HSplitLoRA: A Heterogeneous Split Parameter-Efficient Fine-Tuning Framework for Large Language Models

Zheng Lin, Yuxin Zhang, Zhe Chen, *Member, IEEE*, Zihan Fang, Xianhao Chen, *Member, IEEE*, Praneeth Vepakomma, *Member, IEEE*, Wei Ni, *Fellow, IEEE*, Jun Luo, *Fellow, IEEE*, and Yue Gao, *Fellow, IEEE*

Abstract—Recently, large language models (LLMs) have achieved remarkable breakthroughs, revolutionizing the natural language processing domain and beyond. Due to immense parameter sizes, fine-tuning these models with private data for diverse downstream tasks has become mainstream. Though federated learning (FL) offers a promising solution for fine-tuning LLMs without sharing raw data, substantial computing costs hinder its democratization. Moreover, in real-world scenarios, private client devices often possess heterogeneous computing resources, further complicating LLM fine-tuning. To combat these challenges, we propose HSplitLoRA, a heterogeneous parameter-efficient fine-tuning (PEFT) framework built on split learning (SL) and low-rank adaptation (LoRA) fine-tuning, for efficiently fine-tuning LLMs on heterogeneous client devices. HSplitLoRA first identifies important weights based on their contributions to LLM training. It then dynamically configures the decomposition ranks of LoRA adapters for selected weights and determines the model split point according to varying computing budgets of client devices. Finally, a noise-free adapter aggregation mechanism is devised to support heterogeneous adapter aggregation without introducing noise. Extensive experiments demonstrate that HSplitLoRA outperforms state-of-the-art benchmarks in training accuracy and convergence speed.

Index Terms—Distributed learning, split learning, large language model, parameter-efficient fine-tuning.

I. INTRODUCTION

Recently, large language models (LLMs) have achieved tremendous success across a broad spectrum of pivotal sectors due to their exceptional ability in handling high-complexity and large-scale datasets [1]–[5]. Notable examples include OpenAI's GPT series [1], Google's Gemini [2], and Meta's LLaMA [3], all of which have gained significant attention in both industry and academia. These models excel in extracting intricate patterns from vast datasets and adapting to diverse downstream tasks, ranging from natural language understanding and generation to specialized domains such as smart healthcare [6], [7], finance [8], and intelligent transportation [9], [10]. However, due to immense parameter sizes, training LLMs from scratch is typically infeasible [11], rendering fine-tuning with private data for specific downstream tasks become mainstream. The workflow of fine-tuning LLMs typically follows a three-stage process: i) pre-training LLMs from scratch on extensive text corpora (e.g., Wikipedia) using substantial computing resources (e.g., training GPT3-1.3B model requires 64 Tesla V100 GPUs running for one week [12]); ii) fine-tuning pre-trained LLMs to adapt to specific downstream tasks; iii) deploying the fine-tuned LLMs on client devices to perform inference.

Unfortunately, privacy concerns often hinder the collaborative fine-tuning of LLMs across multiple parties, even when they share the same downstream task and could mutually benefit. For example, several freelance financial analysts may wish to collaboratively fine-tune an LLM for a special financial risk prediction task via their personal devices, but they are reluctant to share customer data [8]. Federated learning (FL) [13], [14] offers a promising solution to this privacy issue [15], [16] by enabling participants to fine-tune LLMs using local data without sharing the raw data [17], [18]. However, fine-tuning LLMs via FL is non-trivial, particularly for resource-constrained client devices. In typical configurations, client devices are equipped with low-cost commercial GPUs, such as the NVIDIA GeForce RTX 3050 in a laptop. While these GPUs are sufficient for conventional deep learning tasks such as object detection [19] and image classification [20], they can be overwhelmed by the immense computing workload of full parameter fine-tuning of LLMs within standard FL frameworks. To combat this issue, some recent studies [21], [22] focus on integrating FL with parameter-efficient fine-tuning (PEFT) methods, such as Adapter [23] and low-rank adaptation (LoRA) [11]. Despite these efforts, fine-tuning LLMs via FL on client devices remains prohibitively time- and resource-consuming. For instance, fine-tuning LLaMA 3-8B [3] requires over 20GB of GPU memory, which exceeds the capabilities of most client devices.

Split learning (SL) [24] has emerged as a compelling alternative capable of overcoming the weaknesses of FL. SL offloads the primary training workload to a central server

Z. Lin, Y. Zhang, Z. Chen, Z. Fang, and Y. Gao are with the Institute of Space Internet, Fudan University, Shanghai 200438, China, and the School of Computer Science, Fudan University, Shanghai 200438, China (e-mail: zlin20@fudan.edu.cn; zhfang19@fudan.edu.cn; zhechen@fudan.edu.cn; yxzhang24@m.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).

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).

W. Ni is with Data61, CSIRO, Marsfield, NSW 2122, Australia, and the School of Computing Science and Engineering, and the University of New South Wales, Kennington, NSW 2052, Australia (e-mail: wei.ni@ieee.org).

J. Luo is with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: junluo@ntu.edu.sg).
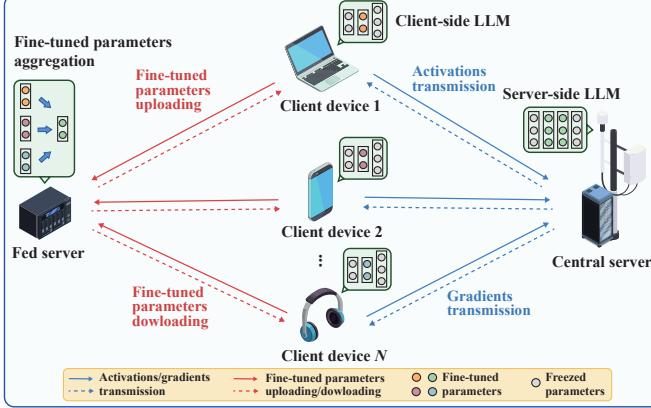
(Corresponding author: Yue Gao)

Fig. 1: A scenario of LLM SL via PEFT with client devices.

via layer-wise model partitioning, alleviating the computing burden on resource-constrained client devices [25]. Fig. 1 illustrates how PEFT integrates with the state-of-the-art SL implementation, split federated learning (SFL) [26], for LLM fine-tuning. The pre-trained LLM is split into two sub-models: client-side LLM deployed on the client devices and server-side LLM deployed on a central server. Each client device executes forward propagation on the client-side LLM and transmits activations to the central server. Then, the central server leverages the received activations to fine-tune the server-side LLMs and send gradients to client devices for client-side back-propagation. After the training is completed, the fed server aggregates the client-side fine-tuned parameters and redistributes them to client devices. This process repeats until the LLM reaches a pre-defined accuracy threshold.

While integrating SL and PEFT techniques holds significant promise, its implementation encounters several critical challenges. First, the limited computing resources[1] of client devices render fine-tuning all trainable parameters impractical, necessitating selectively fine-tuning a portion of trainable weights. However, this paradigm remains unexplored in the SL paradigm. Second, the inherent heterogeneity of computing resources across client devices leads to a severe device unavailability problem[2] [30]–[32], ultimately degrading the training performance. Last but not least, the varying computing budgets across client devices result in structural discrepancies in fine-tuning adapters, making it infeasible to aggregate these adapters across diverse client devices. Empirical studies have been conducted to validate these challenges, as will be discussed in Section II.

To tackle these challenges, we propose a <u>h</u>eterogeneous PEFT framework built on <u>s</u>plit learning and <u>LoRA</u> fine-tuning

---

[1]In this paper, we use GPU memory footprint as a proxy metric to quantify computing resource availability [27]–[29]. This is because GPU memory footprint directly correlates with a model's computing demands, especially for LLM fine-tuning, where memory-intensive operations such as multi-head self-attention and large-scale matrix multiplications dominate. GPU memory footprint provides an effective proxy for evaluating the feasibility of model fine-tuning on resource-constrained client devices.

[2]The device unavailability problem occurs when participating devices fail to perform local model training or data transmission due to disconnections, network instability, or insufficient computing resources, thus hindering global model updates and degrading training performance.

method, named HSplitLoRA. First, in order to efficiently fine-tune the trainable weights under computing resource constraints, we design an important weight identification scheme to identify important weights based on their contributions to LLM training. Second, to accommodate the heterogeneous computing capabilities of edge servers, we propose a dynamic rank and model splitting configuration to dynamically adjust the decomposition ranks of LoRA adapters and the model split point to align with client devices' computing budgets. Finally, to seamlessly aggregate heterogeneous LoRA adapters, we develop a noise-free adapter aggregation that meticulously concatenates low-rank decomposition matrices to support heterogeneous adapter aggregation without introducing noise. The key contributions of this paper are summarized as follows:

- To the best of our knowledge, HSplitLoRA is the first heterogeneous LLM fine-tuning framework that integrates SL and PEFT, enabling distributed collaborative learning across heterogeneous client devices.
- We devise an important weight identification scheme to quantify the contribution of trainable weights to training performance to prioritize important weights for fine-tuning.
- We propose an adaptive rank and model splitting configuration to adjust the decomposition ranks of LoRA adapters and the model split point to accommodate heterogeneous computing budgets of client devices.
- We develop a noise-free adapter aggregation that meticulously concatenates low-rank decomposition matrices to support heterogeneous adapter aggregation without introducing additional noise.
- We empirically evaluate HSplitLoRA with extensive experiments. The results demonstrate that HSplitLoRA outperforms state-of-the-art benchmarks, significantly in training accuracy and convergence speed.

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III elaborates on the system design of HSplitLoRA. Section IV presents the system implementation, followed by the performance evaluation in Section V. Related works and technical limitations are discussed in Section VI. Finally, conclusions and future remarks are presented in Section VII.

II. BACKGROUND AND MOTIVATION

In this section, we begin by introducing the background of PEFT for LLM SL. Then, we conduct extensive pilot measurement studies to investigate the design challenges of SL on LLM PEFT, serving as the foundation for the design motivation behind HSplitLoRA.

A. Injecting PEFT into Split Learning for LLMs

The prevalent LLMs typically consist of billions or even hundreds of billions of parameters. The immense scale of LLMs demands substantial computing resources, rendering full-parameter fine-tuning (FT) [34] (i.e., initializing the model with pre-trained weights and updating all parameters to adapt to various downstream tasks) of LLMs infeasible, especially on resource-constrained client devices. While the SL can offload primary computing burdens to central servers via
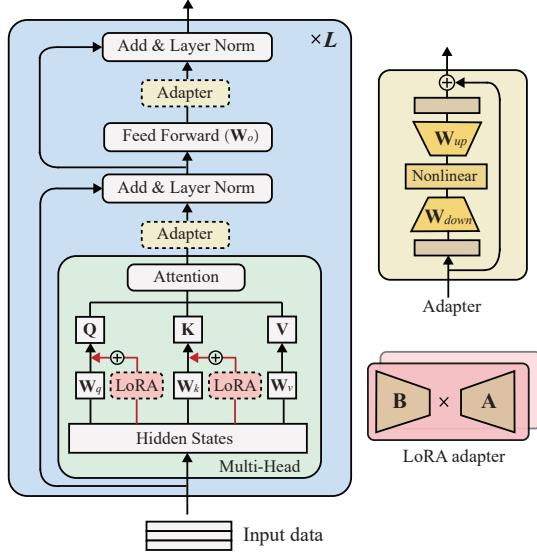
Fig. 2: An illustration of Adapter [23] and LoRA [11] fine-tuning methods, where $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, and $\mathbf{W}_o$ are trainable weights [33].

model partitioning [33], fine-tuning client-side LLMs still leads to intolerable latency. For instance, the client-side LLM composed of the first 10 Transformer blocks from the LLama-2-7B model [3] contains 8.75 billion parameters—60 times more than the parameter count of traditional models like VGG-16 [35], which has only 139 million parameters.

To address these challenges, PEFT offers a promising solution by fine-tuning only a small portion of LLM parameters, significantly reducing the computing cost. The current two widely adopted PEFT methods are Adapter [23] and LoRA [11]. As shown in Fig. 2, Adapter works by inserting a few additional trainable layers into the model backbone [23] and only fine-tuning those layers in a pre-trained model. In contrast, LoRA concatenates two trainable low-rank decomposition matrices in parallel to weights [11] but freezes all weights of a pre-trained model. For LoRA fine-tuning, the model update can be expressed as $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}$. Here, $\Delta\mathbf{W} = \mathbf{BA}$ denote incremental model update, $\mathbf{B} \in \mathbb{R}^{d_i \times r}$ and $\mathbf{A} \in \mathbb{R}^{r \times d_o}$ are low-rank decomposition matrices. The rank is $r \ll \min(d_i, d_o)$ (e.g., $r = 4$ when $d_i = d_o = 1024$), and dimensions of input and output of $\mathbf{B}$ and $\mathbf{A}$ are d_i and d_o, respectively. The LoRA adapter is the cascade structure composed of $\mathbf{B}$ and $\mathbf{A}$. Compared to Adapter, LoRA offers two distinct advantages: First, LoRA requires lower computing costs by updating only two low-rank matrices with few parameters. Second, LoRA can support LLM fine-tuning without changing their backbone architectures, showcasing better scalability and flexibility. Several studies [11], [36], [37] have demonstrated the superior performance of LoRA over Adapter, and hence we focus exclusively on LoRA in the following discussions and designs.

To better understand the computing and communication bottlenecks in LLM SL, we integrate the LoRA fine-tuning technique into one of the most representative SL frameworks, SFL [26], and take the well-known LLaMA-2-7B model as the
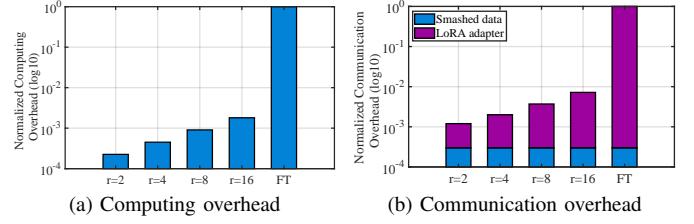

Fig. 3: The normalized computing and communication overheads of LoRA and FT on LLaMA-2-7B, where r is the decomposition rank of LoRA adapter.

global model. We select the E2E dataset [38] as the training dataset. Unless otherwise specified, this experimental setup remains the same throughout the whole Section II. Fig. 3a and 3b present the normalized computing and communication overhead[3] for LoRA and FT. The results reveal that the computing and communication overhead of FT is more than 500 and 100 times that of LoRA. Apparently, instead of the FT, PEFT can significantly mitigate the computing and communication overhead of LLM SL. However, we still meet three key challenges in integrating PEFT into LLM SL, which will be discussed in the following sections.

B. Limited Computing Resources

While LoRA significantly reduces the computing overhead for fine-tuning LLMs (see Section II-A), configuring LoRA adapters to all trainable weights on resource-constrained client devices is still impractical as LLM scale up [11], [39], [40]. In SL, this limitation also holds for the central server, which needs to handle heavy workloads offloaded from multiple client devices. Despite its more powerful computing capability, the central server still struggles to accommodate LoRA configurations for all trainable weights as the number of served client devices increases [20], [41]. Therefore, it is imperative to selectively fine-tune trainable weights under resource constraints.

To understand how the selection of trainable weights impacts the training performance of SL, we conduct two motivating experiments. Perplexity (PPL) serves as the performance metric to evaluate the training performance of the model, with a smaller PPL indicating better performance. First, we fine-tune the LLama-2-7B model for individual trainable weights (i.e., $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, and $\mathbf{W}_o$ in Fig. 2) and combinations with varying numbers of trainable weights. Fig. 4a illustrates that different trainable weights contribute unequally to the model's training performance, with some weights yielding more significant improvements than others. Fig. 4b demonstrates that more trainable weights achieve better training performance, but this comes at the cost of higher computing overhead. Second, we configure LoRA adapters for four trainable weights across two distinct training rounds to investigate the dynamic impact of trainable weights on model performance. Fig. 5a shows the PPL of each trainable weight, and Fig. 5b reports the

---

[3]The communication overhead consists of two components: the smashed data (i.e., activations and gradients) exchange overhead between client devices and the central server for model fine-tuning, and the transmission overhead of LoRA adapters for model aggregation.
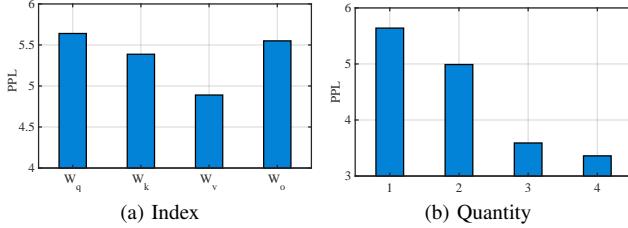
(a) Index   (b) Quantity

Fig. 4: The training performance with different trainable weights on LLaMA-2-7B, where the trainable weights combinations corresponding to quantities 1, 2, 3, and 4 are $\{\mathbf{W}_q\}$, $\{\mathbf{W}_q, \mathbf{W}_k\}$, $\{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v\}$, and $\{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o\}$, respectively.

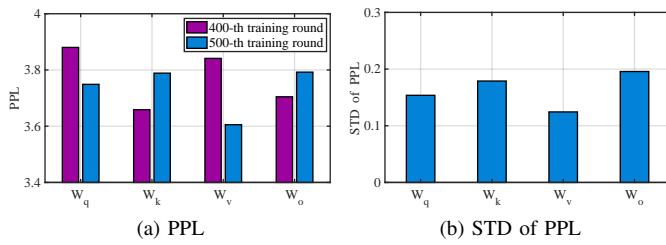

(a) PPL   (b) STD of PPL

Fig. 5: The PPL at the 400-th and 500-th training rounds and the STDs of the PPL across 100 rounds on LLaMA-2-7B.

standard deviation (STD) of PPL. The results indicate that the contribution of trainable weights to training performance varies throughout training, rendering the fixed fine-tuning scheme infeasible. Therefore, it is paramount to dynamically select the combination of trainable weights for LoRA adapter configuration.

C. Severe Device Unavailability Problem

Current state-of-the-art SL frameworks [24], [26] typically assume sufficient and homogeneous computing resources across client devices for synchronous local model training, thereby leading to good training performance. However, in practice, the computing capabilities of client devices often vary due to differences in hardware architecture [42] and the priority of on-demand running programs [43]. Client devices with weak computing power may fail to complete local model training, resulting in a severe device unavailability problem [44]–[46] and thus severely degrading training efficiency and performance. To gain a deeper understanding of the impact of computing heterogeneity on LLM SL, we conduct two motivating experiments. Consistent with Section II-B, we integrate the LoRA fine-tuning technique into SFL [26] and evaluate its performance on LLaMA-2-7B and GPT-2-L.

Fig. 7a and Fig. 7b show the converged accuracy and time on LLaMA-2-7B and GPT-2-L under varying device unavailability rates[4]. The results demonstrate that the converged accuracy degrades by 0.23 and 0.21 PPL, and convergence speed slows down by 42% and 40% on LLaMA-2-7B and GPT-2-L, respectively, as the device unavailability rate increases

---

[4]The device unavailability rate is defined as the proportion of devices incapable of local training due to computing limitations, which is calculated as the number of devices unable to complete local training divided by the total number of participating devices.
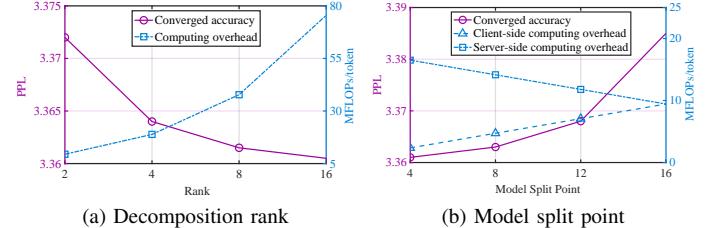

(a) Decomposition rank   (b) Model split point

Fig. 6: The converged accuracy and computing overhead versus decomposition rank and model split point on LLaMA-2-7B.

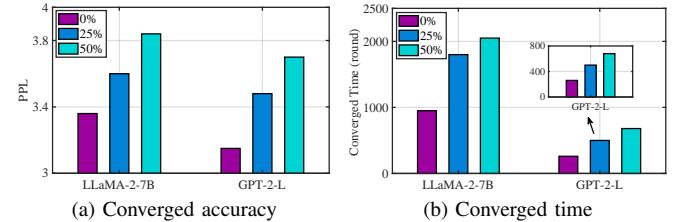(a) Converged accuracy   (b) Converged time

Fig. 7: The converged accuracy and time on LLaMA-2-7B and GPT-2-L under varying device unavailability rates.

from 0% to 50%. Therefore, it is essential to develop an efficient SL fine-tuning framework capable of accommodating heterogeneous computing resources of client devices.

To combat the computing heterogeneity issue, customizing key parameters in LLM SL is critical. For LLM SL, the decomposition ranks of LoRA adapters and the model split point are pivotal for training performance and computing costs. The rank dictates the number of trainable parameters in each weight for fune-tuning [37], [47], while the model split point determines the distribution of computing load between client devices and a central server [24], [26], [33]. To investigate the impact of these two parameters on LLM SL, we fine-tune LLaMA-2-7B with varying decomposition rank and model split point configurations, as shown in Fig. 6. The model split point j is defined as the model connection point between the j-th and $(j + 1)$-th transformer block. It is seen from Fig. 6a that increasing the decomposition rank improves training but incurs higher computing costs. Fig. 6b illustrates that model split point trade-offs the computing load between the client devices and central server while affecting training performance. Therefore, it is vital to customize the optimal decomposition rank and model split point for each client device under heterogeneous computing constraints.

D. Heterogenous Adapter Aggregation

Recall Fig 1, the fed server is responsible for periodically aggregating LoRA adapters from all client devices to consolidate client-side LLM updates in LLM SL. As illustrated in Fig. 8, the current state-of-the-art aggregation scheme [48] assumes homogeneity in LoRA adapters across client devices. It first averages the low-rank decomposition matrices across client devices and then multiplies them to represent the aggregated client-side LLM update. However, this scheme is mathematically inconsistent with the aggregation of client-
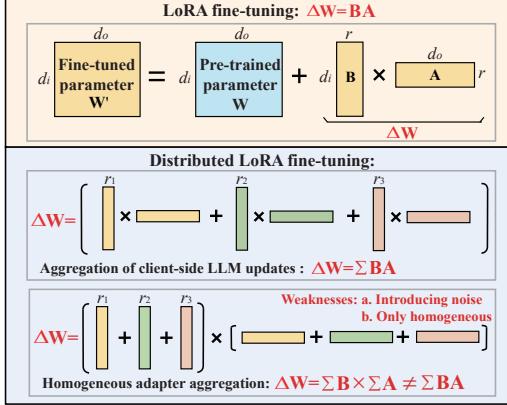
Fig. 8: The comparison of the state-of-the-art homogeneous aggregation scheme [48] with the aggregation of client-side LLM updates, where **B** and **A** are low-rank decomposition matrices.

Fig. 9: An overview of HSplitLoRA architecture.

side LLM updates, which averages the product of the decomposition matrices across all client devices. This inconsistency inevitably causes mathematical errors and introduces additional noise into the adapter aggregation process. As discussed in Section II-C, the inherent computing heterogeneity across client devices results in varying decomposition ranks for LoRA adapters across client devices. This dimensional mismatch between LoRA adapters renders the homogeneous adapter aggregation unsuitable, as it fails to accommodate the structural discrepancies across LoRA adapters. A more sophisticated adapter aggregation scheme is needed—one that can accommodate the heterogeneous nature of LoRA adapters while avoiding additional noise in the adapter aggregation process.

III. SYSTEM DESIGN

A. Overview

To combat address the above-mentioned challenges, we propose HSplitLoRA, a heterogeneous PEFT framework built on SFL [26] and LoRA fine-tuning method [11], as illustrated in Fig. 9. HSplitLoRA consists of the following three meticulously designed components:

- To facilitate efficient fine-tuning under computing constraints, we design an *important weight identification* (Section III-C) to identify the contribution of each trainable weight to training performance, so as to prioritize important weights for fine-tuning.
- To overcome the device unavailability issue, we propose an *adaptive rank and model splitting configuration* (Section III-D), which adjusts the decomposition ranks of LoRA adapters and model split point based on heterogeneous computing budgets of client devices to improve training efficiency.
- To efficiently aggregate heterogeneous adapters, we develop a *noise-free adapter aggregation* (Section III-E) that meticulously concatenates low-rank decomposition matrices to eliminate structural discrepancies across adapters, achieving noise-free heterogeneous adapter ag-
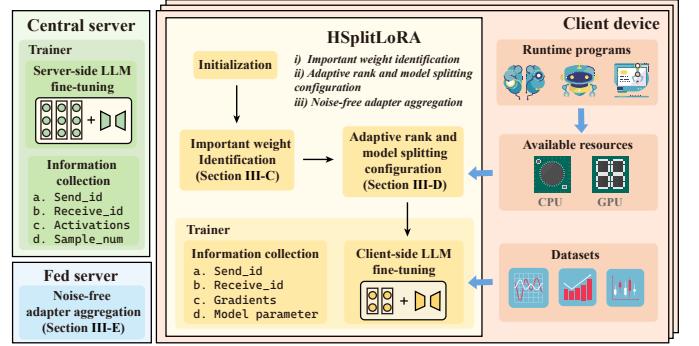
gregation.

The training workflow of HSplitLoRA for each training round follows three steps: i) Each client device utilizes the important weight identification to determine the fine-tuning priority of trainable weights; ii) client devices conduct adaptive rank and model splitting configuration based on computing budgets, and then collaborate with the central server for LLM fine-tuning via activations/gradients exchange; and iii) fed server employs noise-free adapter aggregation to aggregate heterogeneous client-side adapters and distribute aggregated LoRA adapters to all client devices for the next training round. It is noted that the last step can be conducted less often. HSplitLoRA is built upon a common SL framework [26], where aggregation refers to a weighted average of several adapters (similar to FedAvg [13]) but only for adapters trained on the client side.

B. System Model

In this section, we model the split fine-tuning of LLM on client devices with heterogeneous computing budgets, providing a theoretical foundation for the design of HSplitLoRA. As shown in Fig. 10, we consider a typical scenario of HSplitLoRA over an edge computing system, consisting of three components:

- **Client device:** The set of participating client devices is denoted by $\mathcal{N} = \{1, 2, ..., N\}$, where N is the total number of client devices. The computing budget of the n-th client device is $C_{c,n}$, while the client-side LLM on the n-th client device is represented as $\mathbf{W}_{c,n}$. The local dataset residing on the n-th client device with $|\mathcal{D}_n|$ data samples is denoted by $\mathcal{D}_n = \{\mathbf{x}_{n,k}, y_{n,k}\}_{k=1}^{|\mathcal{D}_n|}$, where $\mathbf{x}_{n,k}$ and $y_{n,k}$ are the k-th input data and its corresponding label in $\mathcal{D}_n$, respectively. Consequently, the total dataset across all client devices is $\mathcal{D} = \bigcup_{n=1}^{N} \mathcal{D}_n$.
- **Central server:** The central server is a powerful computing entity tasked with fine-tuning server-side LLM. The computing budget of the central server is denoted by C_s and the server-side LLM is $\mathbf{W}_s$.
- **Fed server:** The fed server is an entity responsible for adapter aggregation, periodically aggregating client-side LoRA adapters from all participating client devices.

The global model of the n-th client device is represented as $\mathbf{W}_n = [\mathbf{W}_s; \mathbf{W}_{c,n}]$. The local loss function of the n-th
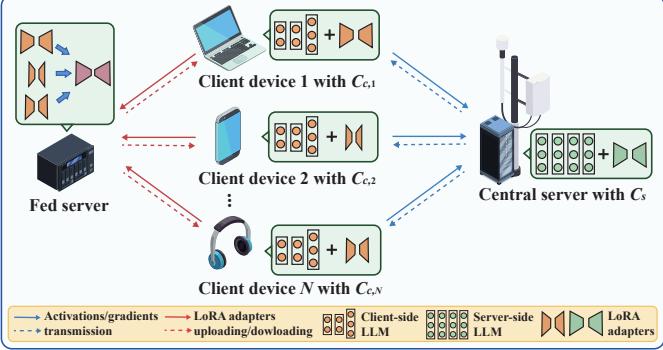
Fig. 10: Integrating LoRA adapters into LLM SL.

client device is $L_n(\mathbf{W}_n) = \frac{1}{|\mathcal{D}_n|} \sum_{k=1}^{|\mathcal{D}_n|} L_{n,k}(\mathbf{x}_{n,k}, y_{n,k}; \mathbf{W}_n)$, where $L_{n,k}(\mathbf{x}_{n,k}, y_{n,k}; \mathbf{W}_n)$ denotes the sample-wise loss function of the k-th data sample in $\mathcal{D}_n$. The objective of SL is to find the optimal global model that achieves good performance across all participating client devices, which can be formulated to minimize the finite-sum nonconvex global loss function:

$$\min_{\mathbf{W}_g} L(\mathbf{W}_g) = \min_{\mathbf{W}_g} \sum_{n=1}^{N} \frac{|\mathcal{D}_n|}{|\mathcal{D}|} L_n(\mathbf{W}_g), \quad (1)$$

where $\mathbf{W}_g = \sum_{n=1}^{N} \frac{|\mathcal{D}_n|}{|\mathcal{D}|} \mathbf{W}_n$.

As discussed in Section II-C, decomposition rank and model split point can significantly impact the LoRA fine-tuning performance. Therefore, we reformulate Eqn. (1) as

$$\min_{S, \mathcal{R}_n, \mathcal{W}_n} \quad \sum_{n=1}^{N} \frac{|\mathcal{D}_n|}{|\mathcal{D}|} L_n(\mathbf{W}_g | S, \mathcal{R}_{c,n}, \mathcal{R}_s, \mathcal{W}_{c,n}, \mathcal{W}_s), \quad (2)$$

$$\text{s.t.} \quad S \in \mathcal{S}, \quad (2a)$$

$$\mathcal{W}_{c,n}, \mathcal{W}_s \subseteq \{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o\}, \quad (2b)$$

$$r_{c,n}^m, r_s^m \in \mathcal{Q}, \quad (2c)$$

$$f_c(\mathcal{W}_{c,n}, \mathcal{R}_{c,n}) \le C_{c,n}, \quad (2d)$$

$$f_s(\mathcal{W}_s, \mathcal{R}_s) \le C_s, \quad (2e)$$

where S is the model split point selection decision, $\mathcal{W}_{c,n}$ and $\mathcal{W}_s$ represent selected combinations of trainable weights for the n-th client device and central server, $\mathcal{R}_{c,n} = \{r_{c,n}^1, r_{c,n}^2, \ldots, r_{c,n}^M\}$ and $\mathcal{R}_s = \{r_s^1, r_s^2, \ldots, r_s^M\}$ denote sets of decomposition ranks of $\mathcal{W}_{c,n}$ and $\mathcal{W}_s$. Here, $r_{c,n}^m$ and r_s^m represent the decomposition rank of the m-th trainable weight for the n-th client device and central server[5], and M is the total number of trainable weights[6].

Eqn. (2a) ensures that the model split point is selected from a predefined set $\mathcal{S}$. Eqn. (2b) specifies that the trainable weights are chosen from the query, key, value, and feed-forward layers of Transformer architecture (see Fig. 2). Eqn. (2c) guarantees that the decomposition rank of the LoRA adapter is selected from a predefined rank set $\mathcal{Q}$. Eqn. (2d)

and Eqn. (2e) represent the computing budgets of the n-th client device and central server, where $f_c(\cdot)$ and $f_s(\cdot)$ map the structure of trainable weights with LoRA adapters into the corresponding computing budget. The optimization problem in Eqn. (2) is a mixed-integer linear programming (MILP) problem, which is typically NP-hard. To solve this problem, we propose HSplitLoRA, a heterogeneous PEFT framework elaborated in Sections III-C to III-E.

C. Important Weight Identification

As explained in Section II-B, limited computing budgets of client devices and central server necessitate selective fine-tuning of trainable weights. To this end, we design an important weight identification scheme to identify and predict important weights (i.e., trainable weights with significant contributions to training performance) for the next training round, enabling efficient fine-tuning under constrained computing budgets. The scheme comprises two key components: a weight importance metric and dynamic weight identification.

a) Weight Importance Metric. Existing method for determining weight importance typically rely on single-dimensional metrics [49]–[52], such as gradients [49], [50], weight magnitudes [51], or computational complexity [52]. These methods fail to capture the multifaceted nature of weight importance. For instance, gradients may inadequately reflect the relative significance of weights in the model [49], [50], while weight magnitudes may ignore the sensitivity to training dynamics [51]. Furthermore, computing costs of different weights vary significantly, and overlooking this may result in overestimating the importance of computation-intensive weights, hence undermining the effectiveness of fine-tuning. This limitation underscores the need for a holistic metric that accounts for both optimization impact and computing demands.

We propose a new metric, named resource-normalized gradient-weight product (RNGWP), to evaluate weight importance by capturing the interplay between weights and gradients, as well as associated computing costs. For an arbitrary trainable weight $\mathbf{W}$, RNGWP is given by

$$\Theta(\mathbf{W}) = \frac{\sum_{j=1}^{J} \left| w_j \nabla_{w_j} L(\mathbf{W}) \right|}{C(\mathbf{W})}, \quad (3)$$

where w_j and $\nabla_{w_j} L(\mathbf{W})$ denote the j-th parameter of the model weight $\mathbf{W}$ and its corresponding gradient, respectively; J is the total number of model parameters in $\mathbf{W}$; and $C(\mathbf{W})$ denotes the fine-tuning cost[7] of trainable weight $\mathbf{W}$.

Remark: The numerator of RNGWP, $\left| w_j \nabla_{w_j} L(\mathbf{W}) \right|$, represents the magnitude of the gradient-weight product, capturing both static and dynamic contributions of weights to training performance: The weight magnitude reflects its inherent significance in the model (i.e., static contribution), while the gradient quantifies the weight's sensitivity to the training

---

[5]If the m-th trainable weight of the n-th client device/central server is not selected for LoRA fine-tuning, its decomposition rank $r_{c,n}^m / r_s^m$ is set to zero.

[6]Recalling Fig. 2, for LoRA fine-tuning, trainable weights are query, key, value, and feed-forward layers, resulting in $M = 4$.

[7]We utilize GPU memory footprint to quantify the fine-tuning cost of the trainable weight [27]–[29], as it represents a critical resource constraint in fine-tuning LLM on client devices. Specifically, the GPU memory footprint comprises model parameters, activations and activations' gradients, optimizer state (depending on the choice of the optimizer, e.g. SGD, Momentum, and Adam), which can be systematically estimated following the methodology in [53], [54].
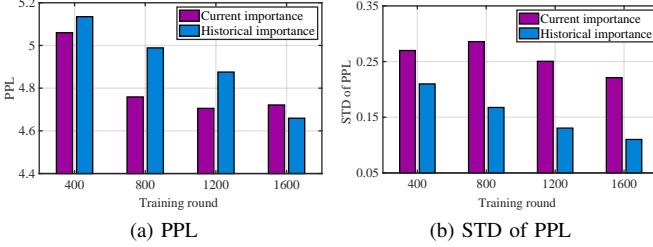
(a) PPL (b) STD of PPL

Fig. 11: The PPL and its STDs of fine-tuning a single weight with the highest current and historical importance on LLaMA-2-7B.

(a) Converged accuracy and time vs. bal-(b) Converged accuracy vs. training ance parameter round

Fig. 12: The converged accuracy and time versus balance parameter (a) and converged accuracy across 200 training rounds with varying balance parameters (b) on LLaMA-2-7B.

loss (i.e., dynamic contribution). This design prevents the overestimation of large weights with negligible gradients and the over-prioritization of small weights with steep gradients. To accommodate varying computing budgets across weights, the gradient-weight product is normalized by the denominator $C(\mathbf{W})$. This normalization ensures that fine-tuning prioritizes weights with high importance relative to their computing costs.
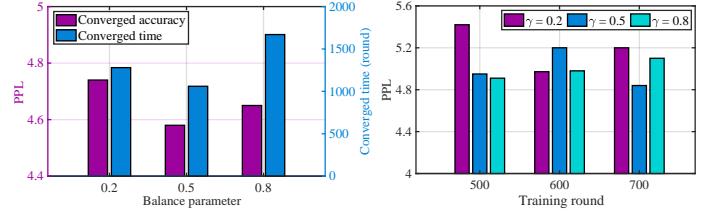
b) Dynamic Weight Identification. As LLM fine-tuning progresses, the weight importance fluctuates due to changes in model parameters and computing budgets across client devices. While the current importance (i.e., the weight importance at the current training round) captures immediate importance fluctuations, relying solely on it is insufficient. LLM fine-tuning is prone to short-term noise or local optima [17], [55], and focusing exclusively on current importance risks overfitting, compromising model convergence and fine-tuning performance [56], [57]. In contrast, historical importance (i.e., the cumulative weight importance over several previous training rounds) reflects long-term trends of weight importance across training rounds [58], [59], enabling the model to resist noise and remain stable model convergence.

To better understand the impact of current and historical importance on training performance, we fine-tune a single weight with the highest current and historical importance on LLaMA-2-7B. Fig. 11 shows that fine-tuning with the current importance achieves rapid PPL reduction in the early stages of training, but exhibits significantly higher standard deviation (STD) than the counterpart with historical importance. In contrast, fine-tuning with the historical importance demonstrates more stable convergence, ultimately achieving superior training performance with lower STD. These results reveal a trade-off: current importance enables rapid adaptation but lacks convergence stability, whereas historical importance improves convergence but struggles with dynamic adaptation. Therefore, designing a dynamic important weight identification that integrates current and historical importance is paramount for improving training accuracy while retaining robust model convergence.

In light of this, we define the weight importance of trainable weight $\mathbf{W}$ at the t-th training round, denoted as $\bar{I}_t(\mathbf{W})$, as a weighted average of historical and current importance:

$$\bar{I}_t(\mathbf{W}) = \gamma_t \bar{I}_{t-1}(\mathbf{W}) + (1 - \gamma_t)\tilde{I}_t(\mathbf{W}), \qquad (4)$$

where γ_t is a balance parameter controlling the trade-off between historical and current importance, $\bar{I}_{t-1}(\mathbf{W})$ is the cumulative historical importance up to the $(t-1)$-th round, and $\tilde{I}_t(\mathbf{W}) = \Theta(\mathbf{W}_{t-1})$ is the current importance in the t-th training round.

The key to identifying important weights lies in the design of the balance parameter γ_t. To gain deeper insights into its impact on training performance, we fine-tune a single weight with the highest weight importance under varying balance parameters on LLaMA-2-7B. Fig. 12a shows that different balance parameters exhibit distinct convergence speed and training accuracy, while Fig. 12b reveals that the optimal balance parameter varies across training rounds. These observations underscore the necessity of dynamically adjusting balance parameters for each training round.

Motivated by these insights, we formulate the balance parameter γ_t by considering two pivotal factors:

- **Training Stage:** In early training stages, the model relies heavily on current importance to adapt to the data due to its limited task understanding. As the model training converges, excessive dependence on current importance may lead to instability or overfitting. The model must gradually shift its focus from current to historical importance. This transition is governed by the ratio of the current round index t to the total training rounds T, i.e., $\frac{t}{T}$.

- **Weight Importance Fluctuations:** The extreme fluctuations in weight importance can negatively impact training efficiency. When a weight becomes excessively important, it may dominate learning and overfit to data-specific patterns, thereby degrading the model's generalization ability. Conversely, if a weight's importance becomes too low, the model may suppress potential information, leading to underfitting or suboptimal convergence. Therefore, it is essential to regularize the weight importance by jointly considering current importance and historical importance. We utilize the importance ratio $\frac{\tilde{I}_t(\mathbf{W})}{\bar{I}_{t-1}(\mathbf{W})}$ to enable bidirectional adjustment: when the current weight importance exceeds the historical importance, we attenuate its influence to avoid overreacting to transient fluctuations; when it falls below, we amplify its effect to prevent the suppression of useful information, ensuring that weights with low current importance are still given a chance to contribute, especially in early training stages
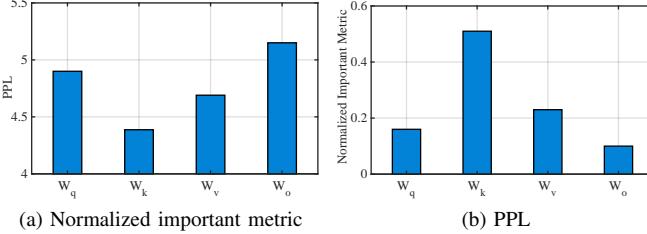
Fig. 13: The normalized weight importance and PPL of different trainable weights on LLaMA-2-7B across 200 training rounds.

where weight importance may be unreliable.

Based on these two factors, we define the balance parameter γ_t for the t-th training round as:

$$\gamma_t = 1 - \exp(-\frac{\tilde{I}_t(\mathbf{W})t}{\bar{I}_{t-1}(\mathbf{W})T}). \quad (5)$$

As shown in Fig. 13, we observe that trainable weights with higher weight importance, as defined in (4), exhibit better fine-tuning performance, i.e., lower PPL values, validating the effectiveness of the proposed weight importance metric. With the weight importance, client devices and the central server can configure the combinations of LoRA adapters and model splitting, as discussed in the following section.

D. Adaptive Rank and Model Splitting Configuration

Recalling Section II-C, the computing heterogeneity across client devices causes the severe device unavailability problem, significantly degrading the training performance of LLM SL. In LLM SL, the fine-tuning computing cost on client devices and the central server is jointly determined by the decomposition rank of the LoRA adapters and the model split point. On the one hand, a shallower model split point reduces the computing burden on client devices but shifts substantial load to the central server [4], [25]. On the other hand, the decomposition rank of the LoRA adapter presents a trade-off between computational cost and fine-tuning performance: A higher rank improves data fitting but incurs more computing resources [4], [11]. The tight coupling of these parameters necessitates a unified optimization solution to enhance fine-tuning performance.

We propose an adaptive rank and model splitting configuration based on the weight importance indicator in Section III-C, as shown in Fig. 14. The strategy consists of two key components: i) rank and model splitting configuration, and ii) adaptive adjustments of rank and model splitting.

a) Rank and Model Splitting Configuration. Given the vast search space, an exhaustive search over all possible combinations of LoRA adapter ranks and model split points is computationally prohibitive. To circumvent this impasse, we leverage the weight importance indicator developed in Section III-C to configure the LoRA adapters and model splitting. For a given model split point S, the client devices and the central server first sort trainable weights (shown in Fig. 2) by importance and then configure the LoRA adapters
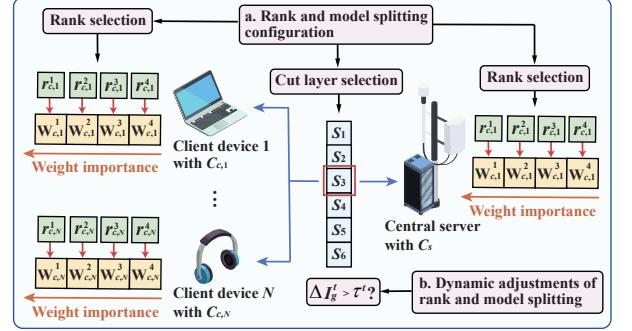

Fig. 14: The illustration of adaptive rank and model splitting configuration solution.
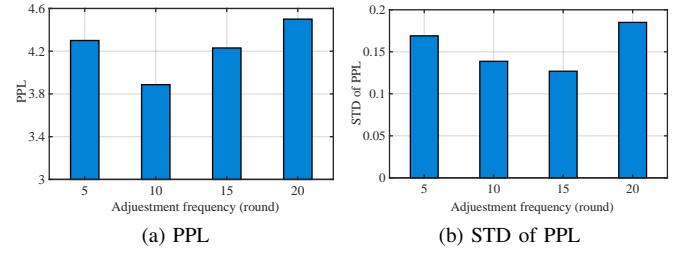

Fig. 15: The PPL and its STDs of fine-tuning LLaMA-2-7B with varying adjustment frequencies.

accordingly. Specifically, the LoRA adapters are assigned to trainable weights in descending order of importance, selecting ranks from a predefined set $\mathcal{Q}$. If the computing cost of the selected rank is within the available budget, the rank is selected; otherwise, the next most important weight is evaluated. This process continues until the remaining computing budget is exhausted, yielding an optimal LoRA configuration for the given split point S.

To identify the most effective combination of model split point and LoRA adapter configuration, we compute the global weight importance for each candidate combination, which captures the overall importance of weights across both the client-side and server-side LLMs, computed as the sum of the average of the weight importance for selected client-side and server-side weights. The global weight importance for the given model split point S is given by

$$I_g(S) = \frac{1}{\sum\limits_{n=1}^{N} |\mathcal{W}_{c,n}|} \sum_{n=1}^{N} \sum_{\substack{r \in \mathcal{R}_{c,n}, \\ \mathbf{W} \in \mathcal{W}_{c,n}}} \Theta(\mathbf{W}_{c,n}|S, r, \mathbf{W})$$
$$+ \frac{1}{|\mathcal{W}_s|} \sum_{\substack{r \in \mathcal{R}_s, \\ \mathbf{W} \in \mathcal{W}_s}} \Theta(\mathbf{W}_s|S, r, \mathbf{W}). \quad (6)$$

By comparing the global weight importance across the predefined set of model split points $\mathcal{S}$, the model split point and LoRA adapter with the highest global weight importance are selected. This ensures that the most important weights are prioritized, enhancing training performance while accommodating the heterogeneous computing budgets of the client devices.

b) Adaptive Adjustments of Rank and Model Splitting.

As mentioned in Section II-C, the computing budgets of client devices and the central server vary during model training. To accommodate these fluctuations, dynamic rank and model split point adjustments are essential. Since LoRA fine-tuning updates only a small portion of the whole model parameters[8], ranks can be flexibly adjusted in each training round to accommodate the fluctuations in computing budgets without compromising model stability [11]. In contrast, the adjustment frequency of model split point must be carefully determined. Frequent adjustments may degrade fine-tuning performance and stability due to the rapid changes in the overall model aggregation frequency [25], whereas infrequent adjustments may hinder the model from adapting to the weight importance variations, leading to a performance decline.

To better understand the impact of adjustment frequency of the model split point on training performance, we fine-tune the LLaMA-2-7B with varying adjustment frequencies. We use the number of training rounds between two consecutive model split point adjustments to represent the adjustment frequency. Fig. 15a demonstrates that a moderate adjustment frequency maximizes accuracy, while Fig. 15b shows that both excessively frequent and infrequent adjustments can result in increased STD, thereby degrading training stability.

We propose an adaptive strategy for adjusting the model splitting point based on the maximum global weight importance difference $\Delta I_{g,t}(S)$, as given by:

$$\Delta I_{g,t}(S) = \max_{S' \in \mathcal{S} \setminus \{S\}} I_{g,t}(S') - I_{g,t}(S). \tag{7}$$

The model split point adjustment is triggered when $\Delta I_{g,t}(S)$ exceeds a predefined threshold τ_t, indicating that the current model splitting configuration exhibits significantly worse performance than alternative options. For substantial importance difference, τ_t should be set lower to trigger the more frequent model split point adjustment for responding promptly to the importance difference. For minimal importance difference, τ_t should be appropriately increased to avoid excessive adjustments, ensuring training stability and efficiency. Thus, the threshold is designed as follows:

$$\tau_t = \tau_{t-1} \max\left(1 - \Delta I_{g,t}(S), \varepsilon\right), \tag{8}$$

where $\varepsilon > 0$ is a small positive constant.

Furthermore, when rank adjustments cannot accommodate changes in the computing budgets of the client devices or central server, a model split point adjustment is triggered. This adaptive strategy dynamically accommodates variations in computing resources and weight importance, ensuring the efficiency and stability of the training process. We summarize the procedure of adaptive rank and model splitting configuration in **Algorithm 1**.

E. Noise-free Adapter Aggregation and Deployment

Current SL frameworks update the client-side model via a weighted average of client-side model parameters [20], [26]. However, for LLMs with billions of parameters, transmitting client-side LLM for model aggregation is bandwidth-intensive

---

[8]The fine-tuning parameters are substantially fewer than those of the global model [11], [60].

Algorithm 1: Adaptive Rank and Model Splitting Configuration

Require: $\mathcal{N}$, $C_{c,n}$, C_s, $\mathcal{S}$, $\mathcal{R}_{c,n}$, $\mathcal{R}_s$, $\mathcal{W}_{c,n}$, $\mathcal{W}_s$.
Data: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_N\}$ where $\mathcal{D}_n$ is the local dataset of the n-th client devices.

1   Adaptive_config($\mathcal{N}$, $\mathcal{S}$):
2   **for** $t = 1, 2, ..., T$ **do**
3   Compute global weight importance change $\Delta I_g(S_t)$
4   **if** $\Delta I_g(S_t) > \tau_t$ **then**
5   **for** $n \in \mathcal{N}$ *in parallel* **do**
6   Rank_model_split_config($C_{c,n,t}$, $\mathcal{R}_{c,n}$, $\mathcal{W}_{c,n}$)
7   Rank_model_split_config($C_{s,t}$, $\mathcal{R}_s$, $\mathcal{W}_s$)
8   **else**
9   Configure adapter ranks to fit $C_{c,n,t}$ and $C_{s,t}$
10   Rank_model_split_config(C, $\mathcal{R}$, $\mathcal{W}$):
11   **for** $S \in \mathcal{S}$ **do**
12   **for** $\mathbf{W} \in \mathcal{W}$ *(sorted by importance)* **do**
13   **for** $r \in \mathcal{R}$ *(descending order)* **do**
14   **if** $f(\mathbf{W}, r) \leq C$ **then**
15   Configure rank r to weight $\mathbf{W}$
             $C \leftarrow C - f(\mathbf{W}, r)$
16   Compute $I_g(S)$ via Eqn. (6)
17   Select optimal split point $S^* = \arg\max_{S \in \mathcal{S}} I_g(S)$

---

and overwhelms the limited computing capabilities of client devices [4] (see Section II-A). LoRA fine-tuning mitigates this issue by encoding model updates as low-rank decomposition matrices, allowing only these matrices to be transmitted for model aggregation. However, aggregating low-rank matrices remains challenging. As discussed in Section II-D, the heterogeneous computing budgets across client devices result in low-rank matrices with varying ranks, rendering direct numerical averaging infeasible. Furthermore, the state-of-the-art aggregation scheme [48] is mathematically inconsistent with the aggregation of client-side LLM updates, introducing additional noise into the adapter aggregation process.

To combat these challenges, as illustrated in Fig. 16, we propose a noise-free adapter aggregation scheme that employs matrix concatenation to enable seamless aggregation of heterogeneous low-rank matrices without introducing additional noise. The proposed scheme consists of the following two stages.

a) LoRA Adapter Concatenation. Though the LoRA adapters received by the fed server exhibit structural heterogeneity, we observe that the total ranks of low-rank decomposition matrices $\mathbf{B}$ and $\mathbf{A}$ across client devices remain consistent. This insight suggests that structural consistency can be achieved by concatenating multiple LoRA adapters along the rank dimensions of $\mathbf{B}$ and $\mathbf{A}$.

As illustrated in Fig. 16, at the t-th training round, the concatenated decomposition matrices of the m-th client-side trainable weight (LoRA adapter) are expressed as

$$\mathbf{B}_c^{m,t} = [\mathbf{B}_{c,1}^{m,t}, \mathbf{B}_{c,2}^{m,t}, \cdots, \mathbf{B}_{c,N}^{m,t}] \in \mathbb{R}^{d_i \times (r_{c,1}^{m,t} + r_{c,2}^{m,t} + \ldots + r_{c,N}^{m,t})} \tag{9}$$
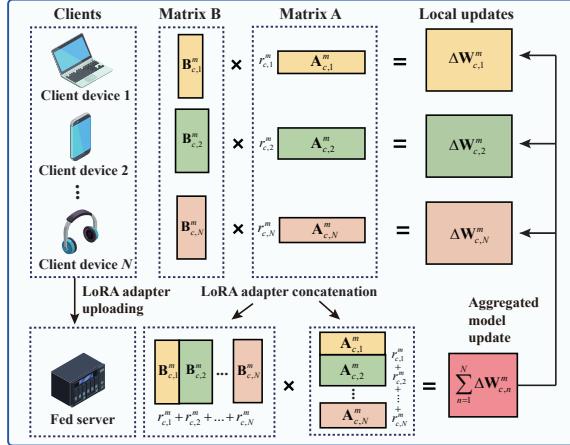
Fig. 16: The noise-free adapter aggregation scheme of HSplit-LoRA framework, where $\mathbf{B}_{c,n}^m$ and $\mathbf{A}_{c,n}^m$ represent low-rank decomposition matrices of the m-th trainable weight for client device n, and its corresponding decomposition rank is denoted by $r_{c,n}^m$.

and

$$\mathbf{A}_c^{m,t} = \begin{bmatrix} \mathbf{A}_{c,1}^{m,t}, \\ \mathbf{A}_{c,2}^{m,t}, \\ \vdots \\ \mathbf{A}_{c,N}^{m,t} \end{bmatrix} \in \mathbb{R}^{(r_{c,1}^{m,t}+r_{c,2}^{m,t}+...+r_{c,N}^{m,t}) \times d_o}, \quad (10)$$

where d_i and d_o denote the input and output dimensions of the low-rank decomposition matrices $\mathbf{B}$ and $\mathbf{A}$, respectively. Therefore, at the t-th training round, the product of the concatenated decomposition matrices, representing the aggregated incremental model update, can be calculated as [61]

$$\Delta\mathbf{W}_c^{m,t} = [\mathbf{B}_{c,1}^{m,t}, \mathbf{B}_{c,2}^{m,t}, \cdots, \mathbf{B}_{c,N}^{m,t}] \begin{bmatrix} \mathbf{A}_{c,1}^{m,t}, \\ \mathbf{A}_{c,2}^{m,t}, \\ \vdots \\ \mathbf{A}_{c,N}^{m,t} \end{bmatrix} = \sum_{n=1}^{N} \mathbf{B}_{c,n}^{m,t} \mathbf{A}_{c,n}^{m,t}. \quad (11)$$

Recall the incremental model update $\Delta\mathbf{W}_{c,n}^m = \mathbf{B}_{c,n}^m \mathbf{A}_{c,n}^m$ from Section II-D, the product of the concatenated decomposition matrices is equivalent to the sum of the client-side incremental model updates from the client devices. This indicates the noise-free property of the concatenate-then-multiply scheme, ensuring accurate LoRA aggregation without introducing additional noise.

b) Incremental Matrix Update. While the concatenate-then-multiply scheme enables the computation of the aggregated incremental model update via Eqn. (11), recovering the low-rank decomposition matrices for each client device is non-trivial. The matrix decomposition is usually non-unique and often yields multiple valid solutions. Moreover, the aggregated incremental model update contains mixed information across LoRA adapters, this mixing effect obscures the unique characteristics of individual adapters, rendering accurate reconstruction of updated adapters for each client device difficult [62]. Therefore, instead of directly updating the decomposition

Fig. 17: HSplitLoRA prototype and testbed.

matrices $\mathbf{B}_{c,n}^m$ and $\mathbf{A}_{c,n}^m$, we merge the aggregated incremental model update $\Delta\mathbf{W}_c^m$ into the pre-trained client-side model $\mathbf{W}_c^m$, as given by

$$\mathbf{W}_c^{m,t+1} = \mathbf{W}_c^{m,t} + \Delta\mathbf{W}_c^{m,t}. \quad (12)$$

After that, $\mathbf{B}_{c,n}^{m,t}$ is initialized to zero and $\mathbf{A}_{c,n}^{m,t}$ is initialized [11], [63] using a random Gaussian distribution at the beginning of the next training round.

IV. IMPLEMENTATION

In this section, we first elaborate on the implementation of HSplitLoRA and then introduce the experiment setup.

A. Implementing HSplitLoRA

We prototype HSplitLoRA for LLM fine-tuning based on a micro services architecture, as illustrated in Fig. 17. The platform consists of Jetson development kits and a high-performance server, with network conditions controlled via *tc* [64]. The central server is emulated by an H3C UniServer R5300 G3 server equipped with eight NVIDIA GeForce RTX 3090 GPUs, dual Intel Xeon Silver 4210R processors (10 cores, 2.84 GHz each), and 8×32 GB DDR4 RAM, running Ubuntu 18.04.6 LTS. The Jetson series kits [65] have been widely recognized as a standard hardware platform for edge AI applications, offering on-device computing resources (e.g., GPU acceleration for deep learning inference and training) within a constrained power budget. Thus, to emulate the client devices, we utilize the Jetson AGX Xavier kits, equipped with a 512-core Volta GPU with Tensor Cores, an 8-core ARM V8.2 64-bit CPU, 32 GB EMMC 5.1 storage, and also running Ubuntu 18.04.6 LTS. The software stack comprises Python 3.7 and PyTorch 1.9.1, which are used for implementing LLM fine-tuning for natural language generation applications.

B. Experiment Setup

Datasets and Models. We adopt the E2E dataset [38] to evaluate the performance of HSplitLoRA. The E2E dataset focuses on the restaurant domain, consisting of 42,000 training, 4,600 validation, and 4,600 test samples. To implement HSplitLoRA, we employ two well-known LLMs, LLaMA-2-7B [3] and GPT-2-L [66]. The LLaMA-2-7B model comprises 32 layers and 3.52 billion parameters, while the GPT-2-L features 24 layers with 355 million parameters and 36 layers with 774 million parameters. In our experiments, we fine-tune the LLaMA-2-7B and GPT-2-L models on the E2E dataset for the natural language generation (NLG) task.
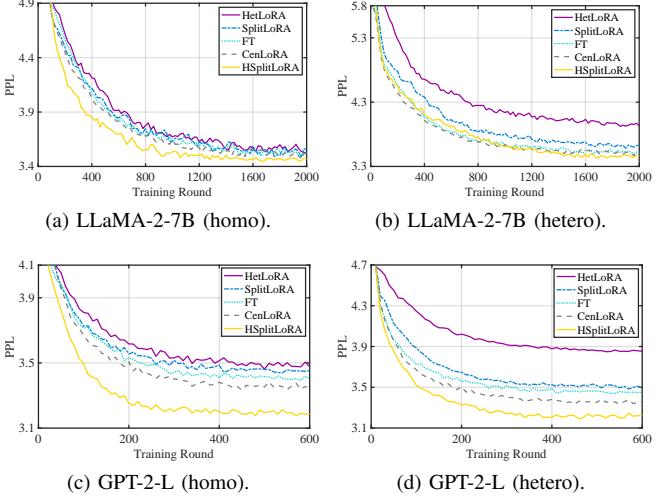
(a) LLaMA-2-7B (homo).

(b) LLaMA-2-7B (hetero).

(c) GPT-2-L (homo).

(d) GPT-2-L (hetero).

Fig. 18: The training performance on LLaMA-2-7B and GPT-2-L models under heterogeneous (hetero) and homogeneous (homo) settings.

(a) LLaMA-2-7B

(b) GPT-2-L

Fig. 19: The converged accuracy on LLaMA-2-7B and GPT-2-L models under heterogeneous and homogeneous settings.

(a) LLaMA-2-7B

(b) GPT-2-L

Fig. 20: The converged time on LLaMA-2-7B and GPT-2-L models under heterogeneous and homogeneous settings.

V. PERFORMANCE EVALUATION

This section provides numerical results to evaluate the training performance of HSplitLoRA framework and the effectiveness of each meticulously designed component.

A. Superiority of HSplitLoRA

Training Performance of HSplitLoRA. Fig. 18 presents the superior training performance of HSplitLoRA compared to four benchmarks under heterogeneous and homogeneous settings. In the homogeneous setting, HSplitLoRA achieves the fastest convergence and highest training accuracy, outperforming FT, CenLoRA, HetLoRA, and SplitLoRA. This advantage stems from HSplitLoRA's important weight identification, which identifies and selectively fine-tunes trainable parameters based on their contributions to LLM training, thereby enhancing training performance. In the heterogeneous setting, computing discrepancies across client devices exacerbate the device unavailability effect in HetLoRA and SplitLoRA, leading to degraded converged accuracy. While HetLoRA mitigates the device unavailability effect by assigning varying ranks of LoRA adapters, its fine-tuning remains inefficient due to the necessity of fine-tuning the whole LLM on the client device. SplitLoRA employs a static rank and model splitting configuration, causing a mismatch between the computing workload and the computing budget of the client device, significantly slowing down model convergence and leading to worse converged accuracy than FT and CenLoRA. In contrast, HSplitLoRA dynamically configures the optimal LoRA adapter ranks and model splitting as model training progresses and computing budgets change, ensuring superior training performance even under the heterogeneous setting. For GPT-2-L, HSplitLoRA still retains a consistent advantage

Benchmarks. To comprehensively evaluate the performance of HSplitLoRA, we compare HSplitLoRA against the following alternatives:

- **Full-parameter fine-tuning (FT):** Central server fine-tunes parameters of the whole LLM, allowing the model to fully adapt to new tasks or datasets while leveraging its pre-trained knowledge [34].
- **Centralized low-rank adaptation (CenLoRA):** Central server updates LoRA adapters of the whole LLM with the full dataset [11].
- **Heterogeneous federated low-rank adaptation (HetLoRA):** HetLoRA allows client devices to fine-tune LLM with varying LoRA rank via FL and employ a sparsity-weighted aggregation scheme to effectively aggregate LoRA adapters [61].
- **Split low-rank adaptation (SplitLoRA):** SplitLoRA partitions the LLM into client-side and server-side LLMs, deployed on the client devices and central server for fine-tuning, and periodically aggregates client-side LoRA adapters [4].

Hyper-parameters. For the NLG task on the E2E dataset for LLaMA-2-7B, we set the mini-batch size, learning rate, and maximum sequence length to 1, 0.0001, and 384, respectively, while these parameters are configured as 4, 0.0002, and 512 for GPT-2-L. The total number of participating client devices is 5, all running in a synchronous mode [67]. For the homogeneous setting, we set the maximum GPU memory of each client device to 20GB for LLaMA-2-7B and 4GB for GPT-2-L. For the heterogeneous setting, the maximum GPU memory of client devices follows a uniform distribution between 5GB and 20GB for LLaMA-2-7B, and between 1GB and 4GB for GPT-2-L. For both homogeneous and heterogeneous settings, the maximum GPU memory of the central server is set to 75GB for LLaMA-2-7B and 15GB for GPT-2-L. The predefined rank set is $\mathcal{Q} = \{1, 2, 4, 8, 16, 32\}$.
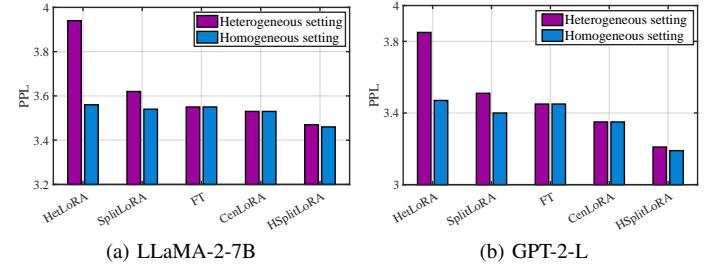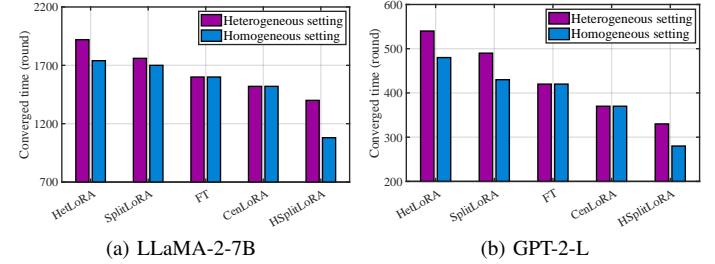
Model	Method	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
LLaMA-2-7B (homo)	FT	66.4	8.45	44.6	69.0	2.35
	CenLoRA	66.7	8.46	44.8	69.1	2.37
	HetLoRA	66.2	8.42	44.3	68.8	2.34
	SplitLoRA	66.5	8.44	44.7	69.0	2.36
	HSplitLoRA (Ours)	**68.1**	**8.66**	**45.7**	**69.8**	**2.45**
LLaMA-2-7B (hetero)	FT	66.4	8.45	44.6	69.0	2.35
	CenLoRA	66.7	8.46	44.8	69.1	2.37
	HetLoRA	61.3	8.07	41.1	64.7	2.09
	SplitLoRA	65.4	8.38	44.0	68.4	2.32
	HSplitLoRA (Ours)	**68.0**	**8.65**	**45.6**	**69.5**	**2.42**
GPT-2-L (homo)	FT	68.2	8.68	45.8	69.9	2.45
	CenLoRA	68.9	8.76	46.5	70.7	2.47
	HetLoRA	68.0	8.64	45.6	69.5	2.42
	SplitLoRA	68.6	8.79	46.3	70.2	2.46
	HSplitLoRA (Ours)	**69.7**	**8.82**	**46.8**	**71.2**	**2.51**
GPT-2-L (hetero)	FT	68.2	8.68	45.8	69.9	2.45
	CenLoRA	68.9	8.76	46.5	70.7	2.47
	HetLoRA	62.2	8.17	41.5	65.2	2.12
	SplitLoRA	67.2	8.57	45.4	69.4	2.42
	HSplitLoRA (Ours)	**69.5**	**8.79**	**46.6**	**70.9**	**2.49**

TABLE I: The performance comparison of various metrics on LLaMA-2-7B and GPT-2-L models under heterogeneous and homogeneous settings.

over all benchmarks, demonstrating superior scalability across diverse LLM architectures. These results underscore HSplit-LoRA's superior adaptability to heterogeneous edge computing environments, improving convergence speed and accuracy.

Converged Accuracy of HSplitLoRA. Fig. 19 illustrates the converged accuracy on LLaMA-2-7B and GPT-2-L under homogeneous and heterogeneous settings. In the heterogeneous setting, HetLoRA and SplitLoRA exhibit a significant performance drop, with PPL increasing by approximately 0.38 and 0.1 for LLaMA-2-7B, and 0.38 and 0.11 for GPT-2-L, compared to the homogeneous setting. The performance degradation of HetLoRA stems from its static LoRA adapter configuration, which lacks dynamic rank adjustments and weight prioritization, leading to inefficiencies under varying computational constraints. SplitLoRA suffers from accuracy loss due to its reliance on fixed LoRA adapters and model splitting, making it susceptible to device unavailability effects and reducing adaptability in heterogeneous settings. In contrast, HSplitLoRA achieves the lowest PPL and outperforms the FT, CenLoRA, HetLoRA, and SplitLoRA for PPL by 0.09 (resp. 0.26), 0.07 (resp. 0.16), 0.1 (resp. 0.28), and 0.08 (resp. 0.21) in the homogeneous setting, and 0.08 (resp. 0.24), 0.06 (resp. 0.14), 0.47 (resp. 0.64), and 0.15 (resp. 0.33) in the heterogeneous setting on LLaMA-2-7B (resp. GPT-2-L). This superiority is attributed to the design of important weight identification to prioritize the fine-tuning of key weights, and the adaptive rank and model splitting configuration to accommodate heterogeneous computing constraints.

Furthermore, HSplitLoRA benefits from noise-free adapter aggregation, facilitating efficient aggregation of client-side updates from client devices without introducing additional noise. While HSplitLoRA, HetLoRA, and SplitLoRA showcase noticeable accuracy degradation in heterogeneous settings, FT and CenLoRA remain unaffected as their high computing demands confine execution to the central server. However, their
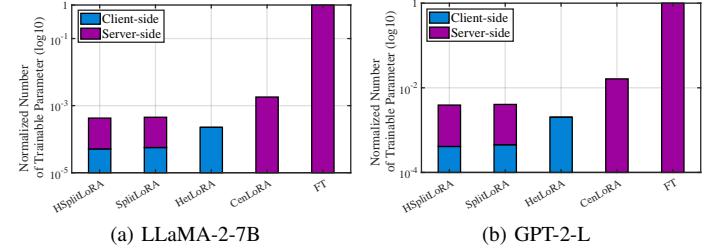

Fig. 21: The number of trainable parameters for LLaMA-2-7B and GPT-2-L models.

reliance on centralized training severely limits their scalability, rendering them impractical for large-scale distributed learning. We also conduct a comprehensive comparison of the accuracy across various metrics in homogeneous and heterogeneous settings, with the results summarized in Table I.

Converged Time of HSplitLoRA. Fig. 20 compares the convergence time of HSplitLoRA against four benchmarks on LLaMA-2-7B and GPT-2-L under the heterogeneous and homogeneous settings. In the homogeneous setting, HSplitLoRA consistently exhibits the fastest convergence speed, surpassing FT, CenLoRA, HetLoRA, and SplitLoRA by factors of approximately 1.4, 1.3, 1.6 and 1.5 for LLaMA-2-7B, and by factors of 1.5, 1.3, 1.7, and 1.5 for GPT-2-L, respectively. This performance gain stems from our design of the important weight identification strategy, which prioritizes the fine-tuning of key weights to expedite model training. FT exhibits the slowest convergence due to the full-parameter fine-tuning, whereas CenLoRA achieves slightly faster model convergence by shrinking the number of trainable parameters via LoRA fine-tuning. In the heterogeneous setting, HetLoRA and Split-LoRA suffer from severe device unavailability effects caused by computing discrepancies across client devices, resulting in slower convergence speed. HSplitLoRA still maintains the
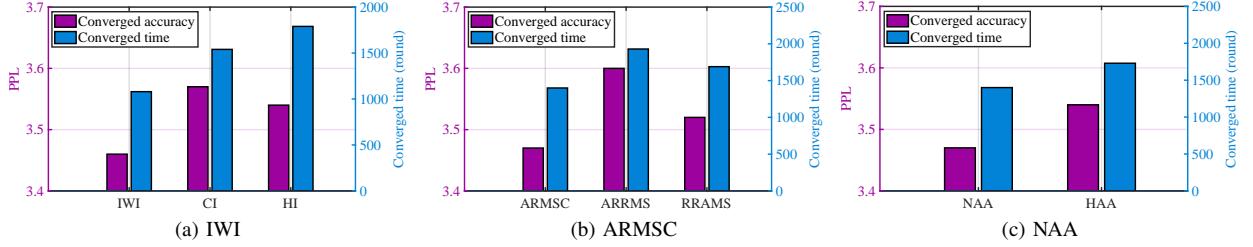
Fig. 22: The ablation evaluation for IWI (a), ARMSC (b), and NAA (c) on LLaMA-2-7B.

Method	E2E NLG Challenge				
	BLEU	**NIST**	**MET**	**ROUGE-L**	**CIDEr**
CI	66.1	8.40	44.2	68.7	2.32
HI	66.6	8.44	44.7	69.1	2.37
IWI (Ours)	**68.1**	**8.66**	**45.7**	**69.8**	**2.45**

TABLE II: The performance comparison of IWI and counterparts relying solely on current importance (CI) or historical importance (HI) on LLaMA-2-7B.

Method	E2E NLG Challenge				
	BLEU	**NIST**	**MET**	**ROUGE-L**	**CIDEr**
ARRMS	65.6	8.39	44.2	68.5	2.33
RRAMS	66.8	8.47	44.8	69.2	2.38
ARMSC (Ours)	**68.0**	**8.65**	**45.6**	**69.5**	**2.42**

TABLE III: The performance comparison of ARMSC, random rank and adaptive model splitting (RRAMS), and adaptive rank and random model splitting (ARRMS) on LLaMA-2-7B.

Method	E2E NLG Challenge				
	BLEU	**NIST**	**MET**	**ROUGE-L**	**CIDEr**
HAA	66.6	8.45	44.8	69.1	2.36
NAA (Ours)	**68.0**	**8.65**	**45.6**	**69.6**	**2.43**

TABLE IV: The performance comparison of NAA and homogeneous adapter aggregation (HAA) [48] on LLaMA-2-7B.

B. Ablation Study

Important Weights Identification (IWI). Fig. 22a compares the training performance of IWI with counterparts relying solely on current importance (CI) or historical importance (HI) on LLaMA-2-7B under the homogeneous setting. The results demonstrate that IWI achieves the lowest PPL while maintaining a convergence speed comparable to CI-based method. This stems from IWI's adaptive weight importance evaluation, which dynamically captures weight variations during training and balances current and historical importance. In contrast, CI-based methods adjust rapidly but are vulnerable to noise and local optima, while the HI-based method exhibits long-term stability but responds sluggishly to dynamic weight importance changes. The comparison of accuracy across various metrics is presented in Table II.

Adaptive Rank and Model Splitting Configuration (ARMSC). Fig. 22b illustrates the training performance of ARMSC compared to random rank and adaptive model splitting (RRAMS) and adaptive rank and random model splitting (ARRMS) on LLaMA-2-7B under the heterogeneous setting. The results indicate that ARMSC achieves the shortest convergence time and lowest PPL. This is attributed to its ability to adjust the LoRA adapter ranks and model split points to accommodate heterogeneous computing budgets across client devices. By adapting LoRA ranks to available computing resources of client devices, ARMSC prevents both overloading and underutilization, while adaptive model splitting balances the computing load between client devices and the central server to mitigate the device unavailability effect. Furthermore, ARMSC leverages global weight importance change as a triggering indicator, ensuring that model split points are updated only when significant changes in weight importance occur, avoiding frequent unnecessary adjustments that could destabilize training. Table III provides training performance on other metrics for further comparison.

Noise-free Adapter Aggregation (NAA). Fig. 22c presents the training performance of NAA and homogeneous adapter

shortest converged time in the heterogeneous setting. This is primarily attributed to adaptive rank and model splitting configuration, which dynamically accommodates heterogeneous computing budgets across client devices.

Number of Trainable Parameters. Fig. 21 presents the average number of trainable parameters of HSplitLoRA and four benchmarks methods for fine-tuning LLaMA-2-7B and GPT-2-L. It is shown that FT exhibits the highest number of trainable parameters, exceeding CenLoRA by at least 550 times on LLaMA-2-7B and by approximately 60 times on GPT-2-L. This is because FT involves updating all parameters of LLM, whereas CenLoRA only updates the parameters of LoRA adapters, which constitute only a small fraction of the global model. HetLoRA has fewer trainable parameters than CenLoRA, as client devices have limited computing resources and cannot support the fine-tuning of the same number of trainable parameters as the central server. HSplitLoRA and SplitLoRA exhibit substantially fewer client-side trainable parameters than HetLoRA. This reduction stems from model splitting, which offloads most trainable parameters to the central server and reduces the computing workload on client devices. Moreover, since HSplitLoRA and SplitLoRA only transmit the client-side LoRA adapters to the fed server for aggregation, a small number of trainable parameters indicates a lower volume of data transmissions for adapter aggregation, thus enhancing fine-tuning efficiency.

aggregation (HAA) [48] on LLaMA-2-7B under the homogeneous setting. NAA consistently outperforms HAA in both accuracy and convergence speed, achieving a lower PPL and a shorter convergence time. This superiority stems from NAA's concatenation-based low-rank adapter aggregation, which preserves mathematical consistency and avoids aggregation-induced noise. In contrast, HAA averages low-rank adapters across client devices before matrix multiplication to approximate the aggregated client-side LLM update. This operation introduces additional noise, degrading training performance. A more comprehensive comparison of NAA and HAA across various metrics is illustrated in Table IV.

VI. RELATED WORK

Parameter-efficient Fine-tuning. PEFT has become a promising approach for efficient LLM fine-tuning, substantially reducing trainable parameters without compromising performance. Among various PEFT approaches, Adapter [23], [68], [69] and LoRA [11], [37] are the most popular two. Adapter introduces lightweight trainable modules in each transformer block, which are fine-tuned while freezing the pre-trained model. LoRA enhances fine-tuning efficiency by decomposing weight updates into two low-rank matrices while keeping the pre-trained model frozen. LoRA has become the dominant PEFT method due to its significant superiority over Adapter, i.e., the substantial reduction of trainable parameters through low-rank decomposition without introducing additional inference latency [11], [36], [37]. Several LoRA variants have been proposed to enhance the efficiency and adaptability in LLM fine-tuning [70]–[72], such as LoRA+ [70] that introduces distinct learning rates for low-rank matrices, and LoRA-drop [71] that prunes less important adapters based on output importance to reduce trainable parameters. VeRA [72] utilizes random projections to initialize low-rank matrices with shared weights, reducing computing overhead without sacrificing training performance.

Split Learning. With the growing demand for efficient distributed learning systems, SL [24] become a compelling distributed framework but suffers from excessive training latency due to its sequential training from one client device to another. To overcome this severe limitation, SFL [26] and parallel SL [73] are proposed to parallelize SL for expediting model training. To further improve the training efficiency of SL, some variants of SL have been developed [20], [25], [41], [74]. EPSL [20] reduces the dimension of activations' gradients via last-layer gradient aggregation to accelerate model training, while CPSL [74] partitions devices into several clusters to reduce training latency. SGLR [41] averages the local gradients at the cut layer to combat the server-side large batch and the backward client decoupling problems. AdaptSFL [25] adaptively controls model splitting and client-side model aggregation to balance communication-computing latency and training convergence.

Split Learning for LLMs. Developing an efficient SL fine-tuning framework for LLMs is still in its infancy. Our prior work, SplitLoRA [4], was the first split parameter-efficient fine-tuning framework for LLMs, built on the SFL framework and LoRA fine-tuning. Another recent framework, SplitLLM [75], proposes a hierarchical cloud-edge-client SL scheme for fine-tuning LLM. Though SplitLoRA and SplitLLM have made progress in SL fine-tuning, they fail to scale effectively on computing entities with heterogeneous computing budgets and cannot adjust the fine-tuning strategy to accommodate dynamic changes in computing resources during model training.

VII. CONCLUSION

Taking an important step towards split parameter-efficient fine-tuning paradigm, we have proposed a heterogeneous parameter-efficient fine-tuning framework built on split learning and LoRA fine-tuning method, named HSplitLoRA. HSplitLoRA consists of three primary components: important weight identification, dynamic rank and model splitting configuration, and noise-free adapter aggregation. First, the important weight identification scheme identifies important weights based on their contributions to LLM training to efficiently fine-tune LLM under computing resource constraints. Second, the dynamic rank and model splitting configuration dynamically adjusts the decomposition ranks of LoRA adapters and model split point to accommodate the heterogeneous computing budgets of client devices. Lastly, the noise-free adapter aggregation mechanism meticulously concatenates low-rank decomposition matrices to support heterogeneous adapter aggregation without introducing additional noise. Extensive experiments demonstrate that HSplitLoRA achieves superior performance compared to state-of-the-art benchmarks.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: A Family of Highly Capable Multimodal Models," *arXiv preprint arXiv:2312.11805*, 2023.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "LLaMA: Open and Efficient Foundation Language Models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] Z. Lin, X. Hu, Y. Zhang, Z. Chen, Z. Fang, X. Chen, A. Li, P. Vepakomma, and Y. Gao, "SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models," *arXiv preprint arXiv:2407.00952*, 2024.

[5] 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.

[6] 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.

[7] Y. Tang, Z. Chen, A. Li, T. Zheng, Z. Lin, J. Xu, P. Lv, Z. Sun, and Y. Gao, "MERIT: Multimodal Wearable Vital Sign Waveform Monitoring," *arXiv preprint arXiv:2410.00392*, 2024.

[8] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, "BloombergGPT: A Large Language Model For Finance," *arXiv preprint arXiv:2303.17564*, 2023.

[9] 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.

[10] S. Hu, Z. Fang, Z. Fang, Y. Deng, X. Chen, and Y. Fang, "AgentsCo-Driver: Large Language Model Empowered Collaborative Driving with Lifelong Learning," *arXiv preprint arXiv:2404.06345*, 2024.

[11] 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," in *Proc. ICLR*, 2022.

[12] B. Yuan, Y. He, J. Davis, T. Zhang, T. Dao, B. Chen, P. S. Liang, C. Re, and C. Zhang, "Decentralized Training of Foundation Models in Heterogeneous Environments," in *Proc. NIPS*, 2022.

[13] 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.

[14] M. Hu, J. Zhang, X. Wang, S. Liu, and Z. Lin, "Accelerating Federated Learning with Model Segmentation for Edge Networks," *IEEE Trans. Green Commun. Netw.*, 2024.

[15] J. Shin, Y. Li, Y. Liu, and S.-J. Lee, "FedBalancer: Data and Pace Control for Efficient Federated Learning on Heterogeneous Clients," in *Proc. MobiSys*, 2022.

[16] K. Panchal, S. Choudhary, N. Parikh, L. Zhang, and H. Guan, "Flow: Per-instance Personalized Federated Learning," in *Proc. NIPS*, 2023.

[17] Z. Wang, Z. Shen, Y. He, G. Sun, H. Wang, L. Lyu, and A. Li, "FLoRA: Federated Fine-Tuning Large Language Models with Heterogeneous Low-Rank Adaptations," in *Proc. NIPS*, 2024.

[18] Y. Zhang, Z. Lin, Z. Chen, Z. Fang, W. Zhu, X. Chen, J. Zhao, and Y. Gao, "SatFed: A Resource-Efficient LEO Satellite-Assisted Heterogeneous Federated Learning Framework," *arXiv preprint arXiv:2409.13503*, 2024.

[19] L. Dong, Z. Yang, X. Cai, Y. Zhao, Q. Ma, and X. Miao, "WAVE: Edge-Device Cooperated Real-Time Object Detection for Open-Air Applications," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4347–4357, 2022.

[20] 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.

[21] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Efficient Federated Learning for Modern NLP," in *Proc. MobiCom*, 2023.

[22] T. Che, J. Liu, Y. Zhou, J. Ren, J. Zhou, V. Sheng, H. Dai, and D. Dou, "Federated Learning of Large Language Models with Parameter-Efficient Prompt Tuning and Adaptive Optimization," in *Proc. EMNLP*, 2023.

[23] 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.

[24] 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.

[25] 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.

[26] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," in *Proc. AAAI*, 2022.

[27] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, and K. Suo, "An Empirical Analysis and Resource Footprint Study of Deploying Large Language Models on Edge Devices," in *Proc. ACM SE*, 2024, pp. 69–76.

[28] D. Xu, W. Yin, H. Zhang, X. Jin, Y. Zhang, S. Wei, M. Xu, and X. Liu, "EdgeLLM: Fast On-device LLM Inference with Speculative Decoding," *IEEE Trans. Mobile Comput.*, 2024.

[29] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile Edge Intelligence for Large Language Models: A Contemporary Survey," *IEEE Commun. Surv. Tutor.*, 2025.

[30] X. Gu, K. Huang, J. Zhang, and L. Huang, "Fast Federated Learning in the Presence of Arbitrary Device Unavailability," in *Proc. NIPS*, 2021.

[31] P. Theodoropoulos, K. E. Nikolakakis, and D. Kalogerias, "Federated Learning under Restricted User Availability," in *Proc. ICASSP*, 2024.

[32] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing Impacts of Heterogeneity in Federated Learning Upon Large-scale Smartphone Data," in *Proc. WWW*, 2021.

[33] 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," *IEEE Commun. Mag.*, 2023.

[34] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, 2019.

[35] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," in *Proc. ICLR*, 2015.

[36] Z. Hu, L. Wang, Y. Lan, W. Xu, E.-P. Lim, L. Bing, X. Xu, S. Poria, and R. Lee, "LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models," in *Proc. EMNLP*, 2023.

[37] 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.

[38] J. Novikova, O. Dušek, and V. Rieser, "The E2E Dataset: New Challenges For End-to-End Generation," in *Proc. SIGDIAL*, 2017.

[39] H. Sun, H. Tian, W. Ni, J. Zheng, D. Niyato, and P. Zhang, "Federated Low-Rank Adaptation for Large Models Fine-Tuning over Wireless Networks," *IEEE Trans. Wireless Commun.*, 2024.

[40] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *Proc. NIPS*, 2024.

[41] S. Pal, M. Uniyal, J. Park, P. Vepakomma, R. Raskar, M. Bennis, M. Jeon, and J. Choi, "Server-side Local Gradient Averaging and Learning Rate Acceleration for Scalable Split Learning," *arXiv preprint arXiv:2112.05929*, 2021.

[42] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and Trends of SRAM-Based Computing-in-Memory for AI Edge Devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[43] X. Zhang and S. Debroy, "Resource Management in Mobile Edge Computing: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–37, 2023.

[44] T. Zhang, L. Gao, S. Lee, M. Zhang, and S. Avestimehr, "TimelyFL: Heterogeneity-aware Asynchronous Federated Learning with Adaptive Partial Training," in *Proc. CVPR*, 2023.

[45] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, "FjORD: Fair and Accurate Federated Learning Under Heterogeneous Targets With Ordered Dropout," *Proc. NIPS*, 2021.

[46] X. Ouyang, Z. Xie, J. Zhou, G. Xing, and J. Huang, "ClusterFL: A Clustering-based Federated Learning System for Human Activity Recognition," *ACM Trans. Sensor Netw.*, vol. 19, no. 1, pp. 1–32, 2022.

[47] S. Li, H. Lu, T. Wu, M. Yu, Q. Weng, X. Chen, Y. Shan, B. Yuan, and W. Wang, "CaraServe: CPU-Assisted and Rank-Aware LoRA Serving for Generative LLM Inference," *arXiv preprint arXiv:2401.11240*, 2024.

[48] J. Zhang, S. Vahidian, M. Kuo, C. Li, R. Zhang, T. Yu, G. Wang, and Y. Chen, "Towards Building the FederatedGPT: Federated Instruction Tuning," in *Proc. ICASSP*, 2024.

[49] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[50] B. Hanin, "Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?" in *Proc. NIPS*, 2018.

[51] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?" in *Proc. MLSys*, 2020.

[52] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling For Convolutional Neural Networks," in *Proc. ICML*, 2019.

[53] Z. Lin, W. Wei, Z. Chen, C.-T. Lam, X. Chen, Y. Gao, and J. Luo, "Hierarchical Split Federated Learning: Convergence Analysis and System Optimization," *IEEE Trans. Mobile Comput.*, 2025.

[54] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and Prediction-based Scheduling in Deep Learning Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 88–100, May. 2021.

[55] J. Zhang, X. Zhang, D. Zhang-Li, J. Yu, Z. Yao, Z. Ma, Y. Xu, H. Wang, X. Zhang, N. Lin *et al.*, "GLM-Dialog: Noise-tolerant Pre-training For Knowledge-grounded Dialogue Generation," in *Proc. ACM KDD*, 2023.

[56] F. Wu, Z. Li, Y. Li, B. Ding, and J. Gao, "FedBiOT: LLM Local Finetuning in Federated Learning Without Full Model," in *Proc. ACM KDD*, 2024.

[57] Y. Wang, S. Si, D. Li, M. Lukasik, F. Yu, C.-J. Hsieh, I. S. Dhillon, and S. Kumar, "Two-stage LLM Fine-tuning with Less Specialization and More Generalization," in *Proc. ICML*, 2024.

[58] S. Kotha, J. M. Springer, and A. Raghunathan, "Understanding Catastrophic Forgetting in Language Models Via Implicit Inference," in *Proc. ICLR*, 2024.

[59] B. Xu, X. Shu, H. Mei, Z. Bai, B. Fernando, M. Z. Shou, and J. Tang, "DoFIT: Domain-aware Federated Instruction Tuning With Alleviated Catastrophic Forgetting," in *Proc. NIPS*, 2024.

[60] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," in *Proc. NIPS*, 2023.

[61] Y. J. Cho, L. Liu, Z. Xu, A. Fahrezi, and G. Joshi, "Heterogeneous LoRA for Federated Fine-tuning of On-device Foundation Models," in *Proc. EMNLP*, 2024.

[62] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "Fedpara: Low-rank Hadamard Product For Communication-efficient Federated Learning," *arXiv preprint arXiv:2108.06098*, 2021.

[63] R. Singhal, K. Ponkshe, and P. Vepakomma, "Exact Aggregation for Federated and Efficient Fine-Tuning of Foundation Models," *arXiv preprint arXiv:2410.09432*, 2024.

[64] J. D. Beshay, A. Francini, and R. Prakash, "On the Fidelity of Single-Machine Network Emulation in Linux," in *Proc. of the 23rd IEEE MASCOTS*, 2015.

[65] (2024) "NVIDIA Jetson Embedded Systems Developer Kits and Modules". Available: https://www.nvidia.cn/autonomous-machines/embedded-systems/?section=jetsonTX2.

[66] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language Models are Unsupervised Multitask Learners," *OpenAI Blog*, 2019.

[67] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More Effective Distributed ML via A Stale Synchronous Parallel Parameter Server," in *Proc. NIPS*, 2013.

[68] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, "Adapter-Fusion: Non-Destructive Task Composition for Transfer Learning," in *Proc. EACL*, 2021.

[69] R. Karimi Mahabadi, J. Henderson, and S. Ruder, "Compacter: Efficient Low-Rank Hypercomplex Adapter Layers," in *Proc. NIPS*, 2021.

[70] S. Hayou, N. Ghosh, and B. Yu, "LoRA+: Efficient Low Rank Adaptation of Large Models," in *Proc. ICML*, 2024.

[71] H. Zhou, X. Lu, W. Xu, C. Zhu, T. Zhao, and M. Yang, "Lora-drop: Efficient LoRA Parameter Pruning Based on Output Evaluation," *arXiv preprint arXiv:2402.07721*, 2024.

[72] D. J. Kopiczko, T. Blankevoort, and Y. M. Asano, "VeRA: Vector-based Random Matrix Adaptation," in *Proc. ICLR*, 2024.

[73] M. Kim, A. DeRieux, and W. Saad, "A Bargaining Game For Personalized, Energy Efficient Split Learning over Wireless Networks," in *Proc. WCNC*, 2023.

[74] 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. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, 2023.

[75] S. Zhang, G. Cheng, Z. Li, and W. Wu, "SplitLLM: Hierarchical Split Learning for Large Language Model over Wireless Network," *arXiv preprint arXiv:2501.13318*, 2025.