FedSubMuon: Communication-Efficient Federated LLM Fine-Tuning via Structured Subspace Muon

Shaolong Chen[1,2,3], Youming Tao[4], Shuzhen Chen[5], Falko Dressler[4], Qingqing Ye[6], Di Wang[1,2]*
[1]King Abdullah University of Science and Technology (KAUST),
[2]Provable Responsible AI and Data Analytics (PRADA) Lab,
[3]Imperial College London, [4]Technische Universität Berlin (TU Berlin),
[5]Ocean University of China, [6]The Hong Kong Polytechnic University

Abstract

Federated fine-tuning adapts large language models (LLMs) to decentralized client data, but its scalability in cross-device training is often limited by the high communication cost. Muon is an optimizer that improves optimization performance by orthogonalizing momentum for matrix-valued parameters. Existing federated Muon methods demonstrate the benefit of matrix-aware optimization in federated learning, but still require transmitting full layer-size updates and optimizer state. A natural way to reduce communication is to directly apply Muon to LoRA factors, but this changes the optimized object and weakens Muon's matrix-aware update geometry. We propose FEDSUBMUON, a communication-efficient federated Muon fine-tuning method that optimizes compact coefficient matrices within shared structured subspaces. This design keeps Muon on a single matrix-valued trainable object, while reducing the client upload to compact coefficient matrices. We further introduce FEDSUBMUON-GT, an accuracy-oriented extension that uses projected gradients to adapt tracked subspace bases toward task-relevant gradient directions. Experiments on instruction tuning and mathematical reasoning show that FEDSUBMUON-GT achieves the best overall accuracy on four of five dataset–model pairs, while FEDSUBMUON performs best under all matched communication budgets. On Dolly-15K, the closest communication baseline requires $5.5\times$ and $1.4\times$ more total communication on Llama-1B and Qwen-4B, respectively.

1 Introduction

Federated fine-tuning adapts large language models (LLMs) on decentralized client data without transferring raw examples to a central server. In cross-device settings, however, this adaptation is often constrained by communication cost: in each communication round, the server sends trainable parameters to selected clients, and clients return local updates for aggregation (Zhang et al., 2024; Liu et al., 2025a). As a result, the size of the communicated update becomes a central design constraint for federated LLM fine-tuning.

Table 1: Optimizer control for LoRA-factor federated baselines on Natural Instructions with Llama 1B. Each cell reports mean ROUGE-L% over two runs.

Method	AdamW	Muon	SGD
FedIT	30.96	30.81	27.94
FLoRA	31.14	30.99	27.85

Muon is a matrix-aware optimizer that improves optimization performance by orthogonalizing momentum before updating hidden-layer weights (Jordan, 2024). Existing work has begun to adapt Muon to federated learning, including FedMuon variants for matrix-orthogonalized federated optimization (Liu et al., 2025b; Takezawa et al., 2026; Zhang and Gao, 2025; Wang et al., 2026). This line of work is relevant because exploiting the matrix structure of model updates has been shown to improve optimization performance across a range of federated settings. However, existing FedMuon work does not remove the communication bottleneck for fine-tuning: applying Muon to full weight matrices requires transmitting full layer-size parameter updates.

A natural alternative is to apply Muon within parameter-efficient federated fine-tuning methods like LoRA, but this changes the object being optimized. LoRA represents an update through two coupled low-rank factors, and prior work shows that applying Muon to these factors is not equivalent to a parametrization-independent update of the underlying matrix (Bogachev et al., 2026). Consistent with this issue, Table 1 shows that replacing the optimizer in federated LoRA baselines with Muon does not improve performance. These observations motivate a federated Muon fine-tuning method that preserves Muon's matrix update geometry while

*Corresponding author.

communicating only a compact trainable object.

We propose FEDSUBMUON, a communication-efficient framework for federated Muon fine-tuning with a structured subspace. FEDSUBMUON reduces communication by transmitting a compact coefficient matrix in a shared matrix subspace. Clients optimize the coefficient matrix with Muon, and upload it for aggregation. At refresh rounds, the server commits the aggregated subspace update into a global accumulator and samples a new coefficient matrix. Then we introduce FEDSUBMUON-GT, which uses projected gradients at refresh rounds to update tracked subspace bases. While FEDSUBMUON emphasizes communication efficiency, FEDSUBMUON-GT adds basis-tracking communication to improve accuracy.

Experiments on instruction tuning and mathematical reasoning with Llama and Qwen backbones show this trade-off. FEDSUBMUON-GT achieves the strongest overall accuracy, ranking first on four of five dataset–model pairs. Under matched communication budgets, FEDSUBMUON performs best on both evaluated backbones, showing that simply lowering the rank of LoRA baselines is not a reliable substitute for the proposed subspace coefficient parameterization.

Our contributions are:

- We propose FEDSUBMUON, which adapts federated Muon to communication-efficient fine-tuning by transmitting compact coefficient matrices rather than full weight matrices.
- We develop FEDSUBMUON-GT, a performance-oriented extension that uses projected gradients to adapt tracked subspace bases to task-relevant gradient directions.
- We evaluate the proposed method on instruction tuning and mathematical reasoning across 1-8B LLMs, demonstrating improved accuracy–communication trade-offs under standard and matched-budget settings.

2 Preliminaries

Federated fine-tuning. We consider cross-device federated fine-tuning of a frozen pretrained backbone W^0 over N clients, where private data remain on clients (McMahan et al., 2017). Client i owns a local objective f_i, and the global objective is

$$\min_{\Delta W} f(\Delta W) := \sum_{i=1}^{N} p_i f_i(W^0 + \Delta W),$$

where $\mathcal{L}$ denotes the index set of selected weight matrices, $\Delta W = \{\Delta W_\ell\}_{\ell \in \mathcal{L}}$ is the update on those matrices, and p_i is the aggregation weight of client i. At round t, we write $W^t := W^0 + \Delta W^t$ for the current model. A prevalent federated fine-tuning strategy adapts selected projection matrices with low-rank matrices (Hu et al., 2022); selected clients train and upload adapter parameters for server aggregation (Zhang et al., 2023b, 2024).

Muon Optimizer. Muon is a matrix-aware optimizer that updates hidden-layer weight matrices by orthogonalizing the momentum (Jordan, 2024). Consider the full-matrix update for a layer ℓ. With global gradient $G_\ell^t = \nabla_{W_\ell} f(W^t)$ and matrix momentum M_ℓ^t, full Muon would use

$$M_\ell^{t+1} = (1-\beta) M_\ell^t + \beta G_\ell^t,$$
$$W_\ell^{t+1} = W_\ell^t - \eta \, \mathrm{Muon}(M_\ell^{t+1}),$$

where $\mathrm{Muon}(A) = \widetilde{U}\widetilde{V}^\top$ for the compact SVD $A = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^\top$; Newton–Schulz (NS) provides an efficient approximation to this direction practically.

Random subspace. Random subspace optimization restricts training to a low-dimensional coordinate inside the full parameter space (Li et al., 2018; Aghajanyan et al., 2021). For a layer ℓ, a generic vectorized random subspace is

$$\mathrm{vec}(\Delta W_\ell) = P_\ell b_\ell,$$

where $P_\ell \in \mathbb{R}^{d_{\mathrm{out}} d_{\mathrm{in}} \times k}$ is a random projection matrix and $b_\ell \in \mathbb{R}^k$ is the trainable subspace coordinate. The projection matrix can be resampled periodically (Yang et al., 2026; Rajabi et al., 2025); switching P_ℓ changes the low-dimensional subspace used by subsequent updates.

3 Method

3.1 Overview

FEDSUBMUON is a structured subspace method for federated Muon fine-tuning. The backbone remains frozen, while the server maintains a global update accumulator over the selected weight matrices. Clients train compact subspace coefficient matrices with fixed bases generated from shared random seeds. This parameterization applies Muon to a matrix-structured coefficient update while reducing the per-round trainable parameter count and communication payload.

Figure 1 illustrates the key workflow of FEDSUBMUON. The server samples clients, sends the

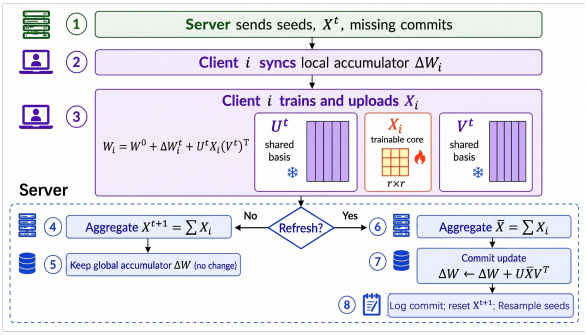


Figure 1: FEDSUBMUON Workflow.

current random seed, active subspace coefficient matrix and any missing commit records to clients. A selected client first replays those commits to synchronize the global update accumulator to its local accumulator. It then regenerates the shared bases from the random seed, runs local Muon updates on the compact subspace coefficient matrix, and uploads the updated matrix for server aggregation.

Refresh rounds separate accumulated model updates from later subspace updates. At the refresh round, the server commits the full model update into the global update accumulator, appends the aggregated coefficient matrix and corresponding seeds to the commit log, resets the coefficient matrix, and samples fresh random seeds for a new basis. If the current communication round is not a refresh round, the server keeps the global update accumulator and random seed unchanged and sets the aggregated value as the active subspace coefficient matrix for the next communication round.

Appendix B provides the full procedural descriptions of both FEDSUBMUON and FEDSUBMUON-GT, corresponding to Algorithms 1 and 2.

3.2 Client Synchronization and Subspace Coefficient Update

We first describe one communication round before separating the refresh branch. A selected client first synchronizes its local accumulator by replaying missing commits; since those records are created at refresh rounds, we give the replay rule in Section 3.3. After this synchronization, all selected clients start local training from the same subspace coefficient matrices X^t and use the same bases.

For a selected layer ℓ, the update used in the current round is separated into two parts: the synchronized local accumulator $\Delta W_{i,\ell}$ and the active subspace coefficient matrix $X_{i,\ell}$. The former stores committed matrix updates, while the latter is the only trainable object during the current round. In FEDSUBMUON, the current bases are generated from the random seed s^t. For each selected layer, the shared seed deterministically generates Gaussian matrices $\Omega_{U,\ell}^t \in \mathbb{R}^{d_{\text{out}} \times r}$ and $\Omega_{V,\ell}^t \in \mathbb{R}^{d_{\text{in}} \times r}$ with i.i.d. $\mathcal{N}(0,1)$ entries, and

$$U_\ell^t = \mathrm{qf}(\Omega_{U,\ell}^t), \qquad V_\ell^t = \mathrm{qf}(\Omega_{V,\ell}^t),$$

where $\mathrm{qf}(\cdot)$ returns the thin-QR orthonormal factor with a fixed deterministic sign convention.

With the current bases fixed, client i optimizes only the subspace coefficient matrix $X_{i,\ell} \in \mathbb{R}^{r \times r}$. The effective local layer used for training is

$$W_{i,\ell}(X_{i,\ell}) = W_\ell^0 + \Delta W_{i,\ell} + U_\ell^t X_{i,\ell} (V_\ell^t)^\top. \quad (1)$$

Eq. (1) is the local training model: the frozen backbone W_ℓ^0, the fixed synchronized accumulator $\Delta W_{i,\ell}$ and fixed subspace bases (U_ℓ^t, V_ℓ^t), only $X_{i,\ell}$ is updated. At the start of local optimization, the client initializes

$$X_{i,\ell}^{t,0} = X_\ell^t, \qquad M_{i,\ell}^{t,0} = 0. \quad (2)$$

For local step e, it computes a stochastic gradient with respect to the coefficient matrix,

$$\widehat{G}_{i,\ell}^{t,e} = \nabla_{X_{i,\ell}} f_i(W_i^t(X_i); \xi_i^{t,e})\big|_{X_i = X_i^{t,e}}. \quad (3)$$

By the chain rule, this is the current layer gradient projected onto the current left and right bases: when the full layer gradient is written as $\nabla_{W_{i,\ell}} f_i$, the corresponding coefficient gradient has the form $(U_\ell^t)^\top (\nabla_{W_{i,\ell}} f_i) V_\ell^t$. Thus Eq. (3) updates the $r \times r$ coefficient matrix, not a full layer-size matrix.

The client then updates the local momentum and applies Muon to this small matrix:

$$\begin{aligned} M_{i,\ell}^{t,e+1} &= (1-\beta) M_{i,\ell}^{t,e} + \beta \widehat{G}_{i,\ell}^{t,e}, \\ X_{i,\ell}^{t,e+1} &= X_{i,\ell}^{t,e} - \eta\, \mathrm{Muon}(M_{i,\ell}^{t,e+1}). \end{aligned} \quad (4)$$

This local update corresponds to step 3 in Figure 1. After E local steps, the client sets $X_{i,\ell}^{t+1} = X_{i,\ell}^{t,E}$. The momentum is reinitialized whenever the client is selected for a new round, so it remains local $r \times r$ optimizer state.

The client uploads $\{X_{i,\ell}^{t+1}\}_{\ell \in \mathcal{L}}$ and the server aggregates the uploaded coefficient matrices with the normalized round weights, $\bar{X}_\ell^{t+1} = \sum_{i \in \mathcal{S}_t} a_i^t X_{i,\ell}^{t+1}, \ell \in \mathcal{L}$. Since all selected clients start from the same active coefficient matrix X_ℓ^t, this aggregation averages compact coefficient updates inside the same current subspace.

3.3 Commit-and-Reset Refresh

After aggregation, the server checks the refresh interval τ to determine whether this round performs a refresh, as shown in Figure 1. If the round is not a refresh round, the server keeps the global accumulator and the current bases unchanged, set:

$$\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t, \qquad X_\ell^{t+1} \leftarrow \bar{X}_\ell^{t+1}.$$

In FEDSUBMUON, the same random seed is also kept for the next round. Thus non-refresh rounds simply continue optimizing the active coefficient matrix in the current subspace.

At a refresh round, the server commits the active coefficient update to the global accumulator and resets the active coefficient matrix:

$$\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t + U_\ell^t \bar{X}_\ell^{t+1} (V_\ell^t)^\top, X_\ell^{t+1} \leftarrow 0. \tag{5}$$

Eq. (5) is step 7 in the refresh branch of Figure 1. The commit uses the bases that were fixed during the current round, because those bases define the meaning of $\bar{X}_\ell^{t+1}$. After the commit, ΔW_ℓ^{t+1} stores the accumulated matrix update for layer ℓ, while the reset $X_\ell^{t+1} \leftarrow 0$ starts the next subspace update from zero.

This separation is important because the global update accumulator after multiple refresh rounds is a sum of matrix updates produced under different bases. A newly sampled random basis generally cannot represent that accumulated update exactly as a fresh coefficient matrix. Therefore, FEDSUBMUON keeps the committed update in the global update accumulator ΔW_ℓ^t, and uses the next basis only to parameterize subsequent coefficient updates. After a refresh, FEDSUBMUON samples fresh random seeds s^{t+1}, and clients in the next round regenerate the shared bases using the Gaussian–QR construction in Section 3.2.

Refreshes are synchronized lazily to handle partial participation. In FEDSUBMUON, the server appends each refresh to the compact log $\mathcal{H}$ as

$$\mathcal{C}_K = \big(s^{[K]} = s^t,\ \{X_\ell^{[K]} = \bar{X}_\ell^{t+1}\}_{\ell \in \mathcal{L}}\big),$$

and increments the commit counter K. The bracketed superscript denotes a commit index, not a communication round. If client i was inactive during some refresh rounds, it does not receive those commits immediately. When it is selected again, the server sends the missing records $\{\mathcal{C}_k\}_{k=v_i}^{K-1}$. For each missing record, the client regenerates the committed bases from $s^{[k]}$ and replays

$$\Delta W_{i,\ell} \leftarrow \Delta W_{i,\ell} + U_\ell^{[k]} X_\ell^{[k]} (V_\ell^{[k]})^\top, \qquad \ell \in \mathcal{L}.$$

It then sets $v_i \leftarrow K$. This synchronization is the step 2 in Figure 1 performed before Eq. (1).

3.4 FEDSUBMUON-GT: Gradient-Tracked Subspace Refresh

FEDSUBMUON uses random refresh to explore new search directions, but the refreshed bases are task-agnostic: they do not use the gradients observed during training, so the effective capture of task-relevant ambient directions can be weak. FEDSUBMUON-GT improves this by keeping the same client-side coefficient update, aggregation, and commit-and-reset rule as FEDSUBMUON, while replacing random basis refresh with gradient-tracked basis refresh. At initialization, the bases are generated from the random seed s^0. But after the first refresh, the server sends tracked subspace bases (U_ℓ^t, V_ℓ^t) instead of sampling a fresh seed. The goal is to make subspace coefficient updates better capture task-relevant full-matrix directions by steering future bases using client gradients.

Updating tracked bases from a full layer gradient would require transmitting full layer-size gradient information. To avoid this cost, selected clients send projected gradients at refresh rounds. After completing the local coefficient update, client i evaluates the updated local model $W_i^t(X_i^{t+1})$ on a small probe batch, uses the resulting layer gradient as the probe gradient, forms

$$H_{U,i,\ell}^t = G_{i,\ell}^{\text{probe},t} V_\ell^t, \quad H_{V,i,\ell}^t = (G_{i,\ell}^{\text{probe},t})^\top U_\ell^t, \tag{6}$$

where $G_{i,\ell}^{\text{probe},t}$ denotes the local gradient used for basis tracking. In non-refresh rounds, clients upload only $\{X_{i,\ell}^{t+1}\}_{\ell \in \mathcal{L}}$. In refresh rounds, they upload $\{X_{i,\ell}^{t+1}, H_{U,i,\ell}^t, H_{V,i,\ell}^t\}_{\ell \in \mathcal{L}}$. The projected gradients keep the basis update dependent on the current local gradient, while avoiding transmission of a full gradient matrix.

The server aggregates these projected gradients with the same weights a_i^t used to aggregate X:

$$\bar{H}_{U,\ell}^t = \sum_{i \in \mathcal{S}_t} a_i^t H_{U,i,\ell}^t, \quad \bar{H}_{V,\ell}^t = \sum_{i \in \mathcal{S}_t} a_i^t H_{V,i,\ell}^t. \tag{7}$$

By linearity, these summaries are equivalent to projecting the weighted probe gradient $\bar{G}_\ell^t = \sum_{i \in \mathcal{S}_t} a_i^t G_{i,\ell}^{\text{probe},t}$ onto the current opposite-side bases: $\bar{H}_{U,\ell}^t = \bar{G}_\ell^t V_\ell^t$ and $\bar{H}_{V,\ell}^t = (\bar{G}_\ell^t)^\top U_\ell^t$.

The basis update follows the principle of gradient subspace tracking. Given the current right basis V_ℓ^t, the left basis should be adjusted to better span

the dominant directions of $\bar{G}_\ell^t V_\ell^t$; the right basis is handled analogously. This can be viewed as two small least-squares subspace fitting problems:

$$\mathcal{J}_U(U) = \min_{B_{U,\ell}} \|UB_{U,\ell} - \bar{H}_{U,\ell}^t\|_F^2,$$
$$\mathcal{J}_V(V) = \min_{B_{V,\ell}} \|VB_{V,\ell} - \bar{H}_{V,\ell}^t\|_F^2.$$

Since U_ℓ^t and V_ℓ^t are orthonormal, the least-squares coordinates at the current bases are

$$B_{U,\ell}^t = (U_\ell^t)^\top \bar{H}_{U,\ell}^t, \qquad B_{V,\ell}^t = (V_\ell^t)^\top \bar{H}_{V,\ell}^t. \tag{8a}$$

The corresponding residual components outside the current bases are $(I - U_\ell^t(U_\ell^t)^\top)\bar{H}_{U,\ell}^t$ and $(I - V_\ell^t(V_\ell^t)^\top)\bar{H}_{V,\ell}^t$. Following the Grassmannian subspace-tracking view (Rajabi et al., 2025), where a subspace is updated using the residual of a least-squares gradient approximation rather than recomputed from scratch, we use the following first-order tangent directions:

$$\begin{aligned} Y_{U,\ell}^t &= (I - U_\ell^t(U_\ell^t)^\top)\bar{H}_{U,\ell}^t(B_{U,\ell}^t)^\top, \\ Y_{V,\ell}^t &= (I - V_\ell^t(V_\ell^t)^\top)\bar{H}_{V,\ell}^t(B_{V,\ell}^t)^\top. \end{aligned} \tag{8b}$$

Here the projection matrices remove the components already represented by the current bases, while multiplication by $(B_{U,\ell}^t)^\top$ and $(B_{V,\ell}^t)^\top$ weights each residual direction by its interaction with the currently represented coefficient-gradient coordinates. Therefore, $Y_{U,\ell}^t$ and $Y_{V,\ell}^t$ do not restart the subspace; instead, they rotate the existing bases toward gradient directions that are useful but currently underrepresented.

Finally, we update the bases:

$$\begin{aligned} U_\ell^{t+1} &= \mathrm{qf}(U_\ell^t + \rho Y_{U,\ell}^t), \\ V_\ell^{t+1} &= \mathrm{qf}(V_\ell^t + \rho Y_{V,\ell}^t). \end{aligned} \tag{9}$$

The subspace learning rate ρ controls the size of the basis rotation, and the QR retraction restores orthonormality. This gives a lightweight first-order approximation to a Grassmannian subspace update, adapted to the structured subspace. The update is applied after committing the current coefficient update, so the basis refresh tracks the gradient directions that are not captured by the already accumulated subspace update.

At the same refresh round, FEDSUBMUON-GT appends a commit record with the tracked bases used for the committed update: $\mathcal{C}_K = \big(\{U_\ell^{[K]} = U_\ell^t,\ V_\ell^{[K]} = V_\ell^t,\ X_\ell^{[K]} = \bar{X}_\ell^{t+1}\}_{\ell\in\mathcal{L}}\big)$. Thus, inactive clients can later replay the exact committed update into their local update accumulators. The difference from FEDSUBMUON is that FEDSUBMUON stores the random seed used to regenerate the committed bases, whereas FEDSUBMUON-GT stores the tracked bases themselves.

3.5 Communication Analysis

In each communication round of FEDSUBMUON, the server sends the current random seed s^t, the active subspace coefficient matrices X^t, and any missing commit records. For each selected layer, a standard round sends an $r \times r$ coefficient matrix to each selected client and receives the updated coefficient matrix back. Therefore the per-client total communication for the active coefficient matrix is $O(r^2)$ downlink plus $O(r^2)$ uplink, independent of the full layer matrix size. Note that r is the side length of a coefficient matrix decoupled from the layer dimensions, whereas the LoRA rank indexes an adapter tied to d_{out} and d_{in}, so the two are not comparable on a shared rank axis. With $r \ll \min\{d_{\text{out}}, d_{\text{in}}\}$, the per-round payload stays at or below the LoRA baselines (Appendix Table 5). This communication reduction does not imply that the committed update is stored in a factorized form: the global update accumulator ΔW_ℓ^t is an ambient matrix for each selected layer, and synchronized clients may materialize the corresponding local accumulator $\Delta W_{i,\ell}$ during training. At deployment time, the accumulated update can be merged into the frozen backbone, so our method does not require an additional full-layer-size parameter store.

Refresh rounds add only compact synchronization state in FEDSUBMUON. The server records $\mathcal{C}_K = (s^{[K]} = s^t, \{X_\ell^{[K]} = \bar{X}_\ell^{t+1}\}_{\ell\in\mathcal{L}})$ in the commit log and samples fresh random seeds. Inactive clients receive and replay missed commit records when they are selected again. FEDSUBMUON-GT follows the same communication path, but after the first refresh it sends tracked subspace bases (U_ℓ^t, V_ℓ^t) in post-refresh communication rounds; its commit records store these bases together with the coefficient matrix. This tracked-basis downlink scales as $O((d_{\text{out}} + d_{\text{in}})r)$ whenever it is sent and can dominate the total traffic. At refresh rounds, selected clients also upload projected gradients $H_{U,i,\ell}^t$ and $H_{V,i,\ell}^t$, which add an amortized $O((d_{\text{out}} + d_{\text{in}})r/\tau)$ uplink term when refreshes occur every τ rounds.

Table 2: Performance comparison across datasets (%). Each cell reports mean ± standard deviation over four runs. **Best** and second-best values are highlighted.

Method	Natural Instructions	Dolly-15K		GSM8K	MATH
	Llama 1B	Llama 1B	Qwen 4B	Qwen 8B	Qwen 8B
FedIT	30.14 ± 0.82	28.32 ± 0.41	36.33 ± 0.17	79.59 ± 0.33	45.24 ± 1.22
FeDeRA	31.07 ± 0.43	29.63 ± 0.53	36.78 ± 0.37	80.29 ± 0.26	47.52 ± 0.65
FLoRA	31.12 ± 0.28	29.30 ± 0.45	36.39 ± 0.36	76.58 ± 0.82	43.48 ± 0.42
FedEx-LoRA	30.29 ± 0.51	28.83 ± 0.10	36.57 ± 0.76	79.51 ± 0.25	45.82 ± 1.26
FLoRG	29.83 ± 0.42	28.53 ± 0.54	36.79 ± 0.70	80.12 ± 0.51	43.48 ± 0.42
FedKRSO	30.56 ± 0.22	29.40 ± 0.71	**37.28 ± 0.33**	81.29 ± 0.49	45.95 ± 1.40
FedSubMuon	30.87 ± 0.61	29.64 ± 0.45	36.49 ± 0.39	81.32 ± 0.50	47.92 ± 1.06
FedSubMuon-GT	**31.37 ± 0.65**	**29.70 ± 0.54**	36.86 ± 0.17	**81.79 ± 0.71**	**48.44 ± 0.12**

4 Experiments

We evaluate whether the proposed structured subspace coefficient parameterization improves the accuracy–communication trade-off in federated LLM fine-tuning. The experiments are designed to answer two questions: whether FedSubMuon is more communication-efficient than federated LoRA and random-subspace baselines, and whether the projected-gradient basis tracking in FedSubMuon-GT translates into stronger downstream performance. We evaluate instruction tuning and mathematical reasoning across 1B–8B LLMs.

4.1 Setup

Models and datasets. We evaluate Llama 3.2 1B, Qwen3 4B, and Qwen3 8B. The instruction-tuning benchmarks are Natural Instructions (Mishra et al., 2022; Wang et al., 2022) and Dolly-15K (Conover et al., 2023); the reasoning benchmarks are GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). Natural Instructions is evaluated on Llama 3.2 1B, Dolly-15K on both Llama 3.2 1B and Qwen3 4B, and GSM8K/MATH on Qwen3 8B.

Evaluation metrics. We report ROUGE-L (Lin, 2004) for Natural Instructions and Dolly-15K, and exact match for GSM8K and MATH. Higher values are better for both metrics.

Baselines. We compare against six representative federated LoRA and random subspace baselines: FedIT (Zhang et al., 2024), FeDeRA (Yan et al., 2026), FLoRA (Wang et al., 2024), FedEx-LoRA (Singhal et al., 2025), FLoRG (Meng et al., 2026), and FedKRSO (Yang et al., 2026), covering factor aggregation, decomposition-based initialization, heterogeneous-rank LoRA, exact-update correction, Gram/Procrustes aggregation, and random subspace optimization.

Implementation details. Following the LoRA experimental setting (Hu et al., 2022), all methods tune only the query and value projection matrices in the attention blocks. Training uses early stopping; therefore, the round numbers below denote the maximum communication rounds rather than mandatory fixed training horizons. The baselines are trained for 40 rounds on Natural Instructions, 60 rounds on Dolly-15K, 20 rounds on GSM8K, and 30 rounds on MATH. FedSubMuon and FedSubMuon-GT use 10 rounds on Natural Instructions and 30 rounds on Dolly-15K with both backbones. On GSM8K and MATH, all methods use the same 20 and 30 rounds, respectively. The shorter FedSubMuon schedules on Natural Instructions and Dolly-15K highlight the convergence advantage under fewer communication rounds, while the matched GSM8K and MATH schedules support fair comparison on reasoning tasks. We use AdamW for the baselines because it is a standard optimizer and directly replacing it with Muon does not improve representative LoRA-factor baselines in our federated setting. Natural Instructions uses task-level heterogeneity, whereas the other datasets are partitioned with a Dirichlet distribution. Additional hyperparameter settings are provided in Appendix C, and dataset processing details are provided in Appendix D.

4.2 Performance

This subsection compares the downstream performance of all methods under the federated fine-tuning settings above. The main goal is to test whether compact subspace coefficient optimization can improve the accuracy of methods that communicate larger adapter or subspace states. Table 2 reports the task performance across datasets and model scales, and the corresponding per-round per-client trainable parameter counts are summarized in Appendix Table 5.

Table 2 shows that across the evaluated datasets and model scales, FEDSUBMUON-GT delivers the most consistently strong accuracy, while FEDSUBMUON remains competitive. FEDSUBMUON-GT is best on four out of five dataset–model pairs and second-best on the remaining one, showing that tracked refresh improves accuracy across both instruction tuning and reasoning settings.

FEDSUBMUON-GT shows its largest gains on the reasoning benchmarks. These suggest that improving the current shared basis, rather than resampling it at random, is particularly useful when stronger task adaptation is required. They are also consistent with the difference in parameterization capacity: in conventional LoRA methods, the trainable budget remains tied to the backbone matrix dimensions, so the practical rank typically has to stay small as model size grows. This pressure is especially acute in FL, where clients have limited local compute, memory, and communication resources. This restriction can make those baselines less expressive on more complex tasks such as mathematical reasoning. By contrast, the FEDSUBMUON subspace coefficient parameterization is decoupled from the full backbone matrix size, which can allocate substantially larger ranks for harder tasks.

4.3 Performance Analysis under Limited Communication

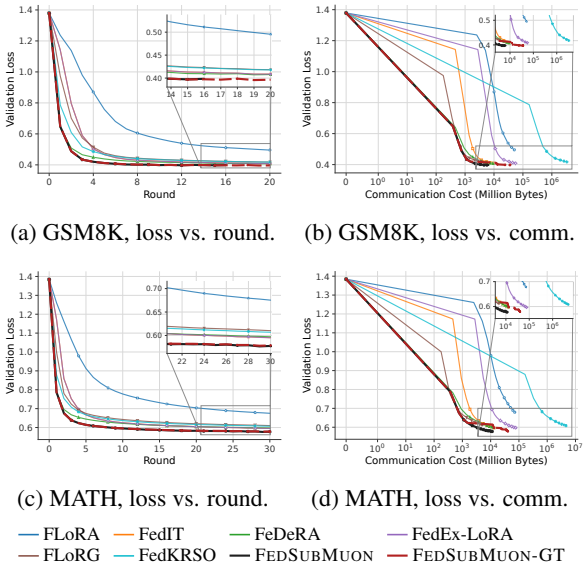
This subsection evaluates whether the accuracy advantage of FEDSUBMUON persists after controlling for *normalized communication*: the accumulated method-dependent training traffic, counting round-wise uploads, downloads, and refresh traffic while excluding the one-time backbone initialization payload common to all methods. We compare all methods under the same normalized communication budget. Table 3 reports the results.

To instantiate the matched-budget comparison, we fix the normalized communication budget to the corresponding FEDSUBMUON run. For the baselines, we sweep ranks in $\{1, 2, 4, 8\}$ and report the best result under this budget. For FEDSUBMUON-GT, the rank is set to the largest feasible even value implied by the budget. The resulting total normalized communication budgets are 220 MB for NI with Llama 1B, 264 MB for Dolly-15K with Llama 1B, and 3457 MB for Dolly-15K with Qwen 4B.

Even with this favorable rank-tuning opportunity for the baselines, FEDSUBMUON is the best method in all three matched-budget settings. FEDSUBMUON-GT is also competitive, obtaining

Table 3: Performance under matched communication budgets. Each cell reports the mean ROUGE-L% over two runs under the matched communication budget.

Method	Model / Dataset		
	Llama/NI	Llama/Dolly	Qwen/Dolly
FedSubMuon	**30.59**	**30.09**	**36.87**
FedSubMuon-GT	29.19	<u>29.32</u>	<u>36.83</u>
FedIT	<u>30.12</u>	24.70	34.62
FeDeRA	30.10	26.68	36.49
FLoRA	10.28	16.95	28.77
FedEx-LoRA	10.24	16.31	28.02
FLoRG	29.94	28.98	35.79
FedKRSO	-	-	-



(a) GSM8K, loss vs. round. (b) GSM8K, loss vs. comm.

(c) MATH, loss vs. round. (d) MATH, loss vs. comm.

Figure 2: Validation-loss dynamics on Qwen 8B.

the second-best result in two of the three settings. FedKRSO cannot run under these budgets because its full-matrix downlink is too large.

The results further show that simply reducing the adapter rank increases the number of feasible updates but limits adaptation capacity, making it an unreliable substitute for the FEDSUBMUON parameterization. FEDSUBMUON instead preserves a larger subspace rank within the matched budget, indicating that its subspace coefficient parameterization uses the available communication more effectively than rank reduction alone.

4.4 Training Loss under Rounds and Communication

This subsection compares the training convergence behavior of all methods. We analyze the Qwen 8B validation-loss trajectories from the preceding experiments in two complementary views: progress per communication round and progress per unit of

Table 4: Communication and computational efficiency on Dolly-15K.

Model	Metric	FedSubMuon	FedSubMuon-GT	FedIT	FeDeRA	FLoRA	FedEx-LoRA	FLoRG	FedKRSO
Llama 1B	Uplink (MB)	**105 (1×)**	923 (8.8×)	920 (8.8×)	1159 (11×)	2045 (19×)	920 (8.8×)	721 (6.9×)	1311 (12×)
	Total Comm. (MB)	**264 (1×)**	11848 (45×)	1840 (7.0×)	2317 (8.8×)	22492 (85×)	11042 (42×)	1442 (5.5×)	85679 (325×)
	GPU Mem. (MB)	**5530**	5555	5761	5762	5764	5762	6567	24136
	Wall time (s)	77.19	**75.33**	83.47	81.25	82.65	84.61	114.18	77.61
Qwen 4B	Uplink (MB)	**1410 (1×)**	7078 (5.0×)	4720 (3.3×)	5782 (4.1×)	7078 (5.0×)	6960 (4.9×)	2490 (1.8×)	13920 (9.9×)
	Total Comm. (MB)	**3457 (1×)**	83802 (24×)	9440 (2.7×)	11564 (3.3×)	77880 (23×)	83519 (24×)	4980 (1.4×)	1127508 (326×)
	GPU Mem. (MB)	**14615**	14825	15230	15226	15178	15230	23427	43259
	Wall time (s)	100.37	**93.14**	108.00	109.89	111.59	109.11	234.04	104.50

normalized communication. Figure 2 reports the loss comparisons on GSM8K and MATH.

Loss versus rounds. The round-based curves show that FEDSUBMUON and FEDSUBMUON-GT converge faster than the baselines and maintain a lower-loss frontier throughout training. On GSM8K, FEDSUBMUON-GT reaches the lowest final validation loss at 0.396, followed closely by FEDSUBMUON at 0.397, while both enter the low-loss region earlier than the strongest non-FEDSUBMUON baselines. On MATH, FEDSUBMUON and FEDSUBMUON-GT are essentially tied, ending at 0.577 and 0.578, respectively, and both remain below the baseline curves. These trajectories therefore show that the proposed subspace parameterization improves not only the final scores in Table 2, but also the speed and quality of optimization during training.

Loss versus communication. The plots examine how validation loss decreases as cumulative normalized communication grows. FEDSUBMUON traces the strongest lower-left frontier on both datasets, reaching the low-loss region with substantially less traffic than the LoRA baselines. FedKRSO also shows a pronounced initial loss drop, but each round communicates more parameters than the LoRA baselines, shifting its curve to the right on the communication axis. FEDSUBMUON-GT shifts this frontier by spending extra basis-refresh communication on tracked subspace bases. This places FEDSUBMUON-GT to the right of FEDSUBMUON in the communication-axis panels, reflecting the cost of basis tracking.

4.5 Communication and Computational Efficiency

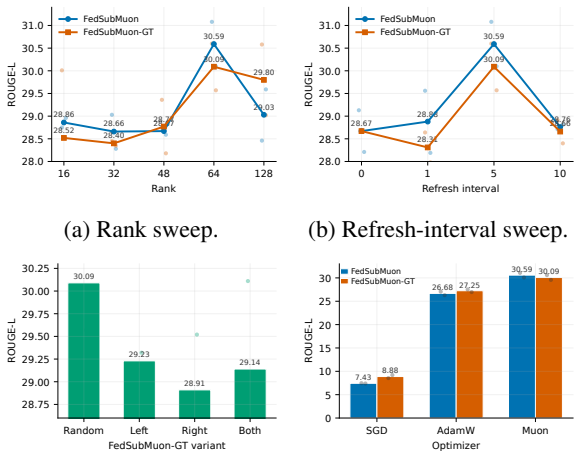
This subsection analyzes Table 4 to compare the communication and computational efficiency of the proposed methods against the baselines. All measurements use the same Dolly-15K settings as Table 2. The total-communication entries report normalized communication, with parenthetical ratios relative to FEDSUBMUON for each backbone; uplink traffic is reported separately. GPU memory denotes the peak footprint, and wall time denotes the average per-round runtime.

Communication efficiency. FEDSUBMUON is the clear communication minimum on both backbones. The closest alternative is FLoRG, which still needs $5.5\times$ more communication on Llama 1B and $1.4\times$ more on Qwen 4B. The LoRA-style baselines and FedKRSO require substantially larger budgets; for example, FedIT uses $7.0\times$ and $2.7\times$ more communication than FEDSUBMUON on the two backbones, while FedKRSO uses $325\times$ and $326\times$ more because it requires broadcasting a full selected-layer matrix update in each round. FEDSUBMUON-GT spends additional communication on gradient-tracked subspace refreshes and lazy catch-up packages ($45\times$ on Llama 1B and $24\times$ on Qwen 4B), but this extra budget is paired with the strongest overall accuracy. Importantly, most of this overhead is not uplink traffic: FEDSUBMUON-GT uploads 923 MB on Llama 1B and 7078 MB on Qwen 4B, only 7.8% and 8.4% of its total communication. This matters in cross-device FL, where uplink and downlink resources are asymmetric and uplink is often the tighter client-side bottleneck.

Computational efficiency. The computation-side metrics show that the stronger accuracy of FEDSUBMUON-GT does not come with heavy runtime or memory overhead. FEDSUBMUON and FEDSUBMUON-GT achieve the best and second-best GPU memory on both backbones, and FEDSUBMUON-GT is the fastest method in wall-clock time, with FEDSUBMUON ranking second.

4.6 Ablation Studies

This section isolates the internal design behind the FEDSUBMUON and FEDSUBMUON-GT configurations, including subspace rank, basis refresh, basis initialization, and optimizer choice. Appendix E additionally varies the federation setting. The re-



(a) Rank sweep.

(b) Refresh-interval sweep.

(c) Basis initialization variants for FEDSUBMUON-GT.

(d) Optimizer sweep.

Figure 3: FEDSUBMUON component ablations on Dolly-15K with Llama 1B. Each plotted value is the mean over two runs.

sults in Figure 3 support four conclusions: (1) moderate subspace ranks are sufficient; (2) basis refresh should be used with a reasonable frequency; (3) random basis initialization outperforms SVD-based alternatives; and (4) Muon is important for optimizing the subspace coefficient matrix.

Rank. Figure 3a shows that increasing the subspace rank improves performance up to a point but does not monotonically improve. FEDSUBMUON reaches its best score at rank 64 (30.59), while FEDSUBMUON-GT also performs best at rank 64 (30.09) and remains competitive at rank 128 (29.80). Lower ranks underfit the update space, whereas the rank-128 setting does not provide a clear gain on this backbone.

Basis refresh. Figure 3b evaluates the refresh interval used by FEDSUBMUON-GT. When the refresh interval is zero, FEDSUBMUON-GT performs the same computation as FEDSUBMUON because the basis is never refreshed. Refreshing every five rounds gives the best FEDSUBMUON-GT result (30.09), while refreshing too frequently ($\tau = 1$) or too rarely ($\tau = 10$) reduces performance to 28.31 and 28.66, respectively. This pattern indicates that basis refresh is useful when it periodically expands the explored subspace, but overly frequent refreshes can destabilize local optimization.

SVD initialization. Figure 3c compares the default random initialization used by FEDSUBMUON-GT against SVD-based initialization variants. Instead of drawing all subspace bases from random seeds, the SVD variants initialize one or both bases from the singular vectors of the frozen pretrained matrix W_ℓ^0. Concretely, if $W_\ell^0 \approx \widetilde{U}_\ell \Sigma_\ell \widetilde{V}_\ell^\top$, the left variant initializes U_ℓ^0 with the leading columns of $\widetilde{U}_\ell$ while keeping V_ℓ^0 random, the right variant initializes V_ℓ^0 with the leading columns of $\widetilde{V}_\ell$ while keeping U_ℓ^0 random, and the both variant initializes both sides from the corresponding singular-vector bases. This test follows the same motivation as SVD-based initializations, which replace random adapter directions with principal weight or activation subspaces (Yan et al., 2026; Meng et al., 2024; Paischer et al., 2025). The random initialization gives the highest score overall (30.09), exceeding the left (29.23), both (29.14), and right (28.91) variants. Among the three SVD-based variants, initializing the left basis performs best. This result indicates that, in this setting, the default random subspace is more effective than replacing it with singular-vector initialization.

Optimizer. Figure 3d shows that the optimizer choice has the largest effect. SGD fails to make meaningful progress, reaching only 7.43 for FEDSUBMUON and 8.88 for FEDSUBMUON-GT. AdamW is substantially better (26.68 and 27.25), but still trails Muon by 3.91 and 2.84 ROUGE-L points. Muon therefore provides a critical optimization advantage for the subspace coefficient matrix, which explains why the main experiments use Muon for both FEDSUBMUON variants.

5 Conclusion

We introduced FEDSUBMUON, a communication-efficient realization of federated Muon that moves matrix-aware optimization from full weight matrices to compact structured subspace coefficients. This design keeps Muon on a single matrix-valued trainable object and reduces client upload to compact coefficient matrices. Empirically, while FEDSUBMUON is the strongest communication-efficient variant, FEDSUBMUON-GT further adds projected-gradient basis tracking and achieves the strongest overall accuracy. These results show that structured subspace coefficients provide an effective path for bringing Muon matrix optimization to federated LLM fine-tuning.

Limitations

The primary limitation of this work is that FEDSUBMUON is evaluated in controlled federated fine-tuning settings rather than in a real-world cross-device deployment. Although our experiments

cover instruction tuning and reasoning tasks with 1B–8B LLMs, practical deployments may involve larger models, more heterogeneous client hardware, unstable network conditions, stragglers, and stricter memory constraints. These factors can affect the efficiency of client synchronization, basis refresh, and update aggregation. We leave large-scale deployment studies with more diverse client environments and larger foundation models to future work.

Another limitation is that our experiments focus on adapting selected attention projection matrices. This setting follows common PEFT practice for LLM fine-tuning, but it does not exhaust all possible adaptation targets or model architectures. Other modules, larger adaptation scopes, or different backbone families may exhibit different accuracy–communication trade-offs. We believe the design principle of optimizing compact matrix-valued subspace coefficients is broadly applicable, but a systematic exploration of broader adaptation targets and model scales remains future work.

Finally, federated learning keeps raw examples decentralized, but it does not by itself provide a formal privacy guarantee. Model updates, coefficient matrices, or gradient-related information may still leak information under strong attacks. Practical deployments on sensitive data should combine methods such as FEDSUBMUON with appropriate privacy and security mechanisms, such as secure aggregation and differential privacy.

Ethical Considerations

We propose a communication-efficient federated fine-tuning framework that reduces communication cost while keeping raw client data decentralized. All experiments in this paper use public benchmark datasets, and we do not collect new personal data or conduct human-subject studies. We have not identified specific additional risks arising from the proposed method beyond those commonly associated with federated learning and LLM fine-tuning.

Acknowledgments

Shuzhen Chen is supported in part by the Fundamental Research Funds for the Central Universities under Grant 202513024, in part by the Qingdao Postdoctoral Research Project under Grant No. QDBSH20250102012. Di Wang is supported in part by the funding BAS/1/1689-01-01, RGC/3/7125-01-01, FCC/1/5940-20-05, FCC/1/5940-06-02, and King Abdullah University of Science and Technology (KAUST) – Center of Excellence for Generative AI, under award number 5940 and a gift from Google.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Vladimir Bogachev, Vladimir Aletov, Alexander Molozhavenko, Denis Bobkov, Vera Soboleva, Aibek Alanov, and Maxim Rakhuba. 2026. LoRA meets riemannion: Muon optimizer for parametrization-independent low-rank adapters. In *International Conference on Learning Representations*.

Rui Chen, Qiyu Wan, Xinyue Zhang, Xiaoqi Qin, Yanzhao Hou, Di Wang, Xin Fu, and Miao Pan. 2026. Wireless-aware energy-efficient federated learning over mobile devices via algorithm and hardware co-design. *IEEE Transactions on Networking*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

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. Databricks blog. Blog post.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Vineet Gupta, Tomer Koren, and Yoram Singer. 2018. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1842–1850. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Keller Jordan. 2024. Muon: An optimizer for hidden layers in neural networks. Keller Jordan blog. Blog post.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Han Liu, Ruoyao Wen, Srijith Nair, Jia Liu, Wenjing Lou, Chongjie Zhang, William Yeoh, Yevgeniy Vorobeychik, and Ning Zhang. 2025a. EcoLoRA: Communication-efficient federated fine-tuning of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20732–20746, Suzhou, China. Association for Computational Linguistics.

Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. 2024. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *International Conference on Learning Representations*.

Junkang Liu, Fanhua Shang, Junchao Zhou, Hongying Liu, Yuanyuan Liu, and Jin Liu. 2025b. FedMuon: Accelerating federated learning with matrix orthogonalization. *Preprint*, arXiv:2510.27403.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR.

Chuiyang Meng, Ming Tang, and Vincent W.S. Wong. 2026. FLoRG: Federated fine-tuning with low-rank gram matrices and procrustes alignment. In *The Fourteenth International Conference on Learning Representations*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, Dublin, Ireland. Association for Computational Linguistics.

Fabian Paischer, Lukas Hauzenberger, Thomas Schmied, Benedikt Alkin, Marc Peter Deisenroth, and Sepp Hochreiter. 2025. Parameter efficient fine-tuning via explained variance adaptation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Sahar Rajabi, Nayeema Nonta, and Sirisha Rambhatla. 2025. Subtrack++: Gradient subspace tracking for scalable LLM training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Raghav Singhal, Kaustubh Ponkshe, and Praneeth Vepakomma. 2025. FedEx-LoRA: Exact aggregation for federated and efficient fine-tuning of large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1316–1336, Vienna, Austria. Association for Computational Linguistics.

Yuki Takezawa, Anastasia Koloskova, Xiaowen Jiang, and Sebastian U. Stich. 2026. FedMuon: Federated learning with bias-corrected LMO-based optimization. In *International Conference on Learning Representations*.

Youming Tao, Cheng-Long Wang, Miao Pan, Dongxiao Yu, Xiuzhen Cheng, and Di Wang. 2024. Communication efficient and provable federated unlearning. *Proceedings of the VLDB Endowment*, 17(5):1119–1131.

Youming Tao, Zuyuan Zhang, Di Wang, Dongxiao Yu, Xiuzhen Cheng, and Falko Dressler. 2026. Byzantine-resilient federated learning under heterogeneity and heavy tails. *IEEE Transactions on Networking*.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. 2025. SOAP: Improving and stabilizing shampoo using adam for language modeling. In *International Conference on Learning Representations*.

Liangyu Wang, Siqi Zhang, Junjie Wang, Yiming Dong, Bo Zheng, Zihan Qiu, Shengkun Tang, Di Wang, Rui Men, and Dayiheng Liu. 2026. Canzona: A unified, asynchronous, and load-balanced framework for distributed matrix-based optimizers. *arXiv preprint arXiv:2602.06079*.

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 Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, and 16 others. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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. In *Advances in Neural Information Processing Systems*, volume 37, pages 22513–22533. Curran Associates, Inc.

Zihang Xiang, Tianhao Wang, Wanyu Lin, and Di Wang. 2023. Practical differentially private and byzantine-resilient federated learning. *Proceedings of the ACM on Management of Data*, 1(2):1–26.

Yuxuan Yan, Shunpu Tang, Yuanchao Shu, Zhiguo Shi, and Qianqian Yang. 2026. FedeRA: Efficient fine-tuning of language models in federated learning leveraging weight decomposition. In *AAAI 2026 Workshop on Machine Learning for Wireless Communication and Networks (ML4Wireless)*.

Guohao Yang, Tongle Wu, Yuanxiong Guo, Ying Sun, and Yanmin Gong. 2026. FedKRSO: Communication and memory efficient federated fine-tuning of large language models. In *IEEE INFOCOM 2026 - IEEE Conference on Computer Communications*, pages 1–10.

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.

Xinwen Zhang and Hongchang Gao. 2025. On provable benefits of muon in federated learning. *Preprint*, arXiv:2510.03866.

Zhong Zhang, Bang Liu, and Junming Shao. 2023a. Fine-tuning happens in tiny subspaces: Exploring intrinsic task-specific subspaces of pre-trained language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1701–1713, Toronto, Canada. Association for Computational Linguistics.

Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. 2023b. FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9963–9977, Toronto, Canada. Association for Computational Linguistics.

A Related Work

We situate FEDSUBMUON relative to three lines of work: communication-efficient federated fine-tuning, random subspace optimization, and matrix-aware optimizers.

A.1 Federated Fine-Tuning

Adapting LLMs under federated constraints requires minimizing per-round communication while preserving adaptation quality. Most approaches combine federated averaging with parameter-efficient tuning to reduce the communicated state (Zhang et al., 2023b, 2024; Tao et al., 2026; Chen et al., 2026; Tao et al., 2024; Xiang et al., 2023).

LoRA-based methods communicate factorized adapter states. FeDeRA initializes adapters from pretrained weight decomposition (Yan et al., 2026); FLoRA supports heterogeneous per-client ranks (Wang et al., 2024); FedEx-LoRA corrects the aggregation mismatch that arises from separately averaging low-rank factors (Singhal et al., 2025). A complementary line replaces factorized adapters with compact subspace coordinates generated from shared random seeds: FLoRG communicates Gram-matrix subspace coordinates (Meng et al., 2026), and FedKRSO uses random subspace projections with memory-efficient aggregation (Yang et al., 2026).

FEDSUBMUON is most closely related to this seeded-subspace group, but differs in two respects: the server maintains an ambient-space matrix accumulator rather than a low-rank adapter, and clients apply Muon to a matrix-valued coefficient matrix rather than optimizing scalar subspace coordinates.

A.2 Random Subspace Optimization

The intrinsic-dimension framework establishes that neural networks can reach competitive task performance when optimization is restricted to a low-dimensional random subspace (Li et al., 2018). For LLMs, this premise extends to fine-tuning: effective adaptation can be achieved in surprisingly small subspaces of the pretrained parameter space (Aghajanyan et al., 2021), and empirical analyses show that fine-tuning trajectories concentrate on compact task-relevant directions (Zhang et al., 2023a). LoRA can be understood as a structured instantiation of this principle, tying the trainable budget to a low-rank factorization (Hu et al., 2022; Ding et al., 2023).

FEDSUBMUON uses the low-dimensionality premise differently from standard LoRA. The subspace coefficient matrix is a per-round optimization variable rather than a persistent adapter: committed updates accumulate in the ambient matrix, so refreshing or tracking the bases redirects future

search without discarding previously accumulated gradient information.

A.3 Muon and Matrix-Aware Optimizers

Matrix-aware optimizers exploit structure invisible to coordinate-wise methods. Shampoo maintains per-dimension Kronecker-factor preconditioners (Gupta et al., 2018); Sophia incorporates lightweight diagonal Hessian estimates (Liu et al., 2024); SOAP stabilizes Shampoo-style preconditioning by maintaining Adam moments in the preconditioner eigenbasis (Vyas et al., 2025). These methods improve optimization geometry but require maintaining and computing preconditioning matrices that introduce memory and computational overhead beyond the weights themselves.

Muon offers a lighter alternative by orthogonalizing momentum via Newton–Schulz iterations and applying the resulting matrix direction to hidden-layer weights (Jordan, 2024). Recent work on federated Muon analyzes convergence and acceleration under matrix-orthogonalized updates (Liu et al., 2025b; Takezawa et al., 2026; Zhang and Gao, 2025), but these methods operate on full weight matrices, incurring full layer-size communication per round. Applying Muon directly to LoRA factors is not equivalent to optimizing the underlying update matrix, because the factors are non-unique coordinates of a low-rank product (Bogachev et al., 2026). FEDSUBMUON resolves this by applying Muon to a $r \times r$ subspace coefficient matrix, preserving Muon's matrix-update geometry while reducing the federated payload to $O(r^2)$ per layer.

B Algorithms

Algorithms 1 and 2 summarize FEDSUBMUON and FEDSUBMUON-GT, respectively.

Algorithm 1 FedSubMuon

Require: Backbone W^0, rounds T, local steps E, client weights $\{p_i\}_{i=1}^N$, selected-matrix index set $\mathcal{L}$, subspace rank r, refresh interval τ, learning rate η

Initialize global update accumulators $\Delta W_\ell^0 \leftarrow 0$, local versions $v_i \leftarrow 0$, local update accumulators $\Delta W_{i,\ell} \leftarrow 0$, trainable subspace coefficient matrices $X_\ell^0 \leftarrow 0$, random seeds s^0, commit log $\mathcal{H} \leftarrow [\,]$, counter $K \leftarrow 0$
for $t = 0, \dots, T-1$ **do**
  Server: Sample clients $\mathcal{S}_t$ and set $a_i^t = p_i / \sum_{j \in \mathcal{S}_t} p_j$; send $\{s^t, X^t\}$ and missing commit records $\{\mathcal{C}_k\}_{k=v_i}^{K-1}$ to each $i \in \mathcal{S}_t$
  // Client synchronization
  for all $i \in \mathcal{S}_t$ **in parallel do**
    For each $\mathcal{C}_k$, generate $(U_\ell^{[k]}, V_\ell^{[k]})$ from $s^{[k]}$ and apply $\Delta W_{i,\ell} \leftarrow \Delta W_{i,\ell} + U_\ell^{[k]} X_\ell^{[k]} (V_\ell^{[k]})^\top$ for all $\ell \in \mathcal{L}$; set $v_i \leftarrow K$
    Generate (U_ℓ^t, V_ℓ^t) from seed s^t; Initialize $X_{i,\ell}^{t,0}$, $M_{i,\ell}^{t,0}$, $W_{i,\ell}(X_{i,\ell}^{t,0})$ by Eqs. (1)–(2)
    // Local Muon update of the subspace coefficient matrix
    Run E local Muon steps on $X_{i,\ell}$ using Eqs. (3)–(4); set $X_{i,\ell}^{t+1} \leftarrow X_{i,\ell}^{t,E}$
    Upload $X_{i,\ell}^{t+1}$ for all $\ell \in \mathcal{L}$
  end for
  // Server aggregation
  Set $\bar{X}_\ell^{t+1} \leftarrow \sum_{i \in \mathcal{S}_t} a_i^t X_{i,\ell}^{t+1}$ for all $\ell \in \mathcal{L}$
  // Refresh
  if $(t+1) \bmod \tau = 0$ **then**
    $\mathcal{C}_K \leftarrow (s^{[K]} = s^t, \{X_\ell^{[K]} = \bar{X}_\ell^{t+1}\}_{\ell \in \mathcal{L}})$ Append $\mathcal{C}_K$ to $\mathcal{H}$; set $K \leftarrow K+1$
    $\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t + U_\ell^t \bar{X}_\ell^{t+1} (V_\ell^t)^\top$ for all $\ell \in \mathcal{L}$; set $X_\ell^{t+1} \leftarrow 0$
    Sample fresh random seeds s^{t+1}
  else
    Keep $\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t$, $s^{t+1} \leftarrow s^t$ Set $X_\ell^{t+1} \leftarrow \bar{X}_\ell^{t+1}$
  end if
end for
return $W^0 + \Delta W_\ell^T + U_\ell^T X_\ell^T (V_\ell^T)^\top$

C Hyperparameter Settings

For Natural Instructions, we use 738 clients and sample 3% of clients per communication round. For Dolly-15K, we use 200 clients with 5% participation. For GSM8K and MATH, we use 100 clients with 10% participation.

For all experiments, each participating client uses a local batch size of 1. All inputs are tokenized with a maximum length of 1024. Each client runs 100 local optimization steps on Natural Instructions and one local epoch per communication round on the remaining datasets. We apply early stopping with patience 5 and min-delta 8.For the baseline methods, we use AdamW and search the learning rate over $\{10^{-4}, 5\times10^{-5}, 10^{-5}, 5\times10^{-6}\}$. The final learning rate is 10^{-5} for FedKRSO and 5×10^{-5} for all other baselines. For all methods, we set the LoRA scaling factor to 1.

For FEDSUBMUON and FEDSUBMUON-GT, we sweep the coefficient-matrix learning rate over $\{10^{-3}, 2\times10^{-3}, 5\times10^{-3}, 10^{-2}\}$. FEDSUBMUON uses a final coefficient-matrix learning rate of 2×10^{-3} on Natural Instructions and 5×10^{-3} on the remaining datasets. For FEDSUBMUON-GT, we additionally search the subspace learning rate over $\{5\times10^{-3}, 10^{-2}, 5\times10^{-2}, 10^{-1}\}$. On Natural Instructions, the final hyperparameters are a coefficient-matrix learning rate of 2×10^{-3} and a subspace learning rate of 10^{-2}. On the remaining datasets, FEDSUBMUON-GT uses a coefficient-matrix learning rate of 5×10^{-3} and a subspace learning rate of 5×10^{-2}.

Algorithm 2 FedSubMuon-GT

Require: Backbone W^0, rounds T, local steps E, client weights $\{p_i\}_{i=1}^N$, selected-matrix index set $\mathcal{L}$, subspace rank r, refresh interval τ, learning rate η, subspace learning rate ρ

Initialize global update accumulators $\Delta W_\ell^0 \leftarrow 0$, local versions $v_i \leftarrow 0$, local update accumulators $\Delta W_{i,\ell} \leftarrow 0$, trainable subspace coefficient matrices $X_\ell^0 \leftarrow 0$, random seed s^0 for the initial subspace, commit log $\mathcal{H} \leftarrow [\,]$, counter $K \leftarrow 0$
for $t = 0, \dots, T-1$ **do**
  Server: Sample clients $\mathcal{S}_t$ and set $a_i^t = p_i / \sum_{j \in \mathcal{S}_t} p_j$; send $\{s^t, X^t\}$ if $t < \tau$, otherwise send $\{U^t, V^t, X^t\}$, and send missing commit records $\{\mathcal{C}_k\}_{k=v_i}^{K-1}$ to each $i \in \mathcal{S}_t$
  // Client synchronization
  for all $i \in \mathcal{S}_t$ **in parallel do**
    For each $\mathcal{C}_k$, apply $\Delta W_{i,\ell} \leftarrow \Delta W_{i,\ell} + U_\ell^{[k]} X_\ell^{[k]} (V_\ell^{[k]})^\top$ for all $\ell \in \mathcal{L}$; set $v_i \leftarrow K$
    Generate (U_ℓ^t, V_ℓ^t) from seed s^t if $t < \tau$; otherwise use received tracked subspace bases (U_ℓ^t, V_ℓ^t); Initialize $X_{i,\ell}^{t,0}$, $M_{i,\ell}^{t,0}$, $W_{i,\ell}(X_{i,\ell}^{t,0})$ by Eqs. (1)–(2)
    // Local Muon update of the subspace coefficient matrix
    Run E local Muon steps on $X_{i,\ell}$ using Eqs. (3)–(4); set $X_{i,\ell}^{t+1} \leftarrow X_{i,\ell}^{t,E}$
    if $(t+1) \bmod \tau = 0$ **then**
      Form projected gradients by Eq. (6)
      Upload $\{X_{i,\ell}^{t+1}, H_{U,i,\ell}^t, H_{V,i,\ell}^t\}_{\ell \in \mathcal{L}}$
    else
      Upload $\{X_{i,\ell}^{t+1}\}_{\ell \in \mathcal{L}}$
    end if
  end for
  // Server aggregation
  Set $\bar{X}_\ell^{t+1} \leftarrow \sum_{i \in \mathcal{S}_t} a_i^t X_{i,\ell}^{t+1}$ for all $\ell \in \mathcal{L}$
  // Refresh
  if $(t+1) \bmod \tau = 0$ **then**
    $\mathcal{C}_K \leftarrow (\{U_\ell^{[K]} = U_\ell^t, V_\ell^{[K]} = V_\ell^t, X_\ell^{[K]} = \bar{X}_\ell^{t+1}\}_{\ell \in \mathcal{L}})$; Append $\mathcal{C}_K$ to $\mathcal{H}$; set $K \leftarrow K+1$
    $\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t + U_\ell^t \bar{X}_\ell^{t+1} (V_\ell^t)^\top$ for all $\ell \in \mathcal{L}$; set $X_\ell^{t+1} \leftarrow 0$
    Aggregate projected gradients by Eq. (7); form least-squares coordinates and tangent directions by Eqs. (8a)–(8b); update $(U_\ell^{t+1}, V_\ell^{t+1})$ by Eq. (9)
  else
    Keep $\Delta W_\ell^{t+1} \leftarrow \Delta W_\ell^t$, keep $s^{t+1} \leftarrow s^t$ if $t+1 < \tau$ and $(U_\ell^{t+1}, V_\ell^{t+1}) \leftarrow (U_\ell^t, V_\ell^t)$ otherwise, and set $X_\ell^{t+1} \leftarrow \bar{X}_\ell^{t+1}$
  end if
end for
return $W^0 + \Delta W_\ell^T + U_\ell^T X_\ell^T (V_\ell^T)^\top$

D Dataset Processing and Prompt Templates

This section summarizes the dataset preprocessing pipeline used in our codebase. For the Qwen backbones, each example is formatted as a single-turn chat request and then processed with the model's chat template. For the non-Qwen backbone, we use plain-text instruction prompts.

Dolly-15K. Each example is normalized into an instruction, an optional context field used as input, a response target, and the original category label. The prompt for Llama follows the standard Alpaca format with an instruction block, an optional input block when the context is non-empty, and a response header. For Qwen, the same content is converted into a user message containing the instruction followed by the optional input field before applying the chat template. Dolly-15K contains eight task categories: brainstorming, classification, closed QA, creative writing, general QA, information extraction, open QA, and summarization. In our split, the summarization category is used as the evaluation set, and the remaining seven categories form the federated training set, which is then partitioned across clients according to the specified IID or Dirichlet setting. The default partition is Dirichlet with concentration 0.5.

Table 5: Per-round per-client trainable parameter counts under the q/v-only adaptation setup.

Model	Method group	Rank	Trainable params (ratio)
Llama 1B	FedSubMuon/ FedSubMuon-GT	64	131,072 (1×)
	FedIT/FeDeRA/FLoRA/ FedEx-LoRA	8	851,968 (6.5×)
	FLoRG	8	327,680 (2.5×)
	FedKRSO	8	1,310,720 (10×)
Qwen 4B	FedSubMuon/ FedSubMuon-GT	128	1,179,648 (1×)
	FedIT/FeDeRA/FLoRA/ FedEx-LoRA	8	2,949,120 (2.5×)
	FLoRG	8	1,032,192 (0.88×)
	FedKRSO	8	5,898,240 (5×)
Qwen 8B	FedSubMuon/ FedSubMuon-GT	256	4,718,592 (1×)
	FedIT/FeDeRA/FLoRA/ FedEx-LoRA	12	5,750,784 (1.22×)
	FLoRG	12	2,211,840 (0.47×)
	FedKRSO	12	8,847,360 (1.88×)

Natural Instructions. Each example is converted into a triple consisting of the task definition, the instance input, and the first reference output. For the Llama backbone, the task definition and instance input are formatted as plain-text instructions; for Qwen, the same content is converted into an equivalent single-turn chat request. Beyond token truncation, we also discard examples whose raw input text is excessively long before tokenization. After filtering, training tasks with fewer than 20 examples are removed; each retained training task is then subsampled to 20% of its examples; and each evaluation task is subsampled to $\max(20, 2\%)$ examples when it contains more than 20 instances. In the federated split, each retained training task is treated as one client.

GSM8K. Each sample is reformatted into the fixed instruction "Solve the following grade-school math problem step by step," with the question as input and the annotated answer as target output.

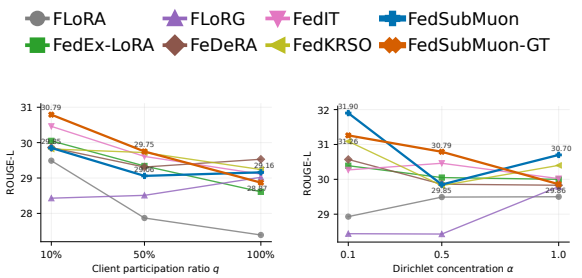
For the Llama backbone, the resulting sample is formatted as a plain-text instruction example, while Qwen receives the corresponding single-turn chat request. The official training split is partitioned for federated training, and the official test split is used for both eval and test. Evaluation is measured by exact-match accuracy on the final answer.

MATH. Each problem is paired with the fixed instruction "Solve the following competition math problem. Show your reasoning clearly and put the final answer in \boxed{}," using the problem statement as input and the full solution as target output. The preprocessing also stores the extracted boxed final answer, together with the subject and difficulty-level metadata. As with GSM8K, the resulting sample is formatted as plain-text instruction data for Llama and as a single-turn chat request for Qwen. For evaluation, the official training split is divided into a federated training partition and a development set of size 500; the development set is used for evaluation, and the official test split is used for test. Performance is measured by exact-match accuracy on the extracted boxed answer.

E Additional Ablation Studies

This section evaluates whether the FEDSUBMUON variants remain robust when the federation setting changes, complementing the component ablations in Section 4.6. Figure 4 varies client participation and client-data heterogeneity on Dolly-15K with Llama 1B, showing how the methods behave under different partial-participation and non-IID regimes. Both panels in Figure 4 use 20 total clients. In Figure 4a, participation ratios $q = 0.1$, 0.5, and 1.0 correspond to low, medium, and full participation, respectively, with fixed $\alpha = 0.5$. In Figure 4b, the Dirichlet concentration α controls data heterogeneity while q is fixed to 0.1; smaller α indicates a stronger non-IID split.

Client participation ratio. Figure 4a sweeps the client participation ratio across partial and full participation. FEDSUBMUON-GT achieves the best result at low and medium participation, reaching 30.79 at $q = 0.1$ and 29.75 at $q = 0.5$. Relative to FEDSUBMUON, the gain from basis refresh is also large under partial participation: $+0.94$ at $q = 0.1$ and $+0.69$ at $q = 0.5$. At full participation, however, most methods perform similarly. This pattern suggests that the refreshed subspace is most useful in the partial-participation regime targeted by cross-device federated learning, whereas under full participation the aggregation problem becomes easier and the methods become more comparable.



(a) Client participation ratio. (b) Client-data distribution.

Figure 4: Federation ablations over client participation and data heterogeneity on Dolly-15K with Llama 1B; each plotted value is averaged over two runs.

Client-data distribution. Figure 4b sweeps the Dirichlet concentration in the fixed-participation setting. FEDSUBMUON is the strongest method at both ends of the heterogeneity range, reaching 31.90 at $\alpha = 0.1$ and 30.70 at $\alpha = 1.0$. These scores exceed the strongest baseline by 1.33 and 0.30 ROUGE-L, respectively. FEDSUBMUON-GT instead peaks at the intermediate default setting $\alpha = 0.5$, where it reaches 30.79 and improves over the best baseline by 0.33 ROUGE-L. Together, these results show that the FEDSUBMUON variants remain competitive across most client-data heterogeneity settings, from highly skewed splits to distributions closer to IID.