

# SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models

Zheng Lin, Xuanjie Hu, Yuxin Zhang, Zhe Chen, *Member, IEEE*, Zihan Fang, Xianhao Chen, *Member, IEEE*, Ang Li, *Member, IEEE*, Praneeth Vepakomma, *Member, IEEE*, and Yue Gao, *Fellow, IEEE*

arXiv:2407.00952v1 [cs.LG] 1 Jul 2024

**Abstract**—The scalability of large language models (LLMs) in handling high-complexity models and large-scale datasets has led to tremendous successes in pivotal domains. While there is an urgent need to acquire more training data for LLMs, a concerning reality is the depletion of high-quality public datasets within a few years. In view of this, the federated learning (FL) LLM fine-tuning paradigm recently has been proposed to facilitate collaborative LLM fine-tuning on distributed private data, where multiple data owners collaboratively fine-tune a shared LLM without sharing raw data. However, the staggering model size of LLMs imposes heavy computing and communication burdens on clients, posing significant barriers to the democratization of the FL LLM fine-tuning paradigm. To address this issue, split learning (SL) has emerged as a promising solution by offloading the primary training workload to a server via model partitioning while exchanging activation/activation’s gradients with smaller data sizes rather than the entire LLM. Unfortunately, research on the SL LLM fine-tuning paradigm is still in its nascent stage. To fill this gap, in this paper, we propose the first SL LLM fine-tuning framework, named SplitLoRA. SplitLoRA is built on the split federated learning (SFL) framework, amalgamating the advantages of parallel training from FL and model splitting from SL and thus greatly enhancing the training efficiency. It is worth noting that SplitLoRA is the inaugural open-source benchmark for SL LLM fine-tuning, providing a foundation for research efforts dedicated to advancing SL LLM fine-tuning. Extensive simulations validate that SplitLoRA achieves target accuracy in significantly less time than state-of-the-art LLM fine-tuning frameworks, demonstrating the superior training performance of SplitLoRA. The project page is available at <https://fdud-inc.github.io/splitora/>.

**Index Terms**—Distributed learning, split federated learning, client-side model aggregation, model splitting, mobile edge computing.

## I. INTRODUCTION

In recent years, LLMs [1]–[3] have achieved tremendous successes across a broad spectrum of pivotal domains such as

Z. Lin, X. Hu, Y. Zhang, Z. Chen, Z. Fang and Y. Gao are with the School of Computer Science, Fudan University, Shanghai 200438, China (e-mail: zlin20@fudan.edu.cn; xjhu23@m.fudan.edu.cn; yuxinzhang22@m.fudan.edu.cn; zhechen@fudan.edu.cn; zhfang19@fudan.edu.cn; gao.yue@fudan.edu.cn). Z. Lin is also with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pok Fu Lam, Hong Kong, China.

X. Chen is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pok Fu Lam, Hong Kong, China (e-mail: xchen@eee.hku.hk).

A. Li is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD-20742, USA (e-mail: angliice@umd.edu).

P. Vepakomma is with Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, United Arab Emirates, and the Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: vepakom@mit.edu).

(Corresponding author: Yue Gao)

smart healthcare [4], [5], computer vision [6]–[8], intelligent transportation [9]–[11] due to their exceptional capabilities in handling high-complexity models and large-scale datasets [1]–[3], [12]. However, a critical concern accompanying the surge of LLMs has emerged: it is estimated that high-quality public datasets will be depleted before 2026 [13]. The increasing trend of researchers preferring to train data-hungry LLMs by combining existing datasets [14] or utilizing model-generated datasets [15], rather than collecting or generating new dataset, also reflects the current scarcity of public available data. This suggests that the development of current LLMs may encounter significant bottlenecks shortly, as the well-established scaling laws indicate that larger datasets usually lead to better performance [16].

With the rapid proliferation of Internet of Things (IoT) devices and advancements in sensor technology, connected IoT devices are capable of collecting massive data. However, these high-quality data distributed across multiple parties cannot be shared publicly due to issues such as user privacy concerns (e.g., medical [17] and financial [18] data) or physical limitations (e.g., lack of network connectivity). Some leading AI technology companies can gather large volumes of private data to train data-hungry LLMs, exemplified by Google’s Med-PaLM [19], a medical LLM capable of providing expert-level medical guidance and diagnosis. Nevertheless, not every party possesses adequate private data to independently train a high-performing and data-hungry LLM. Considering the increasing scarcity of public data and the difficulties in accessing user-private data, devising collaborative training paradigms for decentralized private data without data sharing is paramount to driving the advancement of modern LLMs.

The federated learning (FL) LLM training/fine-tuning paradigm [20]–[22] has recently been proposed to facilitate collaborative LLM training on distributed private data, where multiple data owners collaboratively training a shared LLM without sharing raw data. In FL LLM paradigm, data owners train local LLMs on their respective local data and then send the LLM parameters rather than raw data to a parameter server for model update [23]–[25]. However, the staggering growth in LLM model sizes imposes heavy computing and communication burdens, posing significant barriers to the democratization of the FL LLM paradigm. To address this issue, split learning (SL) [26] has emerged as a promising distributed ML framework capable of overcoming the weakness of FL by offloading the primary training workload to a server via model partitioning while exchanging activation/activation’s gradients with smaller data sizes rather than the entire LLM [27]–[29].

Unfortunately, research on the SL LLM fine-tuning paradigm is still unexplored.

To fill this gap, in this paper, we propose a concise and research-friendly SL LLM fine-tuning framework, named SplitLoRA. SplitLoRA is built on the split federated learning (SFL) [30] framework that amalgamates the advantages of parallel training from FL and model splitting from SL and is developed on the well-known parameter-efficient fine-tuning (PEFT) technique Low-rank adaptation (LoRA) [31], [32]. Specifically, we start from pre-trained LLMs on a large-scale dataset and then train/fine-tune the LLMs to adapt the downstream tasks via SFL, which consists of several stages: client-side forward propagation (FP), activations transmissions, server-side FP and back-propagation (BP), activations' gradients transmissions, client-side BP, and client-side model aggregation. For experimental configurations, we 1) implement SplitLoRA framework based on GPT-2 [33] on the E2E dataset [34] and evaluate its performance across diverse performance metrics (e.g., BLEU, METEOR, and NIST); 2) adopt the SFL framework, amalgamating the advantages of parallel training from FL and model splitting from SL and thus greatly enhancing the training efficiency. 3) employ the PEFT technique, LoRA, enabling training to be executed on a single consumer-grade GPU (such as the NVIDIA 3090). Besides, we investigate the potential of SplitLoRA for practical deployments by comparing it with state-of-the-art LLM fine-tuning paradigm in terms of converged accuracy, convergence rate, and resource efficiency. It is worth noting that SplitLoRA is the inaugural open-source benchmark for SL LLM fine-tuning, providing a foundation for research efforts dedicated to advancing SL LLM fine-tuning.

In the near future, we anticipate that others will build upon our SplitLoRA framework for further explorations. There are several reasons for this: (1) Some new challenges and directions are emerging in the SL LLM fine-tuning, such as determining optimal model splitting for LLMs and extension of SL LLM paradigm to accommodate heterogeneous computing resources across clients. (2) Tailoring the SL LLM training framework specifically to different application scenarios, e.g., vehicular networks and satellite networks, etc., to achieve higher training efficiency. (3) In this era of LLMs, we advocate future work in the SL LLM domain to modify our framework and assess the performance of their algorithms.

Our contributions are as follows:

- We explore the pipeline for fine-tuning contemporary LLMs on decentralized private data resources via SFL, pointing out a promising development direction for LLMs.
- We propose a concise and research-friendly SL LLM fine-tuning framework named SplitLoRA. It is worth noting that SplitLoRA is the inaugural open-source benchmark for SL LLM fine-tuning, providing a foundation for research efforts dedicated to advancing SL LLM fine-tuning.
- We conduct extensive experiments to compare SplitLoRA with the conventional centralized and federated LLM fine-tuning paradigms in terms of converged accuracy, convergence rate, and resource efficiency, demon-

strating that SplitLoRA is more communication- and computation-efficient.

The remainder of this paper is organized as follows. Section II introduces related work. Section III elaborates on system model and SplitLoRA framework. Section IV provides the simulation results. Section V discusses the future research direction. Finally, concluding remarks are presented in Section VI.

## II. RELATED WORK

### A. Large Language Models

Driven by the maturation of deep learning algorithms, increased computing capabilities, and the availability of large-scale datasets, LLMs have made significant strides across industries. Major players in the AI community [1]–[3], [35]–[37], including OpenAI, Google, Microsoft, and Facebook, are dedicated to developing their own LLMs. For instance, OpenAI's highly acclaimed chat LLM, GPT series [1], [35], [36], and Google's BERT [37] have demonstrated superior performance in a wide spectrum of Natural Language Processing (NLP) tasks like language translation, text generation, question answering [38], and sentiment analysis [39], [40]. Moreover, LLM has extended beyond its original NLP domain to shine in pivotal domains including healthcare [4], [19], autonomous driving [41], [42], and robotic control [43], [44]. For example, in healthcare, Med-PaLM [19] is devised for medical image analysis, clinical document processing, and patient diagnosis, facilitating accurate diagnoses and treatment decisions of healthcare professionals. In the realm of autonomous driving, DriveMLM [41] bridges the gap between language decisions and vehicle control commands, enabling closed-loop autonomous driving in realistic simulators. As the scale of LLM models substantially increases, they exhibit exceptional generalization capabilities—a phenomenon known as “emergence” [5]. One of the most representative examples is GPT-4 [36], due to its staggering model size, can successfully perform arithmetic operations such as numerical multiplication without specific targeted training. The outstanding capabilities of LLMs enable them to be directly applied or easily adapted (e.g., through fine-tuning or instruction adjustment) to various downstream or novel tasks, thereby unleashing unprecedented potential in applications such as chatbots, healthcare assistants, and intelligent transportation [1], [5], [41], [45]–[47].

### B. Split Learning

Split learning (SL) [26] has emerged as a promising distributed ML framework, capable of overcoming the weaknesses of FL [24], [48]. SL offloads the primary training workload to the server via model partitioning [29], [30], [49], while exchanging activation/activation's gradients with smaller data sizes rather than the entire model, thereby significantly reducing client-side computing costs and communication overhead during model training [28], [29]. However, the original SL, entitled vanilla SL [26], employs a sequential training mechanism from one device to another, substantially reducing training efficiency and resulting in excessive training latency.

To address this issue, split federated learning (SFL) [30] has been proposed, which merges the advantages of parallel

training from FL and model splitting from SL. Apart from model splitting, SFL features periodic server-side and client-side sub-model aggregation to achieve model synchronization after multiple training rounds, aligning with the design principle of FL [27], [29]. Due to its salient advantages, SFL has garnered significant attention from academia and industry in recent years. In academia, the predominant research efforts focus on the framework design of SFL to enhance model training efficiency, including model splitting [49], [50], model aggregation [49], client selection [51], and device clustering [50], [52]. In the industry, Huawei deems SFL as a pivotal learning framework for 6G edge intelligence due to its flexibility and inherent advantages [53].

The effectiveness of SFL methods has been validated in image classification and small-scale models (e.g., ResNet-18 [54] and VGG-16 [55]). However, the performance of SFL frameworks in the current LLM training paradigm remains unclear. Therefore, in this study, we are the first to investigate the performance of SFL frameworks in LLM training, providing novel insights into SL LLM training and establishing a foundation for future research.

### C. Parameter-Efficient Fine-Tuning

The conventional full-parameter fine-tuning paradigm (i.e., updating all parameters) is computationally expensive for fine-tuning LLMs on edge devices/servers. Moreover, for distributed learning paradigms such as FL and SL, full-parameter fine-tuning also incurs substantial communication overhead associated stemming from model aggregation, hindering the democratization of distributed learning for LLM fine-tuning paradigm. To tackle these challenges, parameter-efficient fine-tuning (PEFT) techniques are a promising solution. There are several representative parameter-efficient fine-tuning methods for LLM, including Adapter Tuning [56]–[58], Prompt Tuning [59], and Low-rank Adaptation (LoRA) [31], [32]. Adapter tuning inserts well-designed adapter modules between layers for training, while prompt tuning adds tunable prefix markers. LoRA decomposes incremental matrix of attention weight into low-rank matrices for model updating. The underlying principle of these methods is to train a small fraction of parameters, typically less than 1% of the original parameters, thus significantly reducing the number of trainable parameters.

Despite our framework can support many PEFT techniques such as Adapter Tuning [56]–[58], Prefix-Tuning [60], and P-Tuning [61], we prefer to employ LoRA [31] since it requires few trainable parameters for adaptation and does not introduce additional inference latency. LoRA is built on the insights that over-parametrized models essentially reside on a low intrinsic dimension [62], and thus, can be characterized with the reduced number of trainable parameters. Specifically, Let  $\mathbf{W} \in \mathbb{R}^{d \times m}$  denote one trainable weight matrix of LLM, its corresponding model update is represented as  $\mathbf{W} + \Delta\mathbf{W} = \mathbf{W} + \mathbf{AB}$ , where  $\mathbf{A} \in \mathbb{R}^{d \times r}$ ,  $\mathbf{B} \in \mathbb{R}^{r \times m}$  are decomposition matrices, and  $r \ll \min(d, m)$ . In this way, the number of trainable parameters  $\mathbf{W}$  could be less than 1% compared to the full-parameter fine-tuning, improving computing and communication efficiency. Therefore, we deploy LoRA fine-tuning technique in SplitLoRA framework.

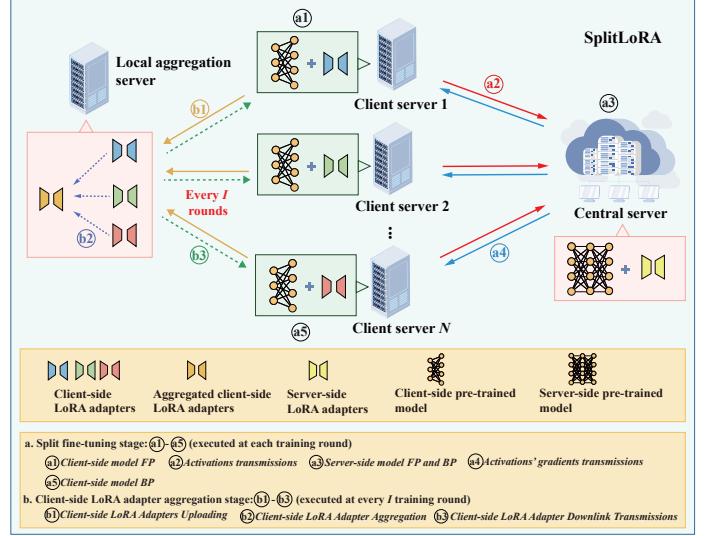


Fig. 1: Overview of proposed SplitLoRA framework.

### D. Split Learning and Large Language Models

Recently, there have been several preliminary papers exploring FL LLM PEFT paradigms [63]–[66]. Even though these frameworks have adopted various PEFT methods for LLM fine-tuning, such as Adapter [56]–[58] and LoRA [31], [32], however, LLM fine-tuning is still computational overload for resource-constrained computing entities. Unfortunately, although SL holds promise in addressing this issue, research on the SL LLM fine-tuning paradigm is still valid.

## III. SPLITLORA FRAMEWORK

In this section, we first present the system model and then overview the training procedures of SplitLoRA.

### A. System Model

In this section, we introduce the system model of SplitLoRA to provide a foundation for the following sections. As illustrated in Fig. 1, we consider a typical scenario of SplitLoRA, which consists of three fundamental components:

- **Client server:** We assume that each client server has sufficient computing capability to execute the forward propagation (FP) and back-propagation (BP) of the client-side pre-trained model. We denote  $\mathcal{N} = \{1, 2, \dots, N\}$  as the set of participating client servers, where  $N$  is the number of client servers. Each client server  $i$  has its local dataset  $\mathcal{D}_i = \{\mathbf{x}_{i,k}, y_{i,k}\}_{k=1}^{D_i}$  with  $D_i$  data samples, where  $\mathbf{x}_{i,k}$  and  $y_{i,k}$  represent the  $k$ -th input data and its corresponding label. The client-side pre-trained model of each client server is denoted as  $\mathbf{W}_c$ .  $\mathcal{R}_{c,i} = \{\mathbf{A}_{c,i}^1, \mathbf{B}_{c,i}^1, \mathbf{A}_{c,i}^2, \mathbf{B}_{c,i}^2, \dots, \mathbf{A}_{c,i}^{M_c}, \mathbf{B}_{c,i}^{M_c}\}$  represents the set of trainable LoRA adapters of the client-side pre-trained model for client server  $i$ , where  $\mathbf{A}_{c,i}^n$  and  $\mathbf{B}_{c,i}^n$  denote the decomposition matrices of  $n$ -th LoRA adapter of client server  $i$  and  $M_c$  is total number of trainable LoRA adapters of client-side pre-trained model.
- **Central server:** The central server is a powerful computing entity responsible for performing

server-side pre-trained model fine-tuning. The server-side pre-trained model is denoted by  $\mathbf{W}_s$ .  $\mathcal{R}_s = \{\mathbf{A}_s^1, \mathbf{B}_s^1, \mathbf{A}_s^2, \mathbf{B}_s^2, \dots, \mathbf{A}_s^{M_s}, \mathbf{B}_s^{M_s}\}$  represents the set of trainable LoRA adapters of the server-side pre-trained model, where  $\mathbf{A}_s^m$  and  $\mathbf{B}_s^m$  denote the decomposition matrices of  $m$ -th LoRA adapter and  $M_s$  is total number of trainable LoRA adapters of server-side pre-trained model.

- **Local aggregation server:** The local aggregation server takes charge of synchronizing client-side LoRA adapters, periodically aggregating them from all participating client servers. Due to privacy concerns, fed and central servers usually do not belong to the identical party since a malicious server could reconstruct the raw data by obtaining client-side pre-trained models and activations [67].

The global pre-trained model is denoted by  $\mathbf{W} = [\mathbf{W}_s; \mathbf{W}_c]$ . The objective of SFL is to find the optimal LoRA adapter configuration  $\mathcal{R}_{c,i}^*$  and  $\mathcal{R}_s^*$  that achieves good performance across all participating devices, which can be formulated as minimizing the following learning objective:

$$\min_{\mathcal{R}_{c,i}, \mathcal{R}_s} \sum_{i=1}^N \frac{|\mathcal{D}_i|}{|\mathcal{D}|} f_i(\mathbf{W} | \mathcal{R}_{c,i}, \mathcal{R}_s) \quad (1)$$

where  $\mathcal{D} = \bigcup_{i=1}^N \mathcal{D}_i$  is total dataset size and  $f_i(\mathbf{W} | \mathcal{R}_{c,i}, \mathcal{R}_s)$  denotes the local loss function of client server  $i$  over local dataset  $\mathcal{D}_i$ .

### B. The SplitLoRA Framework

This section provides a detailed overview of the workflow of the proposed SplitLoRA framework. The distinctive feature of SplitLoRA lies in the integration of the advantages of model splitting from SL and parallel training from FL. Moreover, SplitLoRA employs the state-of-the-art fine-tuning technique, LoRA, to further enhance the model training efficiency.

Before model training begins, the central server initializes the ML model and partitions it into client-side and server-side pre-trained models. Afterward, SplitLoRA conducts model fine-tuning over  $I$  consecutive training rounds, after that the client-side LoRA adapters aggregation is performed. This process loops until the model converges. The training process of SplitLoRA comprises two primary stages: split fine-tuning and client-side LoRA adapters aggregation. The split fine-tuning is executed in each training round, while client-side LoRA adapters aggregation occurs every  $I$  round. As illustrated in Fig. 1, for a training round  $t \in \mathcal{T} = \{1, 2, \dots, T\}$ , the training process of SplitLoRA is detailed as follows.

*a. Split Fine-Tuning Stage:* The split fine-tuning stage involves client-side and server-side fine-tuning for participating client servers and a central server in each training round, which consists of the following five steps.

*a1) Client-side Model Forward Propagation:* In this step, all participating client servers conduct the FP of client-side pre-trained model in parallel. Specifically, each client server  $i$  randomly selects a mini-batch  $\mathcal{B}_i \subseteq \mathcal{D}_i$  with  $b$  data samples from its local dataset for fine-tuning client-side pre-trained model. The input data and corresponding label of mini-batch in the training round  $t$  are denoted by  $\mathbf{x}_i^t$  and  $\mathbf{y}_i^t$ , respectively.  $\mathcal{R}_{c,i}^t = \{\mathbf{A}_{c,i}^{1,t}, \mathbf{B}_{c,i}^{1,t}, \mathbf{A}_{c,i}^{2,t}, \mathbf{B}_{c,i}^{2,t}, \dots, \mathbf{A}_{c,i}^{M_{c,i}}, \mathbf{B}_{c,i}^{M_{c,i}}\}$  represents

the set of trainable LoRA adapters of the client-side pre-trained model for client server  $i$  at the  $t$ -th training round. After feeding a mini-batch of data into the client-side pre-trained model, activations are generated at the cut layer. The activations of client server  $i$  are given by

$$\mathbf{s}_i^t = \varphi(\mathbf{W}_c | \mathcal{R}_{c,i}^{t-1}, \mathbf{x}_i^t), \quad (2)$$

where  $\varphi(w|r, x)$  denotes the mapping relationship between input data  $x$  and its predicted value given model parameter  $w$  and trainable LoRA adapters set  $r$ .

*a2) Activations Transmissions:* After completing the FP of client-side pre-trained models, all participating client servers transmit their respective activations and corresponding labels to the central server (usually over wireless channels). The collected activations from participating client servers are then utilized to fuel server-side model fine-tuning.

*a3) Server-side Model Forward Propagation and Back-propagation:* After receiving activations from participating client servers, the central server feeds these activations into the server-side pre-trained model to execute the server-side FP.  $\mathcal{R}_s^t = \{\mathbf{A}_s^{1,t}, \mathbf{B}_s^{1,t}, \mathbf{A}_s^{2,t}, \mathbf{B}_s^{2,t}, \dots, \mathbf{A}_s^{M_s,t}, \mathbf{B}_s^{M_s,t}\}$  represents the set of trainable LoRA adapters of the server-side pre-trained model at the  $t$ -th training round. The concatenated activation matrix  $\mathbf{S}^t$  is denoted by  $\mathbf{S}^t = [\mathbf{s}_1^t; \mathbf{s}_2^t; \dots; \mathbf{s}_N^t]$ . Thus, the predicted value is expressed as

$$\hat{\mathbf{y}}_i^t = \varphi(\mathbf{W}_s | \mathcal{R}_s^{t-1}, \mathbf{S}^t), \quad (3)$$

The predicted value and labels are utilized to calculate loss function value and further derive the server-side LoRA adapters' gradients. Therefore, the  $m$ -th server-side LoRA adapters can be updated through

$$\mathbf{A}_s^{m,t} \leftarrow \mathbf{A}_s^{m,t-1} - \gamma_s \mathbf{G}_{A,s}^{m,t}, \quad (4)$$

$$\mathbf{B}_s^{m,t} \leftarrow \mathbf{B}_s^{m,t-1} - \gamma_s \mathbf{G}_{B,s}^{m,t}, \quad (5)$$

where  $\mathbf{G}_{A,s}^{m,t}$  and  $\mathbf{G}_{B,s}^{m,t}$  denote the gradients of the decomposition matrices  $\mathbf{A}$  and  $\mathbf{B}$  of  $m$ -th LoRA adapter of server-side pre-trained model, and  $\gamma_s$  is the server-side learning rate.

*a4) Activations' Gradients Transmissions:* After the server-side BP is completed, the central server transmits the activations' gradients to its corresponding participating client servers.

*a5) Client-side Model Back-propagation:* In this step, each client server fine-tunes its client-side pre-trained model based on the received activations' gradients. For client server  $i$ , the  $n$ -th client-side LoRA adapters is updated through

$$\mathbf{A}_{c,i}^{n,t} \leftarrow \mathbf{A}_{c,i}^{n,t-1} - \gamma_c \mathbf{G}_{A,c,i}^{n,t}, \quad (6)$$

$$\mathbf{B}_{c,i}^{n,t} \leftarrow \mathbf{B}_{c,i}^{n,t-1} - \gamma_c \mathbf{G}_{B,c,i}^{n,t}, \quad (7)$$

where  $\mathbf{G}_{A,c,i}^{n,t}$  and  $\mathbf{G}_{B,c,i}^{n,t}$  denote the gradients of the decomposition matrices  $\mathbf{A}$  and  $\mathbf{B}$  of  $n$ -th LoRA adapter of server-side pre-trained model, and  $\gamma_c$  is the client-side learning rate.

*b. Client-side LoRA Adapter Aggregation Stage:* The client-side LoRA adapter aggregation stage primarily focuses on aggregating client-side LoRA adapters on the local aggregation server, which is executed every  $I$  training round. This stage consists of the following three steps.

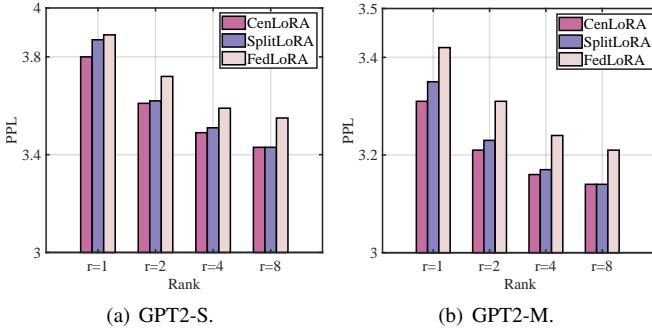


Fig. 2: The converged accuracy for GPT2-S and GPT2-M models, where Perplexity (PPL) is a metric used to measure how well LLMs predict a sample, with lower PPL indicating better predictive performance.

*b1) Client-side LoRA Adapters Uploading:* In this step, each participating client server sends its client-side LoRA adapters to the local aggregation server over the wireless/wired links.

*b2) Client-side LoRA Adapter Aggregation:* The local aggregation server aggregated the received client-side LoRA adapters into aggregated LoRA adapters. The decomposition matrices  $\mathbf{A}$  and  $\mathbf{B}$  of  $n$ -th client-side LoRA adapter are aggregated separately as follows

$$\mathbf{A}_c^{n,t} = \sum_{i=1}^N \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathbf{A}_{c,i}^{n,t}, \quad (8)$$

$$\mathbf{B}_c^{n,t} = \sum_{i=1}^N \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathbf{B}_{c,i}^{n,t}. \quad (9)$$

*b3) Client-side LoRA Adapter Downlink Transmissions:* After completing client-side LoRA adapter aggregation, the local aggregation server sends aggregated client-side LoRA adapter to the participating client servers. Afterward, each client server utilizes the received aggregated LoRA adapter as the initial LoRA configuration for the next training round.

The SplitLoRA training framework is outlined in **Algorithm 1**.

#### IV. PERFORMANCE EVALUATION

This section provides experimental results to evaluate the training performance of the SplitLoRA framework in terms of converged accuracy, convergence rate, and resource efficiency.

##### A. Experimental Setup

**Hardware:** We implement SplitLoRA using Python 3.7 and PyTorch 1.7.1., and train it with the NVIDIA GeForce RTX 3090 GPUs.

**Dataset and task:** We evaluate the training performance of SplitLoRA for natural language generation (NLG) tasks on E2E dataset [34]. The E2E dataset consists of around 42000 training, 4600 validation, and 4600 test examples from the restaurant domain.

**Model:** We employ well-known GPT2 small(GPT-S) and GPT2 medium (GPT2-M) models. GPT2-S is the smallest version in the GPT2 series with 124 million parameters and

---

##### Algorithm 1 The SplitLoRA Training Framework

---

**Input:**  $b, I, \gamma_c, \gamma_s, E, \mathcal{N}, \mathcal{R}_s^0, \mathcal{R}_c^0, \mathbf{W}_c, \mathbf{W}_s$  and  $\mathcal{D}$ .

**Output:**  $\mathcal{R}_s^*, \mathcal{R}_c^*$ .

```

1: Initialization:  $\mathcal{R}_{c,i}^0 \leftarrow \mathcal{R}_c^0$ .
2: for  $t = 1, 2, \dots, T$  do
3:
4:   /** Runs on edge servers **/
5:   for all client server  $i \in \mathcal{N}$  in parallel do
6:      $\mathbf{s}_i^t = \varphi(\mathbf{W}_c | \mathcal{R}_{c,i}^{t-1}, \mathbf{x}_i^t)$ 
7:     Send  $(\mathbf{s}_i^t, \mathbf{y}_i^t)$  to the central server
8:   end for
9:
10:  /** Runs on central server **/
11:   $\mathbf{S}^t = [\mathbf{s}_1^t; \mathbf{s}_2^t; \dots; \mathbf{s}_N^t]$ 
12:   $\hat{\mathbf{y}}_i^t = \varphi(\mathbf{W}_s | \mathcal{R}_s^{t-1}, \mathbf{S}^t)$ 
13:  Calculate loss function value  $f_i(\mathbf{W}^{t-1} | \mathcal{R}_{c,i}, \mathcal{R}_s)$ 
14:   $\mathbf{A}_s^{m,t} \leftarrow \mathbf{A}_s^{m,t-1} - \gamma_s \mathbf{G}_{A,s}^{m,t}$ 
15:   $\mathbf{B}_s^{m,t} \leftarrow \mathbf{B}_s^{m,t-1} - \gamma_s \mathbf{G}_{B,s}^{m,t}$ 
16:  Send activations' gradients to corresponding client servers
17:
18:  /** Runs on edge servers **/
19:  for all edge device  $i \in \mathcal{N}$  in parallel do
20:     $\mathbf{A}_{c,i}^{n,t} \leftarrow \mathbf{A}_{c,i}^{n,t-1} - \gamma_c \mathbf{G}_{A,c,i}^{n,t}$ 
21:     $\mathbf{B}_{c,i}^{n,t} \leftarrow \mathbf{B}_{c,i}^{n,t-1} - \gamma_c \mathbf{G}_{B,c,i}^{n,t}$ 
22:  end for
23:
24:  /** Runs on the fed server **/
25:  if  $T \bmod I = 0$  then
26:     $\mathbf{A}_c^{n,t} = \sum_{i=1}^N \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathbf{A}_{c,i}^{n,t}$ 
27:     $\mathbf{B}_c^{n,t} = \sum_{i=1}^N \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathbf{B}_{c,i}^{n,t}$ 
28:  end if
29: end for

```

---

12-layer Transformer encoders, while GPT2-M is the medium-sized version of GPT2, incorporating 355 million parameters, and 24-layer Transformer encoders.

**Experimental setup:** In the simulations, we deploy  $N = 3$  client servers by default unless specified otherwise. The client-side model of each client server has the first three Transformer layers of the GPT-2 model, while server-side model of central server holds the remaining 9 Transformer layers (GPT2-S) or 21 layers (GPT2-M). The computing capability of each client server is 35.6 TFLOTS (peak performance of one NVIDIA RTX 3090), while the central server's computing capability is set to 284.8 TFLOTS. The communication rate between the client server and the local aggregation server is set to 300 Mbps, and the communication rate between the client server and the central server is set to 600Mbps. We set the mini-batch size, learning rate, and maximum sequence length to 8, 0.0002, and 512 for the GPT2-S model, and 4, 0.0002, and 512 for the GPT2-M model. The rank of the LoRA adapter is set to  $r = \{1, 2, 4, 8\}$ .

GPT2-S	BLEU	NIST	METEOR	ROUGE_L	CIDEr
CenLoRA(r=1)	67.95	8.6973	0.4421	68.96	2.3412
CenLoRA(r=2)	68.49	8.7481	0.4491	68.70	2.3952
CenLoRA(r=4)	69.41	8.7824	0.4610	70.70	2.4713
CenLoRA(r=8)	69.37	8.7735	0.4624	70.96	2.4572
SplitLoRA(r=1)	67.18	8.6601	0.4416	67.71	2.3255
SplitLoRA(r=2)	66.86	8.5667	0.4515	68.50	2.3358
SplitLoRA(r=4)	68.79	8.7259	0.4572	69.84	2.4411
SplitLoRA(r=8)	68.76	8.6931	0.4588	70.17	2.4165
FedLoRA(r=1)	65.66	8.4123	0.4265	67.68	2.1921
FedLoRA(r=2)	67.24	8.6055	0.4398	69.33	2.3025
FedLoRA(r=4)	67.73	8.6148	0.4494	68.59	2.3817
FedLoRA(r=8)	68.39	8.6745	0.4590	70.24	2.4450

TABLE I: The comparison of converged accuracy on GPT2-S for E2E NLG challenge.

GPT2-M	BLEU	NIST	METEOR	ROUGE_L	CIDEr
CenLoRA(r=1)	69.86	8.7679	0.4650	71.20	2.5028
CenLoRA(r=2)	69.97	8.7787	0.4663	71.56	2.5029
CenLoRA(r=4)	69.78	8.7820	0.4667	71.62	2.5301
CenLoRA(r=8)	70.57	8.8557	0.4688	72.17	2.5405
SplitLoRA(r=1)	70.26	8.8274	0.4664	71.73	2.5267
SplitLoRA(r=2)	70.04	8.8031	0.4670	71.68	2.5233
SplitLoRA(r=4)	70.09	8.8075	0.4667	71.60	2.5370
SplitLoRA(r=8)	69.18	8.7189	0.4631	71.30	2.5156
FedLoRA(r=1)	67.02	8.6467	0.4484	68.06	2.3431
FedLoRA(r=2)	69.64	8.7727	0.4633	71.35	2.4900
FedLoRA(r=4)	69.78	8.7836	0.4642	71.87	2.4819
FedLoRA(r=8)	69.55	8.7358	0.4661	71.46	2.4980

TABLE II: The comparison of converged accuracy on GPT2-M for E2E NLG challenge.

**Benchmarks:** To investigate the advantages of SplitLoRA, we compare it with two canonical benchmarks: (1) **Centralized LoRA (CenLoRA):** Client server collects raw data from other participating servers for full model fine-tuning with LoRA adapters. (2) **Federated LoRA (FedLoRA):** Each participating client server locally fine-tunes the full model and then transmits the updated LoRA adapters to the central server for adapter aggregation.

### B. Performance Evaluation

**Converged accuracy:** Fig. 2 compares the training performance of SplitLoRA with other benchmarks on GPT2-S and GPT2-M models for the E2E NLG challenge, with PPL as the performance metric. FedLoRA exhibits lower converged accuracy than SplitLoRA and CenLoRA, with PPL approximately 0.08/0.11 (GPT2-S/GPT2-M) and 0.73/0.09 (GPT2-S/GPT2-M) higher than SplitLoRA and CenLoRA, respectively. The poorer training performance of FedLoRA stems from its fully distributed training paradigm, where the entire model is updated via model aggregation, making it susceptible to model bias caused by data heterogeneity across client servers. SplitLoRA achieves converged accuracy comparable to that of cenLoRA, especially on GPT2-M, where the accuracy difference is less than 0.04. The reason for this is two-fold: One is that server sub-models in SplitLoRA are trained in a centralized manner, making it more resilient to

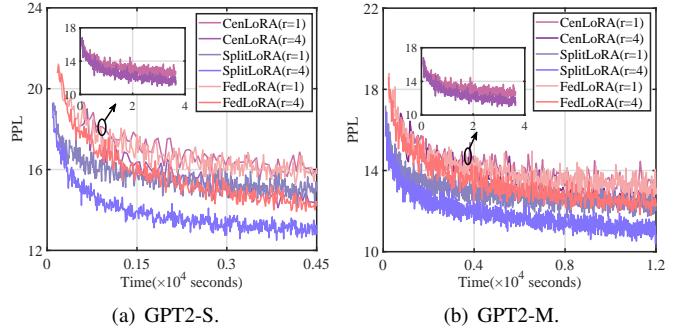


Fig. 3: The training performance on GPT2-S and GPT2-M for E2E NLG challenge.

	GPT2-S	GPT2-M
CenLoRA(r=1)	0.031M	0.089M
CenLoRA(r=2)	0.062M	0.178M
CenLoRA(r=4)	0.124M	0.355M
CenLoRA(r=8)	0.248M	0.710M
SplitLoRA(r=1)	0.008M	0.011M
SplitLoRA(r=2)	0.015M	0.022M
SplitLoRA(r=4)	0.031M	0.044M
SplitLoRA(r=8)	0.062M	0.088M
FedLoRA(r=1)	0.031M	0.089M
FedLoRA(r=2)	0.062M	0.178M
FedLoRA(r=4)	0.124M	0.355M
FedLoRA(r=8)	0.248M	0.710M

TABLE III: The comparison of the number of trainable parameters on GPT2-S and GPT2-M.

data heterogeneity. The other is that SplitLoRA offloads the major workload to a central server, with only a small portion of LLM affected by client data heterogeneity. It can be seen that PPL of SplitLoRA and CenLoRA becomes closer with the rank  $r$  increases. This is because that higher  $r$  implies a larger number of trainable parameters, thereby enhancing the fitting capability of the server-side sub-model. This may partially counteract the effect of data heterogeneity, bringing the performance of SplitLoRA closer to that of CenLoRA. The comparison of training performance on other metrics is shown in Table I and II.

**Convergence rate:** Fig. 3(a) and Fig. 3(b) show the training performance of SplitLoRA and other benchmarks on GPT2-S and GPT-M for E2E NLG challenge. It is clear that SplitLoRA has an overwhelming superiority in convergence speed over FedLoRA and CenLoRA. The training latency of FedLoRA and CenLoRA for achieving model convergence is approximately 1.7 times and 4.7 times and 2.1 times and 4.8 times that of SplitLoRA on GPT-S and GPT-M, respectively. Compared to CenLoRA, FedLoRA expedites model convergence due to its parallel training paradigm of multiple client servers. SplitLoRA is built on FedLoRA by offloading the major computing workload to more powerful central server, further reducing the time required for achieving model convergence while supporting parallel training across multiple client servers.

**Trainable parameter:** Table III presents the number of trainable parameters on GPT2-S and GPT2-M. SplitLoRA re-

duces the computing workload on client servers by partitioning the model and retaining fewer Transformer layers locally. In our experimental setup, for the GPT2-S and GPT2-M models, the parameters involved in fine-tuning on client servers in SplitLoRA constitute only one-fourth and one-eighth of the entire model, respectively. This substantial reduction in local computing requirements renders SplitLoRA highly efficient and suitable for deployment in resource-limited environments. In contrast, CenLoRA and FedLoRA require client servers to handle the entire GPT-2 model for fine-tuning, which imposes significant demands on computing capabilities and memory resources. In particular, in 5G and beyond edge computing systems, small base stations and access points with limited capabilities serve as client servers. Therefore, SplitLoRA not only improves computing efficiency but also enhances the flexibility and adaptability of the system to better cope with varying computational and resource-limited scenarios.

## V. DISCUSSIONS AND FUTURE DIRECTIONS

Previously, we have demonstrated the proposed SplitLoRA framework and its outstanding performance. However, this is not the end of the road, as our framework holds potential for further refinement to adapt diverse scenarios and applications. This section delves into the emerging challenges and intriguing directions for future exploration.

### A. Model Splitting in SplitLoRA

For the SL LLM fine-tuning framework, research on the model splitting is of paramount importance. The cut layer plays a pivotal role in SL, determining the division of computing workload and the volume of data exchanged between client servers/devices and the central server [5], [29]. LLMs typically consist of billions of parameters, with significant variations in the output size between different layers [5], [29], [68]. Therefore, selecting the cut layer can control the data volume transmitted to the central server. Additionally, the cut layer also has impacts on the division of computing workload between client servers/devices and the central server, e.g., deeper cut layers entail a larger computing workload offloaded to the central server. The above observations reveal that the cut layer selection has a dual impact on both communication and computing efficiency, underscoring its significance in SL LLM fine-tuning.

### B. Heterogeneous Configuration in SplitLoRA

In practice, the available resources among different client servers/devices vary greatly, and the resources allocated for training may change during runtime, depending on how the on-demand running procedure prioritizes the resource allocation. More importantly, due to the large-scale parameters of LLM, the resource heterogeneity of client servers/devices can lead to vastly different training times, causing a severe straggler problem in model aggregation. Therefore, it is essential to configure heterogeneous fine-tuning module structures to client servers/devices with heterogeneous resources. For instance, in SplitLoRA, joint selecting cut layers and ranks of LoRA adapters is of paramount importance. Recalling section V-A, cut layer selection determines the division of computing

workload and the volume of data exchanged between client servers/devices and the central server. Therefore, it is an intuitive idea to select distinct cut layers for client servers/devices with heterogeneous resources. Moreover, the rank of client-side LoRA adapters simultaneously affects the computing burden for client-side LLM fine-tuning and the communication overhead required for client-side LoRA adapter aggregation. Thus, for SplitLoRA, how to jointly select cut layers and ranks of LoRA adapters according to heterogeneous resources is a worthwhile endeavor.

### C. Efficiency in SplitLoRA

SplitLoRA typically involves multiple organizations or data centers, each with sufficient computing resources. In this case, seamless training of SplitLoRA is feasible, as each entity conducting local LLM fine-tuning may possess hardware capabilities similar to or exceeding those of the NVIDIA 3090 GPU. This setting allows for more straightforward collaborative LLM fine-tuning, facilitating the handling of larger model parameters and more complex training routines due to the higher computing resources available. Conversely, the practical implementation of SplitLoRA usually necessitates leveraging private data residing on edge devices (e.g., smartphones and laptop) with lower computing and storage resources than data centers for LLM fine-tuning. Training an LLM with billions of parameters across edge devices poses significant barriers due to the limited storage and computing capabilities. Therefore, efficiency becomes even more critical since the LLMs are usually much larger than conventional models used in previous SL literature [26], [27], [29].

Fortunately, recent advancements in model compression, such as knowledge distillation [69] and pruning [70] and quantization [71] techniques, offer promising solutions to reduce model size and computational complexity without significantly compromising performance. These techniques have the potential to enable the deployment of smaller, more efficient versions of LLMs. Therefore, designing efficient model compression and quantization techniques to achieve a more storage, communication, and computation-efficient SL LLM fine-tuning framework is warranted.

### D. Privacy Preservation in SplitLoRA

Deep learning models, those of substantial size, are capable of memorizing training data, which could raise privacy concerns [72], [73]. This risk is particularly severe in LLM fine-tuning due to their powerful capability, which might inadvertently memorize and potentially expose more detailed information [74]. Therefore, it is crucial to design efficient privacy-preserving mechanisms/strategies that guarantee the training performance and effectiveness of SplitLoRA without compromising individual privacy. Two promising directions are worth exploring in the future: one potential avenue is to extend the SplitLoRA framework to a U-shaped paradigm. The SplitLoRA ultimately gets the final inference results in the cloud, however, in the context where training tasks are highly privacy-sensitive (e.g., medical image analysis), this can lead to data privacy leakage of edge devices/servers [75], [76]. The design of the U-shaped framework can fundamentally

address this issue by placing the deeper layers of LLM at the edge devices/servers. The other direction involves the implementation of differential privacy techniques, which add controlled noise to model gradients or updates [77], providing theoretical privacy guarantees for SplitLoRA.

## VI. CONCLUSION

In this paper, we propose the first SL LLM fine-tuning framework, named SplitLoRA. SplitLoRA is built on the split federated learning (SFL) framework, amalgamating the advantages of parallel training from FL and model splitting from SL and thus greatly enhancing the training efficiency. It is worth noting that SplitLoRA is the inaugural open-source benchmark for SL LLM fine-tuning, providing a foundation for research efforts dedicated to advancing SL LLM fine-tuning. Extensive simulations validate that SplitLoRA achieves target accuracy in significantly less time than centralized LLM fine-tuning frameworks, demonstrating the superior training performance of SplitLoRA. We have discussed emerging challenges and research directions in SplitLoRA, where we advocate more future efforts in this realm.

## REFERENCES

- [1] OpenAI, “GPT-4 Technical Report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open Foundation and Fine-tuned Chat Models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, “Palm: Scaling Language Modeling with Pathways,” *arXiv preprint arXiv:2204.02311*, 2022.
- [4] L. Cardenas, K. Parajes, M. Zhu, and S. Zhai, “AutoHealth: Advanced LLM-Empowered Wearable Personalized Medical Butler for Parkinson’s Disease Management,” in *Proc. CCWC*, 2024.
- [5] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, and K. Huang, “Pushing Large Language Models to the 6G Edge: Vision, Challenges, and Opportunities,” *arXiv preprint arXiv:2309.16739*, 2023.
- [6] W. Berrios, G. Mittal, T. Thrush, D. Kiela, and A. Singh, “Towards Language Models that Can See: Computer Vision Through the Lens of Natural Language,” *arXiv preprint arXiv:2306.16410*, 2023.
- [7] Y. Qiu, H. Chen, X. Dong, Z. Lin, I. Y. Liao, M. Tistarelli, and Z. Jin, “Ifvit: Interpretable Fixed-length Representation for Fingerprint Matching via Vision Transformer,” *arXiv preprint arXiv:2404.08237*, 2024.
- [8] W. Wang, Z. Chen, X. Chen, J. Wu, X. Zhu, G. Zeng, P. Luo, T. Lu, J. Zhou, Y. Qiao *et al.*, “Visionllm: Large Language Model is Also An Open-ended Decoder For Vision-centric Tasks,” *Proc. NIPS*, 2024.
- [9] S. Hu, Z. Fang, Z. Fang, X. Chen, and Y. Fang, “AgentsCoDriver: Large Language Model Empowered Collaborative Driving with Lifelong Learning,” *arXiv preprint arXiv:2404.06345*, 2024.
- [10] X. Zhou, M. Liu, B. L. Zagar, E. Yurtsever, and A. C. Knoll, “Vision Language Models in Autonomous Driving and Intelligent Transportation Systems,” *arXiv preprint arXiv:2310.14414*, 2023.
- [11] Y. Tian, X. Li, H. Zhang, C. Zhao, B. Li, X. Wang, and F.-Y. Wang, “VistaGPT: Generative Parallel Transformers for Vehicles with Intelligent Systems for Transport Automation,” *IEEE Trans. Intell. Veh.*, 2023.
- [12] G. Qu, Z. Lin, Q. Chen, J. Li, F. Liu, X. Chen, and K. Huang, “Trim-Caching: Parameter-sharing Edge Caching for AI Model Downloading,” *arXiv preprint arXiv:2404.14204*, 2024.
- [13] P. Villalobos, J. Sevilla, L. Heim, T. Besiroglu, M. Hobbhahn, and A. Ho, “Will We Run Out of Data? An Analysis of the Limits of Scaling Datasets in Machine Learning,” *arXiv preprint arXiv:2211.04325*, 2022.
- [14] Y. Wang, H. Ivison, P. Dasigi, J. Hessel, T. Khot, K. R. Chandu, D. Wadden, K. MacMillan, N. A. Smith, I. Beltagy *et al.*, “How Far Can Camels Go? Exploring the State of Instruction Tuning on Open Resources,” *arXiv preprint arXiv:2306.04751*, 2023.
- [15] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang, “Wizardlm: Empowering large language models to follow complex instructions,” *arXiv preprint arXiv:2304.12244*, 2023.
- [16] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling Laws for Neural Language Models,” *arXiv preprint arXiv:2001.08361*, 2020.
- [17] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, “Large Language Models in Medicine,” *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023.
- [18] S. Wu, O. Irsay, S. Lu, V. Dabrowski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, “Bloomberggpt: A Large Language Model for Finance,” *arXiv preprint arXiv:2303.17564*, 2023.
- [19] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, L. Hou, K. Clark, S. Pfohl, H. Cole-Lewis, D. Neal *et al.*, “Towards Expert-level Medical Question Answering with Large Language Models,” *arXiv preprint arXiv:2305.09617*, 2023.
- [20] R. Ye, W. Wang, J. Chai, D. Li, Z. Li, Y. Xu, Y. Du, Y. Wang, and S. Chen, “OpenFedLLM: Training Large Language Models on Decentralized Private Data via Federated Learning,” *arXiv preprint arXiv:2402.06954*, 2024.
- [21] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, “Efficient Federated Learning for Modern NLP,” in *Proc. Mobicom*, 2023.
- [22] Z. Fang, Z. Lin, Z. Chen, X. Chen, Y. Gao, and Y. Fang, “Automated Federated Pipeline for Parameter-efficient Fine-tuning of Large Language Models,” *arXiv preprint arXiv:2404.06448*, 2024.
- [23] Z. Lin, Z. Chen, Z. Fang, X. Chen, X. Wang, and Y. Gao, “FedSN: A General Federated Learning Framework over LEO Satellite Networks,” *arXiv preprint arXiv:2311.01483*, 2023.
- [24] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated Learning: Strategies for Improving Communication Efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
- [25] Y. Zhang, H. Chen, Z. Lin, Z. Chen, and J. Zhao, “FedAC: A Adaptive Clustered Federated Learning Framework for Heterogeneous Data,” *arXiv preprint arXiv:2403.16460*, 2024.
- [26] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, “Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data,” *arXiv preprint arXiv:1812.00564*, 2018.
- [27] Z. Lin, G. Zhu, Y. Deng, X. Chen, Y. Gao, K. Huang, and Y. Fang, “Efficient Parallel Split Learning over Resource-constrained Wireless Edge Networks,” *IEEE Trans. Mobile Comput.*, 2024.
- [28] S. Lyu, Z. Lin, G. Qu, X. Chen, X. Huang, and P. Li, “Optimal Resource Allocation for U-shaped Parallel Split Learning,” *arXiv preprint arXiv:2308.08896*, 2023.
- [29] Z. Lin, G. Qu, X. Chen, and K. Huang, “Split Learning in 6G Edge Networks,” *IEEE Wireless Commun.*, 2024.
- [30] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, “Splitfed: When Federated Learning Meets Split Learning,” in *Proc. AAAI*, 2022.
- [31] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank Adaptation of Large Language Models,” *arXiv preprint arXiv:2106.09685*, 2021.
- [32] Y. Sheng, S. Cao, D. Li, C. Hooper, N. Lee, S. Yang, C. Chou, B. Zhu, L. Zheng, K. Keutzer *et al.*, “S-LoRA: Serving Thousands of Concurrent LoRA Adapters,” *arXiv preprint arXiv:2311.03285*, 2023.
- [33] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language Models are Unsupervised Multitask Learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [34] J. Novikova, O. Dušek, and V. Rieser, “The E2E Dataset: New Challenges For End-to-End Generation,” in *Proc. SIGDIAL*, 2017.
- [35] R. Dale, “GPT-3: What’s It Good For?” *Nat. Lang. Eng.*, vol. 27, no. 1, pp. 113–118, 2021.
- [36] K. Sanderson, “GPT-4 Is Here: What Scientists Think,” *Nature*, vol. 615, no. 7954, p. 773, 2023.
- [37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [38] L. Jiang, F. Svoboda, and N. D. Lane, “FDAPT: Federated Domain-adaptive Pre-training for Language Models,” *arXiv preprint arXiv:2307.06933*, 2023.
- [39] Z. Englhardt, C. Ma, M. E. Morris, X. Xu, C.-C. Chang, L. Qin, X. Liu, S. Patel, V. Iyer *et al.*, “From Classification to Clinical Insights: Towards Analyzing and Reasoning About Mobile and Behavioral Health Data With Large Language Models,” *arXiv preprint arXiv:2311.13063*, 2023.
- [40] Z. Nan, H. Guan, X. Shen, and C. Liao, “Deep NLP-Based Co-evolution for Synthesizing Code Analysis from Natural Language,” in *Proc. of CC*, 2021.

- [41] W. Wang, J. Xie, C. Hu, H. Zou, J. Fan, W. Tong, Y. Wen, S. Wu, H. Deng, Z. Li *et al.*, “DriveMLM: Aligning Multi-modal Large Language Models with Behavioral Planning States for Autonomous Driving,” *arXiv preprint arXiv:2312.09245*, 2023.
- [42] C. Cui, Y. Ma, X. Cao, W. Ye, Y. Zhou, K. Liang, J. Chen, J. Lu, Z. Yang, K.-D. Liao *et al.*, “A Survey on Multimodal Large Language Models for Autonomous Driving,” in *Proc. WCAC*, 2024.
- [43] J. Mai, J. Chen, B. Li, G. Qian, M. Elhoseiny, and B. Ghanem, “LLM As A Robotic Brain: Unifying Ego-centric Memory and Control,” *arXiv preprint arXiv:2304.09349*, 2023.
- [44] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, “Smart-LLM: Smart Multi-agent Robot Task Planning Using Large Language Models,” *arXiv preprint arXiv:2309.10062*, 2023.
- [45] S. Hu, Z. Fang, H. An, G. Xu, Y. Zhou, X. Chen, and Y. Fang, “Adaptive Communications in Collaborative Perception with Domain Alignment for Autonomous Driving,” *arXiv preprint arXiv:2310.00013*, 2023.
- [46] Z. Lin, L. Wang, J. Ding, B. Tan, and S. Jin, “Channel power gain estimation for terahertz vehicle-to-infrastructure networks,” *IEEE Commun. Lett.*, vol. 27, no. 1, pp. 155–159, 2022.
- [47] S. Hu, Z. Fang, Y. Deng, X. Chen, and Y. Fang, “Collaborative Perception for Connected and Autonomous Driving: Challenges, Possible Solutions and Opportunities,” *arXiv preprint arXiv:2401.01544*, 2024.
- [48] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient Learning of Deep Networks From Decentralized Data,” in *Proc. AISTATS*, 2017.
- [49] Z. Lin, G. Qu, W. Wei, X. Chen, and K. K. Leung, “AdaptSFL: Adaptive Split Federated Learning in Resource-constrained Edge Networks,” *arXiv preprint arXiv:2403.13101*, 2024.
- [50] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, “Split Learning over Wireless Networks: Parallel Design and Resource Management,” *IEEE J. Select. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, 2023.
- [51] X. Liu, Y. Deng, and T. Mahmoodi, “Wireless Distributed Learning: A New Hybrid Split and Federated Learning Approach,” *IEEE Trans. Wireless Commun.*, vol. 22, no. 4, pp. 2650–2665, 2022.
- [52] Z. Cheng, X. Xia, M. Liwang, X. Fan, Y. Sun, X. Wang, and L. Huang, “CHEESE: Distributed Clustering-based Hybrid Federated Split Learning over Edge Networks,” *IEEE Trans. Parallel Distrib. Syst.*, 2023.
- [53] Huawei, *NET4AI: Supporting AI as a Service in 6G*. Cambridge, U.K.: Cambridge Univ. Press, 2022.
- [54] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proc. CVPR*, 2016, pp. 770–778.
- [55] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-scale Image Recognition,” in *Proc. ICLR*, 2015.
- [56] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, “Parameter-Efficient Transfer Learning for NLP,” in *Proc. ICML*, 2019, pp. 2790–2799.
- [57] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, “Adapter-Fusion: Non-Destructive Task Composition for Transfer Learning,” in *Proc. EACL*, 2021, pp. 487–503.
- [58] R. Karimi Mahabadi, J. Henderson, and S. Ruder, “Compacter: Efficient Low-Rank Hypercomplex Adapter Layers,” 2021, pp. 1022–1035.
- [59] L. Li, Y. Zhang, and L. Chen, “Prompt Distillation for Efficient LLM-based Recommendation,” in *Proc. CIKM*, 2023, pp. 1348–1357.
- [60] X. L. Li and P. Liang, “Prefix-tuning: Optimizing Continuous Prompts for Generation,” *arXiv preprint arXiv:2101.00190*, 2021.
- [61] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, “P-tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks,” *arXiv preprint arXiv:2110.07602*, 2021.
- [62] A. Aghajanyan, S. Gupta, and L. Zettlemoyer, “Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning,” in *Proc. IJCNLP*, 2021, pp. 7319–7328.
- [63] C. Chen, X. Feng, J. Zhou, J. Yin, and X. Zheng, “Federated Large Language Model: A Position Paper,” *arXiv preprint arXiv:2307.08925*, 2023.
- [64] T. Fan, Y. Kang, G. Ma, W. Chen, W. Wei, L. Fan, and Q. Yang, “Fate-LLM: A Industrial Grade Federated Learning Framework for Large Language Models,” *arXiv preprint arXiv:2310.10049*, 2023.
- [65] W. Kuang, B. Qian, Z. Li, D. Chen, D. Gao, X. Pan, Y. Xie, Y. Li, B. Ding, and J. Zhou, “FederatedScope-LLM: A Comprehensive Package for Fine-tuning Large Language Models in Federated Learning,” *arXiv preprint arXiv:2309.00363*, 2023.
- [66] J. Zhang, S. Vahidian, M. Kuo, C. Li, R. Zhang, G. Wang, and Y. Chen, “Towards Building the Federated GPT: Federated Instruction Tuning,” *arXiv preprint arXiv:2305.05644*, 2023.
- [67] D. Pasquini, G. Ateniese, and M. Bernaschi, “Unleashing the Tiger: Inference Attacks on Split Learning,” in *Proc. CCS*, 2021.
- [68] G. Qu, Z. Lin, F. Liu, X. Chen, and K. Huang, “TrimCaching: Parameter-sharing AI Model Caching in Wireless Edge Networks,” in *Proc. ICDCS*, 2024.
- [69] B. Wang, Y. J. Zhang, Y. Cao, B. Li, H. B. McMahan, S. Oh, Z. Xu, and M. Zaheer, “Can Public Large Language Models Help Private Cross-device Federated Learning?” *arXiv preprint arXiv:2305.12132*, 2023.
- [70] M. Xia, T. Gao, Z. Zeng, and D. Chen, “Sheared llama: Accelerating Language Model Pre-training Via Structured Pruning,” *arXiv preprint arXiv:2310.06694*, 2023.
- [71] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient Finetuning of Quantized LLMs,” *Proc. NIPS*, 2024.
- [72] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks Against Centralized and Federated Learning,” in *Proc. SP*, 2019.
- [73] S. Gupta, Y. Huang, Z. Zhong, T. Gao, K. Li, and D. Chen, “Recovering Private Text in Federated Learning of Language Models,” *Proc. NIPS*, 2022.
- [74] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, “Extracting Training Data From Large Language Models,” in *USENIX Security*, 2021, pp. 2633–2650.
- [75] T. Ni, G. Lan, J. Wang, Q. Zhao, and W. Xu, “Eavesdropping Mobile App Activity via Radio-Frequency Energy Harvesting,” in *Proc. USENIX Security Symposium*, 2023.
- [76] T. Ni, X. Zhang, and Q. Zhao, “Recovering Fingerprints from In-Display Fingerprint Sensors via Electromagnetic Side Channel,” in *Proc. ACM CCS*, 2023.
- [77] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, “Federated Learning with Differential Privacy: Algorithms and Performance Analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3454–3469, 2020.