RA-LoRA: Rank-Aware Aggregation for Low-Rank Adaptation with Federated Fine-Tuning

Anonymous ACL submission

Abstract

Federated fine-tuning of foundation models is impeded by the need to communicate billions of parameters. Low-rank adaptation (LoRA) alleviates this by updating only compact adapter matrices. However, varying client device capabilities lead to different adapter ranks, causing rank heterogeneity that undermines aggregation, and existing reconciliation methods still incur bias or inefficiency. To address this challenge, we propose *RA-LoRA*, a principled rank-aware aggregation framework that decomposes each update into rank-wise components and aligns them using analytically derived weights. Experiments on both language models and vision transformers demonstrate consistent accuracy improvements in one-shot and three-shot settings.

1 Introduction

Foundation models have achieved state-of-the-art performance across a wide spectrum of tasks (Brown et al., 2020). Notwithstanding these advances, their scale—with billions of parameters—imposes substantial computational and communication overhead, rendering full-parameter updates impractical in federated settings (Wu et al., 2025b). To mitigate this bottleneck, Parameter-efficient fine-tuning (PEFT) techniques have been studied extensively; among these, Low-rank adaptation (LoRA) has gained prominence by freezing pretrained weights and updating only low-rank adapters (Hu et al., 2022).

In the federated LoRA paradigm, clients retain raw data locally and transmit solely the gradients of their low-rank adapters, thereby preserving privacy and dramatically reducing communication costs (Wu et al., 2025a). Each client selects an adapter rank r_i according to its computational capacity (Cho et al., 2024), which induces *rank heterogeneity*—a mismatch in adapter dimensions across clients. Existing remedies employ zero-padding (Cho et al., 2024), replication (Byun and Lee, 2025), or stacking (Wang et al., 2024) to reconcile these differences, yet each heuristic introduces undesirable bias or overhead.

To address these shortcomings, we present **RA-LoRA**, a principled framework that casts rank-aware aggregation as a weighted-alignment optimization. By decomposing adapter updates rank-wise and deriving closed-form weights, RA-LoRA subsumes prior heuristics and balances contributions from clients with disparate ranks.

Our contributions are as follows:

- We propose a unified weighted-alignment framework for heterogeneous-rank aggregation.
- We derive a closed-form, factorized weighting scheme that corrects client-rank bias.
- We validate our approach on federated LoRA for both LLMs and vision transformers.

2 Preliminaries

Low-Rank Adaptation (LoRA). LoRA injects trainable low-rank adapter matrices into a linear layer of a pretrained model, freezing the original weights. Concretely, given a weight matrix $W_0 \in \mathbb{R}^{d\times d}$, LoRA represents the fine-tuned weight as

$$W = W_0 + BA, \tag{1}$$

where $A \in \mathbb{R}^{r\times d}$, $B \in \mathbb{R}^{d\times r}$, $r \ll d$. This reduces the number of trainable parameters from d^2 to $2dr$ (Hu et al., 2022).

Federated LoRA. In the federated setting, each of the K clients fine-tunes and transmits to the server only its adapter $(B^{(k)}, A^{(k)})$. The server aggregates the low-rank updates as

$$\Delta W_{\text{agg}} = \left(\frac{1}{K}\sum_k B^{(k)}\right)\left(\frac{1}{K}\sum_k A^{(k)}\right), \tag{2}$$

yielding the global adapter ΔW_{agg}, which serves as a unified LoRA module that reconciles information from all clients to improve generalization.

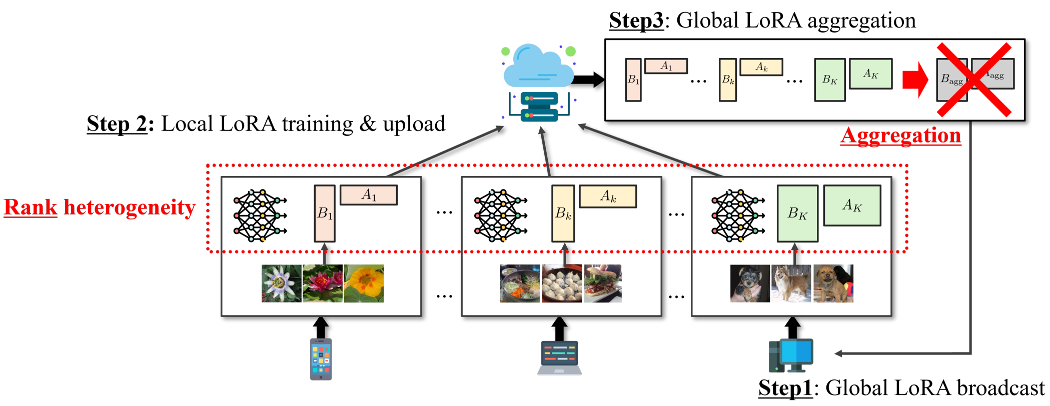
Step3: Global LoRA aggregation
Step 2: Local LoRA training & upload
Rank heterogeneity
Aggregation
B_1
A_1
B_k
A_k
B_K
A_K
B_{agg}
A_{agg}
Step1: Global LoRA broadcast

Figure 1: heterogeneous rank LoRA scenario illustration

Rank Heterogeneity. In the federated LoRA setting, each client k selects its adapter rank r_k based on local capability, resulting in adapter matrices

$$B^{(k)} \in \mathbb{R}^{d \times r_k}, \quad A^{(k)} \in \mathbb{R}^{r_k \times d} \tag{3}$$

of various ranks. Because r_k can differ across clients, directly averaging the low-rank updates $B^{(k)}A^{(k)}$ is ill-posed. In what follows, we introduce three existing method to reconcile these mismatched adapters.

Zero-Padding. A straightforward method for aligning heterogeneous adapters is to pad the adapter matrices of each client to a common rank $R = \max_k r_k$ by appending zeros:

$$B_{\text{zp}}^{(k)} = \left[B^{(k)} \parallel \mathbf{0} \right], \tag{4}$$

$${A_{\text{zp}}^{(k)}}^{\top} = \left[{A^{(k)}}^{\top} \parallel \mathbf{0} \right], \tag{5}$$

with $B_{\text{zp}}^{(k)} \in \mathbb{R}^{d \times R}$ and $A_{\text{zp}}^{(k)} \in \mathbb{R}^{R \times d}$ (Cho et al., 2024). The server then aggregates these zero-padded matrices by

$$\Delta W_{\text{agg}} = \left(\frac{1}{K} \sum_k B_{\text{zp}}^{(k)} \right) \left(\frac{1}{K} \sum_k A_{\text{zp}}^{(k)} \right). \tag{6}$$

A key feature of the zero-padding method is its simplicity and efficiency when aggregating adapters of various ranks. However, the inserted zeros dilute signals from high-rank clients—who possess richer representational capability—biasing the global update toward lower-rank components.

Replication. A replication method avoids the dilution caused by zero entries, by duplicating existing adapter columns and rows to reach the target rank $R = \max_k r_k$. Specifically, let the columns of $B^{(k)}$ be $\mathbf{b}_1^{(k)}, \cdots, \mathbf{b}_{r_k}^{(k)}$ and the rows of $A^{(k)}$ be ${\mathbf{a}_1^{(k)}}^{\top}, \cdots, {\mathbf{a}_{r_k}^{(k)}}^{\top}$. Then, each component is then replicated as follows:

$$B_{\text{rep}}^{(k)} = \left[B^{(k)} \parallel \mathbf{b}_{r_k+1}^{(\text{high rank})}, \cdots, \mathbf{b}_R^{(\text{high rank})} \right], \tag{7}$$

$${A_{\text{rep}}^{(k)}}^{\top} = \left[{A^{(k)}}^{\top} \parallel {\mathbf{a}_{r_k+1}^{(\text{high rank})}}^{\top} \cdots {\mathbf{a}_R^{(\text{high rank})}}^{\top} \right], \tag{8}$$

with $B_{\text{rep}}^{(k)} \in \mathbb{R}^{d \times R}$ and $A_{\text{rep}}^{(k)} \in \mathbb{R}^{R \times d}$ (Byun and Lee, 2025).

Although the replication method eliminates the zero-dilution issue by preserving all adapter entries, it relies on a binary division of clients into high-rank and low-rank groups. An extension of this method to federated scenarios with richer, multi-level rank distributions is non-trivial, limiting scalability and often overemphasizing contributions from high-rank clients.

Stacking. A stacking method concatenates each client's adapter matrices along the rank dimension:

$$B_{\text{stack}} = \left[B^{(1)} \parallel \cdots \parallel B^{(K)} \right], \tag{9}$$

$${A_{\text{stack}}}^{\top} = \left[{A^{(1)}}^{\top} \parallel \cdots \parallel {A^{(K)}}^{\top} \right], \tag{10}$$

with $B_{\text{stack}} \in \mathbb{R}^{d \times (\sum_k r_k)}$, $A_{\text{stack}} \in \mathbb{R}^{(\sum_k r_k) \times d}$. The server then aggregates via

$$\Delta W_{\text{agg}} = B_{\text{stack}} A_{\text{stack}}. \tag{11}$$

While the above method recovers the same update formula as in centralized learning (Wang et al., 2024), the aggregated adapter results in an excessively large rank of $\sum_k r_k$. (i.e., $\sum_k r_k \gg \max_k r_k$) In a federated setting, applying rank-reduction to control such growth incurs significant overhead, effectively negating LoRA's parameter-efficiency benefits.

Discussion. Each aggregation method introduces its own bias and inefficiency, making direct comparison challenging. Zero-padding enforces compatibility at the cost of diluting high-rank signals; replication preserves all non-zero entries but only supports a binary high/low rank split; stacking retains every client-specific direction yet demands excessive communication and expensive compression. Accordingly, a unified framework is required to place these methods on equal footing and enable systematic comparison—an objective we achieve in §3.

3 Method: Rank-Aware Aggregation

Rank-Wise Decomposition. Each client's LoRA update $\Delta W^{(k)}$ admits a decomposition into r_k elementary component matrices, each formed by the product of two basis vectors:

$$\Delta W^{(k)} = \sum_{r=1}^{r_k} \mathbf{b}_r^{(k)} \, \mathbf{a}_r^{(k)\top}, \tag{12}$$

where $\mathbf{b}_r^{(k)}, \mathbf{a}_r^{(k)} \in \mathbb{R}^{d\times 1}$ are the rth column of $B^{(k)}$ and row of $A^{(k)}$, respectively. This basis view exposes each rank component as a standalone matrix of size $d \times d$.

Generalized Aggregation Representation. To unify heterogeneous LoRA updates, we first pad each client's factors $B^{(k)} \in \mathbb{R}^{d\times r_k}$ and $A^{(k)} \in \mathbb{R}^{r_k\times d}$ to a common rank $R = \max_k r_k$:

$$P^{(k)} = \begin{bmatrix} I_{r_k} \\ \mathbf{0}_{(R-r_k)\times r_k} \end{bmatrix} \in \mathbb{R}^{R\times r_k}, \tag{13}$$

$$\tilde{B}^{(k)} = B^{(k)} P^{(k)\top} \in \mathbb{R}^{d\times R}, \tag{14}$$

$$\tilde{A}^{(k)} = P^{(k)} A^{(k)} \in \mathbb{R}^{R\times d}. \tag{15}$$

Stacking across clients, $\tilde{B} = [\tilde{B}^{(1)}, \ldots, \tilde{B}^{(K)}]$ and $\tilde{A} = [\tilde{A}^{(1)}; \ldots; \tilde{A}^{(K)}]$, and any aggregation can be written as

$$\Delta W_{\text{agg}} = \tilde{B}\, W\, \tilde{A},$$

where $W \in \mathbb{R}^{KR\times KR}$ encodes the weighting and alignment of client updates.

Unified Framework for the Exsiting Methods.

- **Zero Padding:** W is a block-diagonal matrix where each diagonal block is a matrix of ones of size $R \times R$:

$$\Delta W_{\text{zp}} = \text{diag}(\underbrace{\mathbf{1}_{R\times R}, \ldots, \mathbf{1}_{R\times R}}_{K \text{ times}}). \tag{16}$$

- **Replication:** W is constructed from client-specific replication weights. Let $\mathbf{C}^{(k)}_{R\times R} := \text{diag}(\gamma_{k,1}, \ldots, \gamma_{k,R}) \cdot \mathbf{1}_{R\times R}$, then

$$\Delta W_{\text{rep}} = \text{diag}(\mathbf{C}^{(1)}_{R\times R}, \ldots, \mathbf{C}^{(K)}_{R\times R}). \tag{17}$$

This compensates for rank mismatch by replicating low-rank contributions. Equivalently, under the rank-wise decomposition,

$$\Delta W \text{rep} = \sum_{k=1}^{K} \sum_{r=1}^{R} \gamma_r^{(k)} \, \mathbf{b}_r^{(k)} \mathbf{a}_r^{(k)\top}, \tag{18}$$

where the binary weight $\gamma_r^{(k)}$ is defined by

$$\gamma_r^{(k)} = \begin{cases} 2, & r > r_{\text{low}}, \\ 1, & r \le r_{\text{low}}. \end{cases}$$

This reformulation suggests that allowing $\gamma_r^{(k)}$ to vary continuously—rather than being restricted to $\{1, 2\}$—could yield more flexible and effective aggregation.

- **Stacking:** Although FLoRA performs stacking and reprojection in practice, it effectively corresponds to $W = I_{KR}$ in the form $\tilde{B} W_{\text{concat}} \tilde{A}$, treating the stacked columns as-is without additional weighting.

Proposed Aggregation Method. Continuous weighting of rank components provides maximal flexibility, but directly optimizing these weights for deep networks is intractable. Instead, we focus on two decisive factors—client data volume and rank rarity—to construct our weights. Specifically, we define:

$$\gamma_r^{(k)} = \alpha_k \, \beta_r, \tag{19}$$

$$\alpha_k = \frac{|D^{(k)}|}{\sum_{j=1}^{K} |D^{(j)}|}, \quad \beta_r = \frac{|D_1|}{|D_r|}, \tag{20}$$

where $|D^{(k)}|$ is the number of examples held by client k, and $|D_r|$ is the total data of all clients with adapter rank at least r. Factor α_k ensures that each client's update is weighted by its data volume, while β_r promotes fairness by up-weighting under-represented rank components. This closed-form design adapts continuously to arbitrary rank heterogeneity and recovers both FedAvg (when all r_k are equal) and binary replication (when β_r is thresholded) as special cases.

	Uniform HETLoRA		Weighted HETLoRA		FLoRA		*RA-LoRA*	
Dataset	**1-Shot**	**3-Shot**	**1-Shot**	**3-Shot**	**1-Shot**	**3-Shot**	**1-Shot**	**3-Shot**
Dolly	39.22	40.13	38.64	39.99	40.00	38.74	**40.87**	**40.38**
Alpaca + Dolly	39.17	39.11	39.64	39.68	38.43	38.63	**39.67**	**40.83**

Table 1: MMLU accuracy (%) under 1-shot and 3-shot communication settings for various LoRA aggregation methods, evaluated on the Dolly and Alpaca+Dolly datasets.

Method	**PPL (WikiText2)**	**PPL (PTB)**
Uniform HETLoRA	19.63	80.62
Weighted HETLoRA	19.15	77.83
FLoRA	14.55	59.12
RA-LoRA	16.76	65.77

Table 2: Comparison of perplexity on Wikitext2 and PTB.

Method	**#Params**	**Complexity**
Uniform HETLoRA	$6,873M$	$O(d^2KR)$
Weighted HETLoRA	$6,873M$	$O(d^2KR)$
FLoRA	$7,229M$	$O(dK^2R^2)$
RA-LoRA	$6,873M$	$O(d^2KR)$

Table 3: Comparison of the number of parameters in the global model and the computational complexity of each aggregation method.

4 Experimental Setup

We evaluate **RA-LoRA** on both language models and vision transformers; more detailed configuration are provided in B.

Model and Datasets. We use the alpaca-native model (Taori et al., 2023) as our base model. All experiments are conducted on the Alpaca and Dolly-15K instruction-response datasets (Databricks, 2023). We distribute the dataset across 10 clients, ensuring non-IID splits by varying per-client dataset sizes (500–2,500 examples). For evaluation, we use the MMLU benchmark (Hendrycks et al., 2020) with 1,444 examples spanning 57 subjects. In addition, we measure perplexity (PPL) on two standard language modeling datasets—WikiText2 (Merity et al., 2017) and Penn Treebank (PTB) (Marcus et al., 1993)—to assess the fluency and generalization of the resulting models.

LoRA Configuration. We attach LoRA adapters only to the query and value projections of the frozen base model. All clients share LoRA hyperparameters: rank r varies per client in $\{4, 16, 64, 128, 256\}$.

Aggregation Methods. We evaluate our RA-LoRA in comparison with three baseline methods—Uniform HETLoRA, Weighted HETLoRA, and FLoRA—using implementations adapted from the FedIT framework (Zhang et al., 2024, 2023).

5 Results

We report the main evaluation results of RA-LoRA on language models below. Results on vision transformers are presented in Appendix A.

As shown in Table 1, RA-LoRA consistently outperforms all baseline methods in both the one-shot and three-shot settings. This indicates the effectiveness of our proposed weighting scheme in reconciling heterogeneous local updates. In contrast, FLoRA exhibits a decline in accuracy as the number of communication rounds increases, likely due to the cumulative rank expansion followed by repeated reduction, which can distort the learned representations.

Table 2 presents perplexity results on WikiText2 and Penn Treebank. FLoRA achieves the lowest perplexity among all methods, which is expected since its evaluation is conducted prior to any rank reduction. Nevertheless, RA-LoRA achieves lower perplexity than the other HETLoRA-based methods, suggesting that it retains better language modeling capabilities under comparable compression settings.

Table 3 compares the parameter and computational efficiency of each method. FLoRA requires substantially more global parameters and exhibits higher aggregation complexity, particularly as the number of clients increases. Nonetheless, RA-LoRA achieves a better trade-off between perplexity and parameter efficiency.

6 Conclusion

RA-LoRA introduces a rank-aware aggregation framework that decomposes LoRA updates into rank-wise components and aligns with analytic weights, correcting rank heterogeneity. Experiments on language and vision models demonstrate consistent accuracy gains.

7 Limitations

Although RA-LoRA delivers strong empirical results, its closed-form weighting remains a heuristic rather than an optimal solution for the aggregation matrix W; end-to-end optimization or learning of these weights could further improve performance. In addition, our evaluation covers only a small set of language and vision benchmarks, and broader experimentation is required to validate the generality of our approach.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yuji Byun and Jaeho Lee. 2025. Towards federated low-rank adaptation of language models with rank heterogeneity. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 356–362.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, and Gauri Joshi. 2024. Heterogeneous lora for federated fine-tuning of on-device foundation models. In *EMNLP*.

Databricks. 2023. Databricks dolly 15k: Instruction-tuned dataset. https://github.com/databrickslabs/dolly. Accessed: 2025-07-01.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ziyao Wang, Zheyu Shen, Yexiao He, Guoheng Sun, Hongyi Wang, Lingjuan Lyu, and Ang Li. 2024. Flora: Federated fine-tuning large language models with heterogeneous low-rank adaptations. *Advances in Neural Information Processing Systems*, 37:22513–22533.

Fei Wu, Jia Hu, Geyong Min, and Shiqiang Wang. 2025a. Adaptive rank allocation for federated parameter-efficient fine-tuning of language models. *arXiv preprint arXiv:2501.14406*.

Yebo Wu, Chunlin Tian, Jingguang Li, He Sun, Kahou Tam, Li Li, and Chengzhong Xu. 2025b. A survey on federated fine-tuning of large language models. *CoRR*.

Jianyi Zhang, Martin Kuo, Ruiyi Zhang, Guoyin Wang, Saeed Vahidian, and Yiran Chen. 2023. Shepherd: A lightweight github platform supporting federated instruction tuning. https://github.com/JayZhang42/FederatedGPT-Shepherd.

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. IEEE.

A Vision Transformer Experiments

Model and Datasets. We extend our evaluation to visual classification by fine-tuning a vision transformer (Vaswani et al., 2017) backbone (pretrained on ImageNet) with LoRA adapters. To demonstrate modality-agnostic robustness, we conduct experiments on Food-101 benchmarks.

LoRA Configuration. For each sampled dataset, we partition the training set across four client groups, assigning adapter ranks of 2, 4, 8, and 16—together covering roughly 0.021% to 0.172% of the model's parameters. Clients perform local updates and aggregation for ten communication rounds, using the same hyperparameters (learning rate, batch size, etc.) as in our language-model experiments.

Results. RA-LoRA consistently accelerates convergence and improves final top-1 accuracy across all three sampled vision benchmarks. Its linear communication and computation scaling—as

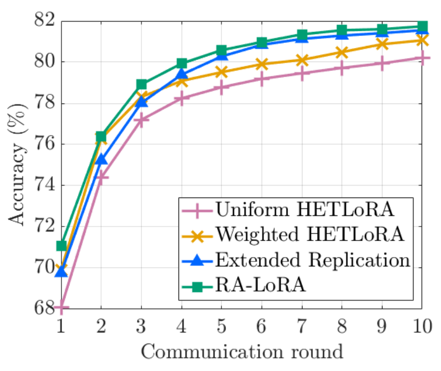

Figure 2: Top-1 accuracy over communication rounds for different aggregation methods on the Food-101 dataset using ViT backbone

opposed to the quadratic blow-up of stacking—enables efficient federated fine-tuning even on high-resolution or medical-image tasks. Future work may explore additional domains (e.g. object detection or segmentation) to further validate our approach.

B Experimental Configuration

We describe additional implementation details used in the experiments reported in Section 4.

Training Setup. Each client trains with a local batch size of 32 and a micro batch size of 16. We use stochastic gradient descent (SGD) as the optimizer with a local learning rate of 0.0003 and apply linear learning rate decay. Training is performed for one local epoch per communication round. LoRA adapters are inserted into the query and value projection layers, with LoRA alpha set to 16 and dropout rate set to 0.05. Clients train on inputs only, without label supervision, and sequence length grouping is disabled.

Communication Setup. We conduct up to three communication rounds for each aggregation method. After each round, clients transmit their adapted parameters to the server for aggregation.

Hardware. All experiments are run on a machine equipped with three NVIDIA RTX 6000 Ada Generation GPUs, each with 48 GB of memory.