SPARFL: SPARSE LOW-RANK FEDERATED LEARNING FOR MEMORY-CONSTRAINED CLIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated Learning (FL) enables collaborative training across decentralized data sources, but standard FL methods typically assume that each participating client can store and optimize the full global model. We study federated Transformer training in the *no-client-fits* regime, where no participating client can store and train the full dense global model within its local memory budget. To address this setting, we propose **SpaRFL** (**Spa**rse Low-**R**ank **F**ederated **L**earning), a client-specific one-shot sparse+low-rank projection method that constructs trainable client models under heterogeneous memory constraints while preserving a dense server model. Across vision, natural language understanding, and language modeling, SpaRFL consistently outperforms the considered model-heterogeneous baselines under severe resource constraints. Under a projected-linear training-state budget 90% below dense training, SpaRFL remains within 1% of the dense baseline on MRPC. In federated pretraining on C4, LLaMA-130M trained at a 70% reduction in the same budget achieves lower perplexity than the dense LLaMA-60M baseline. Overall, SpaRFL provides a practical approach to federated training under strict client memory constraints.

1 INTRODUCTION

Federated Learning (FL) (McMahan et al., 2017; Sani et al., 2024) allows multiple clients to jointly optimize a shared model while keeping their training data on-device. As model performance increasingly depends on larger models and more diverse data, FL offers a natural way to use data distributed across institutions and devices without centralizing it. Standard FL, however, makes a strong systems assumption: every participating client can store the global model, run local optimization, and return an update. This assumption breaks when the global model exceeds every client's local training budget.

Training needs far more memory than storage: under AdamW, each trainable parameter carries its weight, gradient, and two optimizer moments (about 14 bytes), versus 2 bytes for a half-precision weight, a sevenfold gap that grows in absolute terms with model size. In cross-institutional settings such as hospitals, where patient data must stay on site and hardware varies across sites, a client may therefore hold a model it cannot train.

Two straightforward responses are to shrink the global model until it fits the weakest clients, or to exclude under-powered clients entirely. Both sacrifice exactly what makes FL attractive: access to broad private data and the ability to train high-capacity models. This trade-off is costly because scaling laws link performance to dataset and model size (Kaplan et al., 2020; Hoffmann et al., 2022), while heterogeneous FL studies show that excluding resource-limited clients can significantly degrade performance (Ilhan et al., 2023).

It is therefore natural to ask the following: *How can we train robust Transformer architectures in a federated setting where no individual client has the resources to train the entire global model?*

To isolate this setting, we study the *no-client-fits* regime: no participating client can hold the full training state of the dense server model, even when it can store its weights. This is not ordinary model compression. The client model is not a final deployment artifact, but a temporary training view of the dense server model, constructed so local optimization can proceed under the client's memory budget.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

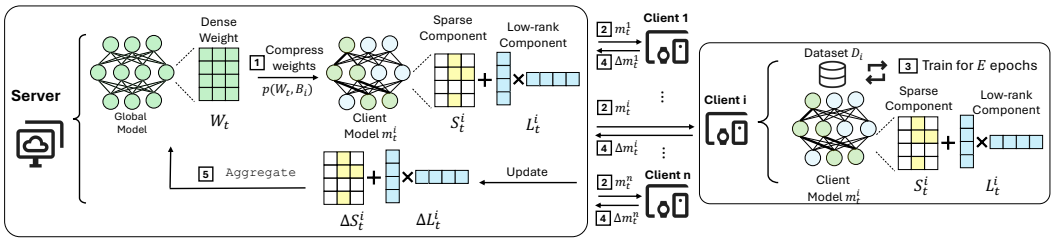

Figure 1: **Overview of SpaRFL.** The server maintains a dense global model, but each participating client receives a budget-feasible sparse+low-rank projection tailored to its memory constraint $\mathcal{B}_i$. Clients train only these projected models on private data and return updates, which the server materializes and aggregates back into the dense parameter space. Thus, SpaRFL turns sparse+low-rank structure into a training interface for the *no-client-fits* regime, where full local training is infeasible.

This distinction matters for existing alternatives. Model-heterogeneous FL methods typically adapt width, depth, or parameter coverage, which becomes brittle under severe memory reductions. Parameter-efficient fine-tuning (PEFT) is effective when a pretrained base fits on every client, but its footprint is bounded below by that frozen base and it cannot train a model from scratch; quantization reduces storage but does not by itself provide client-specific models or an aggregation mechanism for heterogeneous FL.

We introduce **SpaRFL** (**Spa**rse Low-**R**ank **F**ederated **L**earning) to address this regime. SpaRFL preserves a dense server model while exposing client-specific sparse+low-rank projections during local training. In every round, the server projects the current dense weights into a budget-feasible representation for each client, clients train only this projected model, and the server materializes the returned updates back into the dense parameter space for standard aggregation.

Contributions. We make four main contributions. (1) We formulate federated Transformer training in the *no-client-fits* regime, expressing client budgets as optimizer-aware persistent training state that distinguishes trainable from frozen values, rather than as trainable-parameter counts. (2) We introduce a client-specific sparse+low-rank projection framework for this regime that preserves dense server aggregation; for NLP models, we instantiate it with an activation-space sparse+low-rank projection. (3) We provide a convergence analysis for factor-space local optimization under both model- and delta-based aggregation, conditional on a collective tangent-coverage assumption, together with practical insight into the sparse–low-rank allocation across training regimes. (4) Across six datasets spanning vision, natural language understanding, and language modeling, SpaRFL consistently outperforms the considered baselines under severe resource constraints, remains within 1% of full-model training on MRPC at the 90% projected-state budget reduction, and integrates effectively with quantization.

2 Related Work

Heterogeneous Federated Learning. System heterogeneity remains a central challenge in FL since clients often have widely different compute and memory budgets (Diao et al., 2021; Horvath et al., 2021). Existing methods typically handle this by training heterogeneous subnetworks derived from a shared dense model, most often via width- or depth-based pruning. Width-based approaches prune channels (Diao et al., 2021; Horvath et al., 2021), whereas depth-based methods remove layers (Kim et al., 2022; Ilhan et al., 2023), which may under-train deeper layers. Other variants increase parameter coverage by rotating or randomizing subnetwork assignments across clients and rounds (Alam et al., 2022; Su et al., 2024). However, all of these methods remain fundamentally constrained by subnetwork extraction from a dense backbone, which becomes increasingly brittle under severe compression and limited communication budgets.

Alternatively, Parameter-Efficient Fine-Tuning (PEFT) methods (Hyeon-Woo et al., 2021; Babakniya et al., 2023a; Cho et al., 2024; Zhang et al., 2025; Yan et al., 2025; Guo et al., 2024) address FL heterogeneity through adaptive ranks or layer-wise allocation. However, these methods typically assume clients can store the full base model and are mainly limited to fine-tuning. In contrast, SpaRFL

targets extreme memory constraints by re-parameterizing the model to fit local budgets, enabling both pre-training and fine-tuning.

Sparsity and Low-Rank Decompositions. Sparsity is a standard approach for reducing memory use and improving the efficiency of deep neural networks (Hoefler et al., 2021). For LLMs, one-shot pruning methods use activation information or approximate second-order information to identify weights that can be removed (Sun et al., 2024; Frantar & Alistarh, 2023). Beyond unstructured pruning, structured sparse parameterizations such as Monarch matrices (Dao et al., 2022) and Block Tensor-Train structures (Qiu et al., 2024) provide hardware-friendly alternatives to dense layers (Qiu et al., 2024). However, these representations can be too rigid for heterogeneous settings that must operate across a wide range of client resource budgets. Hybrid sparse+low-rank decompositions address this limitation and have been used for model compression (Yu et al., 2017; Li et al., 2023), memory-efficient pretraining (Han et al., 2024), and PEFT, where RoSA (Nikdan et al., 2024) uses this parameterization to approach full fine-tuning performance. In FL, FedLRS (Luo et al., 2023) jointly trains sparse and low-rank components to reduce communication and computation. It does not construct client-specific projections of a dense server model under heterogeneous memory budgets. These approaches motivate sparse+low-rank representations for heterogeneous FL, where client-specific allocations can accommodate different local budgets.

Federated Sparse Training. A related line of work studies sparse training directly in the federated setting to reduce communication and computational overhead. ZeroFL (Qiu et al., 2022) uses local sparsity to accelerate on-device training, FLASH (Babakniya et al., 2023b) uses adaptive sparse hashing, and SparsyFed (Guastella et al., 2025) dynamically evolves sparsity patterns. However, these methods still assume a single shared global architecture, whereas SpaRFL preserves the full capacity of the server model.

In summary, SpaRFL differs from prior work along four axes (see Appendix C for more details). Unlike width/depth-based heterogeneous FL, it does not obtain client models by shrinking the architecture, but by projecting dense weights into client-specific sparse+low-rank representations. Unlike PEFT methods, it does not assume the full base model fits on device, and it supports both fine-tuning and from-scratch pretraining. SpaRFL constructs one-shot sparse+low-rank projections of the dense server model tailored to each client's budget. For NLP, these projections incorporate activation information. Finally, unlike federated sparse training methods, SpaRFL preserves a dense global model on the server and exposes a standard aggregation interface after reconstruction.

3 SPARFL

3.1 PROBLEM FORMULATION

We consider an FL system with a central server and a set of n heterogeneous clients, indexed by $i \in [n]$. Each client i has a local dataset $\mathcal{D}_i$ and is limited by a specific memory budget $\mathcal{B}_i$. At each communication round t, the server maintains a *global dense model* $\mathcal{W}_t = \{W_t^l\}_{l=1}^L$ consisting of L layers with fully connected weights, where each layer is represented by a weight matrix $W_t^l \in \mathbb{R}^{m_l \times n_l}$.

Due to the heterogeneity in client budgets $\{\mathcal{B}_i\}_{i=1}^n$, it is often infeasible for a client i to host or train the entire global model $\mathcal{W}_t$. The server therefore constructs a *client-specific model* $\tilde{\mathcal{W}}_{t,i} = \{\tilde{W}_{t,i}^l\}_{l=1}^L$, with each layer given by $\tilde{W}_{t,i}^l = p(W_t^l, \mathcal{B}_i)$. Here, $p(\cdot, \mathcal{B}_i) : \mathbb{R}^{m_l \times n_l} \to \mathbb{R}^{m_l \times n_l}$ maps the global dense weights to a compressed or re-parameterized format that satisfies the local resource constraint $\mathcal{B}_i$.

We define the local objective for client i as the empirical risk over its local data:

$$F_i(\tilde{\mathcal{W}}) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\ell(f(x;\tilde{\mathcal{W}}), y)\right], \tag{1}$$

where $f(\cdot;\tilde{\mathcal{W}})$ denotes the model architecture parameterized by $\tilde{\mathcal{W}}$, which encompasses the re-parameterized dense layers alongside other components (e.g., LayerNorm), and $\ell(\cdot,\cdot)$ is a standard loss function. Since evaluation and aggregation operate on the dense server model, we take the

data-weighted dense risk as the reference objective:

$$\min_{\mathcal{W}} \Phi(\mathcal{W}) \triangleq \sum_{i=1}^{n} q_i F_i(\mathcal{W}), \qquad q_i \triangleq \frac{|\mathcal{D}_i|}{\sum_{j=1}^{n} |\mathcal{D}_j|}. \tag{2}$$

Clients cannot evaluate gradients at $\mathcal{W}_t$ directly. They initialize at $p(\mathcal{W}_t, \mathcal{B}_i)$ and optimize F_i through the resulting sparse+low-rank factors. The projection, tangent-space, and aggregation effects are made explicit in Appendix B. This formulation does not prescribe a server aggregation rule.

3.2 The Algorithm

Figure 1 and Algorithm 1 summarize SpaRFL as a projection-based interface for federated training in the no-client-fits regime. At round t, the server samples clients $\mathcal{C}_t$, projects the dense model $\mathcal{W}_t$ into client-feasible sparse+low-rank models under their respective budgets, and sends these models for local optimization. The clients return their updates, which the server reconstructs and aggregates into $\mathcal{W}_{t+1}$. The paragraphs below describe the projection variants, budget allocation, and aggregation in detail.

Projection Operator Design. The central algorithmic challenge in SpaRFL is to design a projection operator $p(\cdot, \mathcal{B}_i)$ that converts a dense server model into a client-feasible training model in a single shot, without client gradients and without changing the dense server-side aggregation workflow. Classically, projection operators for weight compression have sought structured approximations that minimize distortion, typically defined as the reconstruction error between dense weights and their compressed counterparts. However, the FL environment imposes two critical constraints: (i) the projection must be performed in a *one-shot* manner due to the immediate budget requirements of participating clients, precluding iterative or incremental sparsification; and (ii) the server must operate without access to client gradients to preserve data privacy.

Algorithm 1 SpaRFL

Input: model $\mathcal{W}$, rounds T, local steps E, budgets $\mathcal{B}$, aggregation rule $\mathcal{A}$
for $t = 0, \ldots, T-1$ **do**
  Sample clients $\mathcal{C} \subseteq [n]$
  for $i \in \mathcal{C}$ **in parallel do**
    $(\mathcal{S}_{t,i}, \mathcal{L}_{t,i}) \leftarrow p(\mathcal{W}_t, \mathcal{B}_i)$
    $\tilde{\mathcal{W}}_{t,i} \leftarrow (\mathcal{S}_{t,i}, \mathcal{L}_{t,i})$
  end for
  Send $\tilde{\mathcal{W}}_{t,i}$ to each client $i \in \mathcal{C}$
  for $i \in \mathcal{C}$ **in parallel do**
    $\tilde{\mathcal{W}}_{t,i,0} \leftarrow \tilde{\mathcal{W}}_{t,i}$
    $\tilde{\mathcal{W}}_{t,i,E} \leftarrow$ Train factors for E local steps
    Send $\tilde{\mathcal{W}}_{t,i,E}$ to server
  end for
  $\mathcal{W}_{t+1} \leftarrow \mathcal{A}(\mathcal{W}_t, \{\tilde{\mathcal{W}}_{t,i,0}, \tilde{\mathcal{W}}_{t,i,E}\}_{i \in \mathcal{C}})$
end for

We represent each client layer as the sum of a low-rank component and a sparse residual. The low-rank term provides a compact approximation of broadly distributed weight structure, while the sparse term retains important coefficients that are poorly represented by the low-rank approximation. This complementary parameterization can be adjusted to satisfy different client memory budgets without changing the layer dimensions. SpaRFL uses this representation as a client-specific, budget-conditioned projection of the dense server model and reconstructs the resulting client models for dense aggregation. We tailor the projection mechanism to the distinct characteristics of vision models and LLMs. Concretely, for each round t, client i, and layer l, we define

$$\begin{aligned} \tilde{W}^l_{t,i} \;&=\; p(W^l_t, \mathcal{B}_i) \;\equiv\; S^l_{t,i} + L^l_{t,i}, \\ &\text{such that} \;\; L^l_{t,i} \;=\; A^l_{t,i} B^l_{t,i}. \end{aligned} \tag{3}$$

where $W^l_t \in \mathbb{R}^{m_l \times n_l}$ is the dense server weight, $S^l_{t,i}$ is sparse, and $L^l_{t,i}$ is low-rank with factors $A^l_{t,i} \in \mathbb{R}^{m_l \times r^l_{t,i}}$ and $B^l_{t,i} \in \mathbb{R}^{r^l_{t,i} \times n_l}$ chosen to satisfy the client memory budget $\mathcal{B}_i$.

Byte-Level Budget Allocation. Let $B^{\text{proj}}_{i,l}$ denote the persistent-state budget, in bytes, assigned to projected linear layer l of client i, and let $k_{i,l}$ denote a count of retained sparse values. If c_S is the persistent-state cost per sparse value and c_L is the cost per low-rank factor parameter, the layer satisfies

$$B^{\text{proj}}_{i,l} = c_S k_{i,l} + c_L r_{i,l}(m_l + n_l). \tag{4}$$

Under our dtype convention, a trainable parameter costs $c_{\text{tr}} = 2 + 4 + 8 = 14$ bytes for its half-precision value, FP32 gradient, and two FP32 AdamW states, whereas a frozen parameter costs

$c_{\text{fr}} = 2$ bytes. Thus $c_S = c_L = c_{\text{tr}}$ during pretraining, while $c_S = c_{\text{fr}}$ and $c_L = c_{\text{tr}}$ during fine-tuning, when the sparse component is frozen. For a fraction $\gamma \in (0, 1)$ assigned to the low-rank factors, we allocate

$$r_{i,l} = \left\lfloor \frac{\gamma B_{i,l}^{\text{proj}}}{c_L(m_l + n_l)} \right\rfloor, \qquad k_{i,l} = \left\lfloor \frac{(1-\gamma) B_{i,l}^{\text{proj}}}{c_S} \right\rfloor. \tag{5}$$

This separates byte budgets from parameter counts and guarantees that the allocated persistent state does not exceed the layer budget, up to unused bytes introduced by flooring.

Weight-Space Projection for Vision Tasks. For vision-centric architectures, we prioritize preserving the original weight matrices by applying a truncated Singular Value Decomposition (SVD) and then pruning the residual. Given the byte-level rank and sparse-support allocations in Equation (5), we first compute the rank-$r_{t,i}^l$ approximation $L_{t,i}^l = U_r \Sigma_r V_r^\top$ of W_t^l. We then form the residual $R_t^l = W_t^l - L_{t,i}^l$ and define the sparse mask $\mathcal{M}_{t,i}^l$ by selecting the top-$k_{t,i}^l$ entries of R_t^l by magnitude:

$$\mathcal{M}_{t,i}^l = \text{TopK}(|R_t^l|, k_{t,i}^l), \quad S_{t,i}^l = \mathcal{M}_{t,i}^l \odot R_t^l, \tag{6}$$

where $\odot$ denotes the Hadamard product. Further discussion on alternative mask selection strategies is provided in Appendix F.

Activation-Aware Projection for NLP Tasks. For Large Language Models (LLMs), one-shot magnitude pruning often fails because small-magnitude weights can significantly impact output when paired with high-magnitude activations (Sun et al., 2024; Frantar & Alistarh, 2023). We assume the server has access to a small, public calibration dataset $\mathcal{D}_{\text{cal}}$ (see Appendix M) to estimate input activation statistics. Let $X^l \in \mathbb{R}^{T \times n_l}$ denote the matrix of input activations for layer l collected over tokens T, and define the diagonal scaling

$$D^l \triangleq \text{diag}\big(\|X_{:,1}^l\|_2, \ldots, \|X_{:,n_l}^l\|_2\big). \tag{7}$$

The projection uses the diagonally activation-weighted distortion

$$\mathcal{E}_{D^l}(S, L) \triangleq \|(W_t^l - S - L)D^l\|_F^2. \tag{8}$$

The corresponding activation-aware importance score[1] is

$$\text{score}_{uv}^l = |(W_t^l)_{uv}|(D^l)_{vv} = |(W_t^l)_{uv}| \cdot \|X_{:,v}^l\|_2. \tag{9}$$

The order of the two components follows the training regime. When a sparse component is retained during fine-tuning, we first choose its support by the score above and then compute the best rank-$r_{t,i}^l$ approximation of the weighted residual $(W_t^l - S_{t,i}^l)D^l$. During pretraining, we first compute the rank-$r_{t,i}^l$ approximation of $W_t^l D^l$ and then retain the largest weighted entries of the remaining residual under the prescribed support budget. In either order, the truncated SVD splits the singular values symmetrically in weighted coordinates, yielding factors $A_{t,i}^l = U_r \Sigma_r^{1/2}$ and $B_{t,i}^l = \Sigma_r^{1/2} V_r^\top (D^l)^\dagger$. This is exactly the diagonal activation-weighted projection characterized in Proposition B.14. The trade-off between the sparse and low-rank component in NLP regimes is further analyzed in Section 4.

Server Aggregation. After local training, each participating client $i \in \mathcal{C}_t$ returns its updated layer parameters in our sparse+low-rank form, i.e., $\{S_{t,i,E}^l, A_{t,i,E}^l, B_{t,i,E}^l\}_{l=1}^L$. To aggregate client updates with standard federated optimizers, the server first reconstructs a dense view of each layer by materializing the corresponding operator:

$$\widehat{W}_{t,i,E}^l \;\triangleq\; S_{t,i,E}^l + A_{t,i,E}^l B_{t,i,E}^l \;=\; \big(S_{t,i,E}^l + A_{t,i,E}^l B_{t,i,E}^l\big) I, \tag{10}$$

where I is the identity matrix and the last equality emphasizes that we can obtain $\widehat{W}_{t,i}^l$ by applying the client operator to I (a server-side operation, since the server is not memory constrained). Once materialized, SpaRFL becomes compatible with any off-the-shelf FL aggregation rule (see Appendix D). For example, the model form of FedAvg aggregates the reconstructed client models layer-wise as

$$W_{t+1}^l \;=\; \sum_{i \in \mathcal{C}_t} \frac{|\mathcal{D}_i|}{\sum_{j \in \mathcal{C}_t} |\mathcal{D}_j|} \widehat{W}_{t,i,E}^l, \tag{11}$$

[1]This computation is efficiently parallelized at the server level.

whereas an update-based rule applies the reconstructed local change $\widehat{W}^l_{t,i,E} - \widehat{W}^l_{t,i,0}$ to W^l_t. These are two aggregation interfaces of the same client parameterization; they coincide when every client initialization equals W^l_t, while client-specific projections generally introduce a mean projection displacement between them. The reconstruction step nevertheless isolates the sparse low-rank client parameterization from the choice of server optimizer, enabling SpaRFL to interface with FedAvg, FedOpt variants, and other aggregation schemes.

Convergence Perspective. Our analysis models the client optimization at the level of the sparse+low-rank representation: local Adam acts on the active sparse values and the factors A, B, and the resulting change is mapped back to dense weight space. Theorem B.10 gives a unified stationarity bound for both model- and delta-based aggregation under full participation, fixed data weights, and a constant base stepsize. The bound identifies three representation-specific quantities: the projection residual, the collective coverage of dense descent directions by the client factor tangent spaces, and, for model aggregation, the mean projection displacement. The delta form removes the last quantity (Corollary B.11), while the model form controls it by the same raw projection residual characterized in our experiments (Corollary B.12). Full assumptions and proofs are given in Appendix B.

4 EXPERIMENTS

We conduct a comprehensive set of experiments to evaluate the performance of SpaRFL for FL on resource constrained clients. Our experiments span vision, natural language understanding (NLU), and natural language generation (NLG) tasks.

We provide a detailed list of hyperparameters and experimental configurations in Appendix A.

Datasets and Models. For vision, we pretrain ViT-Tiny/16 (Dosovitskiy et al., 2021; Steiner et al., 2021) from scratch on CIFAR-10 (Krizhevsky et al., 2009). For NLU, we fine-tune pretrained RoBERTa-base (Liu et al., 2019) on GLUE (Wang et al., 2018), focusing on SST-2, MRPC, and QNLI, which cover sentiment classification, paraphrase detection, and sentence-pair inference. For NLG, we consider two settings. First, we pretrain LLaMA-style models from scratch on C4 in 60M and 130M parameter variants, evaluating them using validation perplexity on C4. Second, we fine-tune Llama-3.2-1B (Grattafiori et al., 2024) on WikiText-2 (Merity et al., 2016). Together, these experiments cover multiple modalities, training regimes, and model sizes under varying memory constraints.

Implementation. We use Flower (Beutel et al., 2020) to simulate federated learning and coordinate server-client communication. We implement SpaRFL in the Compositional Linear Algebra (CoLA) library (Potapczynski et al., 2023), which represents structured matrices as linear operators and uses compositional dispatch to generate memory-efficient routines. This enables efficient integration of sparse and low-rank components on memory-constrained devices.

Baselines. We compare SpaRFL against a *Dense* baseline, which serves as an empirical upper bound in which all clients can train the full global model. We also compare against model-heterogeneous FL baselines: HeteroFL (Diao et al., 2021), which uses static width-based sub-models; FjORD (Horvath et al., 2021), which uses Ordered Dropout to construct nested width-varying sub-models; and FIARSE (Wu et al., 2024), which performs importance-aware sub-model extraction. Our additional comparisons include FedLRS (Luo et al., 2023), which jointly trains sparse and low-rank components; the PEFT methods RoSA (Nikdan et al., 2024) and QLoRA (Dettmers et al., 2023); and the federated LoRA methods Fed-PLoRA (Zhang et al., 2026) and FLASC (Kuo et al., 2026). Because the PEFT-based methods require a pretrained frozen backbone, they are not applicable to our from-scratch pretraining setting; we therefore report their matched-budget fine-tuning comparison in Appendix G.

Memory Budget Formulation. We assign resource tiers using the persistent training state of the linear layers modified by the compared heterogeneous methods. For a trainable parameter, the accounting includes its weight, gradient, and optimizer states; under our AdamW configuration, these require 2, 4, and 8 bytes, respectively. A frozen parameter requires only its 2-byte half-precision value. We therefore allocate each layer budget directly in bytes, accounting separately for trainable sparse

values, frozen sparse values, and trainable low-rank factors as specified in Equations (4) and (5). Embeddings, normalization layers, output heads, and runtime-dependent tensors are common to the controlled budget assignment and are reported separately in Appendix J. Accordingly, a $b\%$ budget reduction denotes a reduction in projected-linear persistent training state relative to the corresponding dense layers.

Our heterogeneous experiments partition the client population uniformly across resource tiers. For computer vision, the projected-state budget reductions are $b \in \{20\%, 40\%, 50\%, 70\%, 80\%\}$; for NLP, they are $b \in \{20\%, 40\%, 50\%, 70\%, 90\%\}$. Further details, including unprojected state, sparse metadata, and analytical runtime-memory components, are provided in Appendix J.

Aggregation Protocol. Unless stated otherwise, the reported experiments use reconstructed-model aggregation. Each client returns its locally optimized sparse and low-rank factors, the server materializes the corresponding dense client model, and these models are averaged at the server. This is the $\lambda_{\text{agg}} = 1$ interface analyzed in Theorem B.10. Delta aggregation is analyzed as an alternative interface.

Homogeneous Clients. Figure 2 illustrates the performance of SpaRFL relative to the baseline methods in the *homogeneous* setting, where all participating clients possess an identical resource budget. On the fine-tuning tasks, SpaRFL outperforms all baselines at every budget; on CIFAR-10, it matches them at mild reductions and pulls ahead as the budget tightens.

Notably, while baseline methods maintain competitive performance during pretraining on CIFAR-10, they exhibit significant degradation during fine-tuning tasks. This suggests that existing resource-adaptive algorithms lack a robust projection mechanism capable of effective knowledge transfer during compression; as the model is scaled down, critical representations acquired during pretraining are lost. In contrast, SpaRFL demonstrates remarkable stability, maintaining a performance gap within 1% relative to the Dense baseline on MRPC even at the 90% projected-state budget reduction.

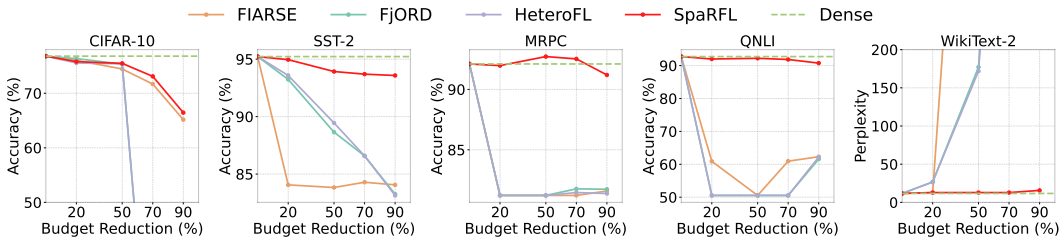

Figure 2: Performance under homogeneous resource constraints. SpaRFL consistently outperforms the considered model-heterogeneous baselines at severe projected-state budget reductions. On MRPC, it remains within 1% of the dense baseline at the 90% budget reduction.

Heterogeneous Clients. Following prior work (Wu et al., 2024; Diao et al., 2021), Table 1 reports results when clients are uniformly distributed across resource tiers. SpaRFL outperforms all considered baselines across tasks; on MRPC, it remains within one F1 point of Dense, whereas each baseline loses more than ten points. These results indicate that SpaRFL continues to extract useful updates from highly constrained clients, for which the baseline submodel approaches degrade substantially.

Table 1: **Heterogeneous** FL results across vision and NLP tasks. SpaRFL outperforms all considered baselines. Notably, SpaRFL remains competitive with the Dense baseline even where baseline methods fail to converge or show substantial degradation.

Dataset	CIFAR-10 (Acc. ↑) $\alpha = 0.3$	CIFAR-10 (Acc. ↑) IID	SST-2 (Acc.↑)	MRPC (F1↑ / Acc.↑)	QNLI (Acc.↑)	WikiText-2 (PPL↓)
Dense	71.73 ± 0.15	75.66 ± 0.24	95.22 ± 0.24	92.35 ± 0.23 / 89.46 ± 0.42	92.74 ± 0.17	11.68 ± 0.00
HeteroFL	52.63 ± 1.15	64.26 ± 0.28	84.52 ± 0.30	81.22 ± 0.00 / 68.38 ± 0.00	55.55 ± 4.71	1599.17 ± 127.18
FjORD	54.00 ± 1.26	64.12 ± 0.41	84.44 ± 0.29	81.69 ± 0.40 / 69.53 ± 0.99	58.07 ± 1.85	1464.94 ± 23.57
FIARSE	66.56 ± 0.34	69.06 ± 0.74	84.17 ± 0.30	81.71 ± 0.45 / 69.61 ± 1.12	65.92 ± 8.51	464.78 ± 1.09
SpaRFL	**68.21 ± 0.64**	**70.30 ± 0.07**	**93.39 ± 0.24**	**91.57 ± 0.90 / 88.15 ± 1.63**	**89.06 ± 1.54**	**13.09 ± 0.27**

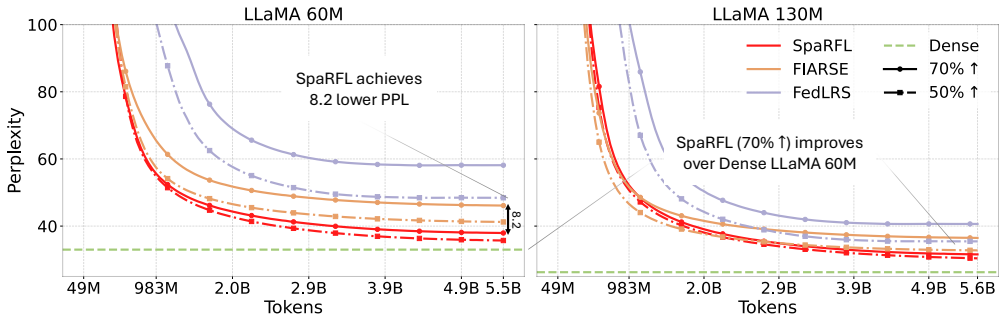

Figure 3: **Training LLaMA from scratch.** Perplexity curves of SpaRFL, FIARSE, and FedLRS for 60M and 130M models on C4 in a homogeneous FL setting under 50% and 70% projected-state budget reductions.

Training LLaMA from Scratch. We study federated pretraining from scratch on C4 using LLaMA-60M and LLaMA-130M in a homogeneous setting, comparing SpaRFL against FIARSE, the best-performing considered model-heterogeneous baseline, and against the federated sparse+low-rank method FedLRS. We follow the LLaMA architecture and training setup of Zhao et al. (2024) (see Appendix A). Figure 3 plots evaluation perplexity on C4 throughout training and highlights three main findings. First, at matched projected-state budgets, SpaRFL consistently outperforms both baselines; at the 70% projected-state budget reduction, it improves perplexity over FIARSE by 8.2 points on LLaMA-60M, with the same trend holding for LLaMA-130M, and over FedLRS by 20.1 and 9.4 points on LLaMA-60M and LLaMA-130M, respectively. Second, this advantage is large enough that SpaRFL at the 70% reduction outperforms both baselines even at the more favorable 50% reduction on both model scales. Third, and most notably, at the 70% reduction, the larger LLaMA-130M model trained with SpaRFL achieves lower perplexity than the dense LLaMA-60M baseline. This result shows that SpaRFL uses the controlled projected-state budget more effectively, enabling federated pretraining of larger models under strict resource constraints.

Sparse vs Low-Rank. Table 2 (a) shows that allocating 95% of the byte budget to the low-rank component and 5% to the sparse residual yields the highest CIFAR-10 accuracy. Compared with the low-rank-only configuration, the 95/5 split improves accuracy by 2.25 percentage points at the same byte budget, indicating that a small sparse residual complements the low-rank approximation. This suggests that low-rank capacity captures the diffuse structure required for training from scratch, while a small sparse component preserves important residual coefficients. The preferred allocation nevertheless depends on the training regime: the LLaMA pretraining analysis in Appendix E instead favors a sparse-heavy decomposition.

Sensitivity to Low-Rank Dimension. We conduct a sensitivity analysis to investigate the impact of the low-rank dimension r on fine-tuning performance across NLP tasks. Table 2 (b) illustrates the performance of SpaRFL as the rank of the low-rank component varies while the total client budget $\mathcal{B}_i$ remains fixed. As established in our fine-tuning protocol, the sparse component S remains frozen to act as a structural anchor for pre-trained knowledge, while only the low-rank factors are updated.

Crucially, under a fixed budget, the rank r and the sparse support are inversely related; increasing the rank necessitates a corresponding reduction in the number of preserved sparse parameters. Contrary to our observations in vision-based pre-training, we find that larger sparse components are significantly more beneficial for NLP fine-tuning. For instance, at the 90% projected-state budget reduction, $r = 6$ substantially outperforms $r = 128$. This suggests that the sparse component effectively captures the salient linguistic features acquired during pre-training, and aggressive pruning of these "outlier" weights to accommodate a higher-rank update leads to significant representational collapse. Conversely, at the 70% projected-state budget reduction, the sparse component holds a sufficient number of parameters to preserve the pre-trained knowledge base, leaving the model performance relatively robust to variations in rank.

Table 2: Sensitivity analyses. **Left:** Effect of the sparse/low-rank budget split on CIFAR-10 accuracy in the homogeneous setting under a 70% projected-state budget reduction. **Right:** Effect of the low-rank rank r on WikiText-2 perplexity under 70% and 90% projected-state budget reductions.

(a) Sparse vs. low-rank budget trade-off

Setting	Acc. ↑
S (50%) + L (50%)	66.74
S (5%) + L (95%)	74.86
S (3%) + L (97%)	74.55
S (0%) + L (100%)	72.61

(b) Sensitivity to low-rank dimension

	r=6	r=16	r=64	r=128
70%	12.64	12.03	11.97	12.82
90%	15.67	15.76	28.77	172.73

Quantizing SpaRFL. Following QLoRA (Dettmers et al., 2023), we introduce Q-SpaRFL, which stores the frozen sparse component S in 4-bit NormalFloat (`NF4`) or `INT8` and dequantizes it to `bf16` during computation. This reduces its memory footprint and leaves more of the fixed budget available for retaining sparse weights alongside the trainable low-rank factors.

As shown in Table 3, quantization is most effective under severe constraints: at the 90% projected-state budget reduction, `NF4` lowers WikiText-2 perplexity from 15.67 to 12.64 by enabling more pre-trained weights to be retained. At the 70% reduction, the gain is marginal because sparse capacity is no longer the primary bottleneck.

Table 3: WikiText-2 perplexity of SpaRFL and Q-SpaRFL under 70% and 90% projected-state budget reductions. Lower is better.

	70%	90%
SpaRFL	11.97	15.67
Q-SpaRFL (`INT8`)	11.97	12.65
Q-SpaRFL (`NF4`)	11.85	12.64

Additional Results. The appendix expands the paper's central claims along six dimensions. Appendix D studies the robustness of SpaRFL under alternative federated optimizers, Appendix G compares SpaRFL against PEFT-based baselines, Appendix H stress-tests nonuniform client resource distributions, Appendix L examines how SpaRFL scales as the number of clients increases, Appendix I analyzes projection and runtime overheads, and Appendix K quantifies communication costs and sparse-format metadata overhead. Together, these results clarify both the algorithmic scope and the practical systems trade-offs of SpaRFL.

Limitations and Future Work. We identify several key limitations that also define our roadmap for future work. First, our current analysis excludes long-context sequences where activation memory becomes the primary bottleneck; we provide a detailed discussion of this regime in Appendix J. Second, we have not yet optimized the end-to-end efficiency of SpaRFL. While micro-benchmarks in Appendix I show that SpaRFL can achieve a 2.6x speedup over dense baselines, implementing fused kernels for varying sparse and low-rank ratios remains an ongoing effort. Finally, our model-quality experiments reach 130M parameters for pretraining and 1.2B for fine-tuning, while at 7B we benchmark only server-side projection (Appendix I); larger-scale validation remains future work.

5 Conclusion

In this work, we studied federated training in the *no-client-fits* regime, where no participating client can store and optimize the dense global model. We introduced SpaRFL, a client-specific sparse+low-rank projection framework that enables memory-feasible local training while preserving a dense server model for standard aggregation. Across computer vision, natural language understanding, and language modeling, SpaRFL consistently outperforms the considered model-heterogeneous baselines under severe resource constraints. It remains within 1% of the Dense baseline on MRPC at a 90% projected-state budget reduction, while a LLaMA-130M model pretrained on C4 at a 70% reduction achieves lower perplexity than the dense LLaMA-60M baseline. Our results further show that the sparse–low-rank allocation should reflect the training regime: from-scratch vision training favors low-rank capacity, whereas LLM fine-tuning under tight budgets benefits from retaining sparse pretrained weights. Overall, SpaRFL provides a practical interface between dense global models and memory-constrained clients for both fine-tuning and from-scratch pretraining.

AI USE STATEMENT

We used generative AI tools to assist with writing and editing the manuscript for grammar, clarity, and readability, and with plotting graphs for the paper. The authors developed and implemented the algorithm and carried out all other research tasks, including experimental design, data processing, and analysis and interpretation of results. We reviewed and verified all AI-assisted edits, plotting code, and resulting figures for correctness and consistency with the underlying results.

ETHICS STATEMENT

This work studies a general method for training models on decentralized data under client memory constraints. It does not involve recruiting human subjects or collecting new participant data; the experiments use existing benchmarks in simulated federated settings. The method is not designed for a harmful application. Its ethical impact depends on the data and downstream model, and federated training alone does not provide a formal privacy guarantee. Deployments involving sensitive data or high-stakes decisions should use appropriate privacy, security, and fairness safeguards.

REPRODUCIBILITY STATEMENT

All experiments in this paper are reproducible using the anonymized code and scripts provided at `https://anonymous.4open.science/r/sparfl`. The experimental setup, evaluation protocols, and hyperparameters are documented in Section 4 and Appendix A. Reproducing the complete set of experiments is estimated to require approximately 4,100 GPU hours on NVIDIA GH200 GPUs.

REFERENCES

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. FedRolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems*, 35:29677–29690, 2022.

Sara Babakniya, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy, and Salman Avestimehr. SLoRA: Federated parameter efficient fine-tuning of language models. *arXiv preprint arXiv:2308.06522*, 2023a.

Sara Babakniya, Souvik Kundu, Saurav Prakash, Yue Niu, and Salman Avestimehr. Revisiting sparsity hunting in federated learning: Why does sparsity consensus matter? *Transactions on Machine Learning Research*, 2023b.

Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque De Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL `https://arxiv.org/abs/1604.06174`.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, and Gauri Joshi. Heterogeneous LoRA for federated fine-tuning of on-device foundation models. *arXiv preprint arXiv:2401.06432*, 2024.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=ZPQhzTSWA7`.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Adriano Guastella, Lorenzo Sani, Alex Iacob, Alessio Mora, Paolo Bellavista, and Nicholas D Lane. SparsyFed: Sparse adaptive federated training. *arXiv preprint arXiv:2504.05153*, 2025.

Pengxin Guo, Shuang Zeng, Yanran Wang, Huijie Fan, Feifei Wang, and Liangqiong Qu. Selective aggregation for low-rank adaptation in federated learning. *arXiv preprint arXiv:2410.01463*, 2024.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010. URL https://arxiv.org/abs/0909.4061.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Kumar Jawanpuria, and Bamdev Mishra. SLTrain: a sparse plus low rank approach for parameter and memory efficient pretraining. *Advances in Neural Information Processing Systems*, 37:118267–118295, 2024.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. FedPara: Low-rank Hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*, 2021.

Fatih Ilhan, Gong Su, and Ling Liu. ScaleFL: Resource-adaptive federated learning with heterogeneous clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24532–24541, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Minjae Kim, Sangyoon Yu, Suhyun Kim, and Soo-Mook Moon. DepthFL: Depthwise federated learning for heterogeneous clients. In *The Eleventh International Conference on Learning Representations*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models, 2022. URL https://arxiv.org/abs/2205.05198.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Kevin Kuo, Arian Raje, Kousik Rajesh, and Virginia Smith. FLASC: Federated LoRA with sparse communication. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 990–1008, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400724152. doi: 10.1145/3786335.3813151. URL https://doi.org/10.1145/3786335.3813151.

Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. LoSparse: Structured compression of large language models based on low-rank and sparse approximation. In *International Conference on Machine Learning*, pp. 20336–20350. PMLR, 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Mingkai Luo, Dan Meng, Jun Wang, Yu Fang, and Jiangyong Ying. FedLRS: A communication-efficient federated learning framework with low-rank and sparse decomposition. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 2091–2099, 2023. doi: 10.1109/ICPADS60453.2023.00284.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Mahdi Nikdan, Soroush Tabesh, Elvir Crnčević, and Dan Alistarh. RoSA: Accurate parameter-efficient fine-tuning via robust adaptation. In *International Conference on Machine Learning*, 2024.

Andres Potapczynski, Marc Finzi, Geoff Pleiss, and Andrew G Wilson. CoLA: Exploiting compositional structure for automatic and efficient numerical linear algebra. *Advances in Neural Information Processing Systems*, 36:43894–43917, 2023.

Shikai Qiu, Andres Potapczynski, Marc Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices. *arXiv preprint arXiv:2406.06248*, 2024.

Xinchi Qiu, Javier Fernandez-Marques, Pedro PB Gusmao, Yan Gao, Titouan Parcollet, and Nicholas Donald Lane. ZeroFL: Efficient on-device training for federated learning with local sparsity. *arXiv preprint arXiv:2208.02507*, 2022.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

Lorenzo Sani, Alex Iacob, Zeyu Cao, Bill Marino, Yan Gao, Tomas Paulik, Wanru Zhao, William F Shen, Preslav Aleksandrov, Xinchi Qiu, et al. The future of large language model pre-training is federated. *arXiv preprint arXiv:2405.10853*, 2024.

Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your ViT? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

Shangchao Su, Bin Li, and Xiangyang Xue. FedRA: A random allocation strategy for federated tuning to unleash the power of heterogeneous clients. In *European Conference on Computer Vision*, pp. 342–358. Springer, 2024.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.

Feijie Wu, Xingchen Wang, Yaqing Wang, Tianci Liu, Lu Su, and Jing Gao. FIARSE: Model-heterogeneous federated learning via importance-aware submodel extraction. *Advances in Neural Information Processing Systems*, 37:115615–115651, 2024.

Yunlu Yan, Chun-Mei Feng, Wangmeng Zuo, Rick Siow Mong Goh, Yong Liu, and Lei Zhu. Federated residual low-rank adaptation of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7370–7379, 2017.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://arxiv.org/abs/2312.05821.

Zikai Zhang, Ping Liu, Jiahao Xu, and Rui Hu. Fed-HeLLo: Efficient federated foundation model fine-tuning with heterogeneous LoRA allocation. *arXiv preprint arXiv:2506.12213*, 2025.

Zikai Zhang, Rui Hu, and Jiahao Xu. Heterogeneous federated fine-tuning with parallel one-rank adaptation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=sXPaVl9KU6.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *International Conference on Machine Learning*, 2024.

SUPPLEMENTARY MATERIAL: *SpaRFL: Sparse Low-Rank Federated Learning for Memory-Constrained Clients*

A EXPERIMENTAL CONFIGURATIONS AND HYPERPARAMETERS

The source code for reproducing the results can be found in the anonymized code repository: `https://anonymous.4open.science/r/sparfl`. The repository includes a README file explaining the setup steps. All experiments were run on NVIDIA GH200 GPUs and require around 4,100 GPU hours in total.

To ensure the reproducibility of our results, we provide the detailed hyperparameter configurations used across our computer vision and natural language processing benchmarks. Across all experiments, we employ a cosine annealing learning rate schedule to facilitate stable convergence.

Computer Vision (CIFAR-10). For the CIFAR-10 experiments, we train a ViT-Tiny from scratch, and utilize a population of 10 clients, performing 200 communication rounds. Each client executes 5 local epochs per round with a batch size of 64. Optimization is performed using Stochastic Gradient Descent (SGD) with a base learning rate of 0.1, decayed to 0.0 over the course of training.

NLP Fine-tuning (GLUE Benchmark). We evaluate SpaRFL on the QNLI, MRPC, and SST-2 tasks using the pretrained RoBERTa-base architecture. We simulate a federated environment with 5 clients and a Dirichlet-based data partition ($\alpha = 1.0$) to reflect realistic non-IID conditions. Clients train for 20 communication rounds, performing 1 local epoch per round. We utilize the AdamW optimizer with a sequence length of 128. We adjust the low-rank capacity based on the projected-state budget: for the 70% and 90% reductions, we set a fixed rank $r = 6$ to prioritize sparse density, while a rank of $r = 128$ is used for less-constrained tiers.

Language Modeling (WikiText-2). For the WikiText-2 task we fine-tune a pretrained Llama-3.2-1B, 5 clients are trained for 3 communication rounds using an IID data distribution. We preprocess WikiText-2 by filtering empty lines, tokenizing without adding special tokens, concatenating the resulting tokens into a packed stream (inserting EOS between lines), and chunking into fixed-length 1024-token blocks. We use a micro-batch size of 8 and 8 gradient accumulation steps, resulting in an effective batch size of 64 per client. We use rank $r = 6$ at the 90% projected-state budget reduction, $r = 64$ at the 70% reduction, and $r = 128$ for the remaining tiers.

Language Modeling Pretraining (C4). We pretrain LLaMA-60M and LLaMA-130M from scratch on the `allenai/c4` corpus using the `t5-base` tokenizer with a sequence (block) length of 256 tokens and `bfloat16` precision. The 60M architecture uses 8 layers, hidden size 512, FFN size 1376, and 8 attention heads; the 130M architecture uses 12 layers, hidden size 768, FFN size 2048, and 12 attention heads. We simulate a federated environment with 5 clients under an IID partition; all clients participate in every round and perform one local epoch per round. Each round consists of 1200 (60M) or 600 (130M) microbatches with a per-device batch size of 32 (60M) or 64 (130M) and 16 (60M) or 8 (130M) gradient-accumulation steps. Both configurations therefore perform 75 optimizer updates per round and use an effective batch of 512 sequences per update. We train for 110 (60M) and 112 (130M) communication rounds with AdamW (weight decay 0), an initial learning rate of $1\mathrm{e}{-3}$ decayed via cosine annealing to $1\mathrm{e}{-4}$, and global gradient-norm clipping. Held-out perplexity is evaluated on 10M C4 tokens at the final server checkpoint.

B CONVERGENCE ANALYSIS

We analyze the local parameterization and optimizer explicitly. The result is a finite-time stationarity bound that covers both reconstructed-model and delta aggregation. Its purpose is to expose the quantities created by heterogeneous sparse+low-rank training, rather than to hide them inside a dense local-SGD surrogate. For clarity, the theorem uses full participation and fixed data-proportional weights. Partial participation requires a separate sampling analysis because normalized data weights

Table 4: Summary of Hyperparameters for SpaRFL Benchmarks.

Hyperparameter	**CIFAR-10**	**GLUE**	**WikiText-2**	**C4 (LLaMA-60M/130M)**
Model Architecture	ViT-Tiny	RoBERTa-base	Llama-3.2-1B	LLaMA-60M / LLaMA-130M
Training Regime	Pretrain	Fine-tune	Fine-tune	Pretrain
Communication Rounds	200	20	3	110 / 112
Number of Clients	10	5	5	5
Local Epochs	5	1	1	1
Batch Size	64	64	8 ($\times$8 accum.)	32 ($\times$16 accum.) / 64 ($\times$8 accum.)
Sequence Length	–	128	1024	256
Optimizer	SGD	AdamW	AdamW	AdamW
Initial Learning Rate	0.1	3e-4	2e-5	1e-3
Final Learning Rate	0.0	1e-6	1e-5	1e-4
Learning Rate Schedule	Cosine	Cosine	Cosine	Cosine
Data Partition	IID, Dirichlet ($\alpha = 0.3$)	Dirichlet ($\alpha = 1.0$)	IID	IID
Low-rank allocation (γ or r)	$\gamma = 0.95$	$r \in \{6, 128\}$	$r \in \{6, 64, 128\}$	$\gamma = 0.03$

within an arbitrary sampled set need not be unbiased for the population objective; it is outside the theorem below.

B.1 Dense objective and factor-space local Adam

Let

$$\Phi(\mathcal{W}) \triangleq \sum_{i=1}^{n} q_i F_i(\mathcal{W}), \qquad q_i \triangleq \frac{|\mathcal{D}_i|}{|\mathcal{D}|}, \qquad F_i(\mathcal{W}) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}_i}[\ell(f(x;\mathcal{W}), y)] \tag{12}$$

be the dense federated objective. All matrices below may be read as vectorized; Euclidean and Frobenius norms then coincide. Unless stated otherwise, expectations are over minibatch sampling and any algorithmic randomness up to the indicated round and local step.

At round t, client i has a fixed sparse support $\Omega_{t,i}$ and factor variables $\theta_{t,i,k} = (s_{t,i,k}, A_{t,i,k}, B_{t,i,k})$ during its E local steps. Frozen components are included in the reconstruction map but omitted from $\theta_{t,i,k}$; in particular, a frozen sparse component is a constant term in the map and has no Jacobian column. Writing $P^*_\Omega s$ for scattering the stored sparse values into their matrix locations, the materialized client model is

$$Z_{t,i,k} = R_{t,i}(\theta_{t,i,k}) \triangleq P^*_{\Omega_{t,i}} s_{t,i,k} + A_{t,i,k} B_{t,i,k}, \qquad Z_{t,i,0} = p(\mathcal{W}_t; \mathcal{B}_i). \tag{13}$$

The notation stacks all projected layers; unprojected trainable blocks are included through identity components of $R_{t,i}$. Let $J_{t,i,k}$ be the Jacobian of $R_{t,i}$ at $\theta_{t,i,k}$ and $J^*_{t,i,k}$ its adjoint. On a projected layer, their actions are explicitly

$$\begin{aligned} J_{t,i,k}(d_s, d_A, d_B) &= P^*_{\Omega_{t,i}} d_s + d_A B_{t,i,k} + A_{t,i,k} d_B, \\ J^*_{t,i,k} G &= (P_{\Omega_{t,i}} G,\ G B^\top_{t,i,k},\ A^\top_{t,i,k} G), \end{aligned} \tag{14}$$

with coordinates corresponding to frozen components omitted. For $f_{t,i}(\theta) \triangleq F_i(R_{t,i}(\theta))$, the chain rule gives

$$\nabla f_{t,i}(\theta_{t,i,k}) = J^*_{t,i,k} \nabla F_i(Z_{t,i,k}). \tag{15}$$

Clients use Adam/AdamW (Kingma & Ba, 2015; Loshchilov & Hutter, 2019). Given the stochastic factor gradient $g_{t,i,k}$, the local recursions are

$$\begin{aligned} m_{t,i,k} &= \beta_1 m_{t,i,k-1} + (1-\beta_1) g_{t,i,k}, & \widehat{m}_{t,i,k} &= m_{t,i,k}/(1-\beta_1^{k+1}), \\ v_{t,i,k} &= \beta_2 v_{t,i,k-1} + (1-\beta_2) g^{\odot 2}_{t,i,k}, & \widehat{v}_{t,i,k} &= v_{t,i,k}/(1-\beta_2^{k+1}), \\ H_{t,i,k} &= \Lambda_{t,i}\, \mathrm{diag}\Big((\sqrt{\widehat{v}_{t,i,k}} + \epsilon)^{-1} \Big), & d_{t,i,k} &= H_{t,i,k} \widehat{m}_{t,i,k} + w^{\mathrm{wd}}_{t,i,k}, \\ \theta_{t,i,k+1} &= \theta_{t,i,k} - \eta d_{t,i,k}. \end{aligned} \tag{16}$$

Here $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, and $\Lambda_{t,i}$ contains the fixed per-factor learning-rate multipliers. The vector $w^{\mathrm{wd}}_{t,i,k}$ is the exact decoupled weight-decay direction, including those multipliers; under a single learning rate ($\Lambda_{t,i} = I$) it is simply $\lambda_{\mathrm{wd}} \theta_{t,i,k}$. The displayed bias correction corresponds to moments reset for a newly projected local problem. The theorem below also permits carried moments by retaining their tracking error explicitly. SGD is included as the special case $\widehat{m}_{t,i,k} = g_{t,i,k}$, $H_{t,i,k} = I$, and $w^{\mathrm{wd}}_{t,i,k} = 0$.

Lemma B.1 (Exact materialized factor update)**.** *For each projected layer, the Adam step in Equation* (16) *induces*

$$Z_{t,i,k+1} - Z_{t,i,k} = -\eta J_{t,i,k} d_{t,i,k} + Q_{t,i,k}, \qquad Q^l_{t,i,k} = \eta^2 d^l_{A,t,i,k} d^l_{B,t,i,k}. \tag{17}$$

Thus, the first-order term lies in the sparse+low-rank tangent space and $Q_{t,i,k}$ is the exact bilinear remainder.

Proof. The sparse scatter is linear. For the low-rank term, expand $(A - \eta d_A)(B - \eta d_B) - AB = -\eta(d_A B + A d_B) + \eta^2 d_A d_B$. Stacking the layers proves the claim. □

For example, with unpreconditioned factor SGD, no decay, and a dense layer gradient G, Equations (14) and (17) gives the first-order materialized direction

$$P^*_{\Omega_{t,i}} P_{\Omega_{t,i}} G + G B^\top_{t,i,k} B_{t,i,k} + A_{t,i,k} A^\top_{t,i,k} G, \tag{18}$$

plus the exact bilinear remainder. Thus even the SGD special case is a support- and factor-dependent direction, not a dense SGD surrogate.

B.2 A UNIFIED AGGREGATION RULE

Let

$$u_t \triangleq \sum_{i=1}^n q_i (Z_{t,i,E} - Z_{t,i,0}), \qquad b_t \triangleq \sum_{i=1}^n q_i Z_{t,i,0} - \mathcal{W}_t. \tag{19}$$

Both aggregation interfaces are represented by

$$\mathcal{W}_{t+1} = \mathcal{W}_t + u_t + \lambda_{\text{agg}} b_t, \qquad \lambda_{\text{agg}} \in \{0, 1\}. \tag{20}$$

Here $\lambda_{\text{agg}} = 0$ is delta aggregation, whereas $\lambda_{\text{agg}} = 1$ gives $\mathcal{W}_{t+1} = \sum_i q_i Z_{t,i,E}$, the reconstructed-model form.

Define the projection residual and mean projection displacement by

$$\begin{aligned} \delta_t^2 &\triangleq \sum_{i=1}^n q_i \|p(\mathcal{W}_t; \mathcal{B}_i) - \mathcal{W}_t\|_F^2, & \bar{\delta}_T^2 &\triangleq \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\delta_t^2], \\ \rho_t^2 &\triangleq \mathbb{E}\|b_t\|_F^2, & \bar{\rho}_T^2 &\triangleq \frac{1}{T} \sum_{t=0}^{T-1} \rho_t^2. \end{aligned} \tag{21}$$

Jensen's inequality immediately gives

$$\rho_t^2 \le \mathbb{E}[\delta_t^2], \qquad \bar{\rho}_T^2 \le \bar{\delta}_T^2. \tag{22}$$

The distinction matters: δ_t controls how far individual client models are projected, while ρ_t measures the systematic part that remains after averaging their projection displacements.

Proposition B.2 (Projection-induced gradient mismatch)**.** *If every F_i is L-smooth, then*

$$\sum_{i=1}^n q_i \|\nabla F_i(\mathcal{W}) - \nabla F_i(p(\mathcal{W}; \mathcal{B}_i))\|^2 \le L^2 \sum_{i=1}^n q_i \|\mathcal{W} - p(\mathcal{W}; \mathcal{B}_i)\|_F^2. \tag{23}$$

Proof. Apply L-smoothness term by term and weight by q_i. □

B.3 ASSUMPTIONS AND ERROR QUANTITIES

The assumptions below are standard bounded-trajectory conditions for non-convex adaptive optimization (Défossez et al., 2022) together with a coverage condition specific to heterogeneous parameterizations.

Assumption B.3 (Smoothness, lower boundedness, and heterogeneity)**.** Every F_i is differentiable and L-smooth, $\Phi^\star \triangleq \inf_{\mathcal{W}} \Phi(\mathcal{W}) > -\infty$, and there is $\zeta \ge 0$ such that, for all $\mathcal{W}$,

$$\sum_{i=1}^n q_i \|\nabla F_i(\mathcal{W}) - \nabla \Phi(\mathcal{W})\|^2 \le \zeta^2. \tag{24}$$

Assumption B.4 (Factor-gradient oracle)**.** Conditioned on the local history $\mathcal{F}_{t,i,k}$ before drawing the minibatch at step k, $\mathbb{E}[g_{t,i,k} \mid \mathcal{F}_{t,i,k}] = \nabla f_{t,i}(\theta_{t,i,k})$ and

$$\mathbb{E}[\|g_{t,i,k} - \nabla f_{t,i}(\theta_{t,i,k})\|^2 \mid \mathcal{F}_{t,i,k}] \le \sigma_\theta^2. \tag{25}$$

Assumption B.5 (Bounded factor-Adam dynamics)**.** Along the iterates there are constants $0 < \lambda_- \le \lambda_+$ and $G_\theta, J, \Theta, W_{\mathrm{wd}} < \infty$ such that

$$\begin{aligned} \lambda_- I \preceq \Lambda_{t,i} \preceq \lambda_+ I, \quad \|g_{t,i,k}\| \le G_\theta, \quad \|J_{t,i,k}\|_{\mathrm{op}} \le J, \\ \|\theta_{t,i,k}\| \le \Theta, \quad \|w^{\mathrm{wd}}_{t,i,k}\| \le W_{\mathrm{wd}}. \end{aligned} \tag{26}$$

The gradient bound applies to all gradients entering the moment recursions; thus it also covers correctly bias-corrected moments carried from earlier rounds. The composite factor gradients are L_θ-Lipschitz on this bounded trajectory. Define

$$\underline{h} \triangleq \frac{\lambda_-}{G_\theta + \epsilon}, \qquad \overline{h} \triangleq \frac{\lambda_+}{\epsilon}, \qquad D \triangleq \overline{h} G_\theta + W_{\mathrm{wd}}. \tag{27}$$

The constants in Equation (27) are consequences rather than additional assumptions. Indeed, bias correction makes each coordinate of $\widehat{m}$ and $\widehat{v}$ a convex average of bounded past gradients and their squares. Hence $\underline{h} I \preceq H_{t,i,k} \preceq \overline{h} I$ and $\|d_{t,i,k}\| \le D$. Define the sharp trajectory bound $R \triangleq \sup_{t,i,k} \sum_l \|d^l_{A,t,i,k}\|_F \|d^l_{B,t,i,k}\|_F$. Then

$$\|Q_{t,i,k}\| \le \eta^2 \sum_l \|d^l_{A,t,i,k}\|_F \|d^l_{B,t,i,k}\|_F \le \eta^2 R, \qquad R \le \frac{D^2}{2}. \tag{28}$$

For standard AdamW, a bound on the factor iterates also yields $W_{\mathrm{wd}} \le \lambda_{\mathrm{wd}} \lambda_+ \Theta$ under the multiplier convention in Equation (16).

Assumption B.6 (Collective tangent coverage)**.** There is $\kappa > 0$ such that, for every round t, with $G_t \triangleq \nabla \Phi(\mathcal{W}_t)$,

$$\frac{1}{E} \sum_{i=1}^n q_i \sum_{k=0}^{E-1} \mathbb{E} \|J^*_{t,i,k} G_t\|^2 \ge \kappa \, \mathbb{E} \|G_t\|^2. \tag{29}$$

Assumption B.6 is the natural identifiability condition for heterogeneous low-dimensional clients: directions invisible to every client cannot be optimized. It requires collective, rather than per-client, coverage and therefore permits each individual tangent space to be low-dimensional.

Proposition B.7 (A checkable sufficient condition for coverage)**.** *For a projected layer l, let*

$$\begin{aligned} c^l_t &\triangleq \min_{u,v} \frac{1}{E} \sum_{i=1}^n q_i \sum_{k=0}^{E-1} \mathbf{1}\{(u,v) \textit{ is a trainable sparse coordinate of client } i\}, \\ C^l_{A,t} &\triangleq \frac{1}{E} \sum_{i=1}^n q_i \sum_{k=0}^{E-1} A^l_{t,i,k} (A^l_{t,i,k})^\top, \\ C^l_{B,t} &\triangleq \frac{1}{E} \sum_{i=1}^n q_i \sum_{k=0}^{E-1} (B^l_{t,i,k})^\top B^l_{t,i,k}. \end{aligned} \tag{30}$$

Set

$$\kappa^{\mathrm{suf}}_t \triangleq \min_l \left\{ c^l_t + \lambda_{\min}(C^l_{A,t}) + \lambda_{\min}(C^l_{B,t}) \right\}, \tag{31}$$

where an unprojected trainable block contributes the value 1 to the minimum. If $\kappa^{\mathrm{suf}}_t \ge \kappa > 0$ almost surely for every round, then Assumption B.6 holds with that κ.

Proof. For any dense matrix G^l, Equation (14) gives

$$\begin{aligned} \frac{1}{E} \sum_{i=1}^n q_i \sum_{k=0}^{E-1} \|(J^l_{t,i,k})^* G^l\|_F^2 &= \frac{1}{E} \sum_{i,k} q_i \|P_{\Omega^l_{t,i}} G^l\|_F^2 + \mathrm{tr}((G^l)^\top C^l_{A,t} G^l) \\ &\quad + \mathrm{tr}(G^l C^l_{B,t} (G^l)^\top) \\ &\ge \left[c^l_t + \lambda_{\min}(C^l_{A,t}) + \lambda_{\min}(C^l_{B,t}) \right] \|G^l\|_F^2. \end{aligned}$$

Summing over blocks, taking the smallest block coefficient, and then taking expectations proves Equation (29). □

Assumption B.6 is trajectory-specific and requires the aggregate client tangent spaces to retain a non-vanishing component of the current dense gradient. It does not require any individual client tangent space to span the dense parameter space. Proposition B.7 provides a sufficient certificate based on sparse-coordinate coverage and factor Gram matrices averaged across clients and local steps; this certificate may be zero even when the trajectory-specific condition is positive. Our theorem should therefore be interpreted as a conditional stationarity result that makes the required representation coverage explicit, rather than as a claim that every sparse+low-rank trajectory automatically satisfies the condition.

The certificate also makes the role of heterogeneous parameterizations explicit: different sparse supports and factor subspaces can cover directions that no single client covers. A frozen sparse component contributes $c_t^l = 0$, while trainable sparse entries contribute directly. The implementation splits the retained singular values symmetrically as $U_r\Sigma_r^{1/2}$ and $\Sigma_r^{1/2}V_r^\top$ (in weighted coordinates for the NLP projection) before undoing the activation scaling. Consequently, the two factor terms in Equation (30) start from a canonical SVD scaling rather than an arbitrary rescaling of the product.

We next retain errors at the level of the *aggregated dense direction*; this preserves cancellation across clients. Define the positive semidefinite dense operator

$$\mathcal{P}_{t,i,k} \triangleq J_{t,i,k}H_{t,i,k}J_{t,i,k}^*, \qquad \mathcal{M}_t \triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} \mathcal{P}_{t,i,k}. \tag{32}$$

The aggregated representation/client mismatch and optimizer-tracking vectors are

$$\begin{aligned} a_t &\triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} \mathcal{P}_{t,i,k}\big(\nabla F_i(Z_{t,i,k}) - G_t\big), \\ c_t &\triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} J_{t,i,k}H_{t,i,k}\left[\widehat{m}_{t,i,k} - \nabla f_{t,i}(\theta_{t,i,k}) + H_{t,i,k}^{-1}w_{t,i,k}^{\mathrm{wd}}\right], \\ e_t &\triangleq a_t + c_t, \qquad \varepsilon_t^2 \triangleq \mathbb{E}\|e_t\|^2. \end{aligned} \tag{33}$$

For an explicit optimizer bound, retain the factor-space quantity

$$\omega_t^2 \triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} \mathbb{E}\left\|\widehat{m}_{t,i,k} - \nabla f_{t,i}(\theta_{t,i,k}) + H_{t,i,k}^{-1}w_{t,i,k}^{\mathrm{wd}}\right\|^2. \tag{34}$$

These quantities are not assumed to vanish. The next result bounds them while preserving exact cancellation of the client gradients at $\mathcal{W}_t$.

Lemma B.8 (Aggregated representation and optimizer error)**.** *Let*

$$U \triangleq JD + \eta R, \qquad \Gamma \triangleq \overline{h}J^2, \qquad \upsilon_t \triangleq \operatorname{ess\,sup}\max_{i,k} \|\mathcal{P}_{t,i,k} - \mathcal{M}_t\|_{\mathrm{op}}, \tag{35}$$

and define the non-cancelling heterogeneity component

$$\xi_t \triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} (\mathcal{P}_{t,i,k} - \mathcal{M}_t)(\nabla F_i(\mathcal{W}_t) - G_t), \qquad \Xi_t^2 \triangleq \mathbb{E}\|\xi_t\|^2. \tag{36}$$

Under Assumptions B.3 and B.5,

$$\Xi_t^2 \le \upsilon_t^2\zeta^2, \tag{37}$$

$$\mathbb{E}\|a_t\|^2 \le 4\Gamma^2L^2\left(\mathbb{E}[\delta_t^2] + \eta^2(E-1)^2U^2\right) + 2\Xi_t^2, \tag{38}$$

$$\varepsilon_t^2 \le 8\Gamma^2L^2\left(\mathbb{E}[\delta_t^2] + \eta^2(E-1)^2U^2\right) + 4\Xi_t^2 + 2J^2\overline{h}^2\omega_t^2. \tag{39}$$

In particular, $\Xi_t = 0$ whenever the dense operators $\mathcal{P}_{t,i,k}$ are common across clients and local steps, irrespective of ζ; moreover $\upsilon_t \le 2\Gamma$, so the coarse heterogeneity bound is always finite under Assumption B.5.

Proof. Because $\|\mathcal{P}_{t,i,k}\|_{\mathrm{op}} \le \Gamma$ and $\sum_i q_i(\nabla F_i(\mathcal{W}_t) - G_t) = 0$, the client term at the common point satisfies

$$\frac{1}{E}\sum_{i,k} q_i \mathcal{P}_{t,i,k}(\nabla F_i(\mathcal{W}_t) - G_t) = \xi_t.$$

Jensen's inequality and Equation (24) give $\Xi_t^2 \le \upsilon_t^2 \zeta^2$.

By Equations (17) and (28), $\|Z_{t,i,k+1} - Z_{t,i,k}\| \le \eta U$ and hence

$$\frac{1}{E}\sum_{i,k} q_i \mathbb{E}\|Z_{t,i,k} - \mathcal{W}_t\|^2 \le 2\mathbb{E}[\delta_t^2] + 2\eta^2(E-1)^2U^2. \tag{40}$$

Adding and subtracting $\nabla F_i(\mathcal{W}_t)$ in the definition of a_t, and using Jensen's inequality and L-smoothness, therefore gives

$$\mathbb{E}\|a_t\|^2 \le 2\Gamma^2 L^2 \frac{1}{E}\sum_{i,k} q_i \mathbb{E}\|Z_{t,i,k} - \mathcal{W}_t\|^2 + 2\Xi_t^2,$$

which proves Equation (38). Similarly, Jensen's inequality applied to c_t gives $\mathbb{E}\|c_t\|^2 \le J^2 \overline{h}^2 \omega_t^2$. Finally use $\|a_t + c_t\|^2 \le 2\|a_t\|^2 + 2\|c_t\|^2$. □

Lemma B.9 (Adam moment tracking)**.** *Suppose the moments are reset at the start of each projected local problem. For*

$$a_{k,j} \triangleq \frac{(1-\beta_1)\beta_1^{k-j}}{1-\beta_1^{k+1}}, \quad \nu_E \triangleq \frac{1}{E}\sum_{k=0}^{E-1}\sum_{j=0}^{k} a_{k,j}^2, \quad \tau_E \triangleq \frac{1}{E}\sum_{k=0}^{E-1}\sum_{j=0}^{k} a_{k,j}(k-j)^2, \tag{41}$$

Under Assumptions B.4 and B.5,

$$\omega_t^2 \le 4\nu_E \sigma_\theta^2 + 4L_\theta^2 \eta^2 D^2 \tau_E + \frac{2W_{\mathrm{wd}}^2}{\underline{h}^2}. \tag{42}$$

Here $\nu_E \le 1$ and $\tau_E \le (E-1)^2$; retaining them exposes the dependence on Adam's first-moment averaging rather than replacing both by worst-case constants. For carried moments, Theorem B.10 remains valid with the realized ω_t^2 in Equation (34)*. The same statement covers gradient clipping or other deterministic optimizer modifications through that realized tracking term; the explicit bound above uses the unbiased oracle in Assumption B.4.*

Proof. Bias correction writes $\widehat{m}_{t,i,k} = \sum_{j=0}^{k} a_{k,j} g_{t,i,j}$, where the coefficients in Equation (41) are nonnegative and sum to one. Add and subtract the corresponding true factor gradients. The martingale-difference property in Assumption B.4 makes cross terms vanish, so the noise contribution is at most $\sigma_\theta^2 \sum_{j=0}^{k} a_{k,j}^2$. Lipschitzness and $\|\theta_{t,i,j} - \theta_{t,i,k}\| \le \eta D|k-j|$ bound the squared gradient-drift contribution, by Jensen, by $L_\theta^2 \eta^2 D^2 \sum_{j=0}^{k} a_{k,j}(k-j)^2$. Applying $\|x+y\|^2 \le 2\|x\|^2 + 2\|y\|^2$ first to noise versus drift and then to the weight-decay term, together with $\|H_{t,i,k}^{-1}\|_{\mathrm{op}} \le 1/\underline{h}$, and averaging over clients and local steps proves the result. □

B.4 Unified stationarity theorem

Define $\bar{\varepsilon}_T^2 = T^{-1}\sum_t \varepsilon_t^2$, $\bar{\Xi}_T^2 = T^{-1}\sum_t \Xi_t^2$, and $\bar{\omega}_T^2 = T^{-1}\sum_t \omega_t^2$. Also write

$$\mu \triangleq \underline{h}\kappa, \qquad C_\lambda \triangleq \frac{4}{\mu^2} + \frac{8L(1+\lambda)\eta E}{\mu}. \tag{43}$$

Theorem B.10 (Factor-Adam bound for both aggregation interfaces)**.** *Under Assumptions B.3, B.5, and B.6, the full-participation iterates generated by Equations* (16) *and* (20) *with constant base stepsize $\eta > 0$ satisfying*

$$\eta E \le \frac{\mu}{8L(1+\lambda_{\mathrm{agg}})\Gamma^2} \tag{44}$$

satisfy

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla\Phi(\mathcal{W}_t)\|^2 \le \frac{4(\Phi(\mathcal{W}_0)-\Phi^\star)}{\eta ET\mu} + C_{\lambda_{\mathrm{agg}}}\bar{\varepsilon}_T^2 + \frac{8\eta^2R^2}{\mu^2} + \frac{4L(1+\lambda_{\mathrm{agg}})\eta^3ER^2}{\mu} + \lambda_{\mathrm{agg}}\left(\frac{8\bar{\rho}_T^2}{\eta^2E^2\mu^2} + \frac{4L\bar{\rho}_T^2}{\eta E\mu}\right), \tag{45}$$

where $\Gamma = \overline{h}J^2$. The terms respectively represent finite-time optimization, aggregated direction error, the exact bilinear factor remainder, and—only for reconstructed-model aggregation—mean projection displacement.

Proof. Fix a round and write $G_t = \nabla\Phi(\mathcal{W}_t)$. From Equations (15) and (33), the average first-order materialized direction is exactly

$$v_t \triangleq \frac{1}{E}\sum_{i=1}^{n} q_i \sum_{k=0}^{E-1} J_{t,i,k}d_{t,i,k} = \mathcal{M}_tG_t + e_t. \tag{46}$$

Since $H_{t,i,k} \succeq \underline{h}I$, collective tangent coverage gives

$$\mathbb{E}\langle G_t, \mathcal{M}_tG_t\rangle \ge \mu\,\mathbb{E}\|G_t\|^2. \tag{47}$$

Consequently, Young's inequality yields

$$\mathbb{E}\langle G_t, v_t\rangle \ge \frac{3\mu}{4}\mathbb{E}\|G_t\|^2 - \frac{\varepsilon_t^2}{\mu}. \tag{48}$$

Furthermore $\|\mathcal{M}_t\|_{\mathrm{op}} \le \Gamma$, so

$$\mathbb{E}\|v_t\|^2 \le 2\Gamma^2\mathbb{E}\|G_t\|^2 + 2\varepsilon_t^2. \tag{49}$$

Summing Equation (17) over local steps and clients yields $u_t = -\eta Ev_t + r_t$, with $\|r_t\| \le \eta^2ER$. It also gives $\|u_t\| \le \eta EU$. By L-smoothness and Equation (20),

$$\Phi(\mathcal{W}_{t+1}) \le \Phi(\mathcal{W}_t) + \langle G_t, u_t + \lambda_{\mathrm{agg}}b_t\rangle + \frac{L}{2}\|u_t + \lambda_{\mathrm{agg}}b_t\|^2. \tag{50}$$

For $\lambda \in \{0,1\}$,

$$\|u_t + \lambda b_t\|^2 \le (1+\lambda)\|u_t\|^2 + 2\lambda\|b_t\|^2, \qquad \|u_t\|^2 \le 2\eta^2E^2\|v_t\|^2 + 2\|r_t\|^2. \tag{51}$$

Young's inequality also gives

$$\langle G_t, r_t\rangle \le \frac{\eta E\mu}{8}\|G_t\|^2 + \frac{2\|r_t\|^2}{\eta E\mu},$$

and, when $\lambda_{\mathrm{agg}} = 1$,

$$\langle G_t, b_t\rangle \le \frac{\eta E\mu}{8}\|G_t\|^2 + \frac{2\|b_t\|^2}{\eta E\mu}.$$

Substitute Equations (48), (49) and (51) into Equation (50). The stepsize condition Equation (44) absorbs the resulting $2L(1+\lambda_{\mathrm{agg}})\eta^2E^2\Gamma^2\mathbb{E}\|G_t\|^2$ term. Using $\|r_t\|^2 \le \eta^4E^2R^2$ then gives

$$\mathbb{E}\Phi(\mathcal{W}_{t+1}) \le \mathbb{E}\Phi(\mathcal{W}_t) - \frac{\eta E\mu}{4}\mathbb{E}\|G_t\|^2 + \left(\frac{\eta E}{\mu} + 2L(1+\lambda_{\mathrm{agg}})\eta^2E^2\right)\varepsilon_t^2 + \frac{2\eta^3ER^2}{\mu} + L(1+\lambda_{\mathrm{agg}})\eta^4E^2R^2 + \lambda_{\mathrm{agg}}\left(\frac{2\rho_t^2}{\eta E\mu} + L\rho_t^2\right).$$

Sum over t, use $\Phi(\mathcal{W}_T) \ge \Phi^\star$, and divide by $\eta ET\mu/4$. □

For compactness, denote the primitive upper bound on the aggregated direction error by

$$\mathfrak{E}_T \triangleq 8\Gamma^2 L^2\big(\bar{\delta}_T^2 + \eta^2(E-1)^2U^2\big) + 4\bar{\Xi}_T^2 + 2J^2\overline{h}^2\bar{\omega}_T^2. \tag{52}$$

Corollary B.11 (Delta aggregation). *For $\lambda_{\mathrm{agg}} = 0$, combining Theorem B.10 and Lemma B.8 gives*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla\Phi(\mathcal{W}_t)\|^2 \le \frac{4(\Phi(\mathcal{W}_0)-\Phi^\star)}{\eta ET\mu} + C_0\mathfrak{E}_T + \frac{8\eta^2R^2}{\mu^2} + \frac{4L\eta^3ER^2}{\mu}. \tag{53}$$

For reset moments, Lemma B.9 supplies an explicit bound on $\bar{\omega}_T^2$, while $\bar{\Xi}_T^2 \le T^{-1}\sum_t \upsilon_t^2\zeta^2$. Thus only the part of client heterogeneity that interacts with variation among the client tangent preconditioners remains in the bound.

Corollary B.12 (Reconstructed-model aggregation). *For $\lambda_{\mathrm{agg}} = 1$, Theorem B.10 and Lemma B.8 give*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla\Phi(\mathcal{W}_t)\|^2 \le \frac{4(\Phi(\mathcal{W}_0)-\Phi^\star)}{\eta ET\mu} + C_1\mathfrak{E}_T + \frac{8\eta^2R^2}{\mu^2} + \frac{8L\eta^3ER^2}{\mu} + \frac{8\bar{\rho}_T^2}{\eta^2E^2\mu^2} + \frac{4L\bar{\rho}_T^2}{\eta E\mu}, \qquad \bar{\rho}_T^2 \le \bar{\delta}_T^2. \tag{54}$$

In particular, if the mean projection displacement obeys $\bar{\rho}_T^2 \le \eta^2E^2\varepsilon_{\mathrm{agg}}^2$, the additional neighborhood is at most

$$\frac{8\varepsilon_{\mathrm{agg}}^2}{\mu^2} + \frac{4L\eta E\varepsilon_{\mathrm{agg}}^2}{\mu}. \tag{55}$$

Thus model aggregation is stable when client projection displacements cancel on average or remain commensurate with the magnitude of a local optimization round. Delta aggregation is the special case that removes this displacement identically.

Dense-limit sanity check. For a deterministic full-gradient step with a common identity parameterization, $E = 1$, $H = I$, and no factor remainder, one has $\mu = \Gamma = 1$ and $\bar{\delta}_T = \bar{\Xi}_T = \bar{\omega}_T = R = 0$. The delta corollary reduces to

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla\Phi(\mathcal{W}_t)\|^2 \le \frac{4(\Phi(\mathcal{W}_0)-\Phi^\star)}{\eta T}, \qquad \eta \le \frac{1}{8L}, \tag{56}$$

so the bound recovers ordinary full-gradient stationarity without a spurious heterogeneity floor.

Interpretation. The theorem separates effects that are absent from a dense local-SGD model. κ measures collective access to dense descent directions; $\bar{\delta}_T$ controls local mismatch caused by projected initialization; $\bar{\Xi}_T$ measures the interaction of client heterogeneity with variation among the client tangent preconditioners; $\bar{\omega}_T$ records Adam's stochastic moment tracking and decoupled weight decay (with the second-moment preconditioner represented by $\underline{h}, \overline{h}$); R is the exact second-order effect of updating both low-rank factors; and $\bar{\rho}_T$ is the systematic projection displacement retained by model aggregation. The result is deliberately a neighborhood guarantee: with fixed client budgets or persistent stochasticity, these quantities need not vanish.

B.5 Deterministic characterization of the projection residual

The next two propositions characterize projection error using familiar spectral quantities. The vision result directly bounds the raw Frobenius residual in Theorem B.10; the NLP result controls the activation-weighted residual optimized by the activation-aware projection.

Proposition B.13 (Vision projection residual; SVD+top-k). *For a layer $W \in \mathbb{R}^{m\times n}$, let $L_r = U_r\Sigma_rV_r^\top$ be the best rank-r approximation of W via the truncated SVD, let $R_r = W - L_r$, and let $H_k(R_r)$ retain the k largest-magnitude entries of R_r (zeroing the rest). The vision projection $p(W) = L_r + H_k(R_r)$ of Equation (6) satisfies*

$$\|W - p(W)\|_F^2 \;=\; \|R_r - H_k(R_r)\|_F^2 \;\le\; \sum_{j>r}\sigma_j(W)^2. \tag{57}$$

Proof. The equality follows because H_k is the optimal k-sparse approximation of the fixed residual R_r in Frobenius norm. The inequality follows from the Eckart–Young theorem: $\|R_r\|_F^2 = \sum_{j>r} \sigma_j(W)^2$, and $\|R_r - H_k(R_r)\|_F^2 \le \|R_r\|_F^2$. □

This does not claim global optimality of p over all rank-r plus k-sparse decompositions; it characterizes the one-shot projection used by SpaRFL. The bound says SpaRFL's vision residual is at most the spectral tail of W beyond rank r.

Proposition B.14 (NLP projection residual; activation-weighted)**.** *Let $X^l \in \mathbb{R}^{T\times n_l}$ be calibration activations for layer l, and let $D^l = \mathrm{diag}(\|X^l_{:,1}\|_2, \dots, \|X^l_{:,n_l}\|_2)$ be the diagonal activation scaling in Equation* (7)*.*

For the sparse-first order, fix S^l, let $Q^l = (W^l - S^l)D^l$, and let Q^l_r be its best rank-r approximation. Then $L^l = Q^l_r(D^l)^\dagger$ minimizes $\mathcal{E}_{D^l}(S^l, L)$ over rank-r matrices, and

$$\|(W^l - S^l - L^l)D^l\|_F^2 \;=\; \sum_{j>r} \sigma_j(Q^l)^2. \tag{58}$$

For the low-rank-first order, let $Q^l = W^l D^l$, set $L^l = Q^l_r(D^l)^\dagger$, and let S^l retain any chosen entries of the residual $W^l - L^l$. Then

$$\|(W^l - S^l - L^l)D^l\|_F^2 \le \sum_{j>r} \sigma_j(Q^l)^2. \tag{59}$$

For a fixed global support budget, selecting the largest values of $|(W^l - L^l)_{uv}|(D^l)_{vv}$ minimizes the left-hand side; the same statement holds row by row when a per-row budget is imposed.

Proof. For sparse-first projection, apply Eckart–Young to Q^l and use the change of variables $L^l = Q^l_r(D^l)^\dagger$. Columns for which $(D^l)_{vv} = 0$ make no contribution to the objective and may be set arbitrarily, which is represented by the pseudoinverse.

For low-rank-first projection, Eckart–Young first gives $\|(W^l - L^l)D^l\|_F^2 = \sum_{j>r} \sigma_j(Q^l)^2$. Because D^l is diagonal and S^l equals the residual on its retained entries, subtracting S^l simply zeros the corresponding entries of $(W^l - L^l)D^l$ and cannot increase its Frobenius norm. The objective is separable across entries, so retaining the largest weighted residual entries is optimal for a global support budget, or row-wise optimal when the budget is imposed per row. □

This is the quantity most aligned with LLM behavior: it measures output distortion on calibration activations rather than raw parameter distance, matching the diagonal objective Equation (8). It does not by itself upper-bound the raw Frobenius residual used in Equation (21): such a conversion additionally requires a positive lower bound on the diagonal entries of D^l. A full-covariance factor can replace D^l in the sparse-first statement, but the entrywise monotonicity used by the low-rank-first inequality is specific to diagonal scaling.

B.6 Empirical characterization of the projection residual

Propositions B.13 and B.14 give worst-case bounds; the actual residual along the SpaRFL iterate path is typically much smaller, since trained Transformer weights have rapidly decaying spectra. We measure both the Frobenius residual $\|\mathcal{W}_t - p(\mathcal{W}_t; \mathcal{B}_i)\|_F / \|\mathcal{W}_t\|_F$ and the activation-weighted residual $\|(\mathcal{W}_t - p(\mathcal{W}_t; \mathcal{B}_i))D_l\|_F / \|\mathcal{W}_t D_l\|_F$ on real model matrices: every Dense layer of LLaMA-130M, at four saved SpaRFL server checkpoints (rounds $20, 40, 60, 80$), at homogeneous client projected-state budget reductions in $\{70\%, 80\%, 90\%\}$, comparing SpaRFL against sparse-only and low-rank-only projections at the same budget.

Figure 4 reports the aggregate $\bar{\delta}_t / \|\mathcal{W}_t\|_F$ across rounds. SpaRFL attains an activation-weighted residual of 0.10 at the 70% projected-state budget reduction by round 80, against 0.27 for sparse-only and 0.59 for low-rank-only at the same budget—a roughly $3\times$ and $6\times$ reduction respectively. The curves are flat to mildly decreasing across rounds (e.g., SpaRFL at 70% improves from 0.113 at round 20 to 0.098 at round 80), showing that projection distortion remains controlled along the observed training trajectory.

Figure 5 resolves the residual at the layer level for the latest checkpoint. Every one of the 84 linear layers (12 transformer blocks $\times$ 7 projection types) has a smaller residual under SpaRFL than under either sparse-only or low-rank-only at every budget. The largest gains over sparse-only occur on the attention output and MLP-down projections, where the low-rank component absorbs a substantial fraction of the weight energy; the largest gains over low-rank-only occur on the attention query/key/value projections, where outlier entries contribute meaningfully and the sparse component captures them efficiently.

Combined with Proposition B.2 and Propositions B.13–B.14, these measurements empirically characterize one of the quantities identified by Theorem B.10: smaller raw projection residual controls both the gradient-mismatch contribution and, through Equation (22), an upper bound on the mean displacement retained by model aggregation. The activation-weighted residual separately measures output distortion on the calibration distribution.

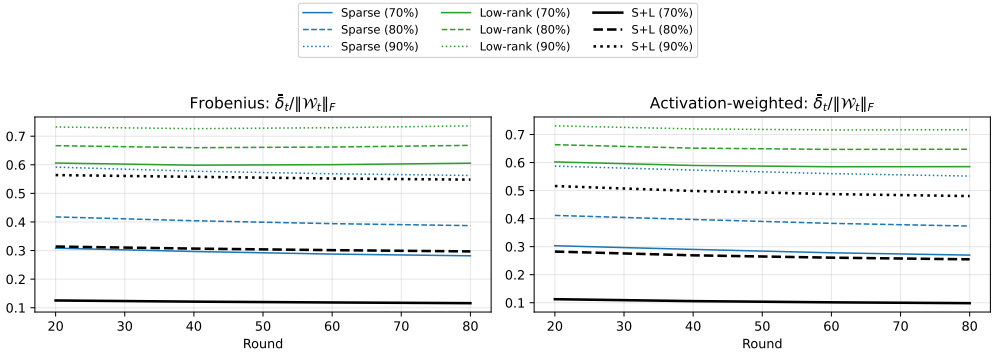

Figure 4: Normalized projection residual along the SpaRFL training trajectory, for three homogeneous budgets and three projection variants, on LLaMA-130M / C4. Left: Frobenius residual $\bar{\delta}_t/\|\mathcal{W}_t\|_F$. Right: activation-weighted residual with diagonal scaling $D_l = \mathrm{diag}(\|X^l_{:,j}\|_2)$. SpaRFL (black) attains the smallest residual at every round and budget; the curves are bounded throughout training and decrease as the budget grows.

C Comparison with Related Work

In Table 5, we summarize how SpaRFL differs from the most closely related method families. The key distinction is that SpaRFL is designed specifically for the *no-client-fits* regime, where no client can store and optimize the dense global model during training.

Sparse+low-rank compression and pretraining methods demonstrate that hybrid sparse–low-rank parameterizations can be effective, but they are not formulated as client-specific one-shot projections inside federated training. FedLRS (Luo et al., 2023) jointly trains low-rank and sparse components to reduce communication and computation, but does not construct client-specific projections of a dense server model under heterogeneous per-client memory budgets. Likewise, sparse+low-rank PEFT methods are designed for fine-tuning and typically assume that the full pretrained base model remains available, which makes them unsuitable for the strict memory regime considered here. Furthermore, prior sparse+low-rank methods operate in weight space (Nikdan et al., 2024; Han et al., 2024; Li et al., 2023), while prior activation-aware methods typically treat sparsity or low-rank structure separately (Sun et al., 2024; Yuan et al., 2025). In contrast, SpaRFL uses a joint sparse+low-rank activation-space projection for federated client models.

PEFT-based FL methods support heterogeneous adaptation during fine-tuning, but they generally require the frozen base model to fit on the client and therefore do not directly address the *no-client-fits* setting. For example, FLASC (Kuo et al., 2026) sparsifies communicated LoRA updates while clients retain the pretrained backbone and locally train the complete LoRA module; it primarily targets communication-efficient fine-tuning rather than client training-model memory. Federated sparse training methods reduce communication and computation through sparsity, but they typically maintain a sparse global model rather than preserving a dense server model with standard dense aggregation.

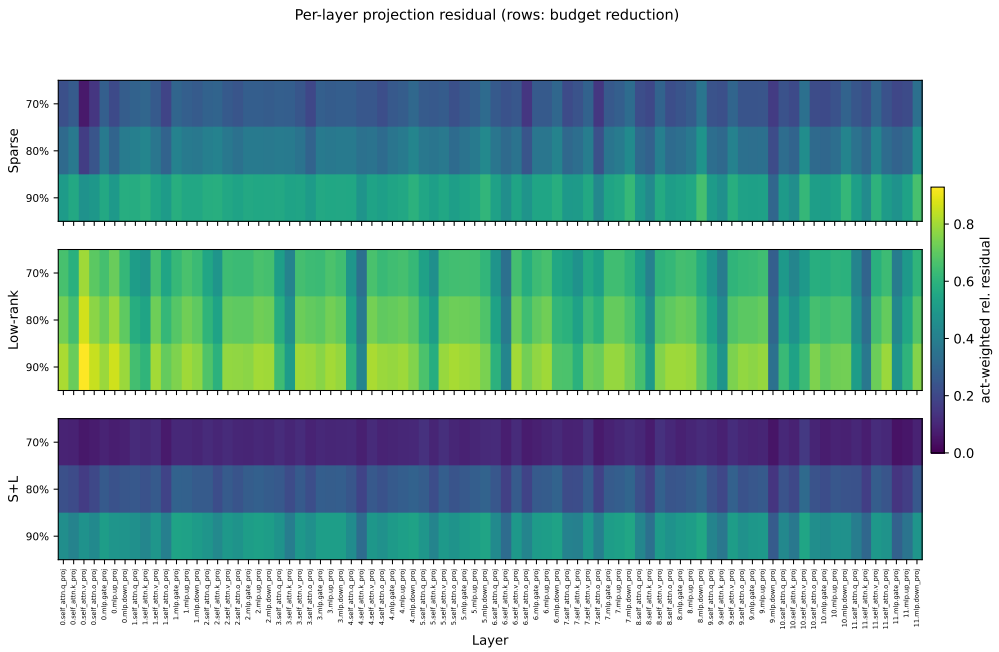

Figure 5: Per-layer activation-weighted projection residual for sparse-only, low-rank-only, and SpaRFL at projected-state budget reductions $\{70\%, 80\%, 90\%\}$, evaluated on the LLaMA-130M server checkpoint at round 80. Columns index the 84 linear layers (q/k/v/o attention plus gate/up/down MLP, across 12 transformer blocks), while rows are method–budget combinations. SpaRFL attains a uniformly smaller residual across layers and budgets.

Among prior model-heterogeneous FL approaches, HeteroFL, FjORD, and FIARSE are the closest comparisons, since they construct client-specific models and remain compatible with standard federated aggregation. However, they do not explicitly target the strict *no-client-fits* regime studied here, and in our experiments they degrade substantially in the fine-tuning setting under severe memory constraints. In contrast, SpaRFL is designed for this regime from the outset and remains effective across both pretraining and fine-tuning.

D Compatibility with Federated Optimizers

While FedAvg (McMahan et al., 2017) remains the de facto standard for federated orchestration, recent advancements have introduced a variety of adaptive optimization strategies to address challenges in convergence and data heterogeneity. To evaluate the optimizer-agnostic robustness of SpaRFL, we conduct experiments using two prominent adaptive strategies from the FedOpt framework: *FedAdam* and *FedAdagrad* (Reddi et al., 2021).

Table 6 summarizes the performance of SpaRFL when integrated with these adaptive optimizers in the homogeneous client setting (see Section 4). We observe that the performance gains of SpaRFL are consistent across all considered optimization strategies. These results further demonstrate the robustness of our approach, confirming that the benefits of our sparse plus low-rank re-parameterization are complementary to advancements in federated optimization.

E Sparse vs. Low-rank for LLaMA Pretraining

In this section, we extend the trade-off analysis between sparse and low-rank budget allocations from Section 4 to training LLaMA-60M from scratch (see Figure 3). Table 7 reports the C4 evaluation perplexity across these configurations. In contrast to the vision setting, increasing the allocation to the sparse component improves results. However, extremal configurations, allocating the budget exclusively to either the sparse or low-rank component, yield suboptimal performance.

Table 5: Comparison of SpaRFL with the most closely related lines of work. *Q1*: Does not require the full base model to reside on the client? *Q2*: Explicitly targets the *no-client-fits* regime? *Q3*: Supports from-scratch pretraining? *Q4*: Remains effective for fine-tuning under severe memory constraints? *Q5*: Uses client-specific model construction? *Q6*: Preserves standard dense server aggregation? Symbols denote full support (🔋), partial support (🪫), and no support (✖). Partial support indicates that a method is only applicable under weaker assumptions, in a restricted setting, or without directly targeting the regime studied in this work.

Method family	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*
Sparse+Low-Rank Compression / Pretraining Han et al. (2024); Li et al. (2023) Yu et al. (2017); Luo et al. (2023)	🔋	✖	🪫	🪫	✖	✖
Sparse+Low-Rank PEFT Nikdan et al. (2024)	✖	✖	✖	🔋	✖	✖
PEFT-based FL Hyeon-Woo et al. (2021); Babakniya et al. (2023a) Cho et al. (2024); Zhang et al. (2025) Yan et al. (2025); Guo et al. (2024) Kuo et al. (2026)	✖	✖	✖	🪫	🔋	🪫
Federated Sparse Training Guastella et al. (2025); Qiu et al. (2022) Babakniya et al. (2023b)	🔋	🪫	🪫	🪫	🪫	✖
HeteroFL Diao et al. (2021)	🔋	🪫	🔋	🪫	🔋	🔋
FjORD Horvath et al. (2021)	🔋	🪫	🔋	🪫	🔋	🔋
FIARSE Wu et al. (2024)	🔋	🪫	🔋	🪫	🔋	🔋
SpaRFL	🔋	🔋	🔋	🔋	🔋	🔋

Table 6: Robustness of SpaRFL across federated optimization strategies. We report WikiText-2 perplexity results when integrating SpaRFL with different federated orchestration algorithms. This demonstrates that SpaRFL is optimizer-agnostic.

PPL ↓	FedAvg	FedAdam	FedAdagrad
SpaRFL (70%)	11.97	12.41	14.00
SpaRFL (90%)	15.67	17.27	17.77

Table 7: Sensitivity analysis of the sparse and low-rank budget trade-off. We report C4 validation perplexity for SpaRFL training LLaMA-60M in the homogeneous setting at the 70% projected-state budget reduction. The hyperparameter γ allocates the available byte budget between the sparse and low-rank components.

Setting	C4 PPL ↓
S (100%) + L (0%)	38.25
S (97%) + L (3%)	37.94
S (95%) + L (5%)	38.50
S (90%) + L (10%)	48.51
S (30%) + L (70%)	119.05
S (5%) + L (95%)	112.54
S (3%) + L (97%)	112.94
S (0%) + L (100%)	114.02

F SPARSE MASK SELECTION AND INITIALIZATION IN VISION.

We ablate the sparse mask selection and initialization strategies for SpaRFL pretraining on CIFAR-10. Since privacy and communication constraints prevent the sharing of client-side first- or second-order statistics, such as gradients or Fisher information, we investigate several heuristic-based alternatives suited for federated environments.

To clarify our analysis, we categorize the masks based on their Selection Strategy (Random vs. Magnitude-based) and their Initialization Value (Zero vs. Residual). We consider the following six configurations:

1. *Random-Zero*: Indices are selected uniformly at random and initialized to zero.
2. *Random-Residual*: Random indices are selected and initialized with their current residual values.
3. *Magnitude-Residual*: Top-k residual indices are selected and initialized with their current residual values.
4. *Hybrid-Zero*: A 50/50 split of random and magnitude-based indices, all zero-initialized.
5. *Hybrid-Residual*: A 50/50 split of random and magnitude-based indices, all residual-initialized.

As shown in Table 8, the Magnitude-Residual strategy achieves superior accuracy. We attribute this performance to its effective norm minimization, which reduces the approximation error more significantly than alternative selection methods.

Table 8: Ablation of sparse mask selection and initialization strategies. We report CIFAR-10 accuracy in the homogeneous setting at the 70% projected-state budget reduction. The Magnitude-Residual mask yields the highest performance.

Setting	Acc. ↑
Random-Zero	72.93
Random-Residual	73.21
Magnitude-Residual	74.55
Hybrid-Zero	72.64
Hybrid-Residual	73.19

G COMPARISON WITH PEFT METHODS.

Hu et al. (2022) introduced Low-Rank Adaptation (LoRA), which freezes the pretrained base model and trains low-rank adapters. QLoRA (Dettmers et al., 2023) additionally quantizes the frozen base, while RoSA (Nikdan et al., 2024) trains sparse and low-rank updates around a frozen dense base. We compare against these centralized PEFT parameterizations in a federated training protocol, as well as two explicitly federated LoRA methods: Fed-PLoRA (Zhang et al., 2026) and FLASC (Kuo et al., 2026). FLASC is the LoRA-based baseline in our matched comparison; it sparsifies communicated LoRA updates while retaining and locally training the complete LoRA module.

The frozen base remains an important memory cost. For a model with P parameters, an unquantized half-precision base alone requires $2P$ bytes. Under the persistent-state accounting in Section 4, this limits unquantized PEFT to a reduction below $1 - 2P/(14P) = 85.71\%$, even before adapter states are included. Thus, unquantized LoRA, RoSA, and Fed-PLoRA are not feasible at a 90% reduction under this accounting; their quantized variants remain applicable.

Table 9 clarifies the intended scope of SpaRFL. When a pretrained base fits in client memory, specialized PEFT approaches are strong: RoSA gives the best 70% results in this comparison, although its WikiText-2 advantage over SpaRFL is only 0.059 perplexity. At the tighter 90% reduction, Q-SpaRFL obtains the lowest WikiText-2 perplexity, improving on QLoRA and QRoSA. Among the explicitly federated LoRA baselines at 70%, SpaRFL outperforms Fed-PLoRA on MRPC, QNLI, and WikiText-2, and outperforms FLASC on every reported task. These results do not position

Table 9: Matched-budget comparison with PEFT baselines. GLUE results use a 70% projected-linear persistent-state budget reduction. For WikiText-2, we report quantized variants of SpaRFL and RoSA at both 70% and 90% reductions; QLoRA is quantized throughout. Fed-PLoRA and FLASC were evaluated at 70% only. Dashes denote settings not evaluated.

Method	SST-2 70% Acc.↑	MRPC 70% F1↑ / Acc.↑	QNLI 70% Acc.↑	WikiText-2 70% PPL↓	WikiText-2 90% PPL↓
Fed-PLoRA (Zhang et al., 2026)	94.27	89.63 / 85.54	91.49	12.59	–
FLASC (LoRA) (Kuo et al., 2026)	92.09	82.88 / 75.00	86.93	16.21	–
QLoRA (Dettmers et al., 2023)	94.19	91.42 / 87.99	78.68	12.471	13.574
RoSA / QRoSA (Nikdan et al., 2024)	**95.37**	**92.72 / 89.87**	**92.81**	**11.791**	13.198
SpaRFL / Q-SpaRFL	93.69	92.53 / 89.71	91.85	11.850	**12.640**

SpaRFL as a universal replacement for PEFT; they show that it remains competitive when PEFT is applicable while also covering stricter regimes.

We separately compare methods under federated pretraining from random initialization. FedLRS (Luo et al., 2023) is a genuine sparse–low-rank pretraining baseline: it jointly trains decomposed model components, but does not perform client-specific one-shot projection of a dense server model. In contrast, QLoRA, RoSA, and FLASC assume a pretrained base. Their entries in Table 10 are therefore stress diagnostics in which the frozen base is randomly initialized, not native pretraining configurations.

Table 10: Federated pretraining from scratch on C4 at a 70% projected-linear persistent-state budget reduction. We report validation perplexity (lower is better). FedLRS is a from-scratch baseline; QLoRA, RoSA-style, and FLASC freeze a randomly initialized base in these diagnostic runs. Dashes denote settings not evaluated.

Method	LLaMA-60M (PPL↓)	LLaMA-130M (PPL↓)
FedLRS (Luo et al., 2023)	58.03	40.91
QLoRA diagnostic (Dettmers et al., 2023)	1539.02	–
RoSA-style diagnostic (Nikdan et al., 2024)	1633.88	–
FLASC diagnostic (Kuo et al., 2026)	12,149.65	–
SpaRFL	**37.94**	**31.56**

SpaRFL improves over FedLRS by 20.09 perplexity points on LLaMA-60M and 9.35 points on LLaMA-130M. The adapter-based diagnostics remain above $1{,}500$ perplexity on LLaMA-60M, with FLASC exceeding $12{,}000$, because freezing a random base leaves most model capacity untrained. Consistently, Fed-PLoRA reaches only 26.32% accuracy before diverging in our heterogeneous CIFAR-10 from-scratch run.

H Robustness to Nonuniform Resource-Tier Distributions

The balanced heterogeneous allocation in Section 4 follows prior model-heterogeneous FL evaluations and enables a controlled comparison in which every resource tier is equally represented. Real deployments, however, need not contain equal numbers of clients at each tier. We therefore vary the tier frequencies while fixing the five CIFAR-10 projected-state budget reductions $(20\%, 40\%, 50\%, 70\%, 80\%)$, the ten-client population, data partition, and training protocol. In the allocation vectors below, entries follow this least-to-most-constrained tier order. All SpaRFL runs use the ablation-selected 95/5 low-rank/sparse budget split with Magnitude-Residual support. We report the three-seed mean and sample standard deviation of the best test accuracy over 200 rounds.

Table 11 shows that SpaRFL is insensitive to the uniform-tier assumption. Relative to the balanced allocation, its accuracy increases by 0.57 percentage points when half of the clients occupy the most constrained tier and by 0.43 points when clients concentrate around the middle tier. More importantly, SpaRFL outperforms FIARSE by 0.79 and 1.71 percentage points under the right-skewed

Table 11: CIFAR-10 accuracy under balanced and nonuniform resource-tier frequencies.

Method	Balanced $(2/2/2/2/2)$	Right-skewed $(1/1/1/2/5)$	Truncated normal $(1/2/4/2/1)$
SpaRFL	$\mathbf{71.04 \pm 0.25}$	$\mathbf{71.61 \pm 0.27}$	$\mathbf{71.47 \pm 0.96}$
FIARSE	69.06 ± 0.74	70.82 ± 0.42	69.76 ± 0.43
HeteroFL	64.26 ± 0.28	49.32 ± 0.65	67.60 ± 0.29

and truncated-normal distributions, respectively. Thus, the heterogeneous gains are not an artifact of assigning equal numbers of clients to every resource tier: SpaRFL remains robust when resource availability is substantially nonuniform.

I Computational Cost and Scaling Analysis

All experiments were executed on four NVIDIA GH200 GPUs; the total compute for reproduction is estimated at 4,100 GPU hours.

I.1 Projection Overhead

In this section, we analyze the server-side overhead incurred by the projection step during model compression. While the SVD projection is the primary computational bottleneck for SpaRFL, we mitigate this by employing a randomized Low-Rank SVD approach (Halko et al., 2010). We further parallelize the projection across clients (one thread per client group), so the per-round projection cost is dominated by a single representative projection rather than the sum across clients. We evaluate the overhead using CIFAR-10 ViT-Tiny and LLaMA-130M pretraining (see Section 4), and additionally benchmark a LLaMA-2-7B pretraining configuration. All measurements use NVIDIA GH200 120GB GPUs.

For ViT-Tiny on CIFAR-10, one full training round takes approximately 533.85 s (~ 8.9 min). For Llama 130M on C4, one round takes approximately 420 s. As shown in Tables 12 and 13, the projection step accounts for less than 0.04% of round time on the ViT setting and less than 0.2% on the Llama 130M setting for every method evaluated, including SpaRFL. This demonstrates that the projection step is not a prohibitive bottleneck in the overall pipeline.

For SpaRFL on Llama 130M, the reported time includes the calibration forward pass over 128 sequences of length 256, which on a GH200 takes only ~ 60 ms for a 130M-parameter model and is therefore not a bottleneck. FIARSE on Llama 130M is dominated instead by a global GPU→CPU magnitude transfer and sort over ~ 85 M projected weights, which is comparatively slow.

Table 12: Server-side projection overhead on CIFAR-10 with ViT-Tiny (sparsity $s = 0.85$, 10 clients, NVIDIA GH200 120GB). We report the per-client wall-clock time to construct one compressed client model and the corresponding overhead relative to one training round (533.85 s).

Method	Wall-clock time (s)	Overhead
HeteroFL	0.051	0.009%
FjORD	0.049	0.009%
FIARSE	0.087	0.016%
SpaRFL	0.192	0.036%

To test whether the server-side compression pipeline remains practical at a substantially larger model scale, we additionally benchmark LLaMA-2-7B on one NVIDIA GH200 at a 70% persistent-state budget reduction. Table 14 separates the activation-calibration pass from the sparse+low-rank projection and compares their combined time with one forward-and-backward training round.

Completing a converged federated pretraining run for a 7B-parameter model is beyond the compute budget of this study. Nevertheless, this benchmark provides systems-level evidence that SpaRFL scales to a 7B pretraining configuration: constructing the compressed client model takes 7.743 s,

Table 13: Server-side projection overhead on C4 with Llama 130M (sparsity $s = 0.97$, 5 clients, NVIDIA GH200 120GB). We report the per-client wall-clock time to construct one compressed client model and the corresponding overhead relative to one training round (~ 420 s). The SpaRFL time includes a Wanda-style calibration forward pass over 128 sequences (length 256) used to weight the residual top-k in activation space.

Method	Wall-clock time (s)	Overhead
FIARSE	0.711	0.169%
SpaRFL	0.203	0.048%

Table 14: Computational-cost scaling for LLaMA-2-7B on one NVIDIA GH200 at a 70% projected-state budget reduction. Calibration uses 128 synthetic sequences of length 256. The reported overhead is total server compression divided by one forward-and-backward training round.

Step	Time (s)
Calibration (128 sequences, length 256)	6.438
Sparse+low-rank projection (224 linear layers)	1.306
Total server compression	7.743
Forward+backward compute (one round)	10,439.472
Server-compression overhead	0.0742%

only 0.0742% of the measured per-round forward-and-backward time. The expensive projection operations remain server-side, and their cost is negligible relative to training even at this scale. We therefore interpret the result as evidence of projection and execution scalability, rather than as a 7B model-quality result.

I.2 Sparse+Low-Rank Matrix Multiplication Runtime

In this section, we benchmark sparse+low-rank (S+L) matrix multiplication against a dense GEMM baseline executed via `torch.matmul` (dispatched to cuBLAS). For a target total sparsity, we allocate 3% of the non-zero elements to the sparse component and 97% to the low-rank component. We sweep the Cartesian product of weight shapes $(d_{\text{in}} \times d_{\text{out}})$, batch sizes $M \in \{1, 4, 16, 64, 128\}$, and total sparsity levels $s_{\text{tot}} \in \{70, 90\}$. The weight shapes include $(8192, 8192)$, $(16384, 16384)$, and $(32768, 32768)$, as well as asymmetric projections such as $(16384, 53248)$ and $(8192, 32768)$ to represent typical LLM FFN blocks. Batch sizes M are chosen to span memory-bound inference and compute-bound prefill regimes.

Each configuration is evaluated over 15 warm-up and 50 timed iterations. We report the mean latency in milliseconds and define speedup $= t_{\text{dense}}/t_{\text{S+L}}$. Across all configurations, the coefficient of variation remained below 1%. Experiments are conducted on an NVIDIA GH200 120GB (Grace Hopper Superchip) using Python 3.12.3, PyTorch 2.6.0, Triton 3.0.0, and CUDA 12.6. All activations, weights, and sparse values are stored in `bfloat16` precision.

Figure 6 illustrates the experimental setup and speedup over cuBLAS, while Tables 15 and 16 provide full measurements for 70% and 90% sparsity, respectively. For 90% sparsity, the kernel achieves up to a 2.6× speedup over cuBLAS on large-scale matrices typical of models like Llama3-405B. At 70% sparsity, the peak speedup reaches 2.12×. We observe that speedups are more pronounced at smaller batch sizes; this is expected, as Tensor Core utilization in the dense baseline improves significantly with larger batches. Furthermore, the CSR format outperforms ELLPACK in most evaluated configurations.

Although the primary focus of this work is not hardware optimization, these results demonstrate substantial potential for efficiency gains. We leave the development of a fully fused kernel for future work. Such an implementation should prioritize end-to-end performance, particularly for regimes where the sparse component contains the majority of the non-zero parameters.

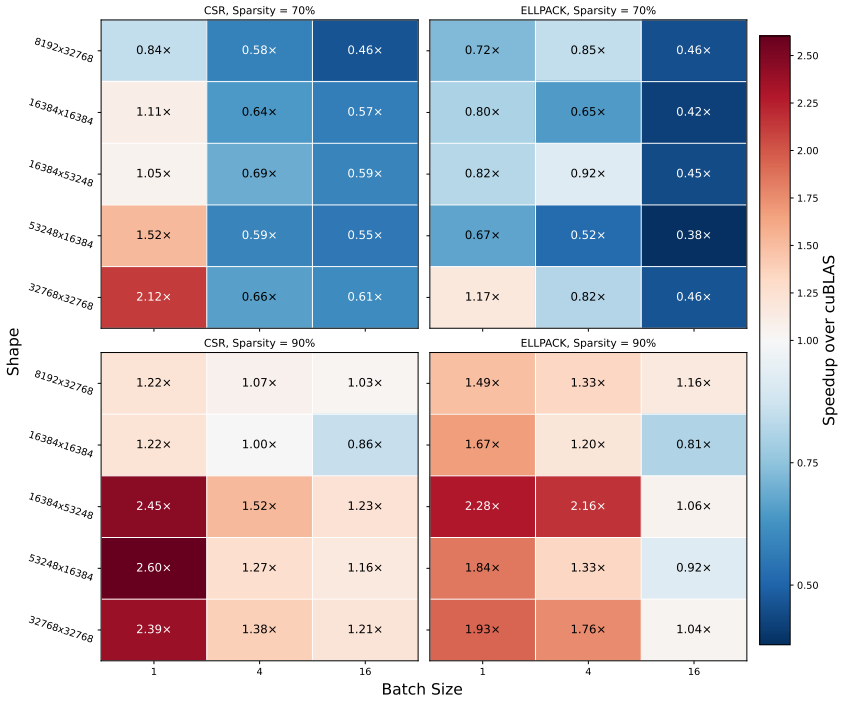

Figure 6: Speedup of CSR and ELLPACK kernels relative to cuBLAS across varying matrix shapes, batch sizes, and sparsity levels (70% and 90%). All benchmarks are conducted using bfloat16 precision.

Table 15: Benchmark results for S+L matrix multiplication at 70% total sparsity. Latencies are reported in milliseconds; speedups are relative to the dense cuBLAS baseline. (NVIDIA GH200 120GB, `bfloat16`).

Shape	B	Rank	Dense	CSR		ELL	
			(ms)	(ms)	speedup	(ms)	speedup
16384×16384	1	2384	0.166	0.150	1.11×	0.206	0.80×
	4	2384	0.160	0.249	0.64×	0.247	0.65×
	16	2384	0.166	0.291	0.57×	0.395	0.42×
32768×32768	1	4768	0.581	0.274	2.12×	0.498	1.17×
	4	4768	0.582	0.883	0.66×	0.712	0.82×
	16	4768	0.583	0.954	0.61×	1.257	0.46×
16384×53248	1	3646	0.493	0.467	1.06×	0.600	0.82×
	4	3646	0.500	0.724	0.69×	0.543	0.92×
	16	3646	0.507	0.859	0.59×	1.136	0.45×
53248×16384	1	3646	0.486	0.320	1.52×	0.724	0.67×
	4	3646	0.487	0.829	0.59×	0.931	0.52×
	16	3646	0.485	0.875	0.55×	1.284	0.38×
8192×32768	1	1907	0.165	0.195	0.85×	0.227	0.73×
	4	1907	0.166	0.286	0.58×	0.196	0.85×
	16	1907	0.166	0.358	0.46×	0.365	0.46×

Table 16: Benchmark results for S+L matrix multiplication at 90% total sparsity. Latencies are reported in milliseconds; speedups are relative to the dense cuBLAS baseline. (NVIDIA GH200 120GB, `bfloat16`).

Shape	B	Rank	Dense	CSR		ELL	
			(ms)	(ms)	speedup	(ms)	speedup
16384×16384	1	795	0.163	0.134	1.22×	0.098	1.67×
	4	795	0.165	0.165	1.00×	0.138	1.20×
	16	795	0.165	0.192	0.86×	0.203	0.81×
32768×32768	1	1589	0.583	0.244	2.39×	0.302	1.93×
	4	1589	0.584	0.422	1.38×	0.332	1.76×
	16	1589	0.583	0.484	1.21×	0.561	1.04×
16384×53248	1	1215	0.495	0.202	2.45×	0.217	2.28×
	4	1215	0.501	0.329	1.52×	0.232	2.16×
	16	1215	0.509	0.415	1.23×	0.480	1.06×
53248×16384	1	1215	0.488	0.187	2.60×	0.265	1.84×
	4	1215	0.488	0.383	1.27×	0.366	1.33×
	16	1215	0.485	0.418	1.16×	0.527	0.92×
8192×32768	1	636	0.164	0.135	1.22×	0.111	1.49×
	4	636	0.165	0.154	1.07×	0.124	1.33×
	16	636	0.169	0.163	1.03×	0.145	1.16×

J Memory Analysis

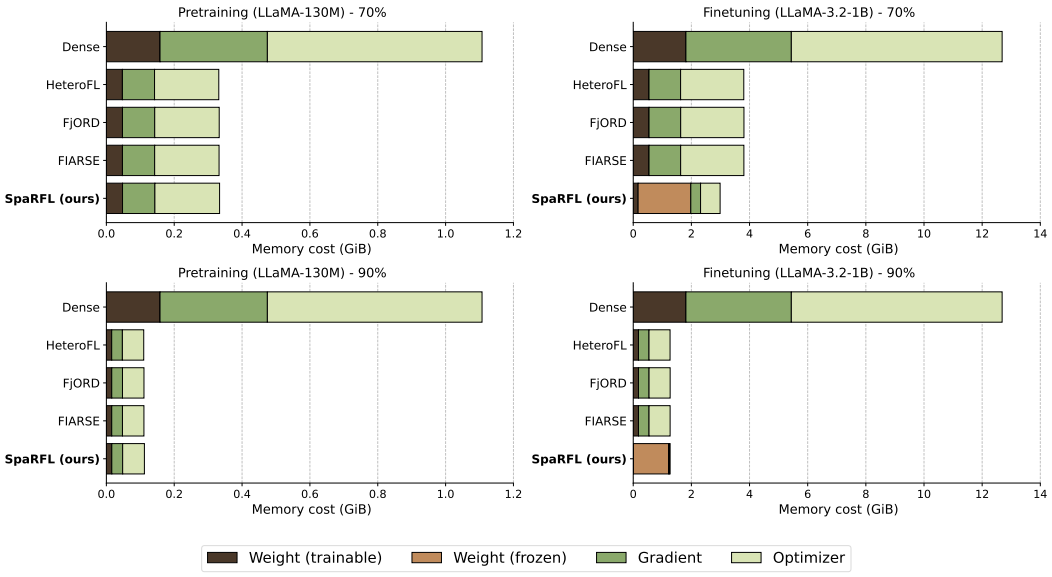

Figure 7: Projected-linear persistent-state breakdown under the 70% and 90% budget reductions. The bars include the trainable and frozen weights, gradients, and optimizer states associated with the linear layers modified by each heterogeneous method. Components shared across methods, including embeddings, normalization layers, and output heads, are excluded from tier assignment and analyzed separately in Figure 8. All quantities are reported in GiB.

This section separates the controlled projected-linear persistent-state budget from broader analytical memory composition. Figure 7 shows the 70% and 90% projected-state budget reductions for LLaMA-130M pretraining and LLaMA-3.2-1B fine-tuning (Section 4). A 70% reduction allocates

30% of the dense projected-linear persistent-state budget to the client parameterization. The bars are expressed in GiB (2^{30} bytes); they are analytical state accounting, not measured CUDA peaks.

A noticeable distinction arises in the fine-tuning regime. Compared to baseline methods, SpaRFL allocates a larger proportion of the budget to maintaining frozen weights, which proves effective as demonstrated in Section 4. At the 90% projected-state budget reduction during fine-tuning, only a small fraction of the controlled budget is spent on trainable parameters.

Under our fixed-interface implementations, HeteroFL, FjORD, FIARSE, and SpaRFL preserve the original Transformer input and output dimensions, and the width-based baselines pad selected outputs back to the dense interface. Consequently, the checkpoint-boundary backbone tensors stored by activation checkpointing (Korthikanti et al., 2022; Chen et al., 2016) are identical across these implementations. This does not imply identical end-to-end activation peaks: recomputation temporaries and kernel workspaces may differ, and a specialized compact implementation of a width-reduced baseline could reduce activation storage further.

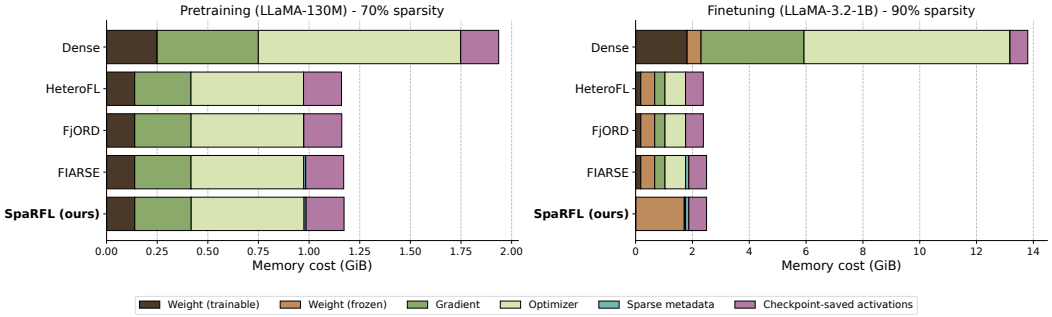

Figure 8: Expanded analytical memory composition for representative LLaMA-130M pretraining and LLaMA-3.2-1B fine-tuning configurations. In addition to the projected linear layers used for budget assignment, the bars include unprojected parameter groups, sparse-format metadata, and checkpoint-saved tensors. This accounting excludes transient runtime allocations such as logits, loss intermediates, recomputation buffers, kernel workspaces, and allocator effects, and therefore should not be interpreted as measured CUDA peak memory. All quantities are reported in GiB.

Index-storage overhead of the sparse component. Storing the indices of non-zero entries is a long-standing line of research, and different sparse formats are best suited to different sparsity regimes (Hoefler et al., 2021). For an $n_r \times n_c$ matrix with m non-zero entries we consider the two formats most relevant to our setting: a *bitmap* (BM), which stores a single presence bit per matrix entry and therefore costs $o_{\text{BM}} = n_r n_c$ bits independently of the number of non-zeros; and *compressed sparse row* (CSR), which stores a column-index of $\lceil \log_2 n_c \rceil$ bits for every non-zero *plus* a row-pointer array of length n_r with $\lceil \log_2 m \rceil$ bits per entry, giving $o_{\text{CSR}} = m \lceil \log_2 n_c \rceil + n_r \lceil \log_2 m \rceil$ bits. Intuitively, CSR pays a (small) constant number of extra bits per non-zero on top of a fixed-size row directory, making it attractive precisely when m is small relative to the matrix area.

Specifically, at the 70% projected-state budget reduction, where the sparse component receives 5% of the controlled budget for ViT on CIFAR-10, the total overhead across all layers using CSR is 0.111 MiB, representing a 0.52% overhead. Similarly, pretraining LLaMA at the 70% reduction, with 97% allocated to the sparse component using a bitmap, requires 10.125 MiB, which constitutes a 2.98% overhead. Client tiers are assigned using the projected-state accounting in Section 4, while index storage is reported separately for completeness. FIARSE incurs slightly higher costs because its matrices are fully sparse and require more indexing data.

K Communication Cost Analysis

We evaluate the per-round communication cost required for each client to transmit its compressed weights in `bfloat16`. For sparse methods (SpaRFL and FIARSE), we include the indexing overhead by selecting the most efficient format from Hoefler et al. (2021) based on the operating density (see Appendix J). Specifically, we use CSR for SpaRFL on ViT and a bitmap for LLaMA-130M, while

FIARSE utilizes a bitmap in both cases. HeteroFL and FjORD transmit contiguous dense sub-matrices and therefore incur no indexing overhead.

Table 17 summarizes these costs under a 70% projected-state budget reduction. SpaRFL achieves lower communication costs than FIARSE on ViT, where CSR provides a more compact representation. On LLaMA-130M, SpaRFL matches the communication cost of FIARSE because both methods operate within the bitmap-optimal density regime. While more efficient than FIARSE in some cases, SpaRFL remains slightly more expensive than HeteroFL and FjORD due to the inherent metadata requirements of sparse representations.

Table 17: Per-round, one-directional communication cost (MB) under a 70% projected-state budget reduction.

Method	ViT-Tiny / CIFAR-10 (MB)	LLaMA-130M / C4 (MB)
Dense	10.125	162.000
FIARSE	3.670	58.725
FjORD	3.038	48.600
HeteroFL	3.038	48.600
SpaRFL	3.148	58.725

L Client Scalability

We evaluate the scalability of SpaRFL by varying the number of clients from 10 to 100 using a heterogeneous CIFAR-10 partition. Following the previous experimental setup (see Section 4), we train a ViT-Tiny network from scratch for 500 rounds with 5 local epochs per round. In each round, we randomly sample 20% of the clients to participate in training. We employ an SGD optimizer with a batch size of 64, an initial learning rate of 0.1, and a cosine learning rate scheduler. Consistent with our 10-client experiments, 95% of the parameter budget is allocated to the low-rank component. As shown in Table 18, SpaRFL consistently outperforms the baselines across all client scales.

Table 18: Client scalability on CIFAR-10 in heterogeneous setting. We report accuracy as the number of clients scales from 10 to 100. SpaRFL outperforms all federated baselines and maintains performance close to the Dense upper bound

Dataset	CIFAR-10 (Acc. ↑)	
	10 clients	100 clients
Dense	75.66 ± 0.24	65.52 ± 0.41
HeteroFL	64.26 ± 0.28	37.45 ± 0.47
FjORD	64.12 ± 0.41	39.07 ± 2.48
FIARSE	69.06 ± 0.74	58.84 ± 0.98
SpaRFL	$\mathbf{70.30 \pm 0.07}$	$\mathbf{60.04 \pm 0.88}$

M Calibration Data for Activation-Aware Projection

The activation-aware NLP projection relies on a small calibration set $\mathcal{D}_{\text{cal}}$ used to estimate input-activation statistics at the server. For every NLP task, the server uses a fixed public subset of 128 C4 sequences, truncated to the context length of the corresponding experiment: 128 tokens for GLUE, 1024 for WikiText-2, and 256 for C4 pretraining. The calibration data are fixed across clients and rounds and never contain client examples. For C4 pretraining, the calibration corpus matches the training domain; for the downstream fine-tuning tasks, it serves as generic public text. Our privacy argument requires only that the calibration set be public and client-independent, not that it be distributionally unrelated to the training task.

The server computes the projection from this public data and the current global weights, without client gradients, client activations, or client samples. The calibration step therefore does not transmit information about client data or weaken any privacy property of the underlying federated optimizer.

N ACTIVATION-SPACE VS. WEIGHT-SPACE PROJECTION

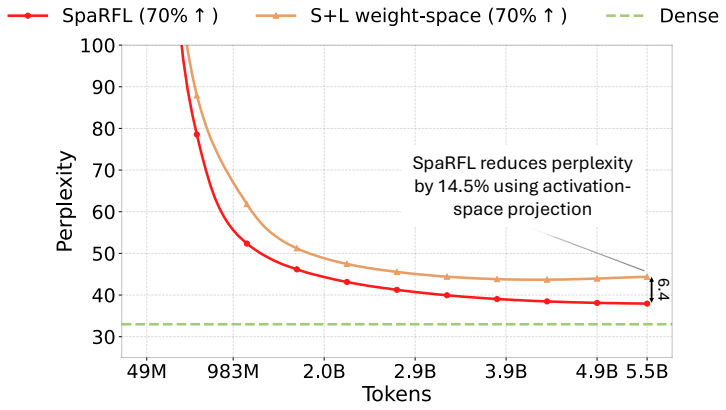

Figure 9: Activation-Space vs. Weight-Space Projection. Federated pretraining of LLaMA-60M on C4 under a 70% projected-state budget reduction. Using activation-space information improves evaluation perplexity by 14.5% compared to the weight-space sparse+low-rank projection.

In Figure 9, we compare SpaRFL against a weight-space sparse+low-rank variant in the same setting as Section 4, namely federated pretraining of LLaMA-60M from scratch on C4. The variant labeled *S+L weight-space* constructs the sparse+low-rank projection without using the calibration subset and optimizes the decomposition only in weight space, rather than in activation space. Figure 9 shows that incorporating activation-space information leads to a clear improvement. In particular, using activation-space projection within SpaRFL reduces C4 perplexity by 14.5% relative to the weight-space variant.