared to Cloud-based paradigm. PrivateLoRA leverages powerful Cloud hardware to achieve parameter-based personalization with limited local resource on edge devices. Parameters containing personal data are stored on edge devices and remain on device during training and inference.

3.2.2 Workload Distribution

In PrivateLoRA, workload distribution is largely balanced so that a closer ratio of compute to processing power is achieved for Cloud and Device. On Cloud, apart from the original decoder stack, A, B of PrivateLoRA adds very marginally additional compute. For LLaMA 2-7B, assuming $r_{C2D} = r_{D2C} = 64$, additional parameters take up 1.4% of original model. Also, A, B can be computed on parallel with W, thus no additional latency is introduced. On edge devices, during forward computation of the decoder stack, the computation workload only involves matrix multiplication of small dimensions. For LLaMA 2-7B, the parameter count of M on edge devices take up less than 0.1% of the original model, thus significantly reducing compute and memory pressure. Moreover, the pure linearity of computation can yield higher utilization of the hardware. Softmax operation in self attention is infamously slow due to low hardware utilization[30, 31] and reduces overall throughput in device-only solutions. In our proposed PrivateLoRA, only linear projections are performed on edge devices which are highly optimized from the perspective of both software and hardware.

3.2.3 Data Locality

During training, A and B are non-trainable and only M is optimized with user private data. M is kept on Device during both training and inference. Since A and B are randomly initialized and kept random, they contain no user data can also be stored on Cloud. Therefore, PrivateLoRA meets the requirement that no private data or its derivative are kept on Cloud. Moreover, A, B and B form a tight pairing similar to public-private key used in encryption. Possessing either component do not result in correct model output.

3.2.4 Target Modules of PrivateLoRA

Several researches have pointed out that a thorough adaptation to every linear projection in transformers, including self attention and MLP, yields better overall performances[32] We only target query, key and value projections in self-attention for adaptation to achieve minimized communication. Target modules don't affect time complexity in single forward scenarios such as generation prefill and training, but can affect generation decoding. During iterative sampling in generation decoding, query projection is only calculated with the newly generated token. Key and Value projections can reutilize KV cache to make marginal cost constant. Whereas computation complexity of other linear projections are proportional to sequence lengths including both prompt and newly generated tokens. Adapting these modules make decoding throughout decay due to increasing communication overhead. Therefore, only adapting query, key and value modules produces consistent communication overhead in all scenarios.

3.3 Paradigm Shift Powered by PrivateLoRA

Based on PrivateLoRA, we propose new paradigm that solves the impossible triangle for efficient LLM personalization.
Parameter-based Personalization Private personal parameters M introduced by PrivateLoRA is optimized with

personal data Z. Despite random and static A, B, residual activations can be effectively transformed by M to output personalized generations.

Model Performance PrivateLoRA leverages models in cloud-based solutions as backend to guarantee better task solving capability compared to device-only solutions. Compared to cloud-based solutions, PrivateLoRA excels in providing tailored generations to better suit user demands. In terms of throughput, thanks to PrivateLoRA, communication overhead is significantly reduced. Leveraging large scale LLM with PrivateLoRA can yield higher overall throughput compared to small models with device-only solutions.

Data Locality Private personal parameters M, derivatives of raw data Z, are optimized with personal data but kept on Device in both training and inference. Public personal parameters A and B are randomly initialized and kept frozen, thus not treated as derivatives of raw data. For the locality of raw data, we deploy word embedding and LM Head on Device and only the decoder stack on Cloud (see Figure 3) so that only human-unreadable activations are transmitted (more discussion in Section 6.2). Therefore, data locality of both raw data and its derivative are protected with PrivateLoRA.

3.3.1 Inference and Training Cycle Powered by PrivateLoRA

To fully understand the paradigm powered by PrivateLoRA, we elaborate on the training and inference cycle of a heterogeneously distributed.

Inference LLMs predict new tokens iteratively. Figure 3 plots the iterative sampling with PrivateLoRA. Starting with a query input by end user, the query is firstly tokenized and go through word embedding to get initial token embeddings. These token embeddings are usable to central decoder stack but cannot be mapped back to token ids with embeddings on Cloud. Token embeddings then go through series of decoder layers. In query, key and value projections in each self attention module, PrivateLoRA is applied to get residual activations from Device that ultimately produce personalized generation. At the end of the decoder stack, activations of the last token are transmitted to Device and decoded by LM Head on Device. The newly obtained token are then tokenized and fed through word embedding for another round of sampling.

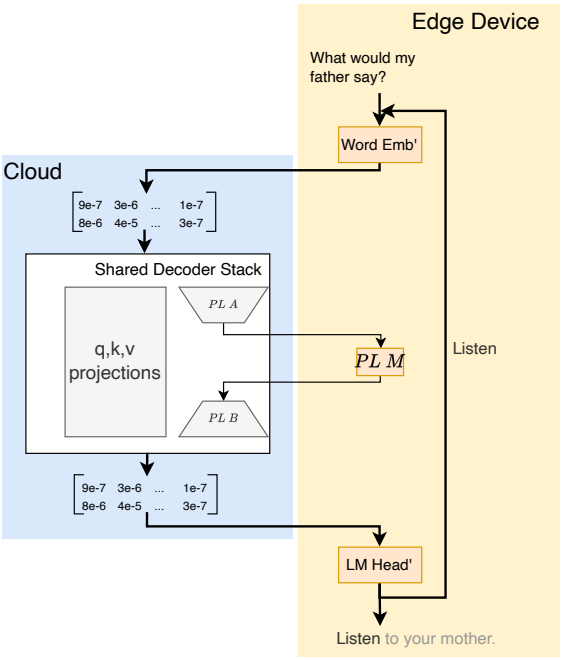


Figure 3: Decode cycle of decoder-only LLM generation with PrivateLoRA. Raw texts are kept on device and only unreadable activations are transmitted. Random and static matrices A, B on Cloud serve as activation encoder-decoder to cut down communication overhead. M on Device steers the residual activations to produce personalized generation.

Training In training cycles, M is tuned with private user data for parameter-based personalization. A, B and the decoder stack is frozen. Training cycle starts with forward computation almost identical to inference except that activations of the entire of the last decoder layer are transmitted to Device. On Device, loss is calculated after feeding received activations to local LM Head. After that, the loss is back propagated through the entire decoder stack. Therefore, throughout training, personal data is always kept on Device and invisible to Cloud.

3.3.2 Memory and FLOPs Analysis

PrivateLoRA is extremely friendly to edge devices with limited computation resources. We numerically estimates required resource and compute to compare deploying complete transformer model and PrivateLoRA counterpart. For memory requirement, we calculate the memory necessary on edge devices to load model parameters into RAM. Total FLOPs on edge devices to complete one forward computation on one token are estimated. The FLOPs are roughly estimated as 2 times of parameter count. Various datatype precisions are considered. And the results are listed in Table 1. With PrivateLoRA, leveraging 7B models of 16-bit datatype only needs 10.6% of memory and 2.0% of FLOPs of 3-bit device-only solutions. Furthermore, PrivateLoRA's efficiency allows it to leverage models up to 30B in size with less memory and total FLOPs than 1B quantized models, underscoring its high resource efficiency.

3.3.3 Throughput Analysis

In this section, we give detailed analysis on throughput in various scenarios. We will decompose the elapsed time of forward computation into several well-known quantity in an end-to-end manner, so that the overall throughput of our proposed heterogeneous distributed system can be numerically estimated in Section 4.1.2.

We firstly start with forward computation and then extend our results to training. For our proposed architecture, time of one forward computation T can be decomposed into time of decoder stack on Cloud $T_C^{Decoder}$, time of LM Head on Device T_{Device}^{LMHead} and overhead introduced by PrivateLoRA $T^{PrivateLoRA}$.

$$T = T_C^{Decoder} + T^{PrivateLoRA} + T_D^{LMHead}. \tag{3}$$

Overhead of PrivateLoRA $T^{PrivateLoRA}$ can be decomposed into network transmission t and local execution time on Cloud T_C^{LRRT} and Device T_D^{LRRT}.

$$T^{PrivateLoRA} = T_{Network}^{PrivateLoRA} + T_D^{LRRT} + T_C^{LRRT}. \tag{4}$$

Since network communication is more critical in our heterogeneous distributed system, we then focus on dissecting network overhead $T_{Network}^{PrivateLoRA}$. Assuming the number of adapted layer is noted as $N' \leq N$, the network communication can be decomposed into the initial and final embedding and decoder stack communication.

$$T_{Network}^{PrivateLoRA} = T_{Network}^{TokenEmb} + T_{Network}^{PrivateLoRA\ Activation} \times N', \tag{5}$$

where $T_{Network}^{TokenEmb}$ are transmission latency of input and output of decoder stack on Cloud, $T_{Network}^{PrivateLoRA\ Activation}$ denotes transmission latency of PrivateLoRA's activation.

Starting from Equation 5, we can then derive a unitary communication time t complete one forward computation on one token:

$$t = Size_{DType} \times (N' \times (\frac{r_{D2C} \times N_{D2C}}{B_{D2C}} + \frac{r_{C2D} \times N_{C2D}}{B_{C2D}})) + T_{Network}^{TokenEmb}, \tag{6}$$

where $N_{D2C}(N_{C2D})$ are number of D2C(C2D) transmissions per decoder layer and $Size_{DType}$ are number of bits of the used datatype. $N_{D2C}(N_{C2D})$ are determined by variable dependency during forward computation. For example, query, key and value projections share common inputs, thus N_{C2D} is 1. They produce three activations of the same dimension, thus N_{D2C} is 3.

Therefore, the elapsed time in Equation 3 for single forward computation writes as,

$$T = T_C^{Decoder} + T_D^{LRRT} + T_C^{LRRT} + T_D^{LMHead} + bs \times l \times t, \tag{7}$$

where bs is the batch size and l is the sequence length of mini-batch.

Inference Throughput With Equation 7, we can compute the throughput measured in tokens per second. Firstly, we note combined throughput for local executing A, B and M as TPS_0^{LRRT}.

$$TPS_0^{LRRT} = \frac{bs \times l}{T_D^{LRRT} + T_C^{LRRT}}. \tag{8}$$

Based on Equation 7, the TPS of PrivateLoRA can be written as

$$TPS = \frac{1}{\frac{1}{TPS_C^{Decoder}} + \frac{1}{TPS_0^{LRRT}} + \frac{1}{TPS_D^{LMHead}} + t}, \tag{9}$$

where $TPS_C^{Decoder}, TPS_D^{LMHead}$ indicates the throughput of decoder stack on cloud and LM Head on Device, respectively. Equation 9 holds for both generation prefill and generation decoding, so we can effectively estimate the inference throughput of generation prefill, single query decoding and batched query decoding.

# Param	# Bit	Memory(MB)	FLOPs (G)
1B	3	491.9	2.6
	4	655.8	2.6
	16	2623.3	2.6
3B	3	999.0	5.3
	4	1333.2	5.3
7B	3	2477.7	13.2
	4	3303.5	13.2
	16 (PL)	265.3	0.27
13B	3	4819.4	25.7
	4	6425.8	25.7
	16 (PL)	331.6	0.33
30B	3	12118.2	64.6
	4	16157.7	64.6
	16 (PL)	431.9	0.43

Table 1: Memory and FLOPs comparison between PrivateLoRA and full model deployment. Memory requirement are calculated to load all parameters and total FLOPs to compute a single token. PrivateLoRA only needs % of. FLOPs are calculated assuming batch size is 1 and sequence length is 1.

Training Throughput Training is much more complicated and many aspects can vary the time cost, such as the optimizer, implementation of computation graph. Following analysis in inference, we decompose the time to complete one training step in an end-end manner. During back-propagation, gradients of the same dimension as activations are transmitted between Cloud and Device in reversed direction.

$$T = T_C^{Decoder} + T_D^{LRRT} + T_C^{LRRT} + T_D^{LMHead} + bs \times l \times (t + t'), \tag{10}$$

where t' represents transmission time of gradients which is different from t due to asymmetric network bandwidth and other terms include both forward and backward computations. The TPS of PrivateLoRA during training is then,

$$TPS = \frac{1}{\frac{1}{TPS_C^{Decoder}} + \frac{1}{TPS_0^{LRRT}} + \frac{1}{TPS_D^{LMHead}} + t + t'}, \tag{11}$$

Therefore, with Equation 9 and 11, we can derive the performance boundary between PrivateLoRA and pure device solutions in both generation and training scenarios.

4 Experiments

This part essentially answers the following two questions (1) Whether PrivateLoRA provides good tuning performances with randomly initialized and non-trainable A and B (2) how fast is PrivateLoRA compared to pure device solution or even running on GPUs? Other properties of PrivateLoRA, such as scalability and integrity, are also studied to present a more comprehensive understanding.

4.1 Experiment Setup

4.1.1 Benchmarks

We use tuning performance on various benchmarks to prove that PrivateLoRA offers good data fitting capability for effective personalization. Our benchmarks involve most commonly used benchmarks including natural language understanding, common sense reasoning and logic arithmetic. For common sense reasoning, we use BoolQ[33]. We evaluate involved methods with LM-Eval Harness[34]. We report zero-shot accuracy except that 5-shot evaluation is adopted for MMLU. Since we also value generation speed, we introduce a custom metric M_S that awards high generation speed and performance improvement via tuning.

$$M_S = \frac{TPS}{TPS_C}(M - M_{NT}), \tag{12}$$

where TPS represents generation speed of tested method, TPS_C denotes generation speed on GPU, M denotes average benchmark score of tested method and M_{NT} denotes task performance of original model without tuning. Larger M_S represents better overall performance.

4.1.2 Generation Speed Benchmark and Estimation

We use decoding speed of single query generation, measured in tokens per second (TPS), as the throughput metric. Throughput are measured on GPU and smart phones representing Cloud and Device runtimes, respectively. Inference speed of PrivateLoRA is numerically estimated according to Equation 9 and modest assumptions on the hardware and network listed in Table 2. Edge Device specifications are equivalent to flagship chipsets of smart phones. GPU specifications are at the level of A100 80G GPU. Network bandwidths are at the level of 5G accessible to average consumers[35, 36]. Detailed calculation is discussed in Section 5.

	Specifications
Device	$MB_D = 42.7GBps$ $FLOPS_D = 15.8T$
Cloud	$MB_C = 1935GBps$ $FLOPS_C = 312T$
Network	$B_{d2c} = 60Mbps$ $B_{c2d} = 100Mbps$

Table 2: Assumptions for numerical estimation of inference latency of PrivateLoRA. Edge Device specifications are equivalent to flagship chipsets of smart phones. Cloud specifications are at the level of single A100 80G GPU. Network bandwidths are at the level of 5G used by average consumers.

4.1.3 Baselines

Various device-only solutions are available. To demonstrate the extreme efficiency of PrivateLoRA, we choose solutions that allow LLMs to run on smart phones as baselines, including 1) small models, 2) quantized models and 3) heterogeneously distributed LoRA on D&C.

Small Models We refer to models with parameters fewer than 3B as small models. Models under this scale can easily fit into smart phone's memory and can run even without parameter quantization. For this category, we include OPT-1B3[37], Falcon-rw-1.3B[38] and StableLM-3B-4e1t[39].

Quantized Models Quantization technique is commonly used to run LLMs on low computation power devices. Quantization reduces parameter precision, thus significantly reduce memory workload and can utilize processor's low precision computation. For this category, we include 4-Bit quantized models of Falcon-rw-1.3B[38],OPT-1B3[37], LLaMA 2-7B and LLaMA 2-13B. We primarily use GPTQ[13] to quantize our model. Aforementioned models except LLaMA 2-13B can be tested on MLC Chat on mobile phones. Despite the availability on phones, we still include 4-Bit quantized LLaMA 2-13B to offer a more comprehensive understanding of impacts of quantization on models. Quantized models weights are either publicly downloadable resources or quantized with open-source software[4].

LoRA on D&C To better demonstrate the communication advantages of PrivateLoRA, we adopt LoRA on D&C runtime. Cloud only holds decoder stack parameters and Device only holds LoRA parameters. Activations are transmitted to complete forward computation but the communication base is dimension of hidden states. We include two LoRA configurations, one optimized for generation speed and another optimized for tuning performance.

- **Speed Oriented** We apply the same latency analysis and get communication budget to make LoRA on D&C reaches comparable speed to PrivateLoRA. With obtained communication budget, we explore among several allocation strategies. We report score from the best performant configuration. We note this configuration as **LoRA$^{\mathbf{S}}$**. Details about the adaptation configurations and budget allocation are listed in Appendix C.
- **Performance Oriented** The equivalent structure of adapting query, key and value projections in every decoder layer in LLaMA. We note this configuration as **LoRA$^{\mathbf{P}}$**

4.1.4 PrivateLoRA Configurations

We adapt only query, key and value projections found in every decoder layer of LLaMA as mentioned in Section 3.2.4. We set $r_{D2C} = r_{C2D} = 128$ to achieve the balance between benchmark performance and generation speed. Under our predefined conditions, PrivateLoRAachieves 30.1 tokens/s for LLaMA 2-7B, 20.1 token/s for LLaMA 2-13B and 10.1 token/s for LLaMA-30B.

4.2 Experimental Results

The following results are obtained with Bfloat16 if not mentioned.

From Table 3, we can draw the following conclusions.

PrivateLoRA outperforms device-only solutions in terms of both benchmark and processing speed. Compared to small models used in device-only solutions, PrivateLoRA is capable of leveraging large scale model as the tuning backend, thus guarantees better task performances. For example, with the target generation speed of around 20 tokens per second, device-only solutions can only leverage 1B model while PrivateLoRA is capable of leverage 7B model. Also, PrivateLoRA can achieve higher generation throughput with backend of 33B model compared to 7B model of device-only solutions, with a huge gap of 23.1 in average benchmark scores, let alone smaller scale models.

Significantly reduced communication overhead allows better adaptation for PrivateLoRA. Combining results of LoRA$^{\mathrm{S}}$ and LoRA$^{\mathrm{P}}$, we find extreme disparity in tuning performance and throughput. For LLaMA 2-13B, LoRA$^{\mathrm{S}}$ fails to improve the benchmark score of every tested task while LoRA$^{\mathrm{P}}$ notably boosts task performance of GSM8K and MMLU. On the other hand, the throughput of LoRA$^{\mathrm{P}}$ is even lower than device-only solutions. Such observed disparity indicates the importance of number of adapted layers. To achieve comparable generation speed, LoRA$^{\mathrm{S}}$ only adapts two layers due to large communication base number of d. But in the case of PrivateLoRA, thanks to significantly reduced communication base, all layers can be adapted and yield good tuning performances.

Tuning performance of PrivateLoRA is comparable to LoRA, thus offering good foundation for personalization. On tested tasks, we find very close average benchmark scores between PrivateLoRA and LoRA$^{\mathrm{P}}$. The gap between PrivateLoRAand LoRA$^{\mathrm{P}}$ are 3.3, 2.1 and 2.0 for LLaMA 2-7B, 13B and 33B model, respectively. In the following ablation study of scaling up ranks, we find the performance gap is even smaller, indicating PrivateLoRA offers reliable tuning performance which can be good foundation for personalization on private data.

4.3 Ablation Study: Integrity of PrivateLoRA

[4]Implemented with AutoGPTQ https://github.com/PanQiWei/AutoGPTQ

Model	Method	Runtime	TPS	GSM8K	HSwag	BoolQ	PIQA	MMLU	AVG.	M_S
Falcon-rw-1.3B	No-tuning	Cloud	49.5	0.8	61.6	62.4	74.7	25.9	45.1	0.0
	4Bit	Device	20.1	0.0	45.6	59.5	74.8	24.6	40.9	-1.7
OPT-1B3	No-tuning	Cloud	49.5	1.0	54.0	59.6	69.0	25.0	41.7	0.0
	4Bit	Device	20.1	0.0	26.5	39.1	55.3	23.4	28.9	-5.2
StableLM-3B	No-tuning	Cloud	37.3	7.7	73.9	75.3	79.2	41.8	55.6	0.0
LLaMA 2-7B	No-tuning	Cloud	37.2	14.6	75.9	77.7	77.8	45.3	58.3	0.0
	4Bit	Device	8.1	3.6	54.8	73.9	77.2	36.2	49.1	-2.0
	LoRA^S	D&C	25.6	14.7	75.9*	81.7	77.8*	50.4	60.1	1.3
	LoRA^P	D&C	2.0	35.7	78.2	88.5	79.2	57.9	67.9	0.5
	PL (Ours)	D&C	27.1	25.1	77.0	88.1	78.5	54.3	64.6	**4.6**
LLaMA 2-13B	No-tuning	Cloud	27.8	23.5	79.4	80.5	79.1	54.8	63.5	0.0
	4Bit	Device	-	6.1	78.6	80.8	78.2	51.4	59.0	-
	LoRA^S	D&C	19.5	23.5*	79.4*	80.5*	79.1*	54.8*	63.5	0.0
	LoRA^P	D&C	1.3	42.7	80.1	89.7	79.7	61.4	70.7	0.3
	PL (Ours)	D&C	20.5	36.5	80.0	88.9	79.3	58.4	68.6	**3.8**
LLaMA-30B	No-tuning	Cloud	16.7	34.8	82.6	83.1	82.3	57.8	68.1	0.0
	LoRA^S	D&C	11.7	34.8*	82.6*	83.1*	82.3*	57.8*	68.1	0.0
	LoRA^P	D&C	0.7	51.9	82.8	90.5	82.6	63.3	74.2	0.3
	PL (Ours)	D&C	12.9	46.7	83.3	88.4	82.6	59.8	72.2	**3.1**

Table 3: PrivateLoRA offers good tuning performance and high throughput comparable to cloud-based solutions. Token generation speed denotes specifically throughput at decoding stage. Cloud speed is measured on single A100 80G GPU and Device speed is measured on flagship smart phones. D&C speed is numerically estimated with modest assumptions listed in Table 2. Starred (*) scores indicate worse benchmark performance after tuning. M_S is a custom metric based on the product of tuning improvement and generation speed. Large M_S represents both good tuning performance and high generation throughput.

A, B are kept random after initialization and serve as activation encoder and decoder. Despite the static nature of A, B, A, B and M form a matched integral pair during optimization of M. We make the hypothesis about integrity of PrivateLoRA that with either of the pair, the model performance with degrade as the residual activations are transformed into noise. In order to prove the integrity of A, B and M pair, we conduct the following ablation study. For a tuned pair of A, B and M, we reinitialize A, B and keep M intact. We then run the benchmark to see impact of our perturbation on performance. We test on two benchmarks, namely MMLU and GSM8K, with LLaMA 2-7B. For each setup, we sample scores for 20 rounds.

Figure 4 plots the benchmark scores of PrivateLoRA-tuned (lighter color), reinitialized A, B (dark color) and original performance without tuning. MMLU of reinitialized A, B are around 40.8 and GSM8K around 10.7, which is inferior to original performance. Overall, the poor performance after reinitializing A, B verifies the integrity of A, B and M.

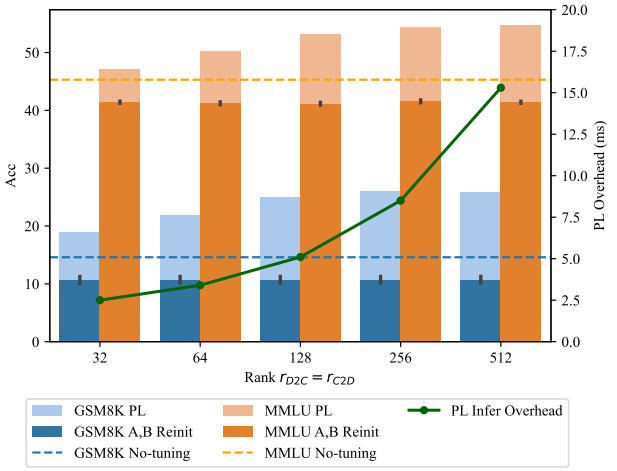
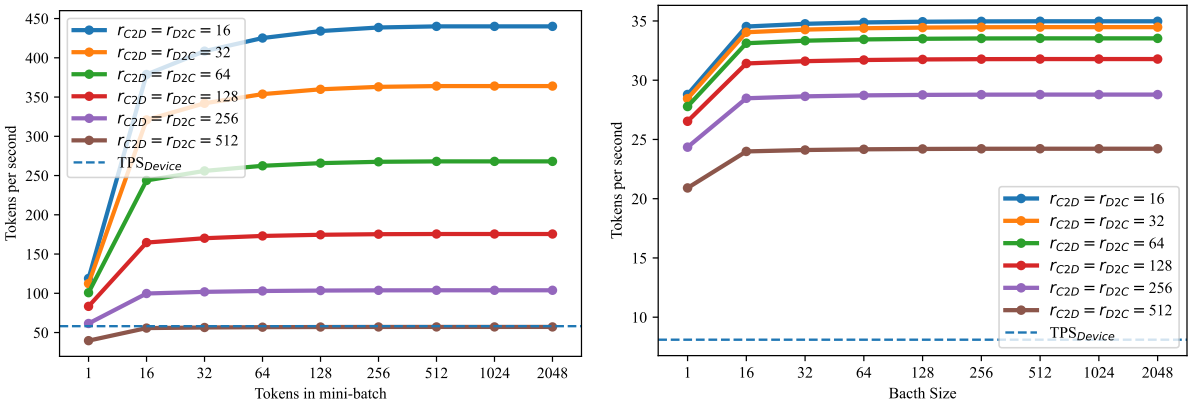


Figure 4: Benchmark scores, PrivateLoRA integrity and inference overhead of PrivateLoRA with scaled up ranks r_{D2C}, r_{C2D}. Unlike LoRA, scaling up ranks benefits tuning performance.

4.4 Ablation Study: Ranks of PrivateLoRA

We empirically choose $r_{D2C} = r_{C2D} = 128$ in our main experiment. To study the effect of ranks r_{D2C}, r_{C2D}, we train LLaMA 2-7B on MMLU and GSM8K with gradually increased r_{C2D}, r_{D2C} from 32 to 512. For simplicity, we set $r_{C2D} = r_{D2C}$. The benchmark scores and resulting inference overhead are plotted in Figure 4.



(a) Prefill.

(b) Decoding.

Figure 5: Inference throughput of PrivateLoRA with backend of LLaMA 2-7B. Prefill and decoding speed for rank of 128 reaches 175.5 and 26.5 tokens per second,respectively. Throughput of both stages is almost 300% of pure device solutions.

Unlike LoRA, increasing ranks r_{D2C}, r_{C2D} of PrivateLoRA increases tuning performances. When scaling ranks from 32 to 128, we see obvious improvement on benchmark scores of MMLU and GSM8K. The benefit in tuning performance decreases as ranks increase and we do not see much of a difference between ranks of 256 and ranks of 512. With this in mind, we can effective adjust the tuning capability on demand. Also, we found that with an equal rank of 32, PrivateLoRA's benchmark score is way lower than LoRA, indicating the impact of untrainable A, B on data fitting capability. However, with increased ranks, as the residual activation becomes of higher rank, the tuning performance is compensated.

Another thing to account for when setting the ranks of PrivateLoRA is the communication overhead. The inference overhead should increase linearly with ranks and Figure 4 verifies it.

4.5 Ablation Study: Comparison with LoRA

Although PrivateLoRA is equally low rank as LoRA, PrivateLoRA exhibits weaker tuning performance compared to LoRA^P. To find out whether the performance difference comes from triplet structure or untrainable A, B , we then carry out an ablation study.

We run all four configurations of A, B with different trainable setting for PrivateLoRA. For LoRA, we run the equivalent structure that only targets query, key and value projections. We benchmark on MMLU and GSM8K as PrivateLoRA falls behind compared to LoRA.

Method	Config	GSM8K	MMLU
PL	$A\times$ $B\times$	25.1	54.3
	$A\times$ $B\surd$	26.7	54.5
	$A\surd$ $B\times$	27.4	55.4
	$A\surd$ $B\surd$	**28.8**	**56.6**
LoRA^P	q,k,v	31.3	57.3

Table 4: Ablation study on structure of PrivateLoRA. Various A, B configurations are tested. Trainable A, B of PrivateLoRA benefits tuning performances. $A\times, B\surd$ means A is non-trainable and B is trainable.

Table 4 lists the benchmark scores of aforementioned methods. We see largest improvements brought by simultaneously trainable A, B. Another interesting thing is that trainable A boosts overall performances more than trainable B, meaning a trainable upstream is more preferable.

5 Throughput Estimation

According to TPS derived in Section 3.3.3, assumptions in Table 2 and measured base throughput (Table 8), we can make numerical estimations on throughput of PrivateLoRA. We discuss throughput of typical scenarios of transformer usage, including generation prefill, generation decoding and training. To simplify the discussion and better present the idea, we use a single A100 80G GPU as Cloud. Therefore, we do not include models larger than 33B. Detailed calculation is discussed in Appendix D.

5.1 Inference Throughput

Figure 5 plots estimated inference throughput of prefill stage and decoding stage of PrivateLoRA with backend of LLaMA 2-7B (more at Appendix D.1). In general, TPS of PrivateLoRA increases with total tokens processed and converges when total tokens exceed certain thresholds. With ranks set to 128, PrivateLoRA can achieve a prefill speed of 175.5 token per second, almost 300% of pure device solutions. As for decoding speed, PrivateLoRA allows a speed of 26.5 TPS, 327% of pure device solutions. If ranks are reduced to 64 or 32, the prefill TPS will be 268 or 364, respectively. Therefore, the tradeoff between tuning performance and throughput is important in PrivateLoRA application.

On larger scale models, execution time of the decoder layer is larger, therefore the impact of PrivateLoRA on throughput becomes smaller. 30B models are impossible to run on current smart phones. However, it can be utilized as personalization backend of PrivateLoRA, and throughput can reach 77.2% of A100 80G GPU speed. Although we do not offer numerical estimations on larger scale model, *e.g.* 65B and 160B, due to the single A100 80G limit, we can still conclude that PrivateLoRA becomes even more competitive on models of these scales.

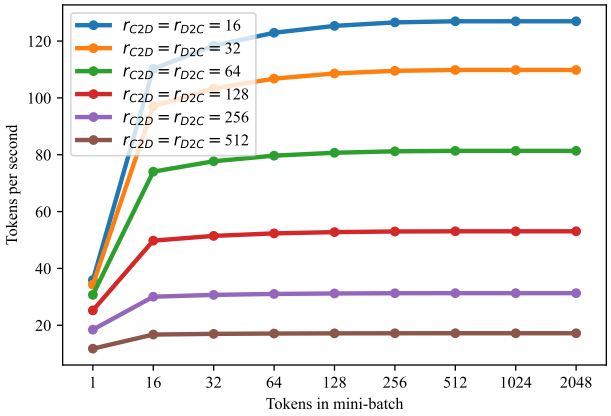


Figure 6: Estimated training throughput of tuning LLaMA 2-7B with PrivateLoRA with different ranks and total tokens in mini-batch. Word embedding, adaptation matrices M and LM Head are set trainable.

5.2 Training Throughput

Training step is much more complex than pure inference. Compared to inference, model training not only adds back-propagation but also involves computations from optimizer, computation graph, *etc*, thus adding highly variable workload to processor and memory. All these aspects make estimation of time cost of training step difficult. Moreover, since there are no publicly available deep learning frameworks that allow tuning LLMs on smart phones, we could not offer empirical training throughput on Device. However, with our estimations, we still find that PrivateLoRA is highly competitive in training efficiency.

Estimated throughput of tuning LLaMA 2-7B with PrivateLoRA of different ranks is plotted in Figure 6. With rank set to 128, the peak training throughput of PrivateLoRA on LLaMA 2-7B is around 53.8 tokens per second, which is almost as fast as prefill speed on Device. On GPU, training throughput is way smaller than prefill throughput and the ratio sits around 10% according to our benchmark. Despite that we can not numerically estimate training throughput on pure Device, we can still conclude that tuning with PrivateLoRA outperforms pure device solutions in terms of speed.

5.3 PrivateLoRA Overhead

In this section, we analyze the overhead introduced by PrivateLoRA, so that throughput under other conditions can be better predicted. In general, despite our modest assumptions, PrivateLoRA introduces very marginal overhead and can produce higher overall throughput if accompanied with SOTA LLM serving technology.

Figure 7 plots detailed inference time decomposition. Under 5G accessible to average consumers, PrivateLoRA adds 5.1ms, 6.4ms and 9.3ms of overhead for 7B, 13B and 30B models, respectively Inference latency overhead introduced by PrivateLoRA is quite marginal and is only proportional to number of adapted layers. Architecture of PrivateLoRA reduces transmission base from hidden dimension to ranks r_{D2C}, r_{C2D} , so that transmission amount is model invariant. Forwarding one token through one decoder layer only yield 8.2 Kb of transmission, while 262.1 Kb is needed in LoRA equivalent architecture and the amount is also relevant to hidden dimension.

Therefore, the marginal latency brought by PrivateLoRA allows for higher throughput if accompanied with dedicated LLM serving technology. Throughput on GPU of this work is measured with Huggingface transformers[40] implementation, which is not dedicated for LLM serving. SOTA LLM serving technology[41, 12] can produce much higher throughput with enhanced FLOPS utilization and multi-GPU support. For instance, running LLaMA 2-70B with MLC LLM[12] on 8 A100 80G GPU yields a decoding speed of 38.8 TPS.

6 Discussion

In previous sections, we discussed the architecture of PrivateLoRA, its impact on LLM service mode and its sheer tuning performance. In this section, we discuss other challenges of PrivateLoRA throughout the entire lifecycle and potential solutions. Solutions are not limited at the level of models since many problems can be better handled in a more systematic manner.

6.1 Further Improve PrivateLoRA Throughput

Various methods can be adopted to improve PrivateLoRA throughput.

With current configuration, the transmission amount per token for LLaMA 2-7B is 4.2Mb. Following the overhead decomposition in Section 3.3.3, we can easily propose several approaches to further reduce the transmission overhead. Firstly, a lower precision data type can be used. Various researches have pointed out that 8bit, even 4bit quantization does not influence much model performance and the impact reduces as model scales up. Therefore, it's reasonable to use PrivateLoRA on a lower precision model and the transmission overhead can be reduced. Secondly, exploiting the redundancy in adaptation configurations. Adapting all modules at all layers may not be optimal and reduction in adapted modules can provide similar performances. Therefore, we can trade fractions of tuning performance with fewer adapted modules. With fewer adapted modules, the activation transmission is also cut down. Thirdly, we can deploy some of the decoder layers on Device. For devices with reasonable hardware, deploy some decoder layers on Device can better utilize the hardware and reduce the transmission amount. As mentioned earlier, this practice also enhances the activation security.

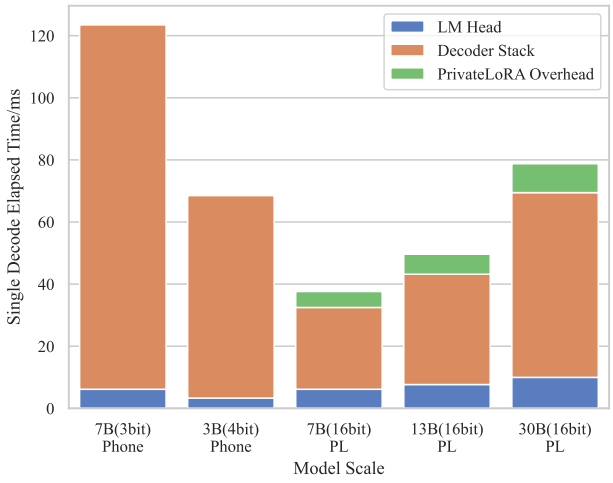


Figure 7: Inference latency decomposition shows that PrivateLoRA adds very marginal overhead. Under 5G accessible to average consumers, PrivateLoRA adds 5.1ms, 6.4ms and 9.3ms of overhead for 7B, 13B and 30B models, respectively. PrivateLoRA allows personalized generation with a backend of 30B models at the speed of 4bit quantized 3B models.

6.2 Activation Privacy

In our work, we protect privacy by ensuring data locality, which refers to the prohibition of persistent storage of raw data and its derivative on Cloud. With PrivateLoRA, raw text and personalized parameters remain on edge devices no matter in training or inference. This protects the user from being identified by visual-based or norm-based comparison[42]. In our proposed paradigm, activations and gradients are transmitted to prevent raw data is uploaded to Cloud. This contrasts with existing literature that focuses on extracting bag of words from general-purpose language model embeddings[43]. However, such method targets final output embeddings and effect on intermediate activations of decoder-only generative LLMs is unknown. To further enhance privacy, we can deploy critical decoder layers on edge devices to reduce the exposure of vulnerable activations[44]. Moreover, our proposed heterogeneous architecture design complicates potential attacks. Looking ahead, our future work will concentrate on bolstering activation privacy, exploring more robust methods to protect against sophisticated data breaches.

6.3 PrivateLoRA Integrity

In Section 4.3, we confirm the integrity of the model components A, B, and M, showing that only matched pairs produce correct outputs. Two critical observations emerge from our analysis. First, we observed an intriguing phenomenon: the perturbed benchmark scores of GSM8K and MMLU both decline by approximately 5% in accuracy compared to the original model, regardless of the noise activation rank. This finding indicates that increasing the rank of noise activations does not further degrade performance. Intriguingly, higher ranks improve tuning performance. Starting from this observation, we may gain valuable insights into the mechanisms of low-rank adaptation of LLM.

Second, we identify a potential vulnerability (e.g., backdoor attack on edge[45]) in the model's design: the simple linear nature of components A, B, and M could allow for 'hacking' of M, assuming A and B are known. By tracking several

rounds of forward computation, one might deduce M, leading to privacy concerns. However, it's crucial to note that the parameters on Device are essentially a black box to the Cloud, and their internal architecture could be more complex than a single linear projection. This realization directs our future work towards enhancing the security and integrity of these model components to achieve responsible[46], secure and ethical use of LLM.

6.4 Low power consumption is the core advantage of PrivateLoRA.

As detailed in Section 3.3.2, PrivateLoRA necessitates a mere 2% of the FLOPs required by a device-only solution, presenting a significant computational advantage. This reduction in computation not only enhances processing speed but also leads to a substantial decrease in power consumption. The key benefit of PrivateLoRA lies in its ability to minimize power usage on edge devices. While advancements in hardware architecture and manufacturing processes may enable larger scale transformers to operate on edge devices, PrivateLoRA's ability to drastically reduce local compute demands remains its standout feature. This is particularly crucial for most edge devices, which aren't typically built for sustained peak performance and where power efficiency is paramount, especially in battery-dependent scenarios. Comparing PrivateLoRA with device-only solutions, there is a notable shift in power consumption from edge devices to the broader network and Cloud infrastructure. Expanding on this, the adoption of PrivateLoRA could have profound implications for the future design and functionality of edge devices.

7 Conclusion and Future Work

In this paper, we propose PrivateLoRA, a Parameter Efficient Fine-Tuning (PEFT) method for heterogeneously distributing LLMs. This novel approach, centered around the concept of Low Rank Residual Transmission, significantly diminishes communication overheads, thus offering a more efficient and privacy-conscious alternative to traditional cloud-based solutions. Our proposed method democratizes access to advanced LLM capabilities and could spur a wave of innovation and new applications across various sectors.

The following directions can be our future work.

- **Communication Budget Allocation** In our experiment, we adapt all decoder layers, but redundancy in adaptation can be exploited to further reduce communication overhead.
- **Activation Privacy** Activation privacy is the next challenge towards a more comprehensive privacy protection in LLM service.
- **Integrity of** A, B, M Unmatched A, B and M only slightly reduces performance but it should work like public private key pair that unmatched pair results in block of access. A method should be found to completely poison the activations so that even semantics can not be deduced.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[2] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[3] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[4] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Trans. Mach. Learn. Res.*, 2022, 2022.

[5] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. Codegeex: A pre-trained model for code generation with multilingual

benchmarking on humaneval-x. In Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye, editors, *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5673–5684. ACM, 2023.

[6] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023.

[7] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics, 2023.

[8] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021.

[9] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR, 2023.

[10] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P. Lillicrap. Android in the wild: A large-scale dataset for android device control. *CoRR*, abs/2307.10088, 2023.

[11] Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *CoRR*, abs/2309.11436, 2023.

[12] MLC team. MLC-LLM, 2023.

[13] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022.

[14] Peiyu Liu, Zikang Liu, Ze-Feng Gao, Dawei Gao, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Do emergent abilities exist in quantized large language models: An empirical study. *CoRR*, abs/2307.08072, 2023.

[15] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.*, 116:1–8, 2018.

[16] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020.

[17] 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. Pytorch FSDP: experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, 2023.

[18] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

[19] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

[20] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. On the effectiveness of adapter-based tuning for pretrained language model adaptation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2208–2222. Association for Computational Linguistics, 2021.

[21] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1–9. Association for Computational Linguistics, 2022.

[22] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *NeurIPS*, 2022.

[23] Edward Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[24] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[25] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019.

[26] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *NeurIPS*, 2022.

[27] Tian Dong, Song Li, Han Qiu, and Jialiang Lu. An interpretable federated learning-based network intrusion detection framework. *arXiv preprint arXiv:2201.03134*, 2022.

[28] Tian Dong, Han Qiu, Jialiang Lu, Meikang Qiu, and Chun Fan. Towards fast network intrusion detection based on efficiency-preserving federated learning. In *2021 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 468–475, 2021.

[29] Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Chen Wang, Wei-Ming Chen, Chuang Gan, and Song Han. Pockengine: Sparse and efficient fine-tuning in a pocket. *CoRR*, abs/2310.17752, 2023.

[30] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

[31] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *CoRR*, abs/2211.05102, 2022.

[32] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *CoRR*, abs/2305.14314, 2023.

[33] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2924–2936. Association for Computational Linguistics, 2019.

[34] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021.

[35] Godfrey Anuga Akpakwu, Bruno J. Silva, Gerhard P. Hancke, and Adnan M. Abu-Mahfouz. A survey on 5g networks for the internet of things: Communication technologies and challenges. *IEEE Access*, 6:3619–3647, 2018.

[36] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. A first look at commercial 5g performance on smartphones. In Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 894–905. ACM / IW3C2, 2020.

[37] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068, 2022.

[38] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

[39] Jonathan Tow, Marco Bellagente, Dakota Mahan, and Carlos Riquelme. Stablelm 3b 4e1t.

[40] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[41] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[42] Tian Dong, Bo Zhao, and Lingjuan Lyu. Privacy for free: How does dataset condensation help privacy? In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5378–5396. PMLR, 2022.

[43] Xudong Pan, Mi Zhang, Shouling Ji, and Min Yang. Privacy risks of general-purpose language models. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1314–1331. IEEE, 2020.

[44] Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. Dola: Decoding by contrasting layers improves factuality in large language models. *CoRR*, abs/2309.03883, 2023.

[45] Tian Dong, Ziyuan Zhang, Han Qiu, Tianwei Zhang, Hewu Li, and Terry Wang. Mind your heart: Stealthy backdoor attack on dynamic deep neural network in edge computing. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1–10, 2023.

[46] Tian Dong, Shaofeng Li, Guoxing Chen, Minhui Xue, Haojin Zhu, and Zhen Liu. Rai ^{2}: Responsible identity audit governing the artificial intelligence. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023*. The Internet Society, 2023.

[47] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *CoRR*, abs/2305.17333, 2023.

A Experiment Setup

A.1 Hyperparameters

Detailed hyperparameters are listed in Table 5. All experiments are carried out in almost identical hyperparameter configuration except that learning rates may vary depending on the task. Learning rates for PrivateLoRA is generally large. Additionally, PrivateLoRA is more sensitive to learning rate. For example, HellaSwag performance vary for different.

Experiment	Method	Hyperparameters	Values
	Shared	Optimizer	AdamW
		Weight Decay	0
		Warmup Ratio	0.1
		LR Scheduler	Linear
		Batch Size $\times$ Num_{GPU}	256
BoolQ	LoRA	Learning Rate	5e-4
	PrivateLoRA	Learning Rate	{5e-3,1e-3}
MMLU	LoRA	Learning Rate	5e-4
	PrivateLoRA	Learning Rate	{5e-3,1e-3}
HellaSwag	LoRA	Learning Rate	5e-4
	PrivateLoRA	Learning Rate	{5e-3,1e-3}
PIQA	LoRA	Learning Rate	5e-4
	PrivateLoRA	Learning Rate	{5e-3,1e-3}
GSM8K	LoRA	Learning Rate	5e-4
	PrivateLoRA	Learning Rate	{5e-3,1e-3}

Table 5: Detailed hyperparameters used in our experiments. We vary learning rates depending on the dataset no matter the model scale. PrivateLoRA generally use larger learning rates. Gradient accumulation step is set to ensure the equality of total train steps for different models.

A.2 Prompt Templates

We follow previous works to build prompts for tuning. More specifically, we follow QLoRA[32] for MMLU, MeZO[47] for BoolQ and LM-Eval Harness[34] for the rest. Despite improvement on benchmark scores in most of our experiments, the following prompts do not guarantee improvement on every model.

Dataset	Task Type	Prompt & Target
MMLU	Multiple Choice	<question> A.<samples>[0] B.<samples>[1] C.<samples>[2] D.<samples>[3] *A/B/C/D*
BoolQ	Classification	<passage><question>? *Yes/No*
GSM8K	Generation	Question: <question> Answer: *<answer>*
HellaSwag	Multiple Choice	<activity_label>: <ctx> ____ A.<endings>[0] B.<endings>[1] C.<endings>[2] D.<endings>[3] Answer:*A/B/C/D. <endings>*

Table 6: Prompt templates used for tuning. Prompts are adapted from QLoRA[32], MeZO[47] and LM-Eval Harness[34]. <question> refers to fields of datasets and *A* denotes target. During training, only loss on target part is calculated for back-propagation. For evaluation, we use standard prompts provided in LM-Eval Harness and make no modification.

B Throughput Measurement

B.1 Inference Throughput Measurement

B.1.1 Quantized Models On Mobile Phones

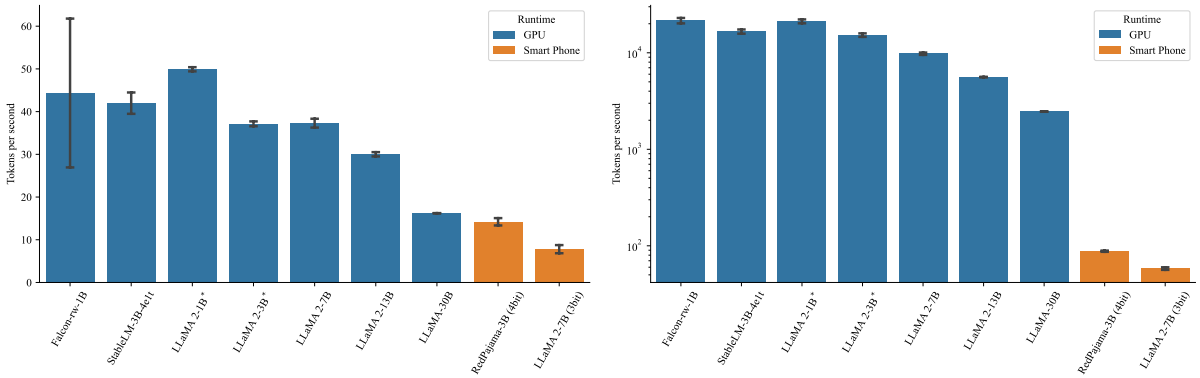
We tested generation speed on mobile phones with MLC Chat[12], currently only solution to running LLMs on mobile devices. We use LLaMA 2-7B (3bit) and RedPajama-3B (4bit) provided in the APP by default for benchmark. We sample on each device for 20 times with inputs of 1024 tokens and report the generation speed from built-in benchmarker. We use long sequences so that prefill stage is guaranteed to be FLOPS bound.

Device Model	LLaMA 2-7B 3bit		RedPajama-3B 4bit	
	Prefill	Decode	Prefill	Decode
iPhone 13 Pro Max	54.3	5.7	81.8	14.3
Xiaomi 13	58.9	8.1	88.1	16.3

Table 7: Token generation speed tested on mobile phones with MLC Chat. Both prefill speed and decoding speed are reported with input sequences of 1024 tokens. Decoding speed is around 10% of prefill speed due to memory boundness caused by non-batched computation.

Table 7 lists prefill and decoding speed of . Decoding speed is much slower than prefill speed because the computation is not batched, thus memory bound.

B.1.2 BF16 Models on a Single A100 80G GPU



(a) Token generation speed of decoding stage.

(b) Throughput rate of prefill stage.

Figure 8: Decoding speed of models of scale between 1B and 30B parameters tested on $1\times$A100 80G GPU. Models with $*$ are not official releases and are created by scaling transformer dimensions.

We use Huggingface Transformers[40] implementations to benchmark generation speed. As for benchmark protocol, we follow vLLM[41] to sample elapsed time of prefilling 1024-token input and decoding 1024 new tokens. Due to implementation, models can produce very results of large variance, *e.g.* Falcon-rw-1.3B. To rule out implementation discrepancy, we scale LLaMA to 1B and 3B according to dimensions of Falcon-rw-1.3B and StableLM-3B-4e1t. Obtained statistics are plotted on Figure 8. In decoding phase, 4bit quantized 3B model on smart phone is as fast as 30B model on a single A100 80 GPU. For prefill stage, the gap between GPU and smart phones are significantly wider. Throughput of 30B model on GPU is almost 3000% of quantized 3B model on smart phones. The low throughput of prefill on smart phones hinders training on personal data.

Model	Batch Size	TPS
LLaMA 2-7B	16	1088.6
LLaMA 2-13B	8	652.8
LLaMA-30B	2	296.0

Table 8: Training throughput of llama series on a single A100 80G GPU.

C Transmission Budget Allocation of LoRA^S

LoRA^S denotes series of LoRA configurations that allows LoRA to reach predefined inference speed under the distributed architecture of PrivateLoRA. Used as baselines in our experiments, we set LoRA^S to reach 70% of Cloud speed and derive the transmission budget per token t for LoRA. We restrict adaptation modules to query, key and value for the same reason as PrivateLoRA. Therefore, the only variable to transmission amount is the number of adapted layers. For LLaMA 2-7B, to reach 70% of Cloud speed, only 2 layers can be adapted. For LLaMA 2-13B and LLaMA-30B, the number of layers are 2, 3, respectively. Therefore, we obtain the following potential configurations. We test obtained configurations by tuning LLaMA 2-7B on MMLU. Ranks of LoRA are set to 32 and α set to 32. Learning rates are 5e-4 and identical for all experiments.

Target TPS	% Cloud TPS	Modules	Layers	MMLU
25.6	∼70%	q,k,v	0:2	47.4
		q,k,v	30:32	45.3*
		q,k,v	15, 31	**50.4**
		q,k,v	0, 15	48.7
		q,k,v	0, 31	45.9
11.7	∼30%	q,k,v	0:16	55.5
		q,k,v	16:32	51.3
		q,k,v	0:32:2	**55.8**
-	-	No-tuning		45.3

Table 9: MMLU scores of LLaMA 2-7B tuned with different LoRA^S configurations. Notations of layers follow python list slicing. 0:16 in layers represents the first 16 decoder layers are adapted. Starred (*) scores indicate worse benchmark performance after tuning.

Table 9 list MMLU scores of various LoRA^S configurations. With a target TPS of 25, LoRA^S provides terrible tuning performances. Adaptation on the 31*st*,32*nd* layers and 1*st*,32*nd* yields worse benchmark performances than no-tuning. Increasing the number of adapted layers also increase the performances. Adapting only the first 16 layers yield similar tuning performance to LoRA^P. Alongside performances of 30% of Cloud TPS, adapting the first few layers gives better tuning performances. Therefore, the general adaptation budget allocation strategy is to adapt shallow layers and final LoRA^S configurations are listed in Table 10

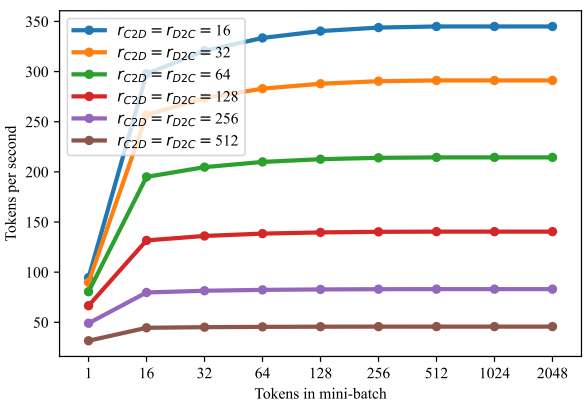
Model	Modules	Layers	TPS	% Cloud TPS
LLaMA 2-7B	q,k,v	15,31	25.6	68.9
LLaMA 2-13B	q,k,v	19,39	19.5	70.0
LLaMA-30B	q,k,v	29,59	12.4	74.6

Table 10: Detailed adaptation configurations of LoRA^S for LLaMA 2-7B, LLaMA 2-13B and LLaMA-30B used in Section 4.2. Adapted modules are restricted to query,key and value to reduce transmission. The adapted layers are chosen with the insight that adapting shallow layers yield more performance gain.

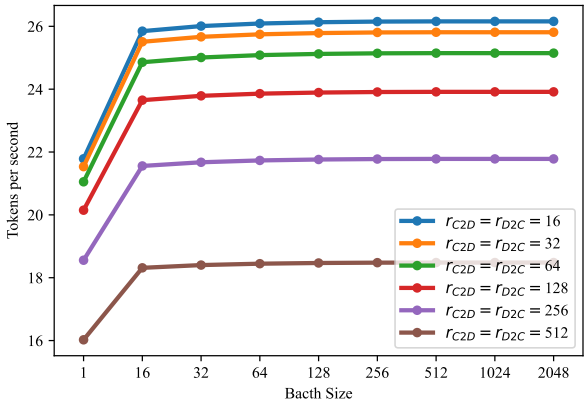
D PrivateLoRA Throughput Estimation

D.1 Inference Throughput

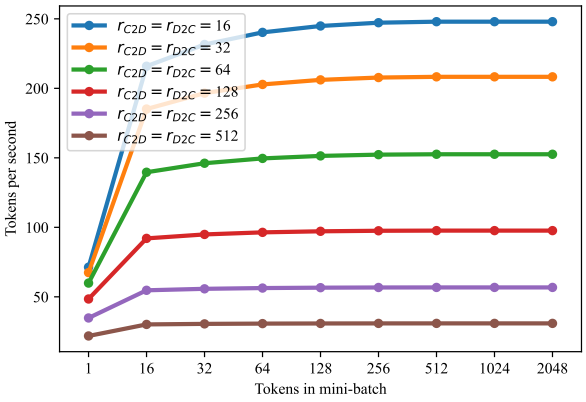
D.2 Training Throughput



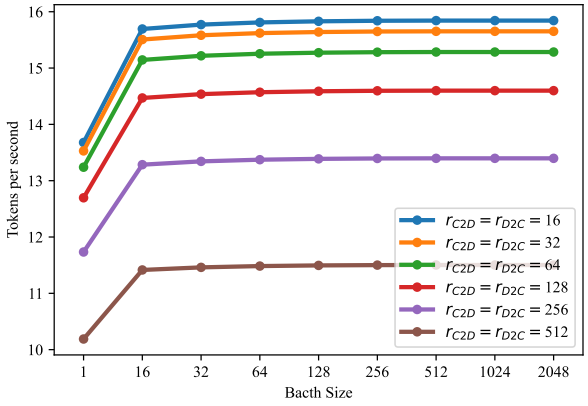
(a) LLaMA 2-13B Prefill.



(b) LLaMA 2-13B Decoding.

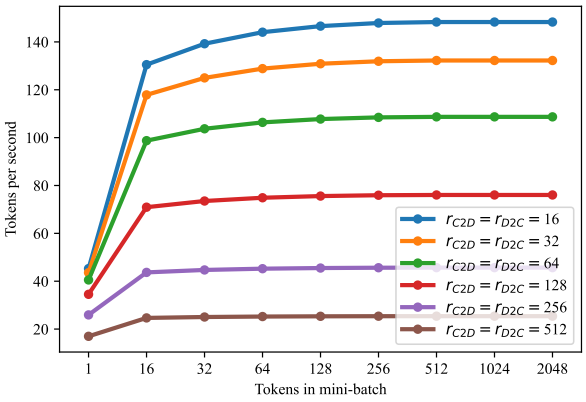


(c) LLaMA 2-30B Prefill.

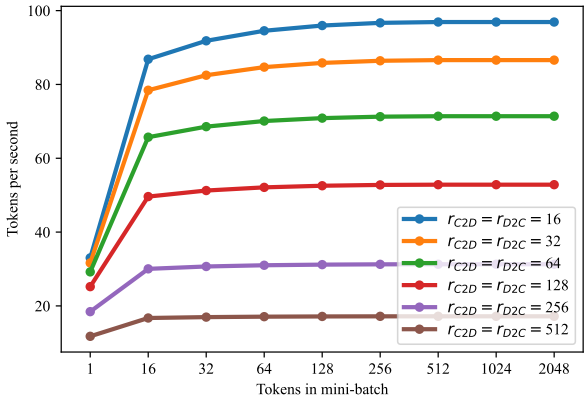


(d) LLaMA 2-30B Decoding.

Figure 9: Estimated inference throughput of PrivateLoRA on different base models.



(a) Training throughput (LLaMA 2-13B).



(b) Training throughput (LLaMA-30B).

Figure 10: Estimated training throughput of PrivateLoRA on different base models.