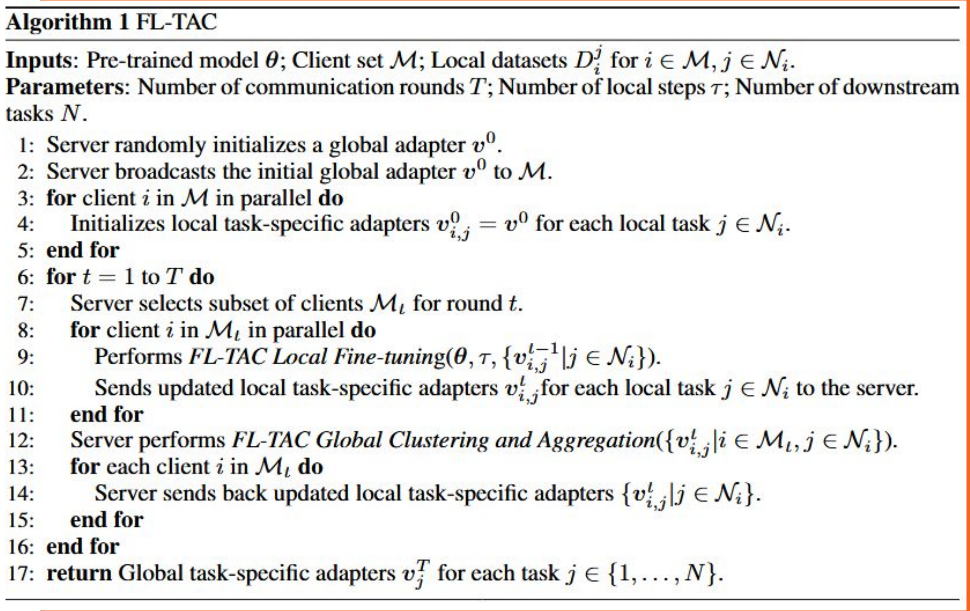
LoRA in Federated Learning

Amirparsa Safari

FL-TAC

- The study addresses the challenge of fine-tuning large-scale pre-trained models within federated learning (FL) frameworks, focusing on reducing communication overhead while maintaining effective task adaptation.
- The primary goal is to develop a parameter-efficient method that enables clients to fine-tune models for multiple downstream tasks using low-rank adapters, and to effectively aggregate these adapters across clients.

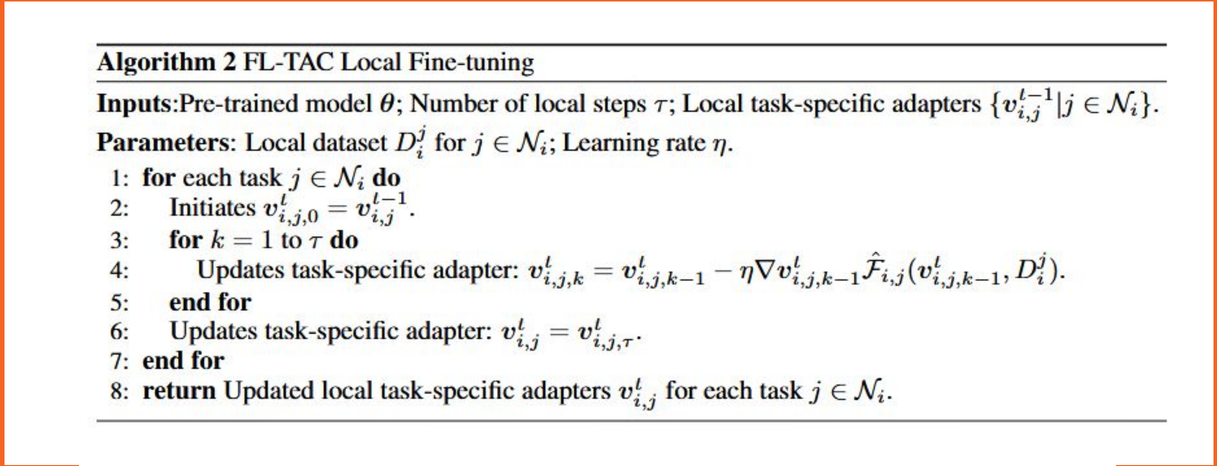


Algorithm 1 FL-TAC

Inputs: Pre-trained model $\boldsymbol{\theta}$; Client set $\mathcal{M}$; Local datasets D_i^j for $i \in \mathcal{M}, j \in \mathcal{N}_i$.
Parameters: Number of communication rounds T; Number of local steps τ; Number of downstream tasks N.

```
1:  Server randomly initializes a global adapter $\boldsymbol{v}^0$.
2:  Server broadcasts the initial global adapter $\boldsymbol{v}^0$ to $\mathcal{M}$.
3:  **for** client $i$ in $\mathcal{M}$ in parallel **do**
4:      Initializes local task-specific adapters $\boldsymbol{v}_{i,j}^0 = \boldsymbol{v}^0$ for each local task $j \in \mathcal{N}_i$.
5:  **end for**
6:  **for** $t = 1$ to $T$ **do**
7:      Server selects subset of clients $\mathcal{M}_t$ for round $t$.
8:      **for** client $i$ in $\mathcal{M}_t$ in parallel **do**
9:          Performs *FL-TAC Local Fine-tuning*$(\boldsymbol{\theta}, \tau, \{\boldsymbol{v}_{i,j}^{t-1} | j \in \mathcal{N}_i\})$.
10:         Sends updated local task-specific adapters $\boldsymbol{v}_{i,j}^t$ for each local task $j \in \mathcal{N}_i$ to the server.
11:     **end for**
12:     Server performs *FL-TAC Global Clustering and Aggregation*$(\{\boldsymbol{v}_{i,j}^t | i \in \mathcal{M}_t, j \in \mathcal{N}_i\})$.
13:     **for** each client $i$ in $\mathcal{M}_t$ **do**
14:         Server sends back updated local task-specific adapters $\{\boldsymbol{v}_{i,j}^t | j \in \mathcal{N}_i\}$.
15:     **end for**
16: **end for**
17: **return** Global task-specific adapters $\boldsymbol{v}_j^T$ for each task $j \in \{1, \ldots, N\}$.
```

FL-TAC

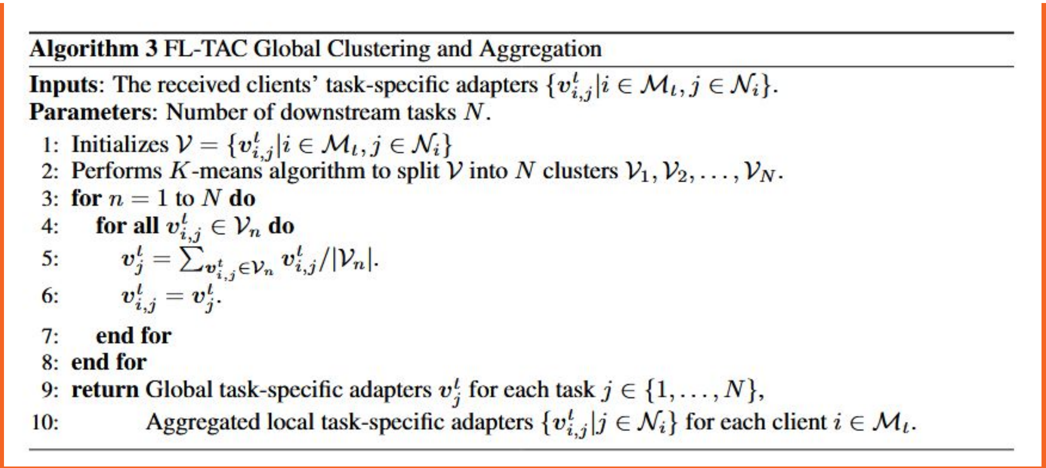


Algorithm 2 FL-TAC Local Fine-tuning

Inputs:Pre-trained model $\boldsymbol{\theta}$; Number of local steps τ; Local task-specific adapters $\{\boldsymbol{v}_{i,j}^{\iota-1} | j \in \mathcal{N}_i\}$.

Parameters: Local dataset D_i^j for $j \in \mathcal{N}_i$; Learning rate η.

```
1: for each task $j \in \mathcal{N}_i$ do
2:     Initiates $\boldsymbol{v}_{i,j,0}^{\iota} = \boldsymbol{v}_{i,j}^{\iota-1}$.
3:     for $k = 1$ to $\tau$ do
4:         Updates task-specific adapter: $\boldsymbol{v}_{i,j,k}^{\iota} = \boldsymbol{v}_{i,j,k-1}^{\iota} - \eta \nabla \boldsymbol{v}_{i,j,k-1}^{\iota} \hat{\mathcal{F}}_{i,j}(\boldsymbol{v}_{i,j,k-1}^{\iota}, D_i^j)$.
5:     end for
6:     Updates task-specific adapter: $\boldsymbol{v}_{i,j}^{\iota} = \boldsymbol{v}_{i,j,\tau}^{\iota}$.
7: end for
8: return Updated local task-specific adapters $\boldsymbol{v}_{i,j}^{\iota}$ for each task $j \in \mathcal{N}_i$.
```



Algorithm 3 FL-TAC Global Clustering and Aggregation

Inputs: The received clients' task-specific adapters $\{\boldsymbol{v}_{i,j}^{\iota} | i \in \mathcal{M}_\iota, j \in \mathcal{N}_i\}$.
Parameters: Number of downstream tasks N.

```
1: Initializes $\mathcal{V} = \{\boldsymbol{v}_{i,j}^{\iota} | i \in \mathcal{M}_\iota, j \in \mathcal{N}_i\}$
2: Performs $K$-means algorithm to split $\mathcal{V}$ into $N$ clusters $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_N$.
3: for $n = 1$ to $N$ do
4:     for all $\boldsymbol{v}_{i,j}^{\iota} \in \mathcal{V}_n$ do
5:         $\boldsymbol{v}_j^{\iota} = \sum_{\boldsymbol{v}_{i,j}^{\iota} \in \mathcal{V}_n} \boldsymbol{v}_{i,j}^{\iota} / |\mathcal{V}_n|$.
6:         $\boldsymbol{v}_{i,j}^{\iota} = \boldsymbol{v}_j^{\iota}$.
7:     end for
8: end for
9: return Global task-specific adapters $\boldsymbol{v}_j^{\iota}$ for each task $j \in \{1, \ldots, N\}$,
10:        Aggregated local task-specific adapters $\{\boldsymbol{v}_{i,j}^{\iota} | j \in \mathcal{N}_i\}$ for each client $i \in \mathcal{M}_\iota$.
```

FedLEASE

- The primary objectives are to develop a method for optimally allocating and selecting LoRA (Low-Rank Adaptation) experts across clients, enabling personalized and efficient model adaptation without excessive communication overhead.
- The significance lies in enhancing domain-specific performance of LLMs in privacy-preserving settings, facilitating scalable, personalized, and resource-efficient federated fine-tuning.

FedLEASE Methodology

1. Core Idea

Client clustering + Adaptive LoRA-based Mixture-of-Experts

2. Expert Creation

- Clients do short local LoRA fine-tuning
- Compute **cosine similarity** of LoRA B matrices
- **Hierarchical clustering** + **silhouette score** → auto-find best number of experts
- Average LoRAs inside each cluster → create global **LoRA experts**

3. Expert Selection (Inference & Training)

- **Adaptive top-M MoE router** → Dynamically picks 1 to M experts per token/client
  Always includes the client's assigned expert + optional others

4. Training Loop

- One-time initial clustering & expert creation
- Clients only train their **assigned LoRA expert + router**
- Server periodically aggregates & updates experts/routers

5. Evaluation Metrics

- Silhouette coefficient → clustering quality
- Cosine similarity → task similarity
- Tested on GLUE & FLAN benchmarks

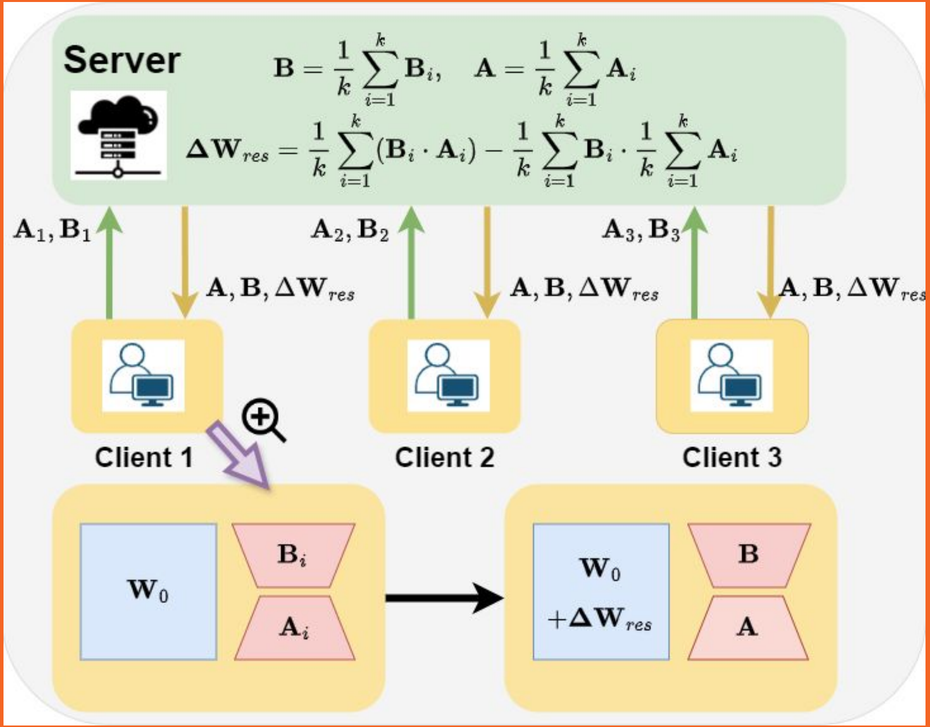
FedEx-LoRA

the problem to answer is **how to do federated fine-tuning with LoRA in a way that yields exact (or near-exact) global updates, preserving the efficacy of fine-tuning while staying efficient in communication and computation.**

Their core question: *Can we adjust the aggregation mechanism in federated LoRA such that the aggregated update across clients matches the "ideal" centralized LoRA update (i.e. the average of the clients' full-updates), while retaining LoRA's parameter efficiency and low communication cost?*To answer it, they propose **FedEx-LoRA**, a modified federated aggregation method: instead of simply averaging the low-rank adapter matrices (as done in the baseline federated-LoRA methods), they compute and transmit an extra **residual (error) term** that accounts for the difference between "average of products" vs "product of averages." That residual is added back to the frozen pretrained base model (not to the low-rank adapters) — thereby restoring the exact global update.
In other words: they developed a solution to eliminate the "inexactness" introduced by naive averaging of low-rank adapters.

Exact (or "ideal") global updates: FedEx-LoRA ensures that the aggregated update after federation matches what would have been obtained by centrally aggregating all client updates — avoiding the "adapter-averaging" mismatch problem.

Potential challenges with rank-heterogeneous settings: The paper notes that extending FedEx-LoRA to settings where different clients use different adapter ranks is nontrivial; the assignment strategy for how to aggregate and assign the low-rank components and residual in that scenario may require further research.


Server
$\mathbf{B} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{B}_i, \quad \mathbf{A} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{A}_i$
$\Delta\mathbf{W}_{res} = \frac{1}{k}\sum_{i=1}^{k}(\mathbf{B}_i \cdot \mathbf{A}_i) - \frac{1}{k}\sum_{i=1}^{k}\mathbf{B}_i \cdot \frac{1}{k}\sum_{i=1}^{k}\mathbf{A}_i$
$\mathbf{A}_1, \mathbf{B}_1$
$\mathbf{A}, \mathbf{B}, \Delta\mathbf{W}_{res}$
$\mathbf{A}_2, \mathbf{B}_2$
$\mathbf{A}, \mathbf{B}, \Delta\mathbf{W}_{res}$
$\mathbf{A}_3, \mathbf{B}_3$
$\mathbf{A}, \mathbf{B}, \Delta\mathbf{W}_{res}$
Client 1
Client 2
Client 3
$\mathbf{W}_0$
$\mathbf{B}_i$
$\mathbf{A}_i$
$\mathbf{W}_0 + \Delta\mathbf{W}_{res}$
$\mathbf{B}$
$\mathbf{A}$

LoRA-FAIR

- The main objectives are to develop a method that effectively combines LoRA with FL by overcoming two key issues: server-side aggregation bias and client-side initialization lag.
- The significance lies in enabling privacy-preserving, computationally efficient fine-tuning of large models across distributed clients, with improved accuracy and reduced communication overhead.

Interventions and Outcomes

- **Interventions:** Implementation of LoRA-FAIR involves reconstructing the ideal global update using a correction term (ΔB), normalizing to preserve shared information, and applying regularization to maintain stability.
- **Measured Outcomes:** Model accuracy across multiple domains, convergence speed, communication overhead, and robustness under non-IID data conditions.
- **Results:** LoRA-FAIR consistently outperforms baseline methods in accuracy and efficiency, effectively mitigating server aggregation bias and client initialization lag without increasing communication costs.

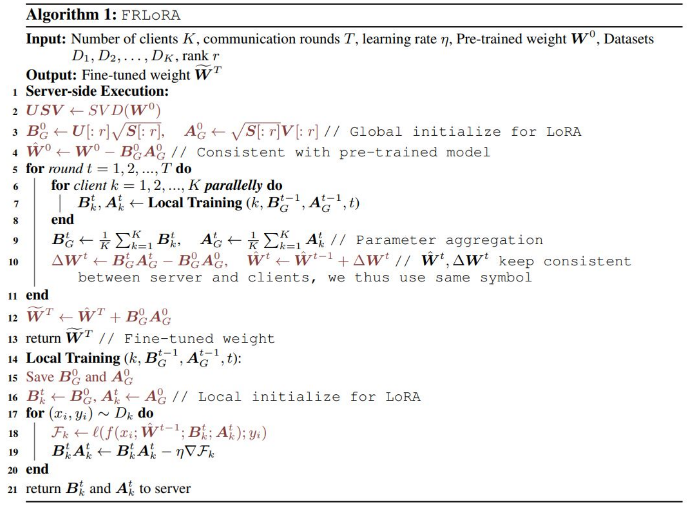
LoRA-FAIR fixes federated LoRA by

(1) adding a tiny ΔB correction so the averaged LoRA behaves like a true global update,

(2) normalizing B matrices so shared knowledge is never destroyed between rounds,

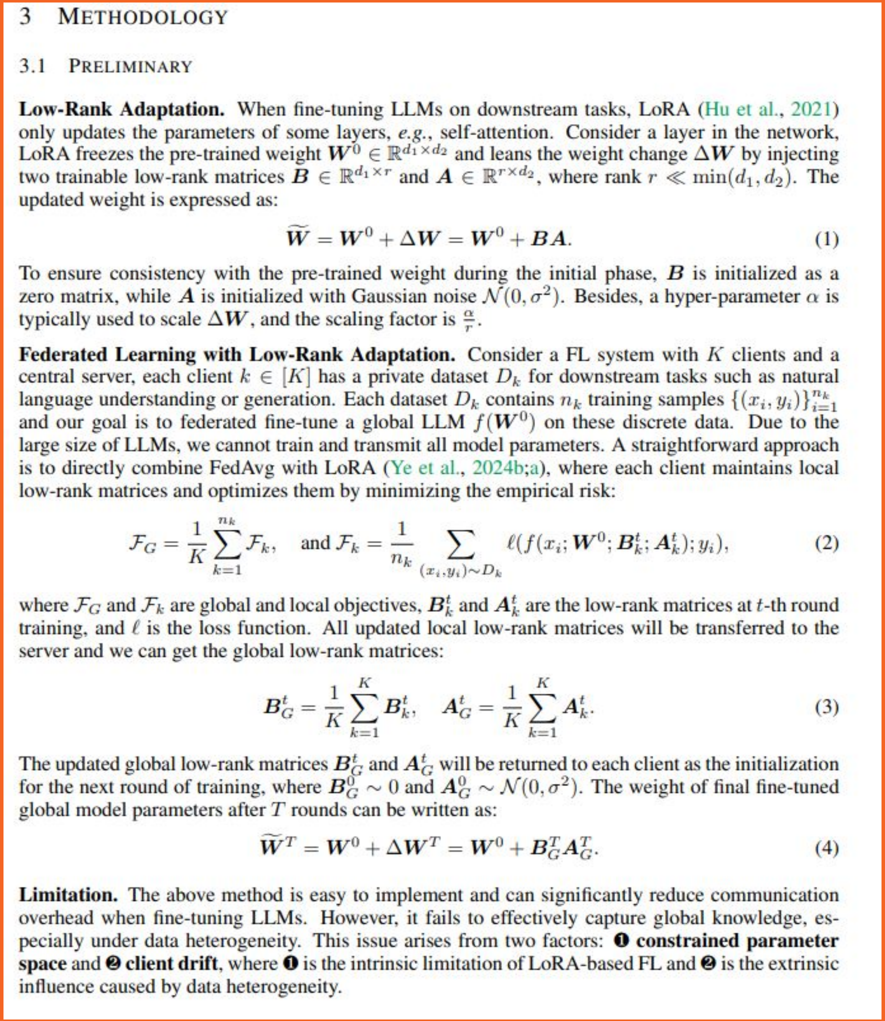
(3) adding light orthogonality + weight decay so training stays stable."

FRLoRA



Algorithm 1: FRLoRA

```
Input: Number of clients $K$, communication rounds $T$, learning rate $\eta$, Pre-trained weight $\boldsymbol{W}^0$, Datasets
       $D_1, D_2, \ldots, D_K$, rank $r$
Output: Fine-tuned weight $\widetilde{\boldsymbol{W}}^T$
Server-side Execution:
$\boldsymbol{USV} \leftarrow SVD(\boldsymbol{W}^0)$
$\boldsymbol{B}_G^0 \leftarrow \boldsymbol{U}[:r]\sqrt{\boldsymbol{S}[:r]}$,   $\boldsymbol{A}_G^0 \leftarrow \sqrt{\boldsymbol{S}[:r]}\boldsymbol{V}[:r]$ // Global initialize for LoRA
$\hat{\boldsymbol{W}}^0 \leftarrow \boldsymbol{W}^0 - \boldsymbol{B}_G^0\boldsymbol{A}_G^0$ // Consistent with pre-trained model
for round $t = 1, 2, ..., T$ do
    for client $k = 1, 2, ..., K$ parallelly do
        $\boldsymbol{B}_k^t, \boldsymbol{A}_k^t \leftarrow$ Local Training $(k, \boldsymbol{B}_G^{t-1}, \boldsymbol{A}_G^{t-1}, t)$
    end
    $\boldsymbol{B}_G^t \leftarrow \frac{1}{K}\sum_{k=1}^K \boldsymbol{B}_k^t$,   $\boldsymbol{A}_G^t \leftarrow \frac{1}{K}\sum_{k=1}^K \boldsymbol{A}_k^t$ // Parameter aggregation
    $\Delta\boldsymbol{W}^t \leftarrow \boldsymbol{B}_G^t\boldsymbol{A}_G^t - \boldsymbol{B}_G^0\boldsymbol{A}_G^0$,   $\hat{\boldsymbol{W}}^t \leftarrow \hat{\boldsymbol{W}}^{t-1} + \Delta\boldsymbol{W}^t$ // $\hat{\boldsymbol{W}}^t, \Delta\boldsymbol{W}^t$ keep consistent
        between server and clients, we thus use same symbol
end
$\widetilde{\boldsymbol{W}}^T \leftarrow \hat{\boldsymbol{W}}^T + \boldsymbol{B}_G^0\boldsymbol{A}_G^0$
return $\widetilde{\boldsymbol{W}}^T$ // Fine-tuned weight
Local Training $(k, \boldsymbol{B}_G^{t-1}, \boldsymbol{A}_G^{t-1}, t)$:
Save $\boldsymbol{B}_G^0$ and $\boldsymbol{A}_G^0$
$\boldsymbol{B}_k^t \leftarrow \boldsymbol{B}_G^0$, $\boldsymbol{A}_k^t \leftarrow \boldsymbol{A}_G^0$ // Local initialize for LoRA
for $(x_i, y_i) \sim D_k$ do
    $\mathcal{F}_k \leftarrow \ell(f(x_i; \hat{\boldsymbol{W}}^{t-1}; \boldsymbol{B}_k^t; \boldsymbol{A}_k^t); y_i)$
    $\boldsymbol{B}_k^t\boldsymbol{A}_k^t \leftarrow \boldsymbol{B}_k^t\boldsymbol{A}_k^t - \eta\nabla\mathcal{F}_k$
end
return $\boldsymbol{B}_k^t$ and $\boldsymbol{A}_k^t$ to server
```



3 METHODOLOGY

3.1 PRELIMINARY

Low-Rank Adaptation. When fine-tuning LLMs on downstream tasks, LoRA (Hu et al., 2021) only updates the parameters of some layers, *e.g.*, self-attention. Consider a layer in the network, LoRA freezes the pre-trained weight $\boldsymbol{W}^0 \in \mathbb{R}^{d_1 \times d_2}$ and leans the weight change $\Delta\boldsymbol{W}$ by injecting two trainable low-rank matrices $\boldsymbol{B} \in \mathbb{R}^{d_1 \times r}$ and $\boldsymbol{A} \in \mathbb{R}^{r \times d_2}$, where rank $r \ll \min(d_1, d_2)$. The updated weight is expressed as:

$$\widetilde{\boldsymbol{W}} = \boldsymbol{W}^0 + \Delta\boldsymbol{W} = \boldsymbol{W}^0 + \boldsymbol{BA}. \tag{1}$$

To ensure consistency with the pre-trained weight during the initial phase, $\boldsymbol{B}$ is initialized as a zero matrix, while $\boldsymbol{A}$ is initialized with Gaussian noise $\mathcal{N}(0, \sigma^2)$. Besides, a hyper-parameter α is typically used to scale $\Delta\boldsymbol{W}$, and the scaling factor is $\frac{\alpha}{r}$.

Federated Learning with Low-Rank Adaptation. Consider a FL system with K clients and a central server, each client $k \in [K]$ has a private dataset D_k for downstream tasks such as natural language understanding or generation. Each dataset D_k contains n_k training samples $\{(x_i, y_i)\}_{i=1}^{n_k}$ and our goal is to federated fine-tune a global LLM $f(\boldsymbol{W}^0)$ on these discrete data. Due to the large size of LLMs, we cannot train and transmit all model parameters. A straightforward approach is to directly combine FedAvg with LoRA (Ye et al., 2024b;a), where each client maintains local low-rank matrices and optimizes them by minimizing the empirical risk:

$$\mathcal{F}_G = \frac{1}{K}\sum_{k=1}^{n_k} \mathcal{F}_k, \quad \text{and } \mathcal{F}_k = \frac{1}{n_k}\sum_{(x_i, y_i) \sim D_k} \ell(f(x_i; \boldsymbol{W}^0; \boldsymbol{B}_k^t; \boldsymbol{A}_k^t); y_i), \tag{2}$$

where $\mathcal{F}_G$ and $\mathcal{F}_k$ are global and local objectives, $\boldsymbol{B}_k^t$ and $\boldsymbol{A}_k^t$ are the low-rank matrices at t-th round training, and ℓ is the loss function. All updated local low-rank matrices will be transferred to the server and we can get the global low-rank matrices:

$$\boldsymbol{B}_G^t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{B}_k^t, \quad \boldsymbol{A}_G^t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{A}_k^t. \tag{3}$$

The updated global low-rank matrices $\boldsymbol{B}_G^t$ and $\boldsymbol{A}_G^t$ will be returned to each client as the initialization for the next round of training, where $\boldsymbol{B}_G^0 \sim 0$ and $\boldsymbol{A}_G^0 \sim \mathcal{N}(0, \sigma^2)$. The weight of final fine-tuned global model parameters after T rounds can be written as:

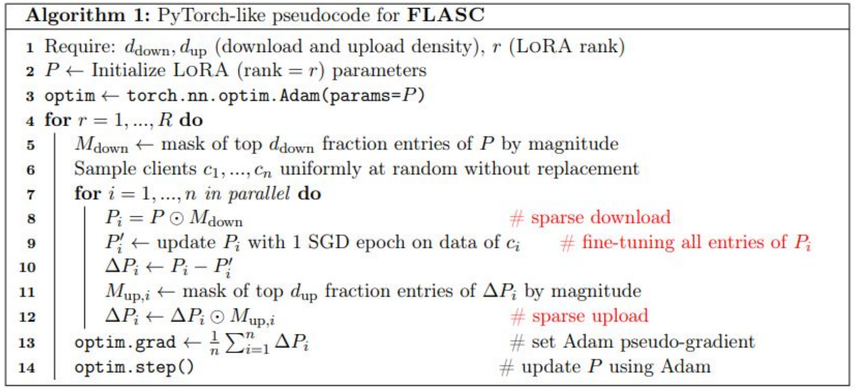
$$\widetilde{\boldsymbol{W}}^T = \boldsymbol{W}^0 + \Delta\boldsymbol{W}^T = \boldsymbol{W}^0 + \boldsymbol{B}_G^T\boldsymbol{A}_G^T. \tag{4}$$

Limitation. The above method is easy to implement and can significantly reduce communication overhead when fine-tuning LLMs. However, it fails to effectively capture global knowledge, especially under data heterogeneity. This issue arises from two factors: ❶ **constrained parameter space** and ❷ **client drift**, where ❶ is the intrinsic limitation of LoRA-based FL and ❷ is the extrinsic influence caused by data heterogeneity.

FLASC

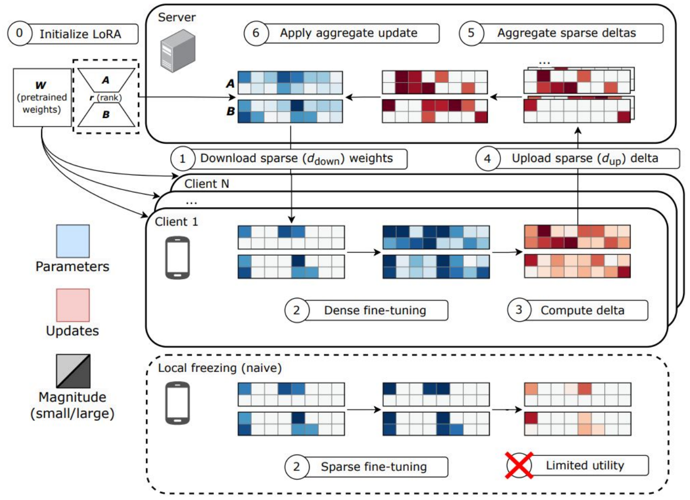
Methodology

- The authors propose **FLASC**, which applies unstructured sparsity (Top-K magnitude masking) to LoRA parameters during communication:
  - Clients download a sparse LoRA module, fine-tune all parameters locally using dense gradients, and then sparsify updates before uploading.
  - Download and upload sparsity patterns are configured independently, allowing adaptation to system constraints like upload bandwidth.
  - The download mask can change across rounds, enabling adaptive parameter freezing/unfreezing strategies.
- The approach is implemented via pseudocode resembling PyTorch, involving:
  - Initialization of LoRA parameters.
  - Random client sampling per round.
  - Dense local fine-tuning.
  - Application of Top-K masks for sparse communication.
  - Parameter updates via Adam optimizer.
- Extensive experiments are conducted across tasks involving image classification (CIFAR10, FLAIR) and text tasks (20NewsGroups, Reddit), with models like ViT-B-16 and GPT2-Small.



Algorithm 1: PyTorch-like pseudocode for **FLASC**

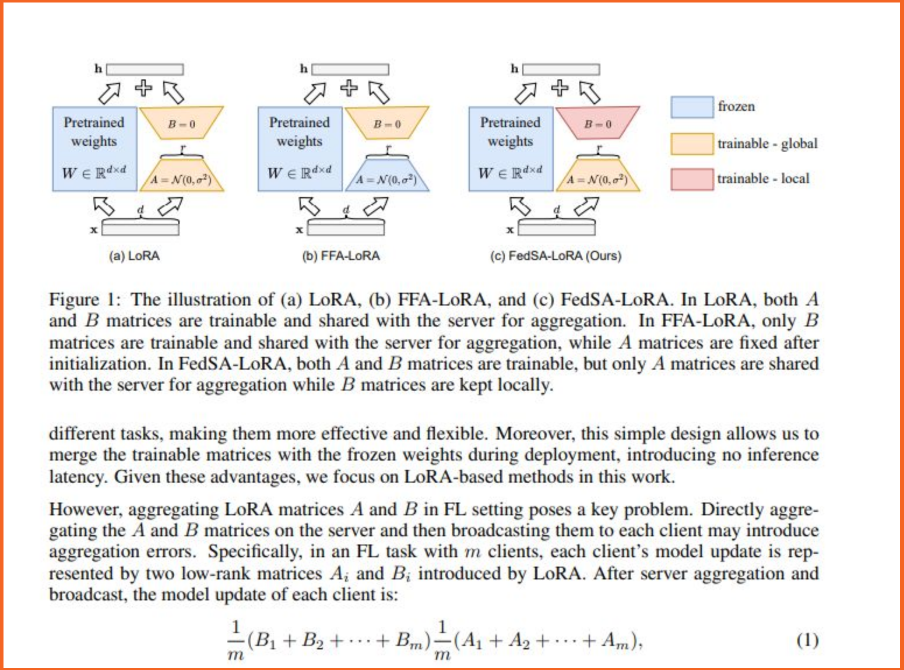
```
Require: $d_{down}, d_{up}$ (download and upload density), $r$ (LoRA rank)
$P \leftarrow$ Initialize LoRA (rank $= r$) parameters
optim $\leftarrow$ torch.nn.optim.Adam(params=$P$)
for $r = 1, ..., R$ do
    $M_{down} \leftarrow$ mask of top $d_{down}$ fraction entries of $P$ by magnitude
    Sample clients $c_1, ..., c_n$ uniformly at random without replacement
    for $i = 1, ..., n$ in parallel do
        $P_i = P \odot M_{down}$                                   # sparse download
        $P'_i \leftarrow$ update $P_i$ with 1 SGD epoch on data of $c_i$    # fine-tuning all entries of $P_i$
        $\Delta P_i \leftarrow P_i - P'_i$
        $M_{up,i} \leftarrow$ mask of top $d_{up}$ fraction entries of $\Delta P_i$ by magnitude
        $\Delta P_i \leftarrow \Delta P_i \odot M_{up,i}$                    # sparse upload
    optim.grad $\leftarrow \frac{1}{n} \sum_{i=1}^{n} \Delta P_i$                # set Adam pseudo-gradient
    optim.step()                                           # update $P$ using Adam
```



FedSA-LoRA

Key Findings

- The A matrices are responsible for capturing general knowledge applicable across clients; B matrices encode client-specific nuances.
- Empirical similarity measures show A matrices are more consistent across clients than B matrices, especially under high data heterogeneity.
- The proposed FedSA-LoRA outperforms existing methods like FFA-LoRA and FedDPA-LoRA in accuracy and communication efficiency.
- Extending FedSA to other LoRA variants (rsLoRA, VeRA) reveals similar asymmetry patterns, confirming the generality of the phenomenon.
- The method reduces communication rounds significantly while maintaining or improving model performance across NLP tasks.



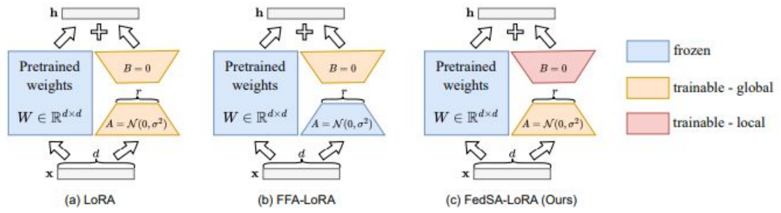


Figure 1: The illustration of (a) LoRA, (b) FFA-LoRA, and (c) FedSA-LoRA. In LoRA, both A and B matrices are trainable and shared with the server for aggregation. In FFA-LoRA, only B matrices are trainable and shared with the server for aggregation, while A matrices are fixed after initialization. In FedSA-LoRA, both A and B matrices are trainable, but only A matrices are shared with the server for aggregation while B matrices are kept locally.

different tasks, making them more effective and flexible. Moreover, this simple design allows us to merge the trainable matrices with the frozen weights during deployment, introducing no inference latency. Given these advantages, we focus on LoRA-based methods in this work.

However, aggregating LoRA matrices A and B in FL setting poses a key problem. Directly aggregating the A and B matrices on the server and then broadcasting them to each client may introduce aggregation errors. Specifically, in an FL task with m clients, each client's model update is represented by two low-rank matrices A_i and B_i introduced by LoRA. After server aggregation and broadcast, the model update of each client is:

$$\frac{1}{m}(B_1 + B_2 + \cdots + B_m)\frac{1}{m}(A_1 + A_2 + \cdots + A_m), \quad (1)$$