CoeF-SFL: Preserving Collaborative Server-Client Learning with Enhanced Communication Efficiency

Anonymous authors
Paper under double-blind review

Abstract

Split Federated Learning (SFL) enables resource-constrained clients to participate in collaborative training, but vanilla SFL exchanges smashed data and gradients at every batch, which incurs significant communication overhead. Recent methods reduce this overhead with an auxiliary network at the client-side cut layer. However, we identify that this approach makes the client optimize a local objective that differs from the end-to-end objective, which fundamentally limits the collaborative training between the client and the server. We propose Compensated Feedback based SFL (CoeF-SFL), a communication-efficient framework that retains the end-to-end objective without any auxiliary network. In CoeF-SFL, the client and the server exchange the smashed data and the gradients once per round and reuse them during local training. Since this reuse makes the gradients stale on the client side, we compensate them with a curvature-based correction in the activation space and develop two variants. CoeF-D approximates the Hessian with a diagonal gradient outer product, while CoeF-J exploits the tractable Jacobian-based Hessian of a surrogate loss that upper-bounds the true loss. We provide the theoretical background of each method, characterizing its compensation. Across vision and language tasks, model capacities, cut layers, and data distributions, CoeF-SFL significantly outperforms auxiliary-network-based methods under the same communication frequency, and the improvement is most substantial on vision tasks. Code is available at https://anonymous.4open.science/r/CoeF-SFL-2686/README.md

1 Introduction

Split Federated Learning (SFL) (Thapa et al., 2022) enables resource-constrained clients to collaboratively train a model without sharing raw data. By splitting the model at a cut layer as in split learning (Gupta & Raskar, 2018; Vepakomma et al., 2018) and training the client-side models in parallel as in federated learning (McMahan et al., 2017), SFL avoids both the heavy client-side computation of federated learning and the sequential training latency of split learning. In SFL, the server updates the server-side model with the smashed data from the clients and returns the corresponding gradients, so that the split models collaboratively optimize the end-to-end (E2E) objective through server-side feedback.

1.1 Auxiliary-network-based SFL

Although SFL addresses computational burden and latency, significant communication overhead remains due to the per-batch exchange between the client and the server. To address this overhead, *AccSFL* (Han et al., 2022) has introduced an auxiliary network into the SFL framework. Specifically, by attaching the network at the cut layer, the client-side models can perform parameter updates without any feedback from the server side. This mechanism eliminates the need to wait for the gradient corresponding to the transmitted smashed data, thereby reducing latency and communication burden. In fact, since AccSFL was proposed, many studies (e.g., Oh et al., 2022; Mu & Shen, 2023; Nair et al., 2025; Shin et al., 2026) have assumed a system model in which an auxiliary network is attached to the client side to achieve high communication efficiency.

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.

Table 1: Preliminary results of vanilla SFL (Thapa et al., 2022), AccSFL (Han et al., 2022), two reference methods, reported as test accuracy (%) and EM/F1 (%) on vision and language tasks with ViT-Tiny and DistilRoBERTa, respectively. Gradient reuse (w/o comp.) is the naive baseline, and Gradient reuse (w/ perfect comp.) is the oracle. For all methods except vanilla SFL, each round, i.e., the interval between two consecutive server feedbacks, corresponds to one local epoch. The other settings, including the SFL setting, local training, and hyperparameter tuning, are described in Section 5.

Method	**CIFAR-100**	**Food-101**	**Tiny-ImageNet**
Vanilla SFL	85.0 ± 0.3	83.8 ± 0.1	78.6 ± 0.2
AccSFL	65.6 ± 1.2	67.5 ± 0.1	47.4 ± 0.2
Gradient reuse (w/o comp.)	78.4 ± 0.4	75.3 ± 2.1	72.8 ± 0.2
Gradient reuse (w/ perfect comp.)	84.5 ± 0.3	83.3 ± 0.0	78.3 ± 0.3

Method	**20 Newsgroups**	**MNLI-m**	**MNLI-mm**	**EM**	**F1**
Vanilla SFL	69.8 ± 0.2	83.3 ± 0.3	83.7 ± 0.3	71.3 ± 0.2	74.5 ± 0.2
AccSFL	66.6 ± 0.3	77.7 ± 0.3	78.5 ± 0.3	61.6 ± 1.2	64.7 ± 1.1
Gradient reuse (w/o comp.)	69.3 ± 0.0	37.1 ± 6.2	37.7 ± 7.1	49.9 ± 0.2	50.0 ± 0.2
Gradient reuse (w/ perfect comp.)	69.3 ± 0.2	83.0 ± 0.1	83.1 ± 0.2	70.2 ± 2.1	73.1 ± 1.5

However, the auxiliary-network-based approach sacrifices a key advantage of the SFL framework, which is that the split models collaboratively optimize the E2E objective through server-side feedback. Although (Nair et al., 2025) pointed out this limitation, noting that such methods "lack server feedback and potentially suffer poor accuracy," why it constitutes a fundamental problem can be explained by the analysis of decoupled training in (Wang et al., 2021), which corresponds to auxiliary-network-based SFL with a single client. Their information-theoretic and empirical analysis shows that such decoupled training with a local objective is suboptimal. Specifically, the local objective drives the features at the cut layer to be over-compressed, which discards task-relevant information that the subsequent layers would otherwise exploit. Furthermore, once this information is discarded, the subsequent layers cannot recover it even with additional capacity. Our preliminary results in Table 1 support this explanation in the context of SFL, showing the corresponding performance degradation of AccSFL compared with vanilla SFL on various vision and language tasks.

1.2 Motivations and Contributions

Motivated by this observation, we argue that the client-side model should be trained with the gradient computed by the server, as in vanilla SFL, while preserving the communication efficiency. To this end, the clients in the proposed method reuse the gradients received at the beginning of the round over multiple local updates, just as the server repeatedly updates its model with the received smashed data in auxiliary-network-based SFL. Although this retains the E2E objective with improved communication efficiency, it causes a mismatch between the activation and the gradient, since the activation of each input changes as the client-side model is updated while the received gradient remains fixed. We refer to this mismatch as the *staleness* of the activation-space gradient, and formulate it after the i-th local update for input x as

$$\varepsilon_i(x) := \underbrace{\nabla_s \mathcal{L}\big(s_i; \theta^s\big)}_{\text{(a) Fresh gradient}} - \underbrace{\nabla_s \mathcal{L}\big(s_0; \theta^s\big)}_{\text{(b) Stale gradient}}, \qquad s_i = f_c\big(x; \theta_i^c\big), \tag{1}$$

where s_i is the activation of x after the i-th local update, and θ^s is the server-side model at the beginning of the round. The fresh gradient matches the current client-side model θ_i^c but is unavailable to the client, which does not hold the server-side model, whereas the stale gradient is computed by the server at the initial activation s_0. As shown by Gradient reuse (w/o comp.) in Table 1, which naively reuses the received gradients, this staleness severely degrades the performance of SFL.

To examine the potential of compensation, Gradient reuse (w/ perfect comp.) in Table 1 denotes an oracle that completely removes the staleness by using the fresh gradient computed with the server-side model at the beginning of the round. It significantly outperforms both AccSFL and Gradient reuse (w/o comp.) and achieves performance close to that of vanilla SFL, which suggests that SFL

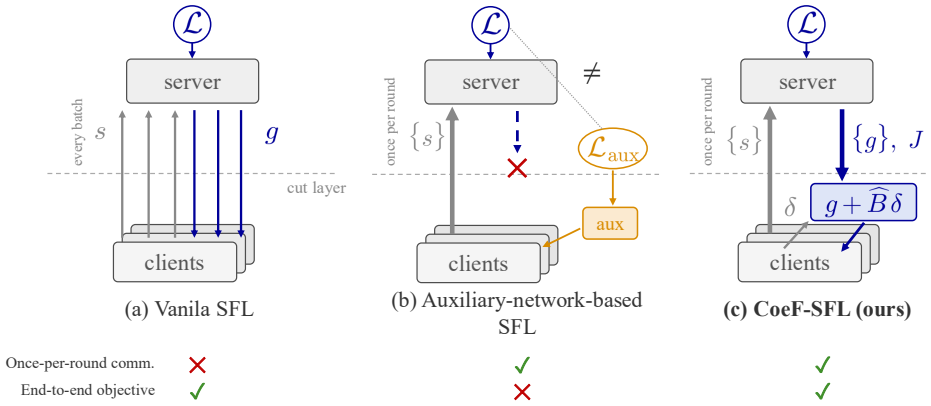

Figure 1: Comparison of SFL frameworks. Vanilla SFL optimizes the end-to-end objective $\mathcal{L}$ through server-side feedback at every batch, which incurs heavy communication. Auxiliary-network-based SFL communicates once per round, but the client optimizes an auxiliary objective $\mathcal{L}_{\text{aux}}$ that differs from $\mathcal{L}$. CoeF-SFL communicates once per round and preserves the end-to-end objective, compensating the received gradients according to the displacement δ of activation.

can retain the communication efficiency while outperforming auxiliary-network-based SFL once the stale gradients are properly compensated. Building on this observation, we propose Compensated Feedback based SFL (CoeF-SFL), which compensates the stale gradients with a curvature-based correction in the activation space and thus achieves both collaborative server-client learning and communication efficiency, as illustrated in Figure 1. We develop two variants, CoeF-D and CoeF-J. CoeF-D approximates the Hessian with a diagonal gradient outer product and requires no additional server-side computation, while CoeF-J exploits the tractable Jacobian-based Hessian of a surrogate loss function that upper-bounds the true loss, which requires only first-order computation on the server but tends to be more robust under non-IID settings and deep cut layers, as shown in Section 5. The design space that we explored for the compensation is discussed in Section 3. Beyond the theoretical and empirical validation of CoeF-SFL, the contributions of this paper are summarized as follows.

Identifying the loss of the end-to-end objective. We identify that auxiliary-network-based SFL, which has been widely adopted in recent SFL studies, loses the E2E objective that the client-side and server-side models collaboratively optimize. To retain this objective with high communication efficiency, we propose a new SFL framework in which the clients reuse the gradients received from the server over multiple local updates.

Analyzing and compensating gradient staleness. To the best of our knowledge, this is the first work that analyzes the staleness of the activation-space gradient, which arises when the client-side model is updated with the gradients received at the beginning of the round. We analyze this staleness both empirically and theoretically, and develop two compensation methods, CoeF-D and CoeF-J, to overcome it.

2 Related Work

Auxiliary-network-based SFL. Several variants have been built on AccSFL (Han et al., 2022), which maintains a separate server-side model for each client. LocFedMix (Oh et al., 2022) additionally introduces data mixup, and CSE-FSL (Mu & Shen, 2023) adopts a single server-side model to reduce the storage requirement of the server. FSL-SAGE (Nair et al., 2025) addresses the lack of server feedback by periodically aligning the auxiliary models with the server-side model on the server, and FedSplitX (Shin et al., 2026) extends the auxiliary-network-based mechanism to environments with system heterogeneity. Since we consider the setting of SplitFed V1 (Thapa et al., 2022) without system-level heterogeneity across clients, and data mixup is orthogonal to the update rule, FSL-SAGE is the only variant whose update rule substantially differs from that of AccSFL in our

setting. Therefore, we adopt AccSFL and FSL-SAGE as the baselines of auxiliary-network-based SFL.

Straggler-resilient SFL. Unlike auxiliary-network-based SFL, GAS (Yang & Liu, 2024) and MU-SplitFed (Liang et al., 2025) consider that clients upload their activations at different times owing to system heterogeneity, and aim to prevent training from being delayed by stragglers. In GAS, each client receives the gradient at every batch as in vanilla SFL, but the gradient is computed with the server-side model at the time of the upload and returned immediately. In MU-SplitFed, the clients perform zeroth-order optimization without backpropagation, and the server performs multiple updates for each client update while returning a scalar feedback instead of the gradient. Although neither method reduces the upload frequency of the clients, which is the focus of this paper, we include them for a more informative comparison by assuming, only for these two methods, that the clients upload their activations sequentially.

Curvature surrogates and upper-bound models. Curvature information in centralized training can be accessed through the generalized Gauss–Newton (GGN) matrix (Schraudolph, 2002; Martens, 2020), stochastic diagonal estimators (Bekas et al., 2007), and low-rank representations (Dangel et al., 2022). The gradient outer product offers another surrogate, which coincides with the empirical Fisher and generally differs from the Hessian (Kunstner et al., 2019). Majorization–minimization (Hunter & Lange, 2004; Böhning & Lindsay, 1988) instead minimizes an upper-bound model of the objective, for which the global curvature bound of the softmax cross-entropy loss (Böhning, 1992) is a well-known ingredient. These methods estimate the curvature with respect to the model parameters, whereas the compensation in our setting requires the curvature with respect to the cut-layer activation, which the client cannot compute without the server-side model.

3 Compensated Feedback based SFL (CoEF-SFL)

Problem setup. We consider a SplitFed V1 setting with K clients and T rounds. For simplicity, we omit the indices of clients and rounds, and denote the client-side and server-side models at the beginning of each round by θ^c and θ^s, respectively. At the beginning of the round, the client generates the smashed data $\boldsymbol{s} = \{f_c(x;\theta^c)\}$ from its local data $\mathcal{D} = \{(x,y)\}$ through the client-side model. The server receives $\{(s,y)\}$ and computes the gradients $\boldsymbol{g} = \{\nabla_s \mathcal{L}(s;\theta^s)\}$ with the server-side model. After the client receives $\boldsymbol{g}$, the client and the server perform I local updates with gradient reuse and smashed-data reuse, respectively.

Starting with $\theta_0^c = \theta^c$, the client-side model after the i-th local update is denoted by θ_i^c. Although the data in the mini-batch differ across local updates, we omit the corresponding index for simplicity. Accordingly, $\boldsymbol{s}_i = \{f_c(x;\theta_i^c)\}$ denotes the smashed data recomputed with θ_i^c, and $\boldsymbol{g}_{0,i}$ denotes the received gradients $\boldsymbol{g}$, both for the data in the mini-batch of the i-th local update. Then, with Gradient reuse (w/o comp.), the client-side model is updated as

$$\theta_i^c = \theta_{i-1}^c - \eta J_{i-1}^\top \boldsymbol{g}_{0,i-1}, \qquad J_i := \frac{\partial \boldsymbol{s}_i}{\partial \theta_i^c}, \tag{2}$$

whereas with Gradient reuse (w/ perfect comp.), it is updated as

$$\theta_i^c = \theta_{i-1}^c - \eta J_{i-1}^\top \nabla_s \mathcal{L}(\boldsymbol{s}_{i-1};\theta^s). \tag{3}$$

Furthermore, for AccSFL without knowledge distillation,

$$\theta_i^c = \theta_{i-1}^c - \eta \nabla_{\theta^c} \mathcal{L}_{\text{aux}}(\boldsymbol{s}_{i-1};\theta_{i-1}^c), \tag{4}$$

where $\mathcal{L}_{\text{aux}}$ denotes the local loss computed through the auxiliary network. In all cases, starting with $\theta_0^s = \theta^s$, the server-side model is updated as

$$\theta_i^s = \theta_{i-1}^s - \eta \nabla_{\theta^s} \mathcal{L}(\boldsymbol{s}_{0,i-1};\theta_{i-1}^s), \tag{5}$$

where $\boldsymbol{s}_{0,i}$ denotes the smashed data $\boldsymbol{s}$ received from the client for the data in the mini-batch of the i-th local update. While the server-side update in equation 5 has no staleness since the received smashed data are fixed, we aim to compensate the staleness of the client-side update in equation 2.

Toward compensating gradient staleness. To resolve this staleness, we explored various approaches. The first approach is to train a regression model on the server, which maps the received smashed data to the computed gradients, and to broadcast it to the clients. Since this model learns

the local landscape of the server-side loss at every round, it can provide the gradient even when the activation is displaced during local training. However, our exploration shows that the target of the regression, i.e., the gradient, has a very high dimension, which makes the regression highly inefficient to train. Consequently, the size of the regression network and the per-round training on the server become impractically burdensome. To mitigate this, we further considered techniques such as MAML (Finn et al., 2017; 2019) and the neural tangent kernel (NTK) (Jacot et al., 2020), but they did not sufficiently alleviate the burden. Therefore, we adopt compensation methods based on the first-order expansion of the gradient, i.e., the curvature, in the activation space, which can be regarded as the most fundamental approach.

With the displacement $\boldsymbol{\delta}_i := \boldsymbol{s}_i - \boldsymbol{s}_{0,i}$, the fundamental theorem of calculus (Nocedal & Wright, 2006) gives the fresh gradient exactly,

$$\boldsymbol{g}_i = \boldsymbol{g}_{0,i} + \bar{H}_i \boldsymbol{\delta}_i, \qquad \bar{H}_i := \int_0^1 \nabla_s^2 \mathcal{L}\big(\boldsymbol{s}_{0,i} + \tau \boldsymbol{\delta}_i; \theta^s\big)\, d\tau. \tag{6}$$

However, the client cannot compute $\bar{H}_i$, CoeF-SFL replaces the curvature with matrix $\widehat{B}_i$ built from what the server transmitted at the round-initial exchange, and the client uses

$$\widetilde{\boldsymbol{g}}_i := \boldsymbol{g}_{0,i} + \widehat{B}_i \boldsymbol{\delta}_i = \boldsymbol{g}_i + \big(\widehat{B}_i - \bar{H}_i\big)\boldsymbol{\delta}_i, \tag{7}$$

in place of $\boldsymbol{g}_{0,i}$ in equation 2. Therefore, the key problem is the reducing the curvature mismatch or the displacement.

CoeF-D: diagonal gradient outer product. Since CoeF-D computes $\widehat{B}_{\mathrm{D}}$ only from the received gradient, it does not require any further feedback from the server. As discussed in Section 2, CoeF-D carries the compensation principle of DC-ASGD (Zheng et al., 2017) from the parameter space to the activation space. Specifically, CoeF-D scales the diagonal of the gradient outer product by a compensation factor λ as

$$\widetilde{\boldsymbol{g}}_{\mathrm{D},i} = \boldsymbol{g}_{0,i} + \widehat{B}_{\mathrm{D},i} \boldsymbol{\delta}_i, \qquad \widehat{B}_{\mathrm{D},i} := \lambda\, \mathrm{diag}\left(\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i}\right), \tag{8}$$

where $\widetilde{\boldsymbol{g}}_{\mathrm{D},i}$ is the compensated gradient of CoeF-D, and $\lambda > 0$ is the compensation factor whose choice is analyzed in Section 4. The outer product is a reasonable surrogate of the Hessian, since it shares the structure of the curvature that it replaces. To show this, let $J_{s,i} := \partial f_s\big(\boldsymbol{s}_{0,i}^*; \theta^s\big)/\partial s$ denote the Jacobian of the server output ϕ computed at $\boldsymbol{s}_{0,i}^*$, and let $r_i := \nabla_\phi \mathcal{L}\big(\boldsymbol{s}_{0,i}^*; \theta^s\big)$ and $\Lambda_i := \nabla_\phi^2 \mathcal{L}\big(\boldsymbol{s}_{0,i}^*; \theta^s\big)$ denote the gradient and the Hessian of the loss with respect to ϕ, respectively. Here, $\boldsymbol{s}_{0,i}^*$ denotes the subset of $\boldsymbol{s}_0$ whose data belong to the mini-batch of the i-th local update at the client, whereas $\boldsymbol{s}_{0,i}$ corresponds to the mini-batch of the i-th local update at the server. By the chain rule, we have

$$\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i} = J_{s,i}^\top \big(r_i r_i^\top\big) J_{s,i}, \tag{9}$$

which has the same form as the generalized Gauss–Newton (GGN) matrix $G_i := J_{s,i}^\top \Lambda_i J_{s,i}$ and agrees with it in expectation up to the mean-reduction weight, as discussed in Proposition 1. The correction is computed elementwise on the client at a cost of $O(d)$ per sample, where d denotes the dimension of the activation space, without any additional server-side computation.

CoeF-J: compensation with an upper bound on the loss. The error of the compensated gradient in equation 7, $\widetilde{\boldsymbol{g}}_i - \boldsymbol{g}_i = (\widehat{B}_i - \bar{H}_i)\boldsymbol{\delta}_i$, is the product of the curvature mismatch and the displacement. CoeF-D reduces this error by approximating the curvature, which is accurate only while the displacement is small. CoeF-J takes a different route. Instead of approximating the curvature, it constructs a surrogate that upper-bounds the loss, and uses the gradient of the surrogate as the compensated gradient, which does not require an accurate curvature.

The surrogate is built from the structure of the curvature. Since the loss depends on the smashed data only through the server output, its Hessian with respect to the smashed data consists of the GGN term $J_{s,i}^\top \Lambda_i J_{s,i}$ and a term that involves the second derivative of the server-side model. The client cannot compute Λ_i, but for standard losses, the curvature with respect to the server output is bounded as $\Lambda_i \preceq MI$, where M is fixed by the loss, e.g., $M = 1/2$ for the softmax cross-entropy loss (Böhning, 1992). Replacing Λ_i with this bound yields

$$\mathcal{L}(\boldsymbol{s}_{0,i} + \boldsymbol{\delta}_i; \theta^s) \le \widehat{F}_{\mathrm{J}}(\boldsymbol{\delta}_i) := \underbrace{\mathcal{L}(\boldsymbol{s}_{0,i}; \theta^s) + \langle \boldsymbol{g}_{0,i}, \boldsymbol{\delta}_i \rangle}_{\text{first-order expansion}} + \underbrace{\tfrac{M}{2}\, \boldsymbol{\delta}_i^\top P_i \boldsymbol{\delta}_i}_{\text{curvature bound}}, \qquad P_i := J_{s,i}^\top J_{s,i}, \tag{10}$$

which holds exactly when the server output changes linearly along the displacement, and approximately otherwise, as analyzed in Proposition 2. Since the surrogate is tangent to the loss at the transmitted smashed data, any step that lowers $\widehat{F}_{\mathrm{J}}$ below its value at $\boldsymbol{\delta}_i = \mathbf{0}$ also keeps the loss below its value at $\boldsymbol{s}_{0,i}$, as in majorization–minimization (Hunter & Lange, 2004). CoeF-J uses the gradient of the surrogate as the compensated gradient,

$$\widetilde{\boldsymbol{g}}_{\mathrm{J},i} = \nabla \widehat{F}_{\mathrm{J}}(\boldsymbol{\delta}_i) = \boldsymbol{g}_{0,i} + \widehat{B}_{\mathrm{J},i}\boldsymbol{\delta}_i, \qquad \widehat{B}_{\mathrm{J},i} := MP_i, \tag{11}$$

which takes the form of equation 7. Therefore, the client needs only the gradient and Jacobian computed at the server with $\boldsymbol{s}$. In practice, the server does not send Jacobian, whose size grows with the output dimension. Instead, it sends R random projections, whose outer products equal P_i in expectation, as will be discussed in Section 4.

CoeF-J: interpretation as sensitivity. The matrix P_i has an intuitive meaning. To first order, a displacement $\boldsymbol{\delta}_i$ changes the server output by $J_{s,i}\boldsymbol{\delta}_i$, and thus

$$\boldsymbol{\delta}_i^\top P_i \boldsymbol{\delta}_i = \|J_{s,i}\boldsymbol{\delta}_i\|^2 \tag{12}$$

measures how strongly the displacement changes the server output. We call P_i the sensitivity matrix, since equation 12 is large for a displacement that changes the server output strongly and zero for a displacement that leaves it unchanged. With $\boldsymbol{g}_{0,i} = J_{s,i}^\top r_i$, the compensated gradient can be written as $\widetilde{\boldsymbol{g}}_{\mathrm{J},i} = J_{s,i}^\top(r_i + MJ_{s,i}\boldsymbol{\delta}_i)$, where $r_i + MJ_{s,i}\boldsymbol{\delta}_i$ is a first-order estimate of the residual at the current smashed data in which Λ_i is replaced by its bound MI. Since $\langle \boldsymbol{\delta}_i, \widehat{B}_{\mathrm{J},i}\boldsymbol{\delta}_i \rangle = M\|J_{s,i}\boldsymbol{\delta}_i\|^2 \geq 0$, a descent step with $\widetilde{\boldsymbol{g}}_{\mathrm{J},i}$ pulls the smashed data back toward $\boldsymbol{s}_{0,i}$, and more strongly for a displacement that changes the server output more. In this sense, CoeF-J reduces the error in equation 7 from the side of the displacement by keeping it small in the directions to which the server output is sensitive, whereas CoeF-D reduces the error from the side of the curvature. We note that this effect is not introduced by an additional regularizer, since it follows from the upper bound in equation 10 with the coefficient M fixed by the loss. We provide the full algorithms of CoeF-D and CoeF-J in Appendix A.

4 Theoretical Analysis

We analyze why the gradient outer product can serve as a curvature surrogate for CoeF-D and when its correction reduces the feedback error. For CoeF-J, we examine when its sensitivity-based surrogate upper-bounds the loss. Throughout this section, the server-side model is fixed at its round-initial value θ^s as in equation 6, and $\phi = f_s(\cdot;\theta^s)$ denotes the server output of length C. The quantities $J_{s,i}$, r_i, Λ_i, and G_i are defined as in Section 3. Our analysis relies on the following assumptions, whose resulting statements are proved in Appendix B.

Assumption 1. $\nabla_s^2 \mathcal{L}(\cdot;\theta^s)$ is L_H-Lipschitz on the segment from $\boldsymbol{s}_{0,i}$ to $\boldsymbol{s}_i$.

Assumption 2. The Hessian of the loss with respect to the server output satisfies $\nabla_\phi^2 \mathcal{L} \preceq MI$ at every server output.

For softmax cross-entropy ℓ, Assumption 2 holds with $M = \frac{1}{2}$ (Böhning, 1992); the output-space curvature of each sample in a mean-reduced mini-batch of size b is further scaled by $w = 1/b$. Let $H_{0,i} := \nabla_s^2 \mathcal{L}(\boldsymbol{s}_{0,i};\theta^s)$. By equation 6 and Assumption 1, the error of the compensated gradient satisfies

$$\|\widetilde{\boldsymbol{g}}_i - \boldsymbol{g}_i\| \leq \|\widehat{B}_i - H_{0,i}\|\,\|\boldsymbol{\delta}_i\| + \frac{L_H}{2}\|\boldsymbol{\delta}_i\|^2. \tag{13}$$

The first term measures the curvature mismatch at the transmitted smashed data; the second accounts for the change of curvature along the displacement.

CoeF-D. Let $D_i := \mathrm{diag}(\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i})$, so that $\widehat{B}_{\mathrm{D},i} = \lambda D_i$ in equation 8. Proposition 1 gives both the curvature interpretation of this diagonal outer product and the condition under which it improves the feedback at a fixed displacement.

Proposition 1 (Curvature Surrogate of CoeF-D)**.** *(i) For the cross-entropy loss averaged over a mini-batch, suppose that its labels are drawn independently from their predictive distributions at the round-initial output. Then*

$$\mathbb{E}[\widehat{B}_{\mathrm{D},i}] = \lambda w\,\mathrm{diag}(G_i).$$

(ii) For a fixed displacement with $D_i\boldsymbol{\delta}_i \neq \mathbf{0}$, let

$$\lambda_i^\star := \frac{\langle D_i\boldsymbol{\delta}_i, \bar{H}_i\boldsymbol{\delta}_i\rangle}{\|D_i\boldsymbol{\delta}_i\|^2}.$$

For $\lambda > 0$, CoeF-D has a smaller feedback error than gradient reuse if and only if $\lambda < 2\lambda_i^\star$.

Part (i) relates the diagonal outer product to the GGN diagonal in the stated predictive-label expectation; it does not assert equality for the observed labels. Part (ii) compares the two feedback errors at the same displacement. The threshold depends on how the CoeF-D correction aligns with the gradient change needed at that displacement.

CoeF-J. CoeF-J uses the sensitivity matrix $P_i = J_{s,i}^\top J_{s,i}$ in equation 10. Assumption 2 bounds the loss change caused by a change Δ in the server output by its first-order term $\langle r_i, \Delta\rangle$ plus $M\|\Delta\|^2/2$. When the server output changes linearly along $\boldsymbol{\delta}_i$, $\Delta = J_{s,i}\boldsymbol{\delta}_i$. Together with $\boldsymbol{g}_{0,i} = J_{s,i}^\top r_i$, this gives the surrogate $\widehat{F}_{\mathrm{J}}$ of equation 10.

Proposition 2 (Loss Upper Bound)**.** *Under Assumption 2, consider the exact sensitivity matrix P_i and the constant coefficient M in equation 10. If the server output is affine on the segment $[\boldsymbol{s}_{0,i}, \boldsymbol{s}_{0,i} + \boldsymbol{\delta}_i]$, then*

$$\mathcal{L}(\boldsymbol{s}_{0,i} + \boldsymbol{\delta}_i; \theta^s) \leq \widehat{F}_{\mathrm{J}}(\boldsymbol{\delta}_i),$$

with equality at $\boldsymbol{\delta}_i = \mathbf{0}$. If instead $\|J_{s,i}\| \leq B_\phi$ and $\|\nabla_s^2 \phi_k\| \leq L_\phi$ on the segment for every output component k, then

$$\mathcal{L}(\boldsymbol{s}_{0,i} + \boldsymbol{\delta}_i; \theta^s) \leq \widehat{F}_{\mathrm{J}}(\boldsymbol{\delta}_i) + O\big(L_\phi\|\boldsymbol{\delta}_i\|^2\big) \quad \text{as } \boldsymbol{\delta}_i \to \mathbf{0},$$

the remainder vanishes when $L_\phi = 0$.

Under the affine condition, lowering $\widehat{F}_{\mathrm{J}}$ below its value at $\boldsymbol{\delta}_i = \mathbf{0}$ also keeps the loss below its value at the transmitted smashed data. For a nonlinear server output, the same comparison includes the stated remainder. The proposition concerns the exact P_i and constant M; the implementation instead uses a single random projection $\widehat{P}_i$ and replaces M by a per-sample coefficient $M_i \leq wM$ computed from the predicted output change $(\boldsymbol{\delta}_i^\top \widehat{P}_i \boldsymbol{\delta}_i)^{1/2}$.

5 Experiments

Models and datasets. Vision experiments use ViT-Tiny and ViT-Base (Dosovitskiy et al., 2021) on CIFAR-100 (Krizhevsky, 2009), Food-101 (Bossard et al., 2014), and Tiny-ImageNet (Le & Yang, 2015); language experiments use DistilRoBERTa and RoBERTa (Liu et al., 2019) on 20 Newsgroups (Lang, 1995), MNLI (Williams et al., 2018), and SQuAD v2.0 (Rajpurkar et al., 2018). All models are fine-tuned with LoRA adapters (Hu et al., 2022) of rank 4 on the attention and MLP projections. Data are partitioned over $K = 50$ clients under an IID split and a Dirichlet split (non-IID) with $\alpha = 0.1$ on the vision tasks and $\alpha = 0.5$ on the language tasks. SQuAD v2.0 is partitioned IID only, since the extractive-QA loader has no label to skew. The cut layer, which determines the number of transformer blocks on the client side, is set to block 4 for the ViT models and RoBERTa and to block 2 for DistilRoBERTa, so that the numbers of trainable parameters of the client-side and server-side models are in a ratio of 1:2. Appendix C.1 varies the cut layer for extended results. Accuracy is reported for classification and exact match (EM) and F1 for SQuAD v2.0.

Baselines. Vanilla SFL exchanges smashed data and gradients at every batch. Gradient reuse (w/ perfect comp.) is oracle and Gradient reuse (w/o comp.) is naive baseline. AccSFL (Han et al., 2022) and FSL-SAGE are the representatives of auxiliary-network-based SFL as we have discussed in Section 2. While AccSFL relatively lacks of server feedback, FSL-SAGE periodically aligns the auxiliary network with server-side model using the collected activation history. GAS (Yang & Liu, 2024) and MU-SplitFed (Liang et al., 2025) are imbalance-update based SFL addressing the straggler issue. For the implementation of GAS and MU-SplitFed, we adopts the straggler setting, but allowing per-batch communication, as we have discussed in Section 2. All experiments, including the baselines, follow SplitFed V1 (Thapa et al., 2022), in which the server keeps one server-side model per client and aggregates them by FedAVG (McMahan et al., 2017) at the end of the round.

Table 2: Comparison results: test accuracy (%) on vision and language tasks, and EM and F1 (%) on SQuAD v2.0. – denotes that the method is not applicable to the task.

Model	Method	CIFAR-100	Food-101	Tiny-ImageNet
ViT-Tiny	Vanilla SFL	85.0 ± 0.3	83.8 ± 0.1	78.6 ± 0.2
	Gradient reuse (w/ perfect comp.)	84.5 ± 0.3	83.3 ± 0.0	78.3 ± 0.3
	Gradient reuse (w/o comp.)	78.4 ± 0.4	75.3 ± 2.1	72.8 ± 0.2
	AccSFL	65.6 ± 1.2	67.5 ± 0.1	47.4 ± 0.2
	FSL-SAGE	66.9 ± 7.2	60.5 ± 0.9	50.9 ± 9.0
	GAS	75.2 ± 0.9	73.8 ± 0.6	71.7 ± 0.2
	MU-SplitFed	1.0 ± 0.0	1.0 ± 0.0	0.5 ± 0.0
	CoeF-D (ours)	**84.5 ± 0.2**	83.4 ± 0.2	**78.3 ± 0.1**
	CoeF-J (ours)	83.9 ± 0.3	**83.6 ± 0.0**	77.8 ± 0.1
ViT-Base	Vanilla SFL	91.9 ± 0.0	90.3 ± 0.1	90.1 ± 0.1
	Gradient reuse (w/ perfect comp.)	91.7 ± 0.1	90.0 ± 0.2	90.0 ± 0.1
	Gradient reuse (w/o comp.)	91.3 ± 0.2	88.5 ± 0.2	89.5 ± 0.3
	AccSFL	89.0 ± 0.3	87.7 ± 0.2	84.1 ± 0.2
	FSL-SAGE	77.3 ± 2.6	73.3 ± 3.1	58.6 ± 8.5
	GAS	90.9 ± 0.2	88.4 ± 0.1	89.2 ± 0.2
	MU-SplitFed	59.1 ± 0.5	65.4 ± 0.3	68.0 ± 0.2
	CoeF-D (ours)	**91.7 ± 0.1**	**90.1 ± 0.1**	**90.0 ± 0.1**
	CoeF-J (ours)	91.4 ± 0.0	**90.1 ± 0.1**	89.9 ± 0.1

Model	Method	20 Newsgroups	MNLI-m	MNLI-mm	SQuAD v2.0	
					EM	F1
DistilRoBERTa	Vanilla SFL	69.8 ± 0.2	83.3 ± 0.3	83.7 ± 0.3	71.3 ± 0.2	74.5 ± 0.2
	Gradient reuse (w/ perfect comp.)	69.3 ± 0.2	83.0 ± 0.1	83.1 ± 0.2	70.2 ± 2.1	73.1 ± 1.5
	Gradient reuse (w/o comp.)	69.3 ± 0.0	37.1 ± 6.2	37.7 ± 7.1	49.9 ± 0.2	50.0 ± 0.2
	AccSFL	66.6 ± 0.3	77.7 ± 0.3	78.5 ± 0.3	61.6 ± 1.2	64.7 ± 1.1
	FSL-SAGE	62.0 ± 0.8	78.8 ± 0.9	79.7 ± 0.7	62.1 ± 0.8	65.2 ± 0.7
	GAS	68.0 ± 0.5	38.9 ± 2.5	39.8 ± 3.1	–	–
	MU-SplitFed	5.1 ± 0.1	33.6 ± 1.3	33.7 ± 1.1	10.7 ± 0.0	11.5 ± 0.0
	CoeF-D (ours)	69.7 ± 0.1	79.7 ± 0.5	80.0 ± 0.3	65.7 ± 5.3	68.2 ± 4.4
	CoeF-J (ours)	**69.8 ± 0.3**	**83.0 ± 0.4**	**83.4 ± 0.2**	**68.6 ± 1.1**	**71.7 ± 0.9**
RoBERTa	Vanilla SFL	70.4 ± 0.5	87.3 ± 0.2	87.1 ± 0.0	79.3 ± 0.4	82.4 ± 0.3
	Gradient reuse (w/ perfect comp.)	70.0 ± 0.4	86.5 ± 0.5	86.4 ± 0.2	77.9 ± 2.0	81.0 ± 1.6
	Gradient reuse (w/o comp.)	69.5 ± 0.6	32.4 ± 0.4	32.6 ± 0.5	47.0 ± 3.9	47.4 ± 3.6
	AccSFL	67.0 ± 0.7	84.3 ± 0.3	84.9 ± 0.2	71.7 ± 0.5	74.7 ± 0.5
	FSL-SAGE	64.5 ± 0.3	83.2 ± 0.5	83.6 ± 0.6	68.6 ± 0.8	71.9 ± 0.8
	GAS	68.9 ± 0.9	35.5 ± 4.9	36.1 ± 6.1	–	–
	MU-SplitFed	5.3 ± 0.0	34.5 ± 1.3	34.5 ± 1.1	10.7 ± 0.0	11.5 ± 0.0
	CoeF-D (ours)	70.1 ± 0.5	**87.2 ± 0.1**	86.8 ± 0.2	77.2 ± 0.6	79.4 ± 0.7
	CoeF-J (ours)	**70.6 ± 0.3**	**87.2 ± 0.1**	**86.9 ± 0.0**	**78.2 ± 0.3**	**81.4 ± 0.3**

Hyperparameters. All methods are trained for $T = 100$ rounds with 10% client participation per round, one local epoch, a batch size of 32, and AdamW on both the client and server sides, after a warm-up of three rounds of vanilla SFL. For each method, the learning rate is selected from $\{10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$. A run whose final accuracy remains at $1/C$ on a C-class task (1.0% on CIFAR-100 and 0.5% on Tiny-ImageNet) is reported as is and regarded as failing to train. CoeF-D uses a constant compensation factor $\lambda = 3000$ with the cross-entropy loss averaged over the mini-batch ($w = 1/32$), which is about 94 times the reference scale $\lambda = 1/w$ of Proposition 1(i). Since the threshold in Proposition 1(ii) depends on the curvature along the displacement rather than on w, we treat λ as a hyperparameter and select it empirically. CoeF-J uses only the projected sensitivity matrix $\widehat{P}_i$ with a single random projection ($R = 1$) on every task, including those with few classes, and replaces M by a per-sample coefficient $M_i \leq wM$ computed from the predicted output change $(\boldsymbol{\delta}_i^\top \widehat{P}_i \boldsymbol{\delta}_i)^{1/2}$; Proposition 2, which assumes the exact P_i and the constant M, does not cover these runs. We report the mean and standard deviation over three seeds, and the entries without standard deviation are single runs. In all tables, the best and second-best results among the methods other than the reference methods are shown in bold and underlined, respectively, in each column.

Comparison results. Table 2 reports the results under IID split. Although CoeF-SFL communicates only once per round, its better variant stays within 0.5 pp of vanilla SFL on every vision task, within 0.3 pp on 20 Newsgroups and MNLI, and within 2.8 pp on SQuAD v2.0. CoeF-J is particularly effective with DistilRoBERTa, where it exceeds CoeF-D by 3.3–3.4 pp on MNLI and by 2.9 pp in EM on SQuAD v2.0 with a much smaller standard deviation. In several columns, CoeF-SFL even

slightly exceeds the oracle, e.g., 87.2% versus 86.5% on MNLI-m with RoBERTa, which may be attributed to the regularization effect of the compensation term.

Gradient reuse (w/o comp.) collapses to near-chance accuracy on MNLI and to around 50% on SQuAD v2.0, and CoeF-SFL recovers the performance by up to 54.8 pp. Among the methods other than the reference methods, CoeF-SFL achieves the best performance in every column, improving on the best auxiliary-network-based method by 16.1–27.4 pp with ViT-Tiny, 2.4–5.9 pp with ViT-Base, and 2.0–6.7 pp on the language tasks.

The auxiliary-network-based methods degrade more on the vision tasks than on the language tasks. With ViT-Tiny, they fall 16.3–31.2 pp below vanilla SFL, whereas their gaps remain within 10.7 pp on all the language tasks. We conjecture that this difference stems from how strongly the intermediate representations are compressed. Vision representations progressively discard label-irrelevant input information (Shwartz-Ziv & Tishby, 2017; Wang et al., 2021), and a local objective at a shallow cut layer accelerates this compression before the server-side model can exploit the information. In contrast, masked language models preserve more information about the input tokens (Voita et al., 2019), so a local objective is less likely to collapse what the subsequent layers require. Consistently, the largest gap among the language tasks appears on SQuAD v2.0, which requires fine-grained token-level information, and with ViT-Base, whose client-side model has a larger capacity, the gap of AccSFL shrinks to 2.6–6.0 pp, in line with (Wang et al., 2021). FSL-SAGE, however, still falls 14.6–31.5 pp behind with ViT-Base and varies by up to 8.5 pp across seeds, which suggests unstable training in our setting.

GAS is competitive with ViT-Base and on 20 Newsgroups, staying within 1.9 pp of vanilla SFL, but it falls 6.9–10.0 pp behind with ViT-Tiny and fails to train on MNLI. It is not applicable to SQuAD v2.0, since its activation generation is conditioned on class labels, which the span-extraction task does not provide. MU-SplitFed yields near-trivial predictions on most tasks, e.g., $1/C$ with ViT-Tiny, on 20 Newsgroups, and on MNLI. We attribute this to the scalar feedback from the server, which conveys too little information for the client-side model to learn without backpropagation under the same number of rounds.

Extended results and discussion. In Appendix C, we provide additional experiments across different cut layers and under the non-IID split. CoeF-SFL consistently outperforms the baselines regardless of the cut layer, and its advantage becomes more pronounced under the non-IID split. In Appendix D.1, we compare the gradient outer product (Zheng et al., 2017) adopted by CoeF-D with two other Hessian approximations, i.e., a low-rank approximation of the Gauss–Newton matrix (Schraudolph, 2002; Martens, 2020) computed with the Lanczos method (Lanczos, 1950) denoted by GGN, and a diagonal approximation estimated with the Hutchinson estimator (Bekas et al., 2007), denoted by Diagonal. The gradient outer product performs best among them in our setting. In Appendix D.2, we further analyze the communication and computation costs of the methods, which reveals the trade-offs among them beyond the performance and shows that CoeF-SFL achieves the highest accuracy without increasing the burden on the clients.

6 Conclusion

In this paper, we identified that auxiliary-network-based SFL reduces communication by replacing the E2E objective with a local objective, which limits the collaborative training between the client and the server. To retain the E2E objective under once-per-round communication, we proposed CoeF-SFL, in which the client reuses the gradients received at the beginning of each round and compensates their staleness with a curvature-based correction in the activation space. CoeF-D realizes this correction with a diagonal gradient outer product that requires no additional server computation, and CoeF-J exploits the Jacobian-based Hessian of a surrogate loss that upper-bounds the true loss. Across vision and language tasks, model capacities, cut layers, and data distributions, CoeF-SFL achieves performance close to that of vanilla SFL and consistently outperforms the auxiliary-network-based methods under the same communication frequency. Extending the analysis to the joint dynamics of the client-side and server-side updates remains an interesting direction for future work.

Reproducibility Statement

We have made the following efforts to ensure reproducibility. Source code for CoeF-SFL and all baselines(FSL-SAGE, AccSFL..etc), together with the scripts used to produce every table and figure, is available at https://anonymous.4open.science/r/CoeF-SFL-2686/. The complete training procedure, including the gradient compensating, is specified in Algorithms A. All datasets used in this study are publicly available: CIFAR-100, Food-101, Tiny-ImageNet, 20 Newsgroups, MNLI, and SQuAD v2.0. Section 5 describes the model and LoRA settings, data partitioning, training procedure, and hyperparameters. Appendix C reports additional results across cut layers and non-IID partitions.

Statement on the Use of AI

We used large language models, including Claude (Anthropic) and Codex (OpenAI), to assist with implementing experiment scripts,, mathematical, and writing assistant. Among the tasks with required disclosure, it was used to assist in implementing the method from the authors' algorithm specification, and to assist in developing and writing the proofs of the theoretical results in the appendix, including suggesting proof steps and intermediate lemmas; the theorem statements were formulated by the authors, and every proof was checked step by step and verified by the authors. It was not used to develop the conceptual framework, to formulate the mathematical claims, to propose hypotheses, to design experiments, or to generate or process data; the remaining required-disclosure tasks are not applicable. Among the tasks with recommended disclosure, it was used to draft and edit parts of the manuscript for readability, resolve LaTeX issues, format references, and write experiment scripts. We have reviewed all AI-assisted work: LLM-generated code was tested by the authors, all proofs and claims were verified by the authors, and all reported numbers come from our experiment records. We take responsibility for the final content of this work.

References

Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57(11–12):1214–1229, 2007.

Dankmar Böhning. Multinomial logistic regression algorithm. *Annals of the Institute of Statistical Mathematics*, 44(1):197–200, 1992.

Dankmar Böhning and Bruce G. Lindsay. Monotonicity of quadratic-approximation algorithms. *Annals of the Institute of Statistical Mathematics*, 40(4):641–663, 1988.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision (ECCV)*, 2014.

Felix Dangel, Lukas Tatzel, and Philipp Hennig. ViViT: Curvature access through the generalized Gauss-Newton's low-rank structure. *Transactions on Machine Learning Research*, 2022.

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. In *International Conference on Learning Representations (ICLR)*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. URL https://arxiv.org/abs/1703.03400.

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning, 2019. URL https://arxiv.org/abs/1902.08438.

Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

Dong-Jun Han, Hasnain Irshad Bhatti, Jungmoon Lee, and Jaekyun Moon. Accelerating federated split learning via local-loss-based training, 2022. URL https://openreview.net/forum?id=SawkGZ3oR2J.

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 (ICLR)*, 2022.

David R. Hunter and Kenneth Lange. A tutorial on MM algorithms. *The American Statistician*, 58 (1):30–37, 2004.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics – Simulation and Computation*, 19(2):433–450, 1990. doi: 10.1080/03610919008812866.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020. URL https://arxiv.org/abs/1806.07572.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical Fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4): 255–282, 1950.

Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 331–339, 1995.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. Technical report, Stanford University, CS231N course report, 2015.

Dandan Liang, Jianing Zhang, Zhe Li, et al. Towards straggler-resilient split federated learning: An unbalanced update approach. *arXiv preprint arXiv:2510.21155*, 2025.

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.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Yujia Mu and Cong Shen. Communication and storage efficient federated split learning. *arXiv preprint arXiv:2302.05599*, 2023.

Srijith Nair, Michael Lin, Peizhong Ju, Amirreza Talebi, Elizabeth Serena Bentley, and Jia Liu. FSL-SAGE: Accelerating federated split learning via smashed activation gradient estimation. *arXiv preprint arXiv:2505.23182*, 2025.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

Seungeun Oh, Jihong Park, Praneeth Vepakomma, Sihun Baek, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. LocFedMix-SL: Localize, federate, and mix for improved scalability, convergence, and latency in split learning. In *Proceedings of the ACM Web Conference (WWW)*, 2022.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Jiyun Shin, Joonwoo Bae, Honggu Kang, Seongah Jeong, et al. FedSplitX: Federated split learning for computationally-constrained heterogeneous clients. *Neurocomputing*, 702:134671, 2026.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. SplitFed: When federated learning meets split learning. In *AAAI Conference on Artificial Intelligence*, 2022.

Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

Elena Voita, Rico Sennrich, and Ivan Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4396–4406, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1448.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. Revisiting locally supervised learning: An alternative to end-to-end training. In *International Conference on Learning Representations (ICLR)*, 2021.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

Jiarong Yang and Yuan Liu. GAS: Generative activation-aided asynchronous split federated learning. *arXiv preprint arXiv:2409.01251*, 2024.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning (ICML)*, 2017.

A ALGORITHM OF COEF-SFL

Algorithm 1 CoeF-SFL for a sampled client in one round. Lines marked (D) and (J) are executed only by CoeF-D and CoeF-J, respectively, and the quantities in (J) lines are computed for each sample in the mini-batch.

Require: client-side model θ^c and server-side model θ^s at the start of the round, I local updates, step size η; (D) compensation factor λ; (J) curvature bound M and number R of random projections

Round-initial exchange
  Client: compute $\boldsymbol{s} = \{f_c(x;\theta^c)\}$ from its local data $\mathcal{D} = \{(x,y)\}$, and send $\boldsymbol{s}$ to the server together with the labels $\{\boldsymbol{y}\}$
  Server: computes the gradients $\boldsymbol{g} = \{\nabla_s \mathcal{L}(s;\theta^s)\}$
    (J) draw $\boldsymbol{v}_1, \dots, \boldsymbol{v}_R \in \{-1,+1\}^C$ with independent random signs, and compute $\boldsymbol{u}_r = J_s^\top \boldsymbol{v}_r$
  send $\boldsymbol{g}$ to the client, together with $\{\boldsymbol{u}_r\}_r$ for CoeF-J
Local updates, starting from $\theta_0^c = \theta^c$ and $\theta_0^s = \theta^s$
for $i = 0, \dots, I-1$ **do**
  Client: $\boldsymbol{\delta}_i \leftarrow \boldsymbol{s}_i - \boldsymbol{s}_{0,i}$
    (D) $\widetilde{\boldsymbol{g}}_i \leftarrow \boldsymbol{g}_{0,i} + \lambda \operatorname{diag}(\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i})\boldsymbol{\delta}_i$ equation 8
    (J) $\widehat{P}_i \boldsymbol{\delta}_i \leftarrow R^{-1} \sum_{r=1}^{R} \boldsymbol{u}_{i,r} \langle \boldsymbol{u}_{i,r}, \boldsymbol{\delta}_i \rangle$, $\widetilde{\boldsymbol{g}}_i \leftarrow \boldsymbol{g}_{0,i} + M\, \widehat{P}_i \boldsymbol{\delta}_i$ equation 11
  Client: $\theta_{i+1}^c \leftarrow \theta_i^c - \eta\, J_i^\top \widetilde{\boldsymbol{g}}_i$
  Server: $\theta_{i+1}^s \leftarrow \theta_i^s - \eta\, \nabla_{\theta^s} \mathcal{L}(\boldsymbol{s}_{0,i}; \theta_i^s)$
end for
Aggregate θ_I^c and θ_I^s over the sampled clients by FedAVG

B PROOFS OF THEORETICAL RESULTS

Throughout this appendix, the server-side model is fixed at its round-initial value θ^s. We write $H(\boldsymbol{s}) := \nabla_s^2 \mathcal{L}(\boldsymbol{s}; \theta^s)$ and $H_{0,i} = H(\boldsymbol{s}_{0,i})$.

B.1 COEF-D

Lemma 1 (Feedback error). *Let $\bar{H}_i := \int_0^1 H(\boldsymbol{s}_{0,i} + \tau \boldsymbol{\delta}_i)\, d\tau$. The identity equation 6 gives $\boldsymbol{g}_i - \boldsymbol{g}_{0,i} = \bar{H}_i \boldsymbol{\delta}_i$. Thus gradient reuse has error $-\bar{H}_i \boldsymbol{\delta}_i$, while the compensated gradient has error $(\widehat{B}_i - \bar{H}_i)\boldsymbol{\delta}_i$. Under Assumption 1, this error satisfies equation 13.*

Proof. Applying the fundamental theorem of calculus to $\tau \mapsto \nabla_s \mathcal{L}(\boldsymbol{s}_{0,i} + \tau \boldsymbol{\delta}_i; \theta^s)$ gives equation 6. Assumption 1 gives

$$\|\bar{H}_i - H_{0,i}\| \le \int_0^1 L_H \tau \|\boldsymbol{\delta}_i\|\, d\tau = \frac{L_H}{2} \|\boldsymbol{\delta}_i\|.$$

Adding and subtracting $H_{0,i}$ in $\widehat{B}_i - \bar{H}_i$ proves equation 13. □

Proof of Proposition 1. (i) For a sample in the mini-batch, let p be the predictive distribution, y its label, and $w = 1/b$. Its output-gradient contribution is $r = w(p - \boldsymbol{e}_y)$, and its output-space Hessian is $\Lambda = w(\operatorname{diag}(p) - pp^\top)$. For $y \sim p$,

$$\mathbb{E}[\boldsymbol{e}_y] = p, \qquad \mathbb{E}[\boldsymbol{e}_y \boldsymbol{e}_y^\top] = \operatorname{diag}(p),$$

so

$$\mathbb{E}[rr^\top] = w^2(\operatorname{diag}(p) - pp^\top) = w\Lambda.$$

By the chain rule, the corresponding activation-gradient outer product has expectation $w J_s^\top \Lambda J_s$. For the full mini-batch, independent predictive labels make the cross-sample terms vanish because each centered gradient contribution has zero mean. Hence $\mathbb{E}[\boldsymbol{g}_{0,i} \boldsymbol{g}_{0,i}^\top] = wG_i$. Taking diagonals and using $\widehat{B}_{\mathrm{D},i} = \lambda \operatorname{diag}(\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i})$ gives the claim. (ii) At the fixed displacement in the proposition,

$$\begin{aligned} \|\widetilde{\boldsymbol{g}}_{\mathrm{D},i} - \boldsymbol{g}_i\|^2 - \|\boldsymbol{g}_{0,i} - \boldsymbol{g}_i\|^2 &= \|\lambda D_i \boldsymbol{\delta}_i - \bar{H}_i \boldsymbol{\delta}_i\|^2 - \|\bar{H}_i \boldsymbol{\delta}_i\|^2 \\ &= \lambda^2 \|D_i \boldsymbol{\delta}_i\|^2 - 2\lambda \langle D_i \boldsymbol{\delta}_i, \bar{H}_i \boldsymbol{\delta}_i \rangle \\ &= \lambda \|D_i \boldsymbol{\delta}_i\|^2 (\lambda - 2\lambda_i^\star). \end{aligned}$$

Since $\lambda > 0$ and $D_i \boldsymbol{\delta}_i \neq \mathbf{0}$, this difference is negative if and only if $\lambda < 2\lambda_i^\star$. □

B.2 CoEF-J

For completeness, the softmax cross-entropy loss satisfies the curvature bound used by Proposition 2. For $p = \mathrm{softmax}(\phi)$,

$$\nabla^2_\phi \ell = \mathrm{diag}(p) - pp^\top \preceq \tfrac{1}{2} I.$$

Indeed, for any $\boldsymbol{v}$, its quadratic form is the variance of v_K for $K \sim p$. This variance is at most one quarter of the squared range of the entries of $\boldsymbol{v}$, and that squared range is at most $2\|\boldsymbol{v}\|^2$. Mean reduction multiplies each sample's curvature by $w = 1/b$; hence $M = \frac{1}{2}$ is valid in Assumption 2.

Proof of Proposition 2. Write $\mathcal{L}_\phi$ for the loss as a function of the server output and put $\bar{\phi} = \phi(\boldsymbol{s}_{0,i})$. Assumption 2 and Taylor's formula give, for any output change Δ,

$$\mathcal{L}_\phi(\bar{\phi} + \Delta) \le \mathcal{L}_\phi(\bar{\phi}) + \langle r_i, \Delta \rangle + \frac{M}{2}\|\Delta\|^2.$$

If the server output is affine on the segment, its actual change is $\Delta = J_{s,i}\boldsymbol{\delta}_i$. Since $\boldsymbol{g}_{0,i} = J_{s,i}^\top r_i$ and $\|J_{s,i}\boldsymbol{\delta}_i\|^2 = \boldsymbol{\delta}_i^\top P_i \boldsymbol{\delta}_i$, the right-hand side is exactly $\widehat{F}_{\mathrm{J}}(\boldsymbol{\delta}_i)$. At $\boldsymbol{\delta}_i = \mathbf{0}$, both sides equal $\mathcal{L}(\boldsymbol{s}_{0,i}; \theta^s)$. For a nonlinear server output, write

$$\phi(\boldsymbol{s}_{0,i} + \boldsymbol{\delta}_i) = \bar{\phi} + J_{s,i}\boldsymbol{\delta}_i + \boldsymbol{\varrho}.$$

Taylor's integral formula and the stated bounds on the output components give $\|\boldsymbol{\varrho}\| \le (\sqrt{C}L_\phi/2)\|\boldsymbol{\delta}_i\|^2$. Using the actual output change in the preceding loss bound instead of $J_{s,i}\boldsymbol{\delta}_i$ adds

$$\langle r_i, \boldsymbol{\varrho} \rangle + M\langle J_{s,i}\boldsymbol{\delta}_i, \boldsymbol{\varrho} \rangle + \frac{M}{2}\|\boldsymbol{\varrho}\|^2.$$

Since $\|J_{s,i}\boldsymbol{\delta}_i\| \le B_\phi \|\boldsymbol{\delta}_i\|$, this addition is at most $\frac{1}{2}\varepsilon(\|\boldsymbol{\delta}_i\|)\|\boldsymbol{\delta}_i\|^2$, where

$$\varepsilon(u) = \sqrt{C}L_\phi(\|r_i\| + MB_\phi u) + \frac{1}{4}CML_\phi^2 u^2.$$

It vanishes when $L_\phi = 0$, completing the proof. □

C Additional Experiments

C.1 Effect of the Cut Layer

Table 3: Effect of the cut layer on ViT-Tiny/CIFAR-100 (IID): test accuracy (%). Cut k places the first k of the twelve transformer blocks on the client, and the last row reports the ratio of trainable parameters, excluding the embedding layer, between the client-side and server-side models.

Method	cut 1	cut 2	cut 4	cut 6	cut 8
Vanilla SFL	85.1 ± 0.1	84.9 ± 0.1	85.0 ± 0.3	84.9 ± 0.3	84.8 ± 0.3
Gradient reuse (w/ perfect comp.)	84.8 ± 0.1	84.8 ± 0.1	84.5 ± 0.3	84.3 ± 0.2	83.9 ± 0.2
Gradient reuse (w/o comp.)	84.5 ± 0.3	84.3 ± 0.2	78.4 ± 0.4	78.5 ± 0.3	77.7 ± 0.3
AccSFL	71.3 ± 1.4	62.0 ± 1.3	65.6 ± 1.2	71.4 ± 0.4	76.0 ± 0.2
FSL-SAGE	47.0 ± 17.2	55.2 ± 7.7	66.9 ± 7.2	69.6 ± 5.7	73.7 ± 4.1
GAS	82.9 ± 0.2	83.1 ± 0.3	75.2 ± 0.9	75.6 ± 0.6	76.1 ± 0.7
MU-SplitFed	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	9.5 ± 0.2
CoeF-D (ours)	84.9 ± 0.2	**85.0 ± 0.2**	**84.5 ± 0.2**	**84.3 ± 0.2**	**83.8 ± 0.3**
CoeF-J (ours)	**85.0 ± 0.1**	84.8 ± 0.1	83.9 ± 0.3	82.5 ± 0.1	81.3 ± 0.1
Client : server	1 : 11.04	1 : 5.02	1 : 2.01	1.01 : 1	1.98 : 1

Table 4: Effect of the cut layer on DistilRoBERTa/20 Newsgroups (IID): test accuracy (%). Cut k places the first k of the six transformer blocks on the client, and the last row reports the ratio of trainable parameters, excluding the embedding layer, between the client-side and server-side models.

Method	cut 1	cut 2	cut 3	cut 4
Vanilla SFL	69.6 ± 0.2	69.8 ± 0.2	69.7 ± 0.3	69.7 ± 0.3
Gradient reuse (w/ perfect comp.)	69.6 ± 0.2	69.3 ± 0.2	69.2 ± 0.3	69.2 ± 0.2
Gradient reuse (w/o comp.)	69.5 ± 0.0	69.3 ± 0.0	68.8 ± 0.2	67.7 ± 0.1
AccSFL	68.2 ± 0.3	66.6 ± 0.3	66.8 ± 0.3	67.8 ± 0.3
FSL-SAGE	59.9 ± 0.7	62.0 ± 0.8	64.8 ± 0.4	66.5 ± 0.0
GAS	68.4 ± 0.5	68.0 ± 0.5	66.9 ± 0.2	66.6 ± 0.7
MU-SplitFed	5.4 ± 0.7	5.1 ± 0.1	5.6 ± 0.2	7.0 ± 0.9
CoeF-D (ours)	69.7 ± 0.1	69.7 ± 0.1	69.3 ± 0.2	69.2 ± 0.3
CoeF-J (ours)	**69.8 ± 0.1**	**69.8 ± 0.3**	**69.8 ± 0.1**	**69.5 ± 0.3**
Client : server	1 : 5.09	1 : 2.04	1 : 1.03	1.92 : 1

C.2 Results under the Non-IID Split

Table 5: ViT-Tiny/vision tasks (non-IID): test accuracy (%).

Model	Method	CIFAR-100	Food-101	Tiny-ImageNet
ViT-Tiny	Vanilla SFL	70.9 ± 1.2	63.9 ± 3.1	59.1 ± 3.3
	Gradient reuse (with perfect comp.)	66.9 ± 0.5	55.6 ± 1.5	54.7 ± 2.9
	Gradient reuse (w/o comp.)	1.0 ± 0.1	1.0 ± 0.0	0.5 ± 0.0
	AccSFL	28.2 ± 1.3	32.5 ± 4.3	18.8 ± 2.1
	FSL-SAGE	33.6 ± 2.0	30.3 ± 5.5	20.0 ± 2.3
	GAS	46.0 ± 10.2	1.0 ± 0.0	0.6 ± 0.1
	MU-SplitFed	1.0 ± 0.0	1.0 ± 0.0	0.5 ± 0.0
	CoeF-D (ours)	67.3 ± 1.5	53.5 ± 2.4	54.9 ± 2.7
	CoeF-J (ours)	**69.1 ± 1.2**	**62.9 ± 3.2**	**57.7 ± 3.4**

Table 6: DistilRoBERTa/language tasks (non-IID): test accuracy (%).

Model	Method	20 Newsgroups	MNLI-m	MNLI-mm
DistilRoBERTa	Vanilla SFL	67.2 ± 0.6	80.4 ± 0.5	80.9 ± 0.7
	Gradient reuse (with perfect comp.)	64.9 ± 1.3	74.6 ± 4.2	74.5 ± 4.8
	Gradient reuse (w/o comp.)	62.5 ± 1.7	33.3 ± 1.5	33.3 ± 1.4
	AccSFL	61.1 ± 1.5	71.4 ± 1.7	71.9 ± 2.2
	FSL-SAGE	53.2 ± 1.6	73.0 ± 1.3	73.4 ± 1.3
	GAS	64.2 ± 0.9	37.9 ± 6.2	38.3 ± 7.0
	MU-SplitFed	5.2 ± 0.1	34.5 ± 1.3	34.5 ± 1.1
	CoeF-D (ours)	65.7 ± 1.0	75.7 ± 0.7	75.6 ± 0.9
	CoeF-J (ours)	**67.1 ± 0.6**	**77.0 ± 0.5**	**77.4 ± 0.9**

D Further Analysis

D.1 Comparison of Hessian Approximations

Here, we compare the gradient outer product (Zheng et al., 2017) adopted by CoeF-D with two other Hessian approximations, i.e., a low-rank approximation of the Gauss–Newton matrix (Schraudolph, 2002; Martens, 2020) computed with the Lanczos method (Lanczos, 1950) denoted by GGN, and a diagonal approximation estimated with the Hutchinson estimator (Bekas et al., 2007), denoted by Hutchinson diagonal. We explain each method and introduce the corresponding first.

Table 7: Notation for further analysis.

Symbol	Meaning
N	samples per client
B	mini-batches per client and round
d	dimension of the smashed data of one sample
f^c, b^c	client forward / backward pass
f^s, b^s	server forward / full backward pass
$\mathrm{b}^s_{\mathrm{in}}$	server backward pass to the smashed data
k	rank of the low-rank Fisher (GGN)
C	number of classes, the dimension of the server output
$\boldsymbol{v}$	input vector of a Gauss–Newton matrix-vector product
J_s	Jacobian of the server output with respect to the smashed data of one sample ($C \times d$)
Λ_i	Hessian of the loss with respect to the server output at the round-initial output of sample i
$J_s^\top \Lambda J_s$	generalized Gauss–Newton matrix of one sample
U	eigenvectors for the k leading eigenvalues of $J_s^\top \Lambda J_s$
Θ	diagonal matrix of the k leading eigenvalues ($k \times k$)
m	Lanczos iterations per sample (low-rank Fisher)
m_d	Rademacher probes per sample (Hutchinson diagonal)
R	random projections per sample (CoeF-J)
$\widehat{B}$	surrogate hessian per sample (GGN, CoeF-D)
λ	compensation factor (CoeF-D)
N_{buf}	samples stored in the FSL-SAGE buffer
E	epochs of auxiliary-network training per alignment
l	rounds between alignments (FSL-SAGE)
a	training cost of the auxiliary network
P	perturbations per update (MU-SplitFed)
τ	server updates per client update (MU-SplitFed)
N_g	activations generated by the server per client and round (GAS)

GGN. The server runs m Lanczos iterations (Lanczos, 1950) on the per-sample generalized Gauss–Newton operator $\boldsymbol{v} \mapsto G_i \boldsymbol{v} = J_{s,i}^\top \Lambda_i J_{s,i} \boldsymbol{v}$, evaluated through Gauss–Newton matrix-vector products without forming the matrix (Schraudolph, 2002), and keeps the k leading eigenpairs (U, Θ). For the cross-entropy loss this operator coincides with the Fisher information (Martens, 2020). The client applies $\widehat{B}_i \boldsymbol{\delta}_i = U \Theta U^\top \boldsymbol{\delta}_i$ at a cost of $k(\mathrm{f}_s + \mathrm{b}_s)$ per sample. The communication cost for downlink grows to $(1 + k)d$ values per sample. With $m = 8$ and $k = 4$, the server needs 41.45 GFLOPs per sample, of which the Lanczos iterations account for 34.23 GFLOPs.

Hutchinson diagonal. The server estimates $\mathrm{diag}(H_{0,i})$ by $m_d^{-1} \sum_{j=1}^{m_d} \boldsymbol{z}_j \odot H_{0,i} \boldsymbol{z}_j$ with Rademacher vectors $\boldsymbol{z}_j$ (Hutchinson, 1990; Bekas et al., 2007). Each product $H_{0,i} \boldsymbol{z}_j$ is obtained by a central finite difference of two server gradient queries at $\boldsymbol{s}_{0,i} \pm \epsilon \boldsymbol{z}_j$, so the server adds $2m_d(\mathrm{f}^s + \mathrm{b}^s_{\mathrm{in}})$ per sample. With $m_d = 8$ this gives $7.226 + 16 \times 3.591 = 64.69$ GFLOPs per sample. The diagonal has the size of $\boldsymbol{g}_{0,i}$, so the downlink doubles, and the client correction is elementwise.

CoeF-D (diagonal outer product). $\widehat{B}_{\mathrm{D}} = \lambda \, \mathrm{diag}(\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i})$ is formed by the client from $\boldsymbol{g}_{0,i}$, so the server cost and the downlink equal those of gradient reuse, and the client adds $O(d)$ per sample.

Choice of curvature approximation. Table 8 compares CoeF-D with two alternatives, which are a low-rank Gauss–Newton approximation computed with the Lanczos method (GGN) and a stochastic estimate of its diagonal (Diagonal). Under IID data, GGN matches CoeF-D but increases the server computation by $5.7\times$ and the communication cost by $5.0\times$, while Diagonal costs $8.9\times$ more computation and barely improves over gradient reuse without compensation. Under non-IID data, both alternatives collapse, and Diagonal falls to the chance level at cut 4. CoeF-D instead

Table 8: Gradient reuse with different curvature approximations on ViT-Tiny/CIFAR-100. The top table reports test accuracy (%), and the bottom table reports the server computation per sample and the communication cost at cut 4. The values are measured in our setting from Section 5

Method	cut 1		cut 4		cut 8	
	IID	non-IID	IID	non-IID	IID	non-IID
GGN	84.8	46.3	84.5	19.4	83.0	7.8
Diagonal	84.7	66.2	78.7	1.0	78.0	7.7
CoeF-D	84.8	72.4	84.5	69.1	83.7	48.8

Method	**Server computation (GFLOPs)**	**Computation complexity**	**Comm. cost (MB)**	**Comm. volume**
Gradient reuse (w/o comp.)	7.23	$2(\mathrm{f}^s + \mathrm{b}^s)$	144.29	$O(Nd)$
GGN	41.45	$(k+1)(\mathrm{f}^s+\mathrm{b}^s)$	721.45	$O((k+1)Nd)$
Hutchinson diagonal	64.69	$\mathrm{f}^s + \mathrm{b}^s + 2m_d(\mathrm{f}^s + \mathrm{b}^s_{\mathrm{in}})$	288.58	$O(2Nd)$
CoeF-D	7.23	$2(\mathrm{f}^s+\mathrm{b}^s)$	144.29	$O(Nd)$

retains 69.1% at cut 4, since its correction scales with $\boldsymbol{g}_{0,i} \odot \boldsymbol{g}_{0,i}$ and thus stays small for samples that are already well fitted. As CoeF-D achieves this with the same computation and communication as gradient reuse without compensation, we adopt the diagonal gradient outer product as the default compensation.

D.2 Communication and Computation Costs

Table 9 summarizes the communication and computation costs of all methods. Compared with AccSFL, CoeF-SFL requires more server computation and uses downlink communication, which AccSFL does not need at all. We argue that neither cost is a major limitation. The server computation of CoeF-D is 7.23 GFLOPs per sample, which is twice that of vanilla SFL and AccSFL because the server additionally computes the gradients of the round-initial smashed data. This overhead is a constant factor that does not grow with the number of local iterations compared to vanilla SFL, and it remains below the server computation of GAS (15.22 GFLOPs) and MU-SplitFed (10.06 GFLOPs). Since the server in SFL typically has far more computing power than the clients, such a constant increase is much easier to absorb than an increase in client computation. The downlink of CoeF-D carries the same volume as that of vanilla SFL (144.29 MB), so it adds no traffic beyond what gradient-based SFL already requires. It is, however, delivered in a single message per round instead of one message per mini-batch, which reduces the number of transfers from 64 to 2 and removes the synchronization between the client and the server during local training.

Tables 10 and 11 indicate that the auxiliary network can be a more restrictive constraint than these costs. AccSFL assumes a small auxiliary network (Han et al., 2022), but even an FC layer without hidden layers amounts to 34.9% of the trainable client-side parameters on ViT-Tiny and 13.9% on DistilRoBERTa in our setting, which is a considerable overhead for the resource-limited clients that SFL targets. We vary the auxiliary network of AccSFL from an FC layer with LoRA adapters, which uses only 1.2% of the trainable client-side parameters, to MLPs with up to 12 hidden layers, which exceed the trainable client-side parameters by a factor of 2.4 on ViT-Tiny and 17 on DistilRoBERTa. The accuracy of AccSFL increases with the size of the auxiliary network and then saturates. On ViT-Tiny/CIFAR-100, it rises from 63.6% with LoRA to 81.9% with eight hidden layers, which is still below the 84.5% of CoeF-D. Even with the largest auxiliary network, AccSFL remains below CoeF-J in every setting and below CoeF-D in all settings except non-IID Food-101. On DistilRoBERTa, the accuracy saturates after a single hidden layer at about 68% under IID data and 64% under non-IID data, whereas CoeF-J reaches 69.8% and 67.1%. FSL-SAGE, whose auxiliary network is 25 times larger than the trainable client-side model on ViT-Tiny, also falls far behind CoeF-SFL, which shows that enlarging the auxiliary network alone does not close the gap.

Furthermore, these results show that the accuracy of auxiliary-network-based SFL is directly tied to the size of an auxiliary network that has to be stored and trained on the client, whose memory and computation are the scarcest resources in SFL. CoeF-SFL instead places its additional cost on the server computation and the downlink, which are less constrained, and achieves higher accuracy with less client computation and without any auxiliary network.

Table 9: Communication per client and round (top) and computation per sample (bottom) on ViT-Tiny/CIFAR-100 at cut 4. Transfers count uplink and downlink messages separately. For FSL-SAGE, entries with a slash give a round without alignment (left) and an alignment round (right, every $l = 10$ rounds), in which the server recomputes the gradients of $N_{\mathrm{buf}} = 64$ stored samples and trains the auxiliary network on them for $E = 40$ epochs, with a counting the double backward of the alignment loss. The values are measured in our setting from Section 5.

Method	**Exch.**	**Transfers**	**Volume**	**Up (MB)**	**Down (MB)**
Vanilla SFL	$O(B)$	64	$O(Nd)$	144.29	144.29
Gradient reuse (w/ perfect comp.)	$O(B)$	65	$O(Nd)$	288.58	144.29
Gradient reuse (w/o comp.)	$O(1)$	2	$O(Nd)$	144.29	144.29
AccSFL	$O(1)$	1	$O(Nd)$	144.29	0.00
MU-SplitFed	$O(1)$	2	$O(Nd)$	432.87	0.000122
FSL-SAGE	$O(1)$	1 / 2	$O(Nd)$	144.29	0.00 / 5.32
GAS	$O(1)$	2	$O(Nd)$	144.29	144.29
CoeF-D (ours)	$O(1)$	2	$O(Nd)$	144.29	144.29
CoeF-J (ours)	$O(1)$	2	$O((1+R)Nd)$	144.29	288.58

	Client		**Server**	
Method	**Complexity**	**GFLOPs**	**Complexity**	**GFLOPs**
Vanilla SFL	$\mathrm{f}^c{+}\mathrm{b}^c$	1.88	$\mathrm{f}^s{+}\mathrm{b}^s$	3.63
Gradient reuse (w/ perfect comp.)	$2\mathrm{f}^c{+}\mathrm{b}^c$	2.77	$2(\mathrm{f}^s{+}\mathrm{b}^s)$	7.23
Gradient reuse (w/o comp.)	$2\mathrm{f}^c{+}\mathrm{b}^c$	2.77	$2(\mathrm{f}^s{+}\mathrm{b}^s)$	7.23
AccSFL	$2\mathrm{f}^c{+}\mathrm{b}^c{+}\mathrm{a}$	2.80	$\mathrm{f}^s{+}\mathrm{b}^s$	3.63
MU-SplitFed	$(1{+}2P)\mathrm{f}^c$	2.69	$2P(\tau{+}1)\mathrm{f}^s$	10.06
FSL-SAGE	$2\mathrm{f}^c{+}\mathrm{b}^c$	2.77	$\mathrm{f}^s{+}\mathrm{b}^s$ / $(1{+}\frac{N_{\mathrm{buf}}}{N})(\mathrm{f}^s{+}\mathrm{b}^s){+}\frac{EN_{\mathrm{buf}}}{N}\mathrm{a}$	3.63 / 14.21
GAS	$2\mathrm{f}^c{+}\mathrm{b}^c$	2.77	$(2{+}\frac{N_g}{N})(\mathrm{f}^s{+}\mathrm{b}^s)$	15.22
CoeF-D (ours)	$2\mathrm{f}^c{+}\mathrm{b}^c{+}O(d)$	2.77	$2(\mathrm{f}^s{+}\mathrm{b}^s)$	7.23
CoeF-J (ours)	$2\mathrm{f}^c{+}\mathrm{b}^c{+}O(Rd)$	2.77	$2\mathrm{f}^s{+}(2{+}R)\mathrm{b}^s$	9.14

Table 10: Auxiliary-network size sweep for AccSFL on ViT-Tiny at cut 4: server-side test accuracy (%). Aux. params gives the trainable parameters of the auxiliary network and their ratio to the trainable client-side parameters (55.3K under LoRA). FC is the auxiliary network of AccSFL in the other tables, and FC + LoRA freezes the FC weight and trains rank-r adapters and the bias. MLP stacks the given number of hidden layers of width $d/2 = 96$ with GELU. FSL-SAGE uses its own auxiliary network, which consists of three server-side blocks followed by a layer norm and an FC head.

	Aux. params		**IID**			**Non-IID**		
Method	#	% client	CIFAR-100	Food-101	Tiny-ImageNet	CIFAR-100	Food-101	Tiny-ImageNet
FC + LoRA ($r{=}2$)	0.7K	1.2	63.6 ± 1.2	64.9 ± 0.4	48.8 ± 0.7	36.6 ± 0.6	37.0 ± 4.5	24.1 ± 2.7
FC + LoRA ($r{=}4$)	1.3K	2.3	63.8 ± 1.0	65.2 ± 0.5	49.4 ± 1.1	37.1 ± 0.8	37.5 ± 4.6	24.5 ± 2.2
FC (AccSFL)	19.3K	34.9	65.6 ± 1.2	67.5 ± 0.1	47.4 ± 0.2	28.2 ± 1.3	32.5 ± 4.3	18.8 ± 2.1
MLP, 1 hidden	28.2K	51.0	78.8 ± 0.2	77.5 ± 0.2	70.7 ± 0.2	54.7 ± 1.9	51.5 ± 3.8	42.0 ± 2.8
MLP, 4 hidden	56.2K	101.6	81.5 ± 0.4	79.9 ± 0.2	74.4 ± 0.1	62.7 ± 1.9	56.9 ± 3.2	51.4 ± 3.2
MLP, 8 hidden	93.4K	168.9	81.9 ± 0.1	80.6 ± 0.1	75.4 ± 0.2	64.6 ± 2.2	58.7 ± 3.7	53.9 ± 3.6
MLP, 12 hidden	130.7K	236.3	81.8 ± 0.2	80.7 ± 0.1	75.5 ± 0.2	64.8 ± 1.8	59.2 ± 3.4	54.2 ± 3.3
FSL-SAGE	1.40M	2524.1	66.9 ± 7.2	60.5 ± 0.9	50.9 ± 9.0	33.6 ± 2.0	30.3 ± 5.5	20.0 ± 2.3
CoeF-D (ours)	0	0.0	84.5 ± 0.2	83.4 ± 0.2	78.3 ± 0.1	67.3 ± 1.5	53.5 ± 2.4	54.9 ± 2.7
CoeF-J (ours)	0	0.0	83.9 ± 0.3	83.6 ± 0.0	77.8 ± 0.1	69.1 ± 1.2	62.9 ± 3.2	57.7 ± 3.4

Table 11: Auxiliary-network size sweep on DistilRoBERTa/20 Newsgroups at cut 2: server-side test accuracy (%). Columns as in Table 10, where the client-side model has 110.6K trainable parameters (LoRA) and MLP uses hidden width $d/2 = 384$. FSL-SAGE uses its own auxiliary network, which consists of two of the four server-side blocks followed by a layer norm and an FC head, with 14.3M parameters.

	Aux. params			
Method	**#**	**% client**	**IID**	**Non-IID**
FC + LoRA (r=2)	1.6K	1.4	65.3 ± 0.6	59.7 ± 1.9
FC + LoRA (r=4)	3.2K	2.9	65.4 ± 0.5	59.8 ± 1.7
FC (AccSFL)	15.4K	13.9	66.6 ± 0.3	61.1 ± 1.5
MLP, 1 hidden	303.0K	274.0	68.1 ± 0.1	64.0 ± 1.1
MLP, 4 hidden	746.5K	675.0	67.9 ± 0.0	64.0 ± 0.6
MLP, 8 hidden	1.34M	1209.7	67.8 ± 0.8	64.4 ± 1.1
MLP, 12 hidden	1.93M	1744.5	67.8 ± 0.7	64.1 ± 1.5
FSL-SAGE	14.3M	115.3	62.0 ± 0.8	53.2 ± 1.6
CoeF-D (ours)	0	0.0	69.7 ± 0.1	65.7 ± 1.0
CoeF-J (ours)	0	0.0	69.8 ± 0.3	67.1 ± 0.6