

# Federated Data-Efficient Instruction Tuning for Large Language Models

Zhen Qin<sup>1</sup>, Zhaomin Wu<sup>2\*</sup>, Bingsheng He<sup>2</sup>, Shuiguang Deng<sup>1\*</sup>

<sup>1</sup>Zhejiang University, <sup>2</sup>National University of Singapore

zhenqin@zju.edu.cn, zhaomin@nus.edu.sg, hebs@comp.nus.edu.sg, dengsg@zju.edu.cn

## Abstract

Instruction tuning is a crucial step in improving the responsiveness of pretrained large language models (LLMs) to human instructions. Federated learning (FL) helps to exploit the use of vast private instruction data from clients, becoming popular for LLM tuning by improving data diversity. Existing federated tuning simply consumes all local data, causing excessive computational overhead and overfitting to local data, while centralized data-efficient solutions are not suitable for FL due to privacy concerns. This work presents FedHDS, a federated data-efficient instruction tuning approach, which tunes LLMs with a representative subset of edge-side data. It reduces the data redundancy at both intra- and inter-client levels without sharing raw data. Experiments with various LLMs, datasets and partitions show that FedHDS improves Rouge-L on unseen tasks by an average of 10.72% over the SOTA full-data federated instruction tuning methods, while using less than 1.5% of the data samples, improving training efficiency by up to tens of times.

## 1 Introduction

Large language models (LLMs) exhibit remarkable performance on a wide range of natural language tasks. Instruction tuning (Wang et al., 2022) is crucial to improve LLMs’ responsiveness to human instructions, whose success hinges on the availability of diverse and high-quality data (Wang et al., 2024, 2023; Li et al., 2024a), particularly for unseen tasks (Wei et al., 2022). As the pool of publicly available data is expected to be exhausted in the near future (Villalobos et al., 2022), exploiting more data sources, such as data on edge devices, becomes essential for continued LLM improvement (Qin et al., 2024a). However, privacy concerns and regulations such as GDPR (Voigt and Von dem Bussche, 2017) complicate the use of edge-side data.

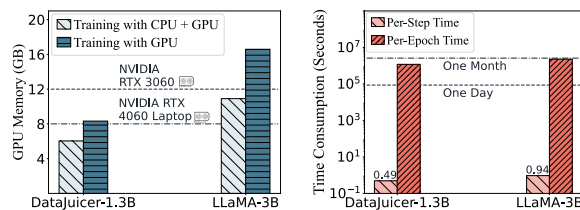


Figure 1: GPU memory cost by training with LoRA<sup>1</sup> v.s. capacity of popular edge-side GPUs<sup>2</sup>.

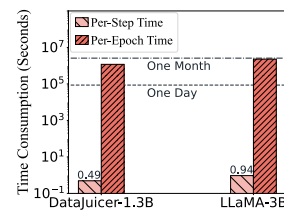


Figure 2: Per-step/epoch time of BP with LoRA on Natural Instructions dataset by CPU+GPU.

Federated learning (FL) (McMahan et al., 2017) has emerged as a promising solution to utilize diverse data from edge devices to tune LLMs (Zhang et al., 2024), especially for generalization to unseen tasks (Qin et al., 2024a; Bai et al., 2024). Prior FL works on LLMs mainly focus on communication and memory costs (Zhang et al., 2024; Babakniya et al., 2023; Zhang et al., 2023; Che et al., 2023; Kuang et al., 2024; Xu et al., 2024; Qin et al., 2024a). Although these methods show promising results, our investigation reveals that they often use all local data for training (Qin et al., 2024a; Xu et al., 2024; Zhang et al., 2024; Ling et al., 2024). This strategy leads to many training steps, and thus causes two main issues in FL contexts:

(1) Efficiency: It is time-consuming to train LLMs, especially when edge devices have limited GPU capacity and thus require CPU+GPU hybrid computing (Figure 1), where iterating through the entire dataset may incur intolerable time consumption (Figure 2). (2) Generalization: FL clients usually hold data covering limited domains. Training on redundant data increases the risk of overfitting, harming generalization to unseen tasks. Thus, in FL, it is necessary to explore tuning LLMs using

<sup>1</sup> CPU+GPU training is implemented with Deepspeed (Rasley et al., 2020) and a batch size of 1 (loaded in 16-bit).

<sup>2</sup> The most popular desktop and laptop GPUs are reported by Steam Hardware Survey (Dec 2024).

\*Corresponding authors.

fewer representative data to improve efficiency and reduce overfitting, i.e., *data-efficient instruction tuning* (Chen et al., 2023; Sachdeva et al., 2024), which has been touched in centralized scenarios, e.g., tuning LLMs with only high-quality samples (Zhou et al., 2023; Li et al., 2024b; Lu et al., 2023; Cui et al., 2023) or representative samples (a.k.a. coreset) (Chen et al., 2023; Wu et al., 2023; Cao et al., 2024). Among them, coreset-based methods usually excel in reducing data volume.

Despite their success in centralized scenarios, applying these works to FL faces two main limitations. **(L1) Low compatibility with FL:** Some methods access all data simultaneously (Chen et al., 2023; Wu et al., 2023; Cao et al., 2024), violating FL’s fundamental principle of restricting direct access to data by third parties, and thus failing to detect inter-client data redundancy. **(L2) Suboptimal data representations:** The selection of coresets significantly impacts the tuning accuracy (Zha et al., 2024; Wang et al., 2024). Current methods rely on data representations from the last Transformer layer (Chen et al., 2023; Wu et al., 2023; Cao et al., 2024), which may not capture the full spectrum of features to distinguish data samples.

Due to the limited research on federated data-efficient instruction tuning and the shortcomings of centralized methods in FL scenarios, we propose **FedHDS**, a novel framework of **federated hierarchical data selection**. It clusters local data on each client to detect *intra-client* data redundancy, and then sends approximate cluster centroids to the server for further clustering to identify *inter-client* data redundancy, cutting down computational costs and enhancing generalization to new tasks by taking the distribution of edge-side data without direct data access, which is compatible with FL (L1). Then, we propose to fuse data features from different Transformer layers, in order to provide better data representations for coreset selection (L2).

This work makes the following contributions:

- We propose FedHDS, a framework that tunes LLMs using coreset while alleviating *intra-client* and *inter-client* data redundancy. To the best of our knowledge, this is the *first* study on federated data-efficient instruction tuning for LLMs.
- We propose a simple yet effective method to fuse features of varying abstraction levels from different Transformer layers. It works within FedHDS to facilitate clustering-based coreset selection for federated instruction tuning.
- We conduct experiments on two widely-used instruction datasets with various LLMs and non-IID partitions, showing that FedHDS improves Rouge-L on unseen tasks by 10.72% on average, using less than 1.5% of the data samples compared to existing practical federated baselines<sup>1</sup>.

## 2 Related Work

**Federated Learning for LLM Tuning** Federated tuning for LLMs gains widespread attention by enabling edge-side data use without direct access. Due to the scale of LLMs, many studies develop federated tuning based on parameter-efficient fine-tuning (PEFT) techniques to reduce memory and communication costs (Zhang et al., 2024; Babakniya et al., 2023; Zhang et al., 2023; Che et al., 2023). Among PEFT techniques, LoRA (Hu et al., 2022a) gains significant attention, with studies on initialization (Babakniya et al., 2023), objective consistency (Sun et al., 2024), and heterogeneous client resources (Bai et al., 2024). Some works focus on specific costs, such as communication by transmitting only loss values and random seeds (Qin et al., 2024a) and memory by employing quantization-aware training (Xu et al., 2024).

During local training, existing works typically consume all local data (Qin et al., 2024a; Xu et al., 2024; Zhang et al., 2024; Wu et al., 2024; Kuang et al., 2024; Sun et al., 2024), leading to a significant computational cost and overfitting to the local data, as client-side data in FL often consists of many samples from only a few domains. Unlike existing methods, this work takes a data-centric perspective (Zha et al., 2024), which focuses on *data efficiency* by selecting a small number of representative data samples for LLM tuning, to achieve better efficiency and generalization on unseen tasks.

**Data-Efficient LLM Instruction Tuning** Existing studies show that LLMs learn knowledge primarily from pretraining, and can be taught to produce better outputs with a limited amount of instruction data (Zhou et al., 2023). Some studies achieve comparable or better tuning results using fewer samples by focusing on data quality or the representativeness of data samples.

Quality-oriented works either filter out low-quality samples by heuristics (Chen et al., 2024), quality indicators (Li et al., 2024c; Chen et al.,

<sup>1</sup>Our codes are available at <https://github.com/zhenqincn/FedHDS>.

2024) or third-party LLMs (Li et al., 2024b), or curate high-quality samples by manual efforts (Zhou et al., 2023) or third-party LLMs (Lu et al., 2023; Cui et al., 2023). Methods based on heuristics or quality indicators can filter out a limited number of samples. Manual or third-party LLM-based methods are unsuitable for FL due to privacy risks.

Representativeness-based works select a few representative samples by clustering algorithms, filtering out much of the data while retaining tuning accuracy (e.g., more than 95% of the dataset) (Chen et al., 2023; Wu et al., 2023). However, existing coreset selection methods have shortcomings in FL contexts: 1) the nature of FL, where data stays on the device, prevents existing methods from being fully integrated, such as detecting data redundancy across clients. 2) existing methods often extract data features for clustering using only the last layer of the Transformer (Chen et al., 2023; Wu et al., 2023; Li et al., 2024c), which may result in a sub-optimal feature space to distinguish coresets.

Table 1: Qualitative comparison between related instruction tuning methods for LLMs and FedHDS.

	Reduce Intra-Client Data Redundancy	Reduce Inter-Client Data Redundancy (Privacy-Preserving)	Full-Layer Feature Fusion
Federated Instruction Tuning	✗	✗	✗
Centralized Data-Efficient Instruction Tuning	✓	✗	✗
<b>FedHDS (ours)</b>	✓	✓	✓

**Summary** As Table 1, existing federated tuning overlooks data redundancy. Centralized data-efficient methods are either impractical for FL due to external data transmission or fail to address inter-client redundancy (L1). They also rely on only the last-layer Transformer outputs to distinguish data samples, potentially causing sub-optimal data representations (L2). Unlike these methods, FedHDS performs instruction tuning using a coreset while alleviating both intra- and inter-client redundancy, with fused features from all Transformer layers.

### 3 Problem Formulation

**Federated Instruction Tuning** Assuming there are  $N$  clients in an FL system, each client  $i$  holds a private dataset  $\mathcal{D}_i$  containing several instances of instruction data. Given an LLM  $\mathbf{w} \in \mathbb{R}^d$  initialized by the pretrained weight  $\mathbf{w}^0$ , federated instruction tuning aims at optimizing  $\mathbf{w}$  with the private data

held by clients towards the following objective,

$$\min_{\mathbf{w}} f(\mathbf{w}) \triangleq \sum_{i=1}^N \lambda_i \cdot \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_i} [\mathcal{L}(\mathbf{w}; \mathbf{x})], \quad (1)$$

where  $\mathcal{L}(\mathbf{w}; \mathbf{x})$  is the loss evaluated on model  $\mathbf{w}$  for data instance  $\mathbf{x}$  sampled from  $\mathcal{D}_i$ , and  $\lambda_i$  is the weight of client  $i$  that follows  $\lambda_i > 0$  and  $\sum_i^N \lambda_i = 1$ . Symbol  $\mathbf{x}$  is used as the batch size is 1 to reduce memory usage (Qin et al., 2024a). To solve Eq. (1), FL iterates multiple rounds. In each round  $r$ , several active clients get the latest model parameters  $\mathbf{w}^r$  from the server and perform several steps of stochastic gradient descent (SGD), as

$$\mathbf{w}_{i,t+1}^r = \mathbf{w}_{i,t}^r - \eta \cdot \nabla_{\mathbf{w}_{i,t}^r} \mathcal{L}(\mathbf{w}_{i,t}^r; \mathbf{x}), \forall \mathbf{x} \in \mathcal{D}_i, \quad (2)$$

where  $\mathbf{w}_{i,t}^r$  is the model of client  $i$  at the  $t$ -th local step in round  $r$ , and  $\eta$  is the learning rate. Typically, the process iterates over the local dataset  $\mathcal{D}_i$  for one or more epochs (Qin et al., 2024a; Xu et al., 2024; Zhang et al., 2024). After local training, each active client sends the updated model to the server. To alleviate communication and memory costs, FL typically adopts PEFT techniques, where only a small subset of model parameters is trained and transmitted. Similarly to Zhang et al. (2024), this work only trains and transmits LoRA adapters.

#### Federated Data-Efficient Instruction Tuning

Assuming each client uses a data selection function  $f(\mathbf{x}; \cdot) \rightarrow \{0, 1\}$  to construct a coreset

$$\tilde{\mathcal{D}}_i = \{\mathbf{x} \in \mathcal{D}_i \mid f(\mathbf{x}; \cdot) = 1\}, \quad (3)$$

and performs local training only on  $\tilde{\mathcal{D}}_i$  as Eq. (2). If  $f$  makes the model  $\tilde{\mathbf{w}}$  trained on  $\{\tilde{\mathcal{D}}_i\}_{i=1}^N$  achieve accuracy comparable to—or even better than—that obtained with the full datasets, while satisfying

$$\sum_{i=1}^N |\tilde{\mathcal{D}}_i| / \sum_{i=1}^N |\mathcal{D}_i| \ll 1, \quad (4)$$

then the method is *data-efficient*, which greatly reduces the computation cost than the full-data ones.

Additionally, by considering the redundancy among local coresets, we can further reduce the local iterations for each client. Assuming an ideal distribution  $\mathcal{P}^*$  that each real-world data  $\mathbf{x}$  follows (Qin et al., 2024b), and data samples in each coreset  $\tilde{\mathcal{D}}_i$  follow  $\mathcal{P}_i$ , where local data distributions  $\{\mathcal{P}_1, \dots, \mathcal{P}_N\}$  share mutual similarity to some extent since all the client-side data can be regarded

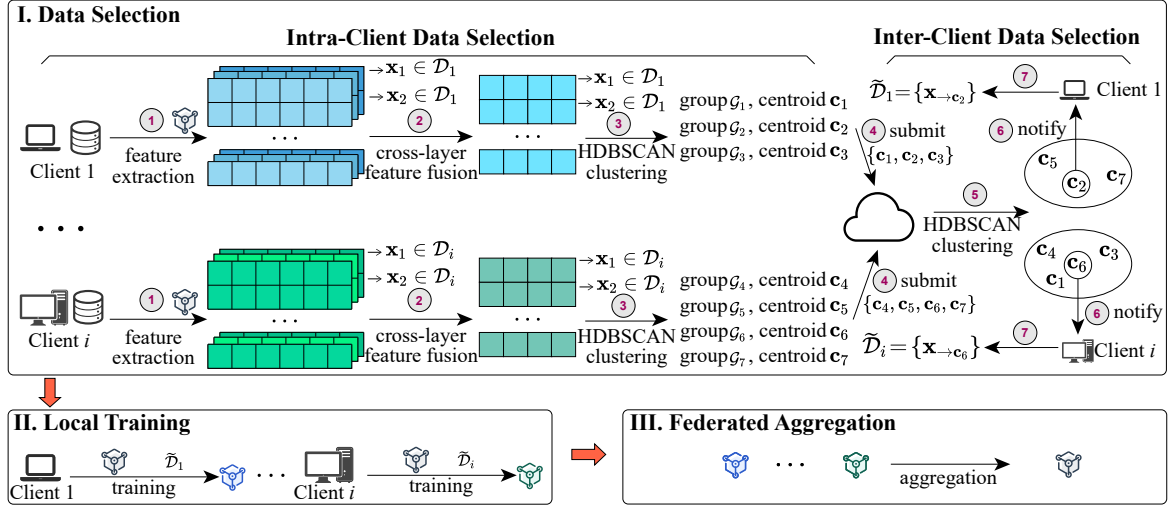


Figure 3: Overview of FedHDS in each round of FL. It differs from vanilla FL (McMahan et al., 2017) by incorporating “I. Data Selection” to select representative data samples before “II. Local Training”. Centroid  $c$  is derived from HDBSCAN and does not correspond to specific samples. Data sample  $\mathbf{x}_{\rightarrow c_j}$  denotes the one closest to the centroid  $c_j$  of corresponding group  $\mathcal{G}_j$ . Set  $\tilde{\mathcal{D}}_i$  contains representative samples in dataset  $\mathcal{D}_i$  of the  $i$ -th client.

as sampled from  $\mathcal{P}^*$ , so that inter-client data redundancy may exist. Experimental results in Section 5.4 also illustrate this. To alleviate intra-client and inter-client data redundancy, we design FedHDS.

## 4 Approach

### 4.1 Overview

FedHDS identifies the representative data samples with their latent features. As shown in Figure 3, in round  $r$ , after downloading the global model  $\mathbf{w}^r$ , each client performs intra-client selection, followed by an inter-client selection performed by the server.

In intra-client selection, client  $i$  gets the hidden state in each Transformer layer with each data sample  $\mathbf{x} \in \mathcal{D}_i$  as the input (① in Figure 3). Then, these features are fused across different Transformer layers (② in Figure 3). Next, clustering is performed to partition the data into several groups (③ in Figure 3). Each group holds an approximate centroid  $c$  that does not correspond to an individual data point. After that, the client sends these centroids  $\{c_1, c_2, \dots\}$  to the server (④ in Figure 3).

Upon receiving the centroids from all active clients, inter-client selection starts. The server clusters received centroids into several groups (⑤ in Figure 3). In each group, the point closest to its centroid  $c_j^H$  is designated as the chosen one. Then, the server notifies each client regarding which of their sent centroids are selected (⑥ in Figure 3). Next, each client  $i$  adds the data sample closest to each of the selected centroid within the corresponding

group to coreset  $\tilde{\mathcal{D}}_i$  (⑦ in Figure 3).

Finally, client  $i$  performs local training on  $\tilde{\mathcal{D}}_i$  as Zhang et al. (2024). The selection processes are summarized in Algorithm 1 of Appendix A. In the following, we detail the design of FedHDS.

### 4.2 Intra-Client Data Selection

Data samples are selected based on features. Given an LLM with  $l$  Transformer layers, it can extract data features layer by layer and token by token, as

$$\begin{bmatrix} \mathbf{h}_j^{1,1} & \mathbf{h}_j^{1,2} & \dots & \mathbf{h}_j^{1,-1} \\ \mathbf{h}_j^{2,1} & \mathbf{h}_j^{2,2} & \dots & \mathbf{h}_j^{2,-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{h}_j^{l,1} & \mathbf{h}_j^{l,2} & \dots & \mathbf{h}_j^{l,-1} \end{bmatrix}, \quad (5)$$

where  $\mathbf{h}_j^{l,b} \in \mathbb{R}^v$  is the hidden states of the  $b$ -th token from the  $l$ -th Transformer layer for the  $j$ -th data sample, with  $b=-1$  denoting the last token. From the token level, FedHDS uses the hidden state of the last token as Li et al. (2024c) since it encapsulates all preceding token information.

At the layer level, prior works often use the last layer (Chen et al., 2023; Wu et al., 2023; Li et al., 2024c), which may not be universally optimal since different layers provide varying abstraction degrees to data representations. We show it with a toy example on Dolly-15K, clustering its 8-category instructions into 8 groups using K-means. Figure 4 evaluates clustering quality with Calinski-Harabasz Index (Caliński and Harabasz, 1974), and  $F_1$ -score



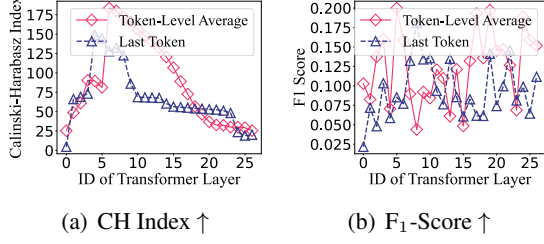


Figure 4: Evaluations of clustered data groups based on features from different Transformer layers, obtained in a centralized scenario with LLaMA-3B on Dolly-15K.

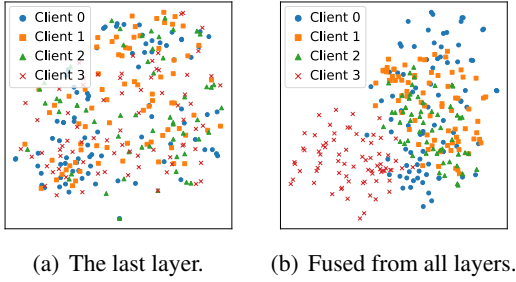


Figure 5: Visualization of features obtained with DataJuicer-1.3B on Dolly-15K ( $\alpha=5.0$ ).

(more evaluations are left in Appendix E.1), showing that the last layer is not universally optimal, and no single layer excels across all metrics.

Predicting the optimal layer is challenging, and computing clustering metrics for all layers is costly. A feasible approach is fusing features from all layers. One naive method is concatenating the last token’s hidden states across layers, as

$$\mathbf{h}_j = [\mathbf{h}_j^{1,-1}, \mathbf{h}_j^{2,-1}, \dots, \mathbf{h}_j^{l,-1}]. \quad (6)$$

However, Figure 4 shows that some layers degrade data separability. Dimensionality reduction may alleviate the impact of inappropriate dimensions, e.g., low-variance dimensions have minimal impact on distance calculation in t-SNE (Van der Maaten and Hinton, 2008). Thus, we apply  $P : \mathbb{R}^{l \times v} \rightarrow \mathbb{R}^k, k \ll l \times v$  to fuse the features from  $l$  layers, as

$$\{\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \dots, \tilde{\mathbf{h}}_{|\mathcal{D}_i|}\} = P(\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{|\mathcal{D}_i|}\}). \quad (7)$$

We choose t-SNE with Barnes-Hut implementation (van der Maaten, 2013), as it is more effective in nonlinear spaces than earlier methods such as PCA (Jolliffe, 2002). The fused feature dimension  $k$  is set to 2 for efficiency. Figure 5 shows the effectiveness of feature fusion, where a random subset of Dolly-15K is partitioned to 4 clients with label distribution skew. If last-layer features are employed,

data among clients tends to be scattered, while fused features form relatively clear boundaries between clients, enhancing sample distinction.

After obtaining fused features of local data  $\mathcal{D}_i$ , denoted by  $\{\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \dots, \tilde{\mathbf{h}}_{|\mathcal{D}_i|}\}$ , HDBSCAN (Campello et al., 2013) is applied to cluster them into groups  $\{\mathcal{G}_1, \mathcal{G}_2, \dots\}$ . Each group  $\mathcal{G}_j$  holds a centroid  $\mathbf{c}_j$  that does not correspond to a real sample. These centroids are sent to the server to determine which groups are selected for tuning.

### 4.3 Inter-Client Data Selection

As discussed in Section 3 and shown in Figure 5, there may be similarities among client-side data. Thus, we cluster the approximate centroids sent from the clients to the server with HDBSCAN to filter redundant data groups among clients, as

$$\mathbb{G}^{\Pi} = \{\mathcal{G}_1^{\Pi}, \mathcal{G}_2^{\Pi}, \dots\} = \text{HDBSCAN}(\{\mathbf{c}_1, \mathbf{c}_2, \dots\}). \quad (8)$$

For each  $\mathcal{G}_j^{\Pi}$  with its approximate centroid  $\mathbf{c}_j^{\Pi}$ , the server identifies the first-layer group  $\mathcal{G}_s$  whose centroid  $\mathbf{c}_s$  closest to  $\mathbf{c}_j^{\Pi}$  as the selected one, as

$$\mathbb{G}^{\text{selected}} = \{\mathcal{G}_s \mid \mathbf{c}_s = \arg \min_{\mathbf{c} \in \mathcal{G}_j^{\Pi}} \|\mathbf{c} - \mathbf{c}_j^{\Pi}\|\} \mid \mathcal{G}_j^{\Pi} \in \mathbb{G}^{\Pi}. \quad (9)$$

Then, the corresponding clients will be notified of the selection of first-layer groups. Given the partitioned data groups  $\mathbb{G}_i = \{\mathcal{G}_1, \mathcal{G}_2, \dots\}$  of client  $i$ , the subset finally used for tuning  $\tilde{\mathcal{D}}_i$  is obtained as:

$$\tilde{\mathcal{D}}_i = \{\mathbf{x} \mid \mathbf{x} \in \mathcal{G}_j, \mathcal{G}_j \in \mathbb{G}_i, \mathcal{G}_j \in \mathbb{G}^{\text{selected}}\}. \quad (10)$$

In each selected group  $\mathcal{G}_j$ , the data sample closest to centroid  $\mathbf{c}_j$  is added in  $\tilde{\mathcal{D}}_i$  ( $\mathbf{x} \rightarrow \mathbf{c}_j$  in Figure 3).

### 4.4 Instruction Tuning with Coresets

After determining coresets, each client  $i$  performs local training using only  $\tilde{\mathcal{D}}_i$  (II in Figure 3), as

$$\mathbf{w}_{i,t+1}^r = \mathbf{w}_{i,t}^r - \eta \cdot \nabla_{\mathbf{w}_{i,t}^r} \mathcal{L}(\mathbf{w}_{i,t}^r; \mathbf{x}), \forall \mathbf{x} \in \tilde{\mathcal{D}}_i, \quad (11)$$

followed by sending the optimized model parameters. Then, the server performs FedAvg (McMahan et al., 2017) on the received parameters, as

$$\mathbf{w}^{r+1} = \frac{1}{\sum_{i \in \mathbb{M}_r} |\tilde{\mathcal{D}}_i|} \sum_{i \in \mathbb{M}_r} |\tilde{\mathcal{D}}_i| \cdot \mathbf{w}_{i,-1}^r, \quad (12)$$

where  $\mathbb{M}_r$  contains the indices of activate clients in the  $r$ -th round of FL. Finally, the next round starts.

FedHDS only performs downsampling on the local training data based on FedAvg, thus its convergence is theoretically supported, as discussed in Appendix B. Besides, we provide discussions on the theoretical speedup of FedHDS in Appendix C.

## 4.5 Further Enhancement for FedHDS

**Efficiency** The efficiency of FedHDS can be further improved by faster feature extraction, since it requires forward propagation across all the local data with the LLM  $w$ . Motivated by studies on retrieval augmented generation which adopt a light-weight retrieval model to extract data features (Fan et al., 2024), we use a smaller language model sharing Transformer architecture and next-token prediction paradigm with  $w$  as a proxy to generate representations for each data, i.e., a small version of GPT-2 (about 124M parameters). We term this approach as FedHDS-Turbo.

**Privacy** Although local centroids sent to the server are two-dimensional vectors that do not correspond to real samples, FedHDS offers more information than vanilla FL. We can adopt a straightforward differential privacy method for scenarios requiring stronger privacy protection (Hu et al., 2022b). We scale elements in local centroids to  $[-1, 1]$  with  $\tanh$  to maintain data distinguishability despite extreme values. Gaussian noise is then added to the scaled centroids before transmission, deriving their differential privacy (Theorem 1). Experiments in Section 5.7 show that FedHDS performs well with a reasonable noise scale.

**Theorem 1.** Let  $\mathbf{c} \in [-1, 1]$  be the original centroid and  $\mathbf{c}' = \mathbf{c} + \mathbf{z}$  be the one noised by Gaussian noise  $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$  with standard deviation  $\sigma$ . Suppose  $\varepsilon, \delta \in (0, 1)$ . Noised centroid  $\mathbf{c}'$  satisfies  $(\varepsilon, \delta)$ -differential privacy if  $\sigma \geq \frac{2\sqrt{2\log(1.25/\delta)}}{\varepsilon}$ .

*Proof.* The proof follows directly by applying the Gaussian mechanism (Dwork et al., 2014), where the sensitivity (i.e., range of  $\mathbf{c}$ ) is 2.  $\square$

## 5 Evaluations

### 5.1 Experimental Setup

**Baselines** We introduce six federated tuning methods using full data as baselines: 1) FedAvg that tunes and transmits the full LLM, included for reference due to high costs; 2&3) FedPTuning and FedPrompt (Kuang et al., 2024) that apply PEFT techniques of P-Tuning (Liu et al., 2023) and Prompt Tuning (Lester et al., 2021) based on FedAvg, respectively, trained with Adam (Kingma and Ba, 2015) optimizer; 4&5) FedIT (Zhang et al., 2024): instruction tuning based on FedAvg with LoRA, optimized with Adam or SGD (FedIT-SGD); 6) FlexLoRA (Bai et al., 2024) that supports

LoRA adapters with varying ranks based on FedIT.

Given the lack of data-efficient works in FL, we develop two federated methods and a centralized one using coresets: 1) Random: It randomly selects a ratio of local data. Although being native, it is a strong baseline (Lin et al., 2024; Sachdeva et al., 2024) by preserving the original data distributions; 2) Perplexity that selects data with lower perplexity scores (Chen et al., 2024); and 3) Coreset-Cent (Chen et al., 2023) that selects a fixed ratio of data by K-means clustering on the last-layer features.

**Datasets and Evaluations** Following Qin et al. (2024a); Kuang et al. (2024), we conduct experiments on Natural Instructions (Wang et al., 2022) (NI) and Dolly-15K (Conover et al., 2023), and employ Rouge-L on held-out tasks as the evaluation metrics. After preprocessing (detailed in Appendix D.4), NI contains 738 training tasks, each of which is assigned to a unique client, providing non-IIDness with feature skew, and the natively provided 119 test tasks are used for evaluation. Dolly-15K contains 8 tasks. The last one is used for evaluation, and the rest are partitioned to 200 clients via Dirichlet distribution with  $\alpha$  set to 0.5 and 5.0, respectively. Experiments on these two datasets provide scenarios where the client has hundreds and dozens of data samples, respectively.

**Implementation** This work targets cross-device FL, thus, 5% of the clients are randomly selected to participate in each round. Limited by space, the implementation is detailed in Appendix D.

### 5.2 Comparison on Accuracy





















We compare these methods in Table 2. The results of full-data FL methods are derived from Qin et al. (2024a) under the same settings, and those of others are obtained within the best hyperparameters.

**Comparison to Full-Data Methods** From Table 2, FedHDS and FedHDS-Turbo outperform full-data FL baselines across the six scenarios with consumed data samples less than 1.5% of them. Particularly, on NI with DataJuicer-1.3B, FedHDS and FedHDS-Turbo relatively improve Rouge-L over FedIT—the practical full-data FL baseline achieving the best average accuracy—by 19.5% and 16.3%, respectively. Averaged across the six scenarios, FedHDS-Turbo improves the Rouge-L score relative to FedIT by 10.72%. Besides, compared to FedIT, FedHDS achieves an average improvement of 4.26% and 4.96% in Rouge-L on

Table 2: Rouge-L (%) comparisons. Parentheses indicate the ratio of consumed data samples compared to full-data methods. Each value is the average Rouge-L obtained in the last round of four runs with different random seeds. **Coreset-Cent** and **FedAvg** are introduced just as references as they are not practical to end devices. Bold and underlined numbers are the best and second-best values among approaches practical to cross-device FL, respectively.

Approach	Natural Instructions (Meta Non-IID)		Dolly-15K ( $\alpha = 0.5$ )		Dolly-15K ( $\alpha = 5.0$ )	
	DataJuicer-1.3B	LLaMA-3B	DataJuicer-1.3B	LLaMA-3B	DataJuicer-1.3B	LLaMA-3B
Coreset-Cent	31.36 $\pm$ 0.80 (1.00%)	34.81 $\pm$ 0.90 (0.01%)	33.27 $\pm$ 0.33 (0.50%)	35.48 $\pm$ 1.08 (1.00%)	33.27 $\pm$ 0.33 (0.50%)	35.48 $\pm$ 1.08 (1.00%)
FedAvg	22.08 $\pm$ 1.52 (100%)	27.88 $\pm$ 0.75 (100%)	32.30 $\pm$ 1.23 (100%)	34.27 $\pm$ 0.45 (100%)	33.38 $\pm$ 1.43 (100%)	33.95 $\pm$ 0.79 (100%)
FedPTuning	19.61 $\pm$ 2.71 (100%)	25.41 $\pm$ 1.14 (100%)	23.98 $\pm$ 3.23 (100%)	30.30 $\pm$ 1.16 (100%)	25.33 $\pm$ 2.48 (100%)	29.08 $\pm$ 1.33 (100%)
FedPrompt	6.04 $\pm$ 0.12 (100%)	8.95 $\pm$ 2.47 (100%)	32.73 $\pm$ 0.87 (100%)	24.50 $\pm$ 4.78 (100%)	32.51 $\pm$ 1.31 (100%)	23.94 $\pm$ 4.15 (100%)
FedIT-SGD	19.40 $\pm$ 1.83 (100%)	28.14 $\pm$ 0.85 (100%)	27.23 $\pm$ 0.68 (100%)	29.28 $\pm$ 0.50 (100%)	27.28 $\pm$ 1.35 (100%)	29.19 $\pm$ 0.89 (100%)
FlexLoRA	23.19 $\pm$ 2.14 (100%)	28.86 $\pm$ 0.55 (100%)	29.81 $\pm$ 1.06 (100%)	32.84 $\pm$ 0.99 (100%)	29.17 $\pm$ 1.35 (100%)	32.18 $\pm$ 1.28 (100%)
FedIT	22.30 $\pm$ 0.42 (100%)	28.13 $\pm$ 0.50 (100%)	30.80 $\pm$ 0.98 (100%)	33.23 $\pm$ 1.51 (100%)	30.97 $\pm$ 0.43 (100%)	33.68 $\pm$ 1.07 (100%)
Random	26.20 $\pm$ 1.71 (0.20%)	31.23 $\pm$ 1.37 (2.00%)	32.59 $\pm$ 0.10 (1.50%)	33.82 $\pm$ 0.82 (1.50%)	32.24 $\pm$ 0.43 (1.50%)	34.29 $\pm$ 0.85 (5.00%)
Perplexity	24.45 $\pm$ 0.77 (5.00%)	30.49 $\pm$ 0.21 (5.00%)	32.73 $\pm$ 0.15 (1.50%)	33.71 $\pm$ 0.51 (1.50%)	32.24 $\pm$ 0.22 (1.50%)	33.88 $\pm$ 0.34 (5.00%)
FedHDS	<b>26.64<math>\pm</math>0.79</b> (0.20%)	32.32 $\pm$ 0.92 (0.15%)	<b>33.38<math>\pm</math>0.40</b> (0.82%)	<b>35.40<math>\pm</math>0.78</b> (1.18%)	<b>33.70<math>\pm</math>0.19</b> (0.88%)	<b>35.79<math>\pm</math>0.43</b> (1.34%)
FedHDS-Turbo	25.93 $\pm$ 0.75 (0.23%)	<b>32.93<math>\pm</math>0.64</b> (0.22%)	33.28 $\pm$ 0.44 (1.31%)	35.01 $\pm$ 0.65 (1.26%)	33.52 $\pm$ 0.20 (1.19%)	35.42 $\pm$ 0.29 (1.28%)

Table 3: Comparisons on 1) client-side time (CTime) i.e., the time a client spends performing local computations during a round of FL, 2) the overall time consumption across all rounds (total time), and 3) the speedup ratio. Time is calculated with training using CPU+GPU due to limited GPU memory on edge devices.

Approach	DataJuicer-1.3B on NI			LLaMA-3B on NI			DataJuicer-1.3B on Dolly-15K			LLaMA-3B on Dolly-15K		
	CTime	Total Time	Speedup	CTime	Total Time	Speedup	CTime	Total Time	Speedup	CTime	Total Time	Speedup
FedIT	489S	8D9H 	1 $\times$	811S	13D21H 	1 $\times$	56.0S	9H20M 	1 $\times$	114S	19H6M 	1 $\times$
Random	13.2S	5H24M 	37.2 $\times$	35.3S	14H29M 	23.0 $\times$	6.5S	43M25S 	12.89 $\times$	25.2S	4H12M 	4.54 $\times$
Perplexity	46.6S	19H9M 	10.5 $\times$	85S	1D11H 	9.51 $\times$	8.9S	59M27S 	9.42 $\times$	27.5S	4H34M 	4.18 $\times$
FedHDS	27.5S	11H17M 	17.8 $\times$	40.7S	16H42M 	19.9 $\times$	5.2S	34M52S 	16.06 $\times$	14.6S	2H25M 	7.90 $\times$
FedHDS-Turbo	10.0S	4H7M 	48.8 $\times$	20.1S	8H15M 	40.4 $\times$	4.5S	29M42S 	18.86 $\times$	17.2S	2H52M 	6.66 $\times$

Dolly-15K when  $\alpha=0.5$  and  $\alpha=5.0$ , respectively, indicating that FedHDS performs better when client-side data has certain similarity. In a cross-device FL scenario with a large scale of clients, it is common for different clients to have similar data distributions. These results demonstrate the effectiveness of our approaches for improving generalization.

**Comparison to Coreset Baselines** Both FedHDS and FedHDS-Turbo outperform Random and Perplexity in 5 of the 6 scenarios. Only on NI with DataJuicer-1.3B, Random slightly outperforms FedHDS-Turbo. From Lin et al. (2024); Sachdeva et al. (2024), Random is a strong baseline as it preserves the data distribution. However, it is affected by data ratios (Cao et al., 2024), and determining the optimal data ratio requires extensive experimentation. Differently, FedHDS and FedHDS-Turbo can automatically determine an appropriate data ratio. Note that in 7 out of the 12 scenarios involving Random and Perplexity, the data ratios with the best Rouge-L are inspired by our approaches. Even so, our approaches outperform them in the vast majority of cases, highlighting the need for a well-designed data selection strategy.

The centralized method, Coreset-Cent, surpasses our approaches on the complex dataset, NI, indicating room for further improvement in FL methods.

### 5.3 Comparison on Time Efficiency

From Table 3, by reducing the number of local training steps, coreset methods improve efficiency over FedIT. FedHDS-Turbo achieves a higher speedup due to the fewer data samples required for better accuracy compared to other coreset FL baselines. Particularly, on NI with DataJuicer-1.3B, FedHDS-Turbo achieves a speedup of 48.8 $\times$  over FedIT. Since FedHDS-Turbo is comparable to FedHDS in accuracy, it may be more applicable. We provide a breakdown of time consumption in Appendix E.4 to clearly present the time consumption of each step in our approach.

### 5.4 Ablation Studies

**Two-Layer Selection** To clarify the contributions of intra- and inter-client selections, we build two methods: 1) FedHDS $\ddagger$ : Removes inter-client selection and groups data by last-layer features. 2) FedHDS $\dagger$ : Uses only intra-client selection, selecting one data sample closest to the centroid in each

Table 4: Rouge-L (%) comparisons for ablation studies, organized in the same manner of Table 2.

Approach	Natural Instructions (Meta Non-IID)		Dolly-15K ( $\alpha = 0.5$ )		Dolly-15K ( $\alpha = 5.0$ )	
	DataJuicer-1.3B	LLaMA-3B	DataJuicer-1.3B	LLaMA-3B	DataJuicer-1.3B	LLaMA-3B
FedHDS $\dagger$	25.45 $\pm$ 0.64 (1.62%)	28.77 $\pm$ 1.91 (3.86%)	31.82 $\pm$ 0.56 (3.71%)	33.66 $\pm$ 0.48 (4.25%)	32.12 $\pm$ 1.66 (3.64%)	33.81 $\pm$ 0.87 (4.32%)
FedHDS $\dagger$	24.48 $\pm$ 0.78 (0.83%)	32.27 $\pm$ 0.12 (1.51%)	32.52 $\pm$ 0.65 (2.93%)	34.11 $\pm$ 0.94 (3.97%)	32.79 $\pm$ 1.15 (2.99%)	33.98 $\pm$ 1.36 (4.33%)
FedHDS	26.64 $\pm$ 0.79 (0.20%)	32.32 $\pm$ 0.92 (0.15%)	<b>33.38 <math>\pm</math> 0.40</b> (0.82%)	<b>35.40 <math>\pm</math> 0.78</b> (1.18%)	<b>33.70 <math>\pm</math> 0.19</b> (0.88%)	<b>35.79 <math>\pm</math> 0.43</b> (1.34%)
w/i PCA	<b>27.84 <math>\pm</math> 0.99</b> (0.09%)	32.62 $\pm$ 0.62 (0.08%)	33.12 $\pm$ 0.10 (0.74%)	34.90 $\pm$ 0.52 (1.86%)	33.06 $\pm$ 0.23 (0.74%)	34.60 $\pm$ 1.08 (2.12%)
w/i KPCA	26.28 $\pm$ 3.93 (0.10%)	28.95 $\pm$ 0.92 (0.08%)	33.21 $\pm$ 0.77 (0.72%)	34.80 $\pm$ 0.95 (1.62%)	33.34 $\pm$ 0.36 (0.75%)	35.11 $\pm$ 0.27 (2.08%)
FedHDS-Turbo	25.93 $\pm$ 0.75 (0.23%)	<b>32.93 <math>\pm</math> 0.64</b> (0.22%)	33.28 $\pm$ 0.44 (1.31%)	35.01 $\pm$ 0.65 (1.26%)	33.52 $\pm$ 0.20 (1.19%)	35.42 $\pm$ 0.29 (1.28%)
w/i PCA	22.93 $\pm$ 2.96 (0.21%)	29.99 $\pm$ 0.78 (0.21%)	33.03 $\pm$ 0.26 (1.40%)	<b>35.35 <math>\pm</math> 0.65</b> (1.87%)	33.15 $\pm$ 0.21 (0.66%)	35.21 $\pm$ 0.54 (1.72%)
w/i KPCA	25.80 $\pm$ 2.78 (0.03%)	28.98 $\pm$ 2.33 (0.20%)	32.86 $\pm$ 1.08 (1.86%)	35.04 $\pm$ 0.73 (1.69%)	33.40 $\pm$ 0.41 (0.71%)	34.88 $\pm$ 1.25 (1.92%)

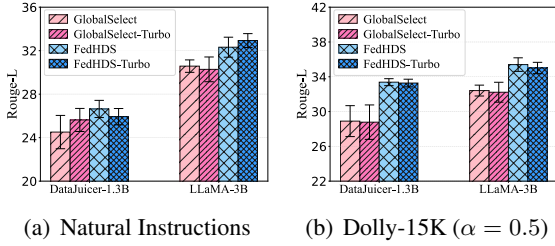


Figure 6: Performance of our approaches and methods that send data features to the server for global selection.

local group. From Table 4, intra- and inter-client selections vary in effectiveness across different scenarios. FedHDS outperforms FedHDS $\dagger$  in all scenarios, showing the efficacy of the two selections.

To emphasize the necessity of hierarchical selection, we construct *GlobalSelect*, which sends all fused features to the server for global clustering, and *GlobalSelect-Turbo* that extracts features with GPT-2. From Figure 6, global selection may cause poor accuracy, potentially because clustering on the entire large-scale dataset yields suboptimal results. Thus, it is necessary to select data hierarchically.

**Feature Fusion** We explore the impact of the dimensionality reduction algorithm by replacing t-SNE with PCA (Jolliffe, 2002) and Kernel PCA (Schölkopf et al., 1997). From Table 4, on the relatively simpler datasets (Dolly-15K), replacing t-SNE brings minimal differences. However, on NI, substituting t-SNE occasionally results in negative effects. Thus, in practical scenarios, we recommend t-SNE for more effective coreset selection.

## 5.5 Communication and Memory Costs

We provide the maximum memory cost and client-side per-round communication cost of our approaches. Besides transmitting LoRA adapters as FedIT, i.e., Comm. (Model), FedHDS additionally

Table 5: Per-round costs by 1.3B models (Dolly-15K).

	Comm. (Model)	Comm. (Features & Cluster Indices)	GPU Mem.
FedIT	12 MB	0	10.56 GB
FedHDS	12 MB	44 Bytes	9.40 GB
FedHDS-Turbo	12 MB	76 Bytes	9.32 GB

transmits 1) 2-dimensional cluster centroids and 2) indices of a few selected clusters. From Table 5, these bring negligible cost, i.e., just a few dozen bytes. Detailed calculations are in Appendix F.

FedHDS’s memory usage is similar to FedIT, with a slight reduction from filtering long samples. Nevertheless, FedHDS makes computation offloading feasible for on-device LLM tuning by significantly reducing the required data samples, while training on the full dataset with enabling offloading incurs a substantial time cost (Figure 2).

## 5.6 Studies on Convergence and Overfitting

To illustrate the convergence trends of these approaches, Figure 7 presents the convergence curves of FedHDS, FedHDS-Turbo, and the baseline methods using LLaMA-3B on Natural Instructions. The involved hyperparameters are aligned with those described in Section 5.1. It can be seen that these methods have nearly reached a convergent state by the 40th federated round, demonstrating that LLMs can be effectively tuned with a limited number of instruction data. Compared to approaches that use the full dataset, those relying on a data subset generally achieve a lower test loss. This is because tuning on the complete local dataset results in excessive local training steps, causing the LLM to overfit the local data and perform worse on unseen tasks, as discussed in Section 5.2.

To better demonstrate that training on a subset can alleviate the overfitting problem to some extent, we provide the training and test loss of FedHDS-



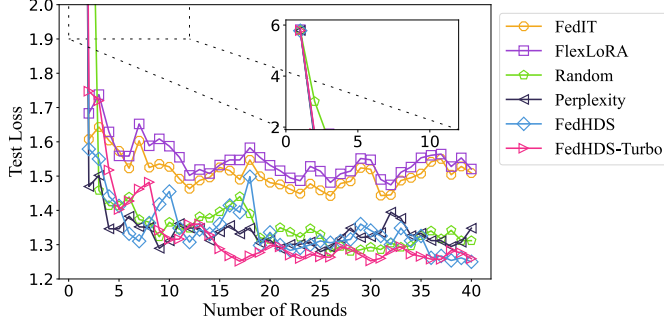


Figure 7: Convergence of the loss value on the test tasks obtained by FedHDS-Turbo and FedIT with LLaMA-3B on NI.

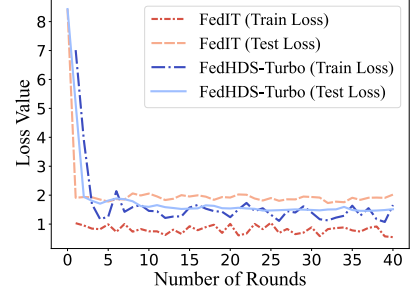


Figure 8: Convergence of training and test loss obtained by FedHDS-Turbo and FedIT with DataJuicer-1.3B on NI.

Turbo with DataJuicer-1.3B on Natural Instructions in Figure 8, together with those of the federated approach using the full dataset that achieves the best average accuracy, i.e., FedIT. As shown in Figure 8, both the training and test loss of FedHDS-Turbo stably decrease. In contrast, the test loss of FedIT quickly stops decreasing while the training loss is still reducing to lower values. Finally, the test loss of FedHDS-Turbo is lower than that of FedIT, causing FedHDS-Turbo to perform better than FedIT, as shown in Table 2.

From the above, we can conclude that FedHDS and FedHDS-Turbo exhibit stable convergence and perform on par with federated approaches that use the full dataset. Additionally, by reducing the amount of data used for instruction tuning, they mitigate overfitting to local data to some extent, leading to higher instruction-tuning accuracy compared to the baseline methods.

### 5.7 Performance with Differential Privacy

FedHDS can perform well with noise sampling variance no greater than 0.1, which improves the privacy of client-side data. Limited by space, detailed experiments on this are left in Appendix E.3.

### 5.8 Performance in Various FL Scenarios

FedHDS outperforms coreset FL baselines with different active client ratios, showing its applicability in various FL scenarios. We demonstrate this with experimental results in Appendix E.5.

## 6 Conclusion

Existing federated instruction tuning methods for LLMs typically train LLMs using all local data, causing significant computation cost and overfitting to local data. This work pioneers an exploration into federated data-efficient instruction tuning, and

proposes FedHDS, a coreset selection approach that solves both intra- and inter-client data redundancy. It fuses data features of varying abstraction levels obtained from different Transformer layers for better data representation. Extensive experiments involving various datasets, LLMs and non-IIDness demonstrate that FedHDS enhances the data efficiency and Rouge-L on unseen tasks over existing federated tuning methods.

## 7 Limitations

Although our approach improves data efficiency and generalization to unseen tasks in FL for LLM fine-tuning, it still has certain limitations. For example, it only selects data based on representativeness but overlooks data quality. Since domain divisions are usually implicit, low-quality data samples may also be treated as a separate domain. In this case, FedHDS may select low-quality data. Therefore, incorporating quality-based filtering mechanisms may help further improve our approach.

## Acknowledgments

This research is supported in part by the National Science Foundation of China under Grants 62125206, the National Key Research and Development Program of China under Grant 2022YFB4500100, and the Zhejiang Provincial Natural Science Foundation of China under Grant LD24F020014. This research is also supported in part by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

## References

- Sara Babakniya, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy, and Salman Avestimehr. 2023. SLoRA: Federated parameter efficient fine-tuning of language models. *arXiv preprint arXiv:2308.06522*.
- Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. 2024. Federated fine-tuning of large language models under heterogeneous tasks and client resources. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27.
- Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer.
- Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. 2024. Instruction mining: Instruction data selection for tuning large language models. In *First Conference on Language Modeling*.
- Tianshi Che, Ji Liu, Yang Zhou, Jiaxiang Ren, Jiwen Zhou, Victor Sheng, Huaiyu Dai, and Dejing Dou. 2023. Federated learning of large language models with parameter-efficient prompt tuning and adaptive optimization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7871–7888.
- Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesun Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. Data-juicer: A one-stop data processing system for large language models. In *Companion of the 2024 International Conference on Management of Data, SIGMOD*, pages 120–134. ACM.
- Hao Chen, Yiming Zhang, Qi Zhang, Hantao Yang, Xiaomeng Hu, Xuetao Ma, Yifan Yanggong, and Junbo Zhao. 2023. Maybe only 0.5% data is needed: A preliminary exploration of low training data instruction tuning. *arXiv preprint arXiv:2305.09246*.
- Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. [Free dolly: Introducing the world’s first truly open instruction-tuned LLM](#).
- Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*.
- David L. Davies and Donald W. Bouldin. 1979. [A cluster separation measure](#). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.
- Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.
- Wenqi Fan, Yajuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501. ACM.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR*. OpenReview.net.
- Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. 2022b. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54(11s):1–37.
- Ian T Jolliffe. 2002. *Principal component analysis for special types of data*. Springer.
- Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*.
- Weirui Kuang, Bingchen Qian, Zitao Li, Daoyuan Chen, Dawei Gao, Xuchen Pan, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. Federatedscope-llm: A comprehensive package for fine-tuning large language models in federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5260–5271.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 3045–3059. Association for Computational Linguistics.
- Ming Li, Lichang Chen, Jiuhai Chen, Shwai He, Jiuxiang Gu, and Tianyi Zhou. 2024a. Selective reflection-tuning: Student-selected data recycling for LLM instruction-tuning. In *Findings of the Association for Computational Linguistics, ACL 2024*, pages 16189–16211.
- Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. 2024b. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL*,

- pages 14255–14273. Association for Computational Linguistics.
- Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024c. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, NAACL 2024, pages 7602–7635. Association for Computational Linguistics.
- Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations, ICLR*. OpenReview.net.
- Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. 2024. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 365–374. ACM.
- Zhenqing Ling, Daoyuan Chen, Liuyi Yao, Yaliang Li, and Ying Shen. 2024. On the convergence of zeroth-order federated tuning for large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1827–1838. ACM.
- Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. GPT understands, too. *AI Open*, pages 208–215.
- Keming Lu, Hongyi Yuan, Zheng Yuan, Runji Lin, Junyang Lin, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. # instag: Instruction tagging for analyzing supervised fine-tuning of large language models. In *The Twelfth International Conference on Learning Representations*. OpenReview.net.
- Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. <https://github.com/huggingface/peft>.
- Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *the International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1273–1282.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.
- Zhen Qin, Daoyuan Chen, Bingchen Qian, Bolin Ding, Yaliang Li, and Shuiguang Deng. 2024a. Federated full-parameter tuning of billion-sized language models with communication cost under 18 kilobytes. In *Proceedings of the 41st International Conference on Machine Learning, ICML*, volume 235, pages 41473–41497.
- Zhen Qin, Daoyuan Chen, Wenhao Zhang, Liuyi Yao, Yilun Huang, Bolin Ding, Yaliang Li, and Shuiguang Deng. 2024b. The synergy between data and multi-modal large language models: A survey from co-development perspective. *arXiv preprint arXiv:2407.08583*.
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.
- Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.
- Noveen Sachdeva, Benjamin Coleman, Wang-Cheng Kang, Jianmo Ni, Lichan Hong, Ed H Chi, James Caverlee, Julian McAuley, and Derek Zhiyuan Cheng. 2024. How to train data-efficient llms. *arXiv preprint arXiv:2402.09668*.
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1997. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer.
- Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. 2024. Improving lora in privacy-preserving federated learning. In *The Twelfth International Conference on Learning Representations, ICLR*. OpenReview.net.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An instruction-following llama model. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca).
- Laurens van der Maaten. 2013. Barnes-hut-sne. In *1st International Conference on Learning Representations, ICLR*.
- Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11):2579–2605.
- Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. 2022. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv preprint arXiv:2211.04325*.
- Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed.*, Cham: Springer International Publishing, 10(3152676):10–5555.

- Jiahao Wang, Bolin Zhang, Qianlong Du, Jiajun Zhang, and Dianhui Chu. 2024. A survey on data selection for llm instruction tuning. *arXiv preprint arXiv:2402.05123*.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-Instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508.
- Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 5085–5109.
- Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR*. OpenReview.net.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.
- Feijie Wu, Zitao Li, Yaliang Li, Bolin Ding, and Jing Gao. 2024. Fedbiot: Llm local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3345–3355. ACM.
- Shengguang Wu, Keming Lu, Benfeng Xu, Junyang Lin, Qi Su, and Chang Zhou. 2023. Self-evolved diverse data sampling for efficient instruction tuning. *arXiv preprint arXiv:2311.08182*.
- Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. 2024. FwdLLM: Efficient federated finetuning of large language models with perturbed inferences. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC*, pages 579–596.
- Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. 2024. Data-centric artificial intelligence: A survey. *ACM Computing Surveys*, 57(5).
- Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Tong Yu, Guoyin Wang, and Yiran Chen. 2024. Towards building the federatedgpt: Federated instruction tuning. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6915–6919.
- Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. 2023. FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL*, pages 9963–9977.
- Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. LIMA: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021.



## Appendix

We provide more discussions and experiments of this work and organize them as follows:

### Table of Contents

<b>A Detailed Algorithm</b>	<b>13</b>
<b>B Convergence Analysis</b>	<b>13</b>
<b>C Analysis on Speedup Ratio</b>	<b>15</b>
<b>D Reproducibility</b>	<b>15</b>
D.1 Experimental Environments for Accuracy Evaluation . . . . .	15
D.2 Experimental Environments for Memory Footprint and Efficiency Statistics . . . . .	15
D.3 Detailed Hyperparameters . . .	15
D.4 Detailed Descriptions on Datasets	16
<b>E Additional Experiments</b>	<b>16</b>
E.1 Evaluation of Features from Different Transformer Layers . . .	16
E.2 Virtualization of Data Features	16
E.3 Performance with Differential Privacy . . . . .	17
E.4 Breakdown of Time Consumption	17
E.5 Performance in Various FL Scenarios . . . . .	18
<b>F Detailed Calculation of Communication Overhead</b>	<b>19</b>

## A Detailed Algorithm

To facilitate a better understanding of each step of the proposed approach, we provide Algorithm 1 to explain how FedHDS selects the coresets for activate clients in each round  $r$ , where lines 2~4 and 7~9 are performed individually by each client, and lines 1 and 5~6 are performed by the server.

## B Convergence Analysis

FedHDS shares the same global objective with FedAvg (FedIT) defined in Eq. (1). The essential difference between FedHDS and FedAvg (FedIT) is that it optimizes only on a subset of the original data, as illustrated by the difference between Eq. (11) and Eq. (2). Given the global objective

of FedHDS defined in Eq. (1), we have the local objective of each client  $i$  during local training as:

$$\min_{\mathbf{w}} f_i(\mathbf{w}) \triangleq \frac{1}{|\tilde{\mathcal{D}}_i|} \sum_{i=1}^{|\tilde{\mathcal{D}}_i|} \mathbb{E}_{\mathbf{x} \sim \tilde{\mathcal{D}}_i} [\mathcal{L}(\mathbf{w}; \mathbf{x})]. \quad (13)$$

Following Li et al. (2020), we first make the following assumptions:

**Assumption 1.** *The local objective of each client  $i$  is  $L$ -smooth, i.e.,  $f_i(\mathbf{v}) \leq f_i(\mathbf{w}) + (\mathbf{v} - \mathbf{w})^T \nabla f_i(\mathbf{w}) + \frac{L}{2} \|\mathbf{v} - \mathbf{w}\|_2^2$ ,  $\forall \mathbf{v} \in \mathbb{R}^d$ ,  $\forall \mathbf{w} \in \mathbb{R}^d$ .*

**Assumption 2.** *The local objective of each client  $i$  is  $\mu$ -convex, i.e.,  $f_i(\mathbf{v}) \geq f_i(\mathbf{w}) + (\mathbf{v} - \mathbf{w})^T \nabla f_i(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{v} - \mathbf{w}\|_2^2$ ,  $\forall \mathbf{v} \in \mathbb{R}^d$ ,  $\forall \mathbf{w} \in \mathbb{R}^d$ .*

**Assumption 3.** *The variance of the stochastic gradients across each client is bounded, i.e.,  $\mathbb{E} \|\nabla f_i(\mathbf{w}_{i,\tau}, \mathbf{x}) - \nabla f_i(\mathbf{w}_{i,\tau})\| \leq \sigma_i^2$ , where  $\mathbf{w}_{i,\tau}$  denotes the model parameters of the  $i$ -th client after  $\tau$  steps of updates.*

**Assumption 4.** *The expected squared norm of the stochastic gradients stays within a uniform bound, i.e.,  $\mathbb{E} \|\nabla f_i(\mathbf{w}_{i,\tau}, \mathbf{x})\|^2 \leq G^2$ .*

Existing works also make similar assumptions on the convergence analysis of federated tuning of LLMs (Ling et al., 2024). We also mildly assume that in each round, there are  $K$  clients on average that will submit their tuned models to the server, and each of them performs  $E$  steps of local training to update their local models.

**Theorem 2.** *Let Assumptions 1~4 hold,  $\mathbf{w}^*$  be the optimal global model,  $\kappa = \frac{L}{\mu}$ ,  $\gamma = \max\{8\kappa, E\}$ ,  $B = \sum_{i=1}^N \lambda_i^2 \sigma_i^2 + 6LT + 8(E-1)^2 G^2$ ,  $C = \frac{4}{K} E^2 G^2$ , and  $\mathbf{E} = \mathbb{E}[f(\mathbf{w}_T)] - f(\mathbf{w}^*)$ , after  $T$  iterations, we have*

$$\mathbf{E} \leq \frac{2\kappa}{\gamma + T} \left( \frac{B + C}{\mu} + 2L \|\mathbf{w}^0 - \mathbf{w}^*\|^2 \right). \quad (14)$$

*Proof.* In fact, FedHDS is based on FedAvg and performs downsampling on local data, thereby affecting the number of local training iterations. With appropriate variable substitution, the convergence of FedHDS can be derived from the proof process in the work of Li et al. (2020).  $\square$

Based on Theorem 2, FedHDS has a convergence guarantee. Compared to FedIT, which directly adopts the training processes of FedAvg,

---

**Algorithm 1: Processes of Data Selection in FedHDS in each round  $r$  of FL.**


---

**Input:**  $\mathbb{M}_r, \{\mathcal{D}_1, \dots, \mathcal{D}_{|\mathbb{M}_r|}\}$ .

**Output:** The selected coreset  $\tilde{\mathcal{D}}_i$  for each client  $i$  active in this round, denoted as  $\{\tilde{\mathcal{D}}_1, \dots, \tilde{\mathcal{D}}_{|\mathbb{M}_r|}\}$ .

```

1 Initialize a list  $\mathbb{C}$  to stage the received centroids
2 for  $i = 1, 2, \dots, |\mathbb{M}_r|$  do
3    $\{\mathbf{c}_1, \mathbf{c}_2, \dots\} = \text{IntraClientSelection}(\mathbf{w}, \mathcal{D}_i)$ 
4    $\mathbb{C} = \mathbb{C} \cup \{\mathbf{c}_1, \mathbf{c}_2, \dots\}$  // ④ in Figure 3
5  $\mathbb{G}^{\text{selected}} = \text{InterClientSelection}(\mathbb{C})$ 
6 send indices of selected groups to corresponding clients // ⑥ in Figure 3
7 for  $i = 1, 2, \dots, |\mathbb{M}_r|$  do
8    $\tilde{\mathcal{D}}_i = \left\{ \mathbf{x} \mid q = \arg \min_{\mathbf{x} \in \mathcal{G}_j} \|\mathbf{x} - \mathbf{c}_j\| \right\}^{\mathcal{G}_j \in \mathbb{G}_i, \mathcal{G}_j \in \mathbb{G}^{\text{selected}}}$  // ⑦ in Figure 3
9 return the selected subset  $\tilde{\mathcal{D}}_i$  for each client  $i \in \mathbb{M}_r$ .

10 Function IntraClientSelection( $\mathbf{w}, \mathcal{D}$ ):
11   for  $j = 1, 2, \dots, |\mathcal{D}|$  do
12     extract features of  $\mathbf{x}_j$  by each layer of  $\mathbf{w}$ , denoted by  $\mathbf{h}_j = [\mathbf{h}_j^{1,-1}, \mathbf{h}_j^{2,-1}, \dots, \mathbf{h}_j^{l,-1}]$ 
13     // ① in Figure 3
14     reduce dimensionality of  $\mathbb{H} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{|\mathcal{D}|}\}$  as
15      $\{\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \dots, \tilde{\mathbf{h}}_{|\mathcal{D}|}\} = P(\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{|\mathcal{D}|}\})$ , obtaining the fused features // ② in Figure 3
16     cluster data in  $\mathcal{D}_i$  based on their fused features, as
17      $\{\mathcal{G}_1, \mathcal{G}_2, \dots\} = \text{HDBSCAN}(\{\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \dots, \tilde{\mathbf{h}}_{|\mathcal{D}|}\})$ , where each  $\mathcal{G}_j$  corresponds to an
18     approximate centroid  $\mathbf{c}_j$  // ③ in Figure 3
19   return  $\{\mathbf{c}_1, \mathbf{c}_2, \dots\}$ 

16 Function InterClientSelection( $\{\mathbf{c}_1, \mathbf{c}_2, \dots\}$ ):
17   perform HDBSCAN to partition received  $\{\mathbf{c}_1, \mathbf{c}_2, \dots\}$  into several groups  $\{\mathcal{G}_1^{\text{II}}, \mathcal{G}_2^{\text{II}}, \dots\}$ 
18   // ⑤ in Figure 3
19   initialize a list  $\mathbb{G}^{\text{selected}}$ 
20   for  $j = 1, 2, \dots, |\{\mathcal{G}_1^{\text{II}}, \mathcal{G}_2^{\text{II}}, \dots\}|$  do
21      $s = \arg \min_{\mathbf{c} \in \mathcal{G}_j^{\text{II}}} \|\mathbf{c} - \mathbf{c}_j^{\text{II}}\|$ 
22     add  $\mathcal{G}_s$  into  $\mathbb{G}^{\text{selected}}$ 
23 return  $\mathbb{G}^{\text{selected}}$ 

```

---

the convergence rate of FedHDS on the training set theoretically has certain disadvantages, which have been experimentally demonstrated in Figure 7. However, the advantage of FedHDS is its ability to handle overfitting to local data. As shown in both Figures 7 and 8, FedHDS and FedHDS-Turbo significantly outperform FedIT in terms of test loss. Therefore, FedHDS and FedHDS-Turbo achieve better Rouge-L on held-out tasks than FedIT, as presented in Table 2.

## C Analysis on Speedup Ratio

In this section, we provide a brief analysis of the speedup achievable by our approaches to better understand their effectiveness in acceleration based on numerical results in Table 3.

For a client with  $M$  data samples, the time complexity of performing t-SNE with Barnes-Hut implementation is  $\mathcal{O}(M \log M)$ , and that of performing HDBSCAN is  $\mathcal{O}(M \log M) \sim \mathcal{O}(M^2)$  based on data sparsity (we adopt the worst complexity). Assume that  $\nu$  and  $\epsilon$  are the scale constants between the actual time consumption and complexity of t-SNE and HDBSCAN, respectively, and  $\xi$  and  $\Xi$  denote the time consumption (e.g., seconds) of performing one-step inference and training with one data sample, respectively. Assuming that the proportion of training data that FedHDS can filter out is  $\varsigma$ , the time consumption of a client by conducting LLM instruction tuning with FedHDS is

$$\nu \cdot M \log M + \epsilon \cdot M^2 + \xi \cdot M + \Xi(1 - \varsigma)M, \quad (15)$$

while that of FedIT, the approach using full data, is  $\Xi \cdot M$ . Therefore, the speeding-up ratio can be formalized as:

$$\frac{\Xi \cdot M}{\nu \cdot M \log M + \epsilon \cdot M^2 + \xi \cdot M + \Xi(1 - \varsigma)M}. \quad (16)$$

For these notations:

- Generally,  $\Xi > \xi$  by several times.
- From the experiments on Dolly-15K, t-SNE on 200 samples takes 0.5 seconds, and HDBSCAN takes only 0.003 seconds. Based on these statistics,  $\nu$  could be in the order of  $10^{-4}$ , and  $\epsilon$  could be in the order of  $10^{-8}$ . Therefore,  $\xi \gg \nu$  and  $\xi \gg \epsilon$ .
- $\varsigma$  can exceed 99% on Natural Instructions.

Therefore, the speedup achieved by our approaches is significant, as experimentally demonstrated in Table 3. Besides, the inference can be significantly accelerated (by reducing  $\xi$ ), leading to a considerable improvement, e.g., FedHDS-Turbo is significantly faster than FedHDS.

## D Reproducibility

### D.1 Experimental Environments for Accuracy Evaluation

We implement these approaches mentioned above with PyTorch (Paszke et al., 2019) 2.0.1, Transformers (Wolf et al., 2020) 4.31.0, scikit-learn 1.5.1, PEFT (Mangrulkar et al., 2022) 0.4.0, and hdbscan 0.8.37. Numerical experiments in Tables 2 and 4 are performed on platforms equipped with NVIDIA A100 or NVIDIA A6000 GPUs, installed with Python 3.10 and CUDA 12.4. Efficiency results in Table 3 are obtained on a platform with an NVIDIA A6000 GPU, installed with Python 3.10, CUDA 12.4 and DeepSpeed 0.15.2.

### D.2 Experimental Environments for Memory Footprint and Efficiency Statistics

The memory footprint and time consumption in Figures 1 and 2 are measured with a maximum token list length of 1,024 where excessively long data will be truncated. The adopted platform is equipped with an NVIDIA A6000 GPU, installed with Python 3.10, CUDA 12.4 and DeepSpeed 0.15.2. For the memory footprint, the 95th percentile is calculated. The selected two GPUs are based on the most popular desktop and laptop GPUs identified from the Steam Hardware Survey (Dec 2024).

### D.3 Detailed Hyperparameters

In approaches involving the LoRA adapter, i.e., FedHDS, FedHDS-Turbo, FedIT and FlexLoRA, the adapters are configured with the same hyperparameter settings, i.e., rank, alpha and dropout of LoRA adapters are set to 8, 16 and 0.05 for FedHDS and FedHDS-Turbo, respectively. Note that although FlexLoRA supports heterogeneous-rank LoRA adapters, we adopt a homogeneous-rank setting to ensure a fair comparison. Coreset-based methods perform 60 rounds of FL on Dolly-15K with LLaMA-3B, and 40 rounds for other scenarios, where the local training is performed on the coreset for one epoch with Adam optimizer. The learning rate and number of rounds for federated

approaches using full data are aligned with those in the work of [Qin et al. \(2024a\)](#).

We perform a preliminary hyperparameter search for federated approaches using coresets, and adopt the advantageous settings for each approach in each scenario for large-scale experiments. Specifically, we first search the learning rate in  $\{3 \times 10^{-4}, 1 \times 10^{-4}, 3 \times 10^{-5}\}$ . Then, for Random and Perplexity, we search the ratio of data samples in the final selected subsets to the full data samples in  $\{0.2\%, 1.5\%, 2\%, 5\%\}$ . Note that the thresholds of 0.2% and 1.5% are inspired by the proportion automatically obtained by FedHDS and FedHDS-Turbo. Considering the importance of an appropriate data ratio ([Cao et al., 2024](#)), these two baselines have benefited to some extent from the data proportion provided by FedHDS and FedHDS-Turbo. The finally adopted values for Tables 2 and 4 as follows: All the federated approaches with coresets adopt a learning rate  $\eta$  of  $3 \times 10^{-5}$  on Dolly-15K. On Natural Instructions, FedHDS and FedHDS-Turbo adopt  $\eta = 3 \times 10^{-4}$ , Random adopt  $\eta = 3 \times 10^{-4}$  with DataJuicer-1.3B and  $\eta = 3 \times 10^{-5}$  with LLaMA-3B, Perplexity adopt  $\eta = 3 \times 10^{-5}$  with DataJuicer-1.3B and  $\eta = 3 \times 10^{-4}$  with LLaMA-3B.

For Random and Perplexity, we searched for the optimal data ratio within  $\{0.2\%, 1.5\%, 2\%, 5\%\}$ , where 0.2% and 1.5% are inspired by those automatically obtained by FedHDS and FedHDS-Turbo. Our approaches adopt learning rate  $\eta = 3 \times 10^{-5}$  on Dolly-15K and  $\eta = 3 \times 10^{-4}$  on NI. FedHDS and FedHDS-Turbo apply HDBSCAN separately in both intra-client and inter-client selection. During intra-client selection, the minimum cluster of HDBSCAN is set to the default value, i.e., 5 for Natural Instructions and 2 for Dolly-15K, considering the relatively small scale of Dolly-15K. During inter-client selection, the minimum cluster of HDBSCAN is uniformly set to 2.

#### D.4 Detailed Descriptions on Datasets

This work adopts the same data preprocessing as reported in ([Qin et al., 2024a](#)). Following [Zhang et al. \(2024\)](#); [Qin et al. \(2024a\)](#), we adopt the prompt template from Alpaca ([Taori et al., 2023](#)).

Natural Instructions includes a large collection of tasks with natural language instructions. We adopt the dataset versioned by v2.8 and the default split, which includes 756 tasks for training and 119 tasks for testing, each with a distinct task definition. Considering the scale of the dataset,

experiments on it are conducted on a randomly sampled subset, containing 20% of the data instances for each training task and 2% for each test task. After the subsampling, each training task with at least 20 data instances is assigned to a unique client. After the above preprocessing, a federated scenario with 738 clients is formed, where the test tasks remain on the server for the held-out evaluation of the tuned LLMs.

Dolly-15K contains 15,015 data samples corresponding to 8 tasks, with the 1,188 data samples from the last task used for testing. The data from the remaining seven tasks is distributed to 200 clients for training, with each task labeled according to its respective task and distributed according to a Dirichlet distribution. We partition data samples to clients via Dirichlet distribution based on the category attribute of each data sample.

After the last FL round is finished, Rouge-L scores are evaluated using the tuned LLM after the final FL round. These scores are calculated based on the data samples in evaluation tasks, with the responses as the ground-truth labels.

## E Additional Experiments

### E.1 Evaluation of Features from Different Transformer Layers

To better demonstrate the statement that the last layer is not universally optimal, and no single layer excels across all metrics in Section 4.2, we conduct evaluations with more LLMs and more metrics including Silhouette Coefficient ([Rousseeuw, 1987](#)) and Davies-Bouldin score ([Davies and Bouldin, 1979](#)), following the experimental setups aligned to that of Figure 4. The additional experimental results in Figure 9 further demonstrate the above statement.

### E.2 Virtualization of Data Features

In Section 4.3, we visualize the features obtained from the last Transformer layer and the fused features with DataJuicer-1.3B on Dolly-15K as an example, as presented in Figure 5. In order to illustrate the differences between fused features and the final layer features in more scenarios, we supplemented the visualizations using different LLMs in various scenarios. As shown in Figure 10, the fused features create more distinct boundaries among clients. Thus, these fused features can be utilized to more effectively distinguish between data samples than existing solutions.



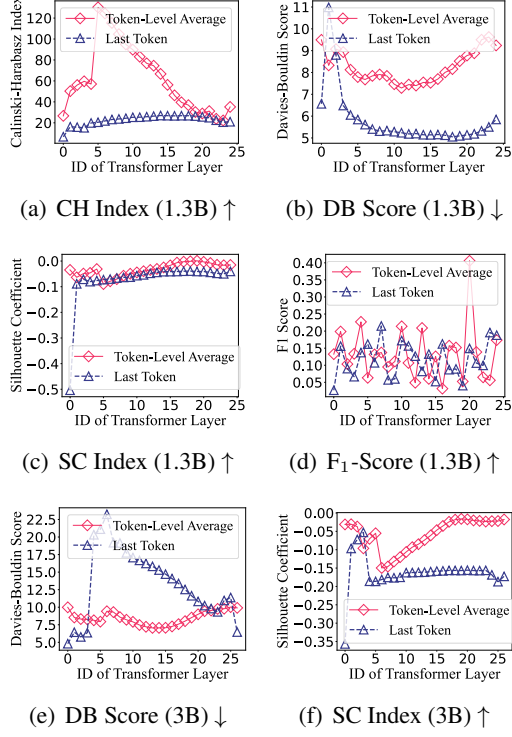
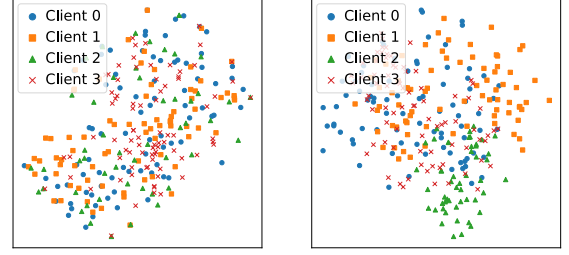


Figure 9: Evaluations of clustered data groups based on features from different Transformer layers, obtained in a centralized scenario with different LLMs on Dolly-15K.

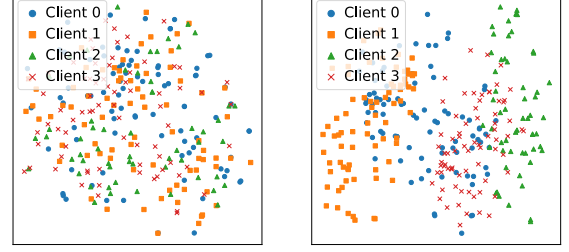
### E.3 Performance with Differential Privacy

Despite that 1) the feature dimensions in our approach are significantly lower compared to the original token-level hidden states, which often have thousands of dimensions, and 2) the shared centroids do not correspond to real data samples, FedHDS still provides additional information compared to the standard FL paradigm (McMahan et al., 2017), such as the spatial distribution of client-side centroids. As discussed in Section 4.5, scaling the elements of the shared centroids to the range  $[-1, 1]$  and then adding Gaussian noise can enhance the privacy protection of the proposed approaches.

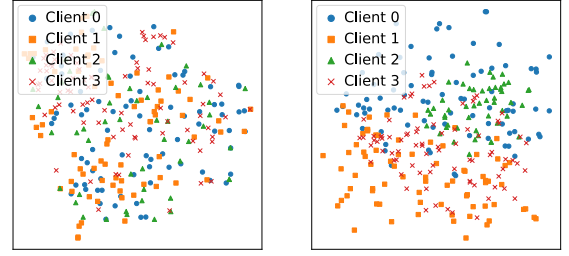
To clarify the impact of adding Gaussian noise, we present example experiments by FedHDS-Turbo on Dolly-15K with DataJuicer-1.3B, adjusting the power of the noise by varying its variance. As presented in Figure 11, when the noise variance is no greater than 0.1, FedHDS-Turbo still outperforms Random. With the noise level increasing further, the performance of FedHDS-Turbo gradually degrades to a level comparable to that of Random as reported in Table 2.



(a) Features by the last Transformer layer with LLaMA-3B in Dolly-15K ( $\alpha=0.5$ ). (b) Fused features by FedHDS with LLaMA-3B in Dolly-15K ( $\alpha=0.5$ ).



(c) Features by the last Transformer layer with LLaMA-3B in Dolly-15K ( $\alpha=5.0$ ). (d) Fused features by FedHDS with LLaMA-3B in Dolly-15K ( $\alpha=5.0$ ).



(e) Features by the last Transformer layer with DataJuicer-1.3B in Dolly-15K ( $\alpha=0.5$ ). (f) Fused features by FedHDS with DataJuicer-1.3B in Dolly-15K ( $\alpha=0.5$ ).

Figure 10: Visualization of features from the last Transformer layer and fused features by FedHDS.

### E.4 Breakdown of Time Consumption

To illustrate the time efficiency of our approach, we provide a detailed breakdown of the time taken by each step, recorded on Natural Instructions (Table 6) and Dolly-15K (Table 7), respectively. From these results, we have the following observations:

- For FedHDS, the time consumption is usually dominated by feature extraction, due to the relatively high time cost of performing inference with an LLM on all local data samples.
- For FedHDS-Turbo, due to its faster feature extraction than FedHDS, its time consumption is mainly caused by training, feature extraction, and feature fusion.

Table 6: Breakdown of time consumption for all steps involving with FedHDS and FedHDS-Turbo **on Natural Instructions**. By default, this table shows the time spent per FL round for each client, with the total time across all clients and rounds in parentheses.

	DataJuicer-1.3B on NI	LLaMA-3B on NI
FedHDS	All Steps: 27.5S (11H17M) Training: 0.2S (3M55S) Feature Extraction: 24.8S (10H11M) Feature Fusion: 2.5S (1H1M) Intra-Client Clustering: $6.7 \times 10^{-3}$ S (9.9S) Inter-Client Clustering: $4.3 \times 10^{-3}$ S (0.2S)	All Steps: 40.7S (16H42M) Training: 6.8S (2H47M) Feature Extraction: 30.9S (12H42M) Feature Fusion: 3.0S (1H13M) Intra-Client Clustering: $6.6 \times 10^{-3}$ S (9.8S) Inter-Client Clustering: $3.6 \times 10^{-3}$ S (0.1S)
FedHDS-Turbo	All Steps: 10.0S (4H7M) Training: 1.4S (34M12S) Feature Extraction: 6.6S (2H43M) Feature Fusion: 2.0S (49M26S) Intra-Client Clustering: $6.7 \times 10^{-3}$ S (9.9S) Inter-Client Clustering: $4.3 \times 10^{-3}$ S (0.2S)	All Steps: 20.1S (8H15M) Training: 11.4S (4H40M) Feature Extraction: 6.7S (2H46M) Feature Fusion: 2.0S (48M20S) Intra-Client Clustering: $6.6 \times 10^{-3}$ S (9.7S) Inter-Client Clustering: $4.6 \times 10^{-3}$ S (0.2S)

Table 7: Breakdown of time consumption for all steps involving with FedHDS and FedHDS-Turbo **on Dolly-15K**. By default, this table shows the time spent per FL round for each client, with the total time across all clients and rounds in parentheses.

	DataJuicer-1.3B on Dolly-15K	LLaMA-3B on Dolly-15K
FedHDS	All Steps: 5.2S (34M52S) Training: 2.0S (13M23S) Feature Extraction: 3.0S (19M42S) Feature Fusion: 0.3S (1M46S) Intra-Client Clustering: $1.6 \times 10^{-3}$ S (0.6S) Inter-Client Clustering: $8.5 \times 10^{-4}$ S ( $3.4 \times 10^{-2}$ S)	All Steps: 14.6S (2H25M) Training: 10.2S (1H41M) Feature Extraction: 4.1S (40M37S) Feature Fusion: 0.3S (3M21S) Intra-Client Clustering: $1.7 \times 10^{-3}$ S (1.0S) Inter-Client Clustering: $1.0 \times 10^{-3}$ S ( $6.2 \times 10^{-2}$ S)
FedHDS-Turbo	All Steps: 4.5S (29M42S) Training: 3.6S (23M45S) Feature Extraction: 0.7S (4M40S) Feature Fusion: 0.2S (1M15S) Intra-Client Clustering: $1.6 \times 10^{-3}$ S (0.6S) Inter-Client Clustering: $1.0 \times 10^{-3}$ S ( $4.0 \times 10^{-2}$ S)	All Steps: 17.2S (2H52M) Training: 16.3S (2H43M) Feature Extraction: 0.7S (6M54S) Feature Fusion: 0.2S (1M49S) Intra-Client Clustering: $1.8 \times 10^{-3}$ S (1.1S) Inter-Client Clustering: $1.7 \times 10^{-3}$ S (0.1S)

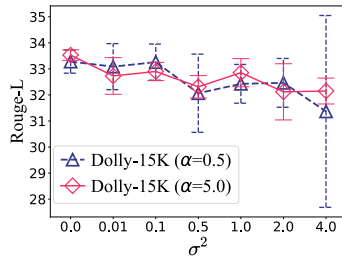


Figure 11: Effects of adding DP noise to FedHDS-Turbo (DataJuicer-1.3B).

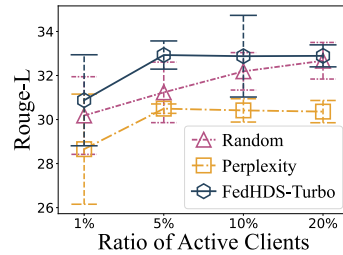


Figure 12: Rouge-L with different active ratio on NI with LLaMA-3B.

### E.5 Performance in Various FL Scenarios

The ratio of active clients in each round of FL affects the number of centroids sent to the server, potentially affecting the effectiveness of inter-client selection. We test approaches with coresets in varying proportions of active clients in each round. As shown in Figure 12, FedHDS-Turbo consistently outperforms Random and Perplexity across varying active client ratios. When the active client ratio is low (1%), all approaches perform unsatisfacto-

rily, where the effectiveness of intra-client selection in FedHDS-Turbo may suffer from insufficient centroids. With an increasing active client ratio, Random demonstrates a robust growth trend, although it still lags behind FedHDS-Turbo. This improvement is likely due to when there are more active clients in each round, the randomly sampled data instances follow a distribution more aligned to the global data distribution. Considering that clients typically participate in FL with a relatively

low active ratio in each round in cross-device FL scenarios (McMahan et al., 2017; Qin et al., 2024a; Xu et al., 2024), FedHDS-Turbo is more suitable than FedHDS for cross-device settings in terms of both accuracy and efficiency.

## F Detailed Calculation of Communication Overhead

Apart from the transmission of model parameters as other federated instruction tuning methods, FedHDS additionally transmits data features of cluster centers and the indices of the selected clusters for data selection. Assuming for a client, there are  $\rho$  clusters after intra-client selection. The client sends the fused features of the  $\rho$  data samples closest to these cluster centers, as:

$$[[e_{1,1}, e_{1,2}], [e_{2,1}, e_{2,2}], \dots, [e_{\rho,1}, e_{\rho,2}]], \quad (17)$$

where  $e$  denotes a floating number, and  $[]$  denotes an array. Then, after the inter-client selection, the server returns indices of selected clusters to corresponding clients, which are just a few integers:

$$[\text{ClusterID}_1, \text{ClusterID}_2, \dots]. \quad (18)$$

Therefore, compared to the widely recognized baseline method FedIT, the additional communication overhead of our approach is negligible.