STREAMSPLIT: DECOUPLING SPLIT FEDERATED LEARNING FOR ASYNCHRONOUS AND CONTINUOUS OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Distributed learning enables collaborative model training across distributed nodes. Split Federated Learning (SFL), a form of distributed learning, partitions model computation between clients and a server, but its tightly coupled execution requires clients and the server to wait for gradients and features from each other for subsequent training. This bidirectional dependency causes frequent execution stalls and severe resource underutilization, particularly under heterogeneous environments, leading to low training efficiency. In this paper, we propose StreamSplit, a decoupled SFL framework that enables clients and the server to optimize asynchronously and continuously while maintaining consistency between them. StreamSplit introduces a lightweight Shadow Server-side Model (SSM) at each client to substitute for the server-side model during local updates, enabling clients to perform continuous training without waiting for gradients. We design a depth-decoupled SSM architecture with a corresponding multi-depth alignment strategy, allowing the SSM to approximate the evolving server-side model and provide reliable surrogate gradients for client updates. Meanwhile, StreamSplit maintains a Latest Feature Table to store and provide candidate features for continuous server-side model training. We also introduce an optimization-aware feature selection strategy to select informative features that facilitate server-side model training and SSM alignment. Experiments show that StreamSplit improves training efficiency and scalability over existing SFL methods under heterogeneous environments. It also performs well on both convolutional and transformer-based models, demonstrating its applicability across model architectures.

1 INTRODUCTION

Distributed learning has become a widely adopted paradigm for collaborative model training across distributed data and computing resources. Federated Learning (FL) (Chen et al., 2025a; Wiesner et al., 2024; Wang & Gao, 2025; Hu et al., 2026; Li et al., 2026) and Split Learning (SL) (Sun & Ochiai, 2024; Wang et al., 2025) are two representative paradigms for distributed model training. Split Federated Learning (SFL) combines the complementary advantages of FL and SL, enabling collaborative training without sharing private data or the full model between clients and the server (Vepakomma et al., 2018; Radovic et al., 2025; Xia et al., 2025).

In SFL, the full model is partitioned at a predefined cut layer into a client-side model deployed on each client and a server-side model deployed on the server. During every training iteration, each client feeds local data into its client-side model and performs forward propagation to obtain an intermediate feature at the cut layer. The feature is then transmitted to the server and fed into the server-side model for subsequent forward propagation, after which the server computes the training loss and performs backpropagation to update the server-side model. The resulting gradient at the cut layer is sent back to the corresponding client, where backpropagation continues through the client-side model to update its parameters. After multiple iterations, a federated server aggregates the client-side models using FedAvg (McMahan et al., 2016). and broadcasts the updated global client-side model back to the clients for subsequent training (Han et al., 2024; Yao et al., 2025; Yan et al., 2026; Yang & Liu, 2025; Tirana et al., 2026).

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.

Existing SFL systems remain fundamentally constrained by the tightly coupled execution between clients and the server. In each training iteration, clients must wait for server-generated cut-layer gradients to continue local updates, while the server must wait for client-generated intermediate features to perform server-side optimization. This bidirectional dependency introduces frequent synchronization stalls, leaving both client and server resources underutilized and reducing training efficiency. The problem is further exacerbated in heterogeneous environments, where variations in client computation and communication speeds cause imbalanced feature arrivals, forcing the server to spend more time waiting for straggling clients and resulting in increased training delays (Liao et al., 2024; Chen et al., 2025b; Tirana et al., 2026).

Existing efforts mainly tackle the inefficiency in SFL from two perspectives. The first category focuses on execution-aware scheduling, which coordinates client feature generation (Xie et al., 2025; Chen et al., 2025c; Xie et al., 2026) and server-side processing (Gao et al., 2025) to mitigate feature arrival gaps caused by client heterogeneity,thereby mitigate server idleness. The second category employs local auxiliary networks (Han et al., 2021; Nair et al., 2025) or gradient-free optimization (Liang et al., 2025) to enable client-side updates without waiting for server-generated gradients.

However, existing approaches do not fully remove the client–server dependency. In execution-aware scheduling approaches, server-side optimization remains constrained by the arrival of client-generated features. Meanwhile, in auxiliary-based and gradient-free approaches, the auxiliary models may deviate from the evolving server-side during independent training, leading to inconsistent optimization guidance for client-side model and suboptimal feature features for server-side learning. These limitations suggest that to achieve full temporal decoupling in SFL and thereby mitigate resource idleness and improve overall training efficiency, two key challenges must be addressed concurrently. First, client-side optimization must operate without real-time gradient feedback from the server, while preserving consistency between client-side and server-side model updates. Specifically, the gradients used to update the client-side model should approximate the cut-layer gradients that would otherwise be provided by the server-side model, while the SSM should exhibit learning behavior consistent with the evolving server-side model, so that the client-side model continues to generate features that are beneficial for subsequent deep semantic extraction on the server. Second, server-side optimization should run continuously and efficiently, instead of halting to wait for the most recent intermediate features produced by clients.

Based on these insights, we propose *StreamSplit*, a decoupled SFL framework that eliminates the bidirectional waiting dependency while maintaining optimization consistency. StreamSplit deploys a lightweight *Shadow Server-side Model (SSM)* on each client, using a depth-decoupled architecture and server-guided multi-depth alignment to track the evolving server-side model and provide surrogate gradients for client-side optimization. Meanwhile, a *Latest Feature Table (LFT)* enables continuous server-side optimization from available client features, with an optimization-aware feature selection strategy to select features beneficial to server-side training and SSM alignment.

Our main contributions are summarized as follows:

- We propose *StreamSplit*, a decoupled SFL framework that removes the bidirectional client–server waiting dependency and enables asynchronous, continuous optimization while preserving optimization consistency between the two sides.
- A lightweight *Shadow Server-side Model (SSM)* is deployed on each client as a surrogate of the evolving server-side model. We design a *depth-decoupled architecture* with server-guided multi-depth alignment, enabling the SSM to generate more consistent surrogate gradients for continuous client-side optimization.
- A continuous server-side optimization mechanism is developed based on a *Latest Feature Table (LFT)* and optimization-aware feature selection, allowing the server to continuously train the server-side model by selecting the feature with the highest optimization utility from LFT, rather than waiting for new features uploaded by clients.
- Extensive experiments demonstrate that *StreamSplit* improves training efficiency and scalability over existing SFL methods under heterogeneous environments. It further achieves competitive performance across convolutional and transformer-based models, validating its applicability across diverse model architectures.

2 Related Work

A line of work improves SFL efficiency through *execution-aware scheduling*, aiming to coordinate client-side computation and server-side processing to reduce server idleness under heterogeneous environments. PipeSFL (Gao et al., 2025) prioritizes the server-side processing of features from slower clients, allowing them to receive gradients earlier and resume local updates sooner, thereby reducing the server's waiting time for their subsequent features. Other approaches address this issue by mitigating the feature arrival gaps caused by client heterogeneity. AdaptivePipe (Xie et al., 2026) dynamically adjusts client batch sizes, while Xie et al. (2025) jointly optimize client selection, model splitting, and bandwidth allocation, and Chen et al. (2025c) jointly consider client-server selection and resource allocation. These approaches thereby reduce server waiting time for stragglers. However, all these approaches retain the client–server dependency, with clients and the server still waiting for data from each other to proceed, while server waiting and idleness can persist under highly heterogeneous client environments.

Another line of work reduces client-side waiting by enabling local updates without conventional server-generated gradients. AN-SFL (Han et al., 2021) introduces a lightweight auxiliary network (AN) for local client optimization, while FSL-SAGE (Nair et al., 2025) periodically transfers and retrains AN at the server and selects one with a similar cut-layer gradient to the server model. MU-SplitFed (Liang et al., 2025) instead adopts zeroth-order optimization to avoid conventional backpropagation through the server. These approaches alleviate client-side dependence on real-time server gradients, but they either allow AN to evolve along trajectories that may diverge from the server-side model or still require server feedback based on client features. Moreover, FSL-SAGE requires storing previous client features and passively selects the closest AN, rather than actively aligning it with the evolving server model.

3 Proposed Method

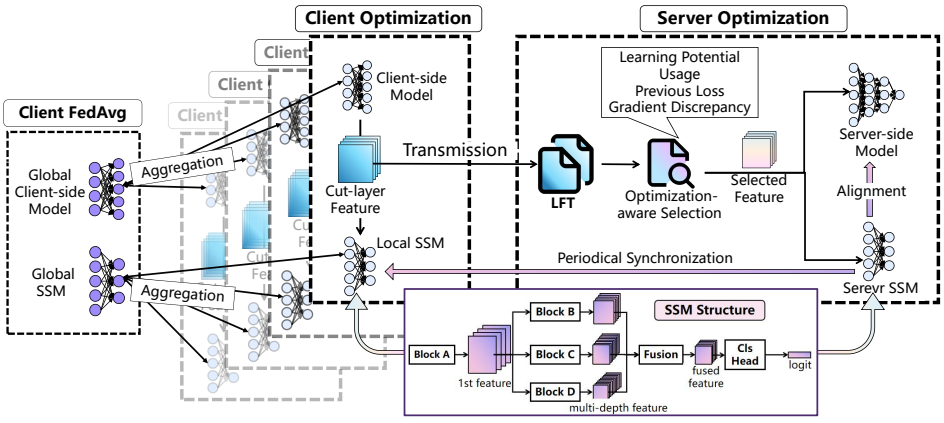


Figure 1: Framework of StreamSplit.

To decouple client-side and server-side optimization while maintaining optimization consistency, we propose StreamSplit, as illustrated in Fig. 1. Specifically, the full model is partitioned into a client-side model and a server-side model, which are deployed on each client and the server, respectively. Each client additionally maintains a local Shadow Server-side Model (SSM), which is jointly optimized with the client-side model, while the server maintains a server SSM alongside the server-side model for server-side optimization and multi-depth alignment. All SSMs adopt the same depth-decoupled architecture. During each local training iteration, the client-side model generates a cut-layer feature and uploads it to the server's LFT. At each server training iteration, an optimization-aware selection is performed over the available features in the LFT, and the selected feature is jointly used for training the server-side model and server SSM, while the server SSM further performs multi-depth alignment with the server-side model. In addition, the server SSM periodically synchronizes with the local SSMs, while the client-side models and local SSMs are peri-

Algorithm 1: StreamSplit

Input: Client datasets $\{\mathcal{D}_i\}_{i=1}^N$, client-side models $\{\theta_c^i\}_{i=1}^N$, local SSMs $\{\theta_{\text{SSM}}^i\}_{i=1}^N$, server-side model θ_s, server-side SSM $\theta_{\text{SSM}}^{\text{server}}$, Latest Feature Table $\mathcal{T}$, maximum server iterations $T_{\max}$

Initialize $\mathcal{T} \leftarrow \emptyset$ and server iteration $t \leftarrow 0$;

while $t < T_{\max}$ **do**

  Client-Side Optimization

  foreach *client* i *in parallel* **do**

    Sample a mini-batch $(\mathbf{x}, \mathbf{y})$ from $\mathcal{D}_i$;

    Generate the cut-layer feature $\mathbf{x}_f$ with θ_c^i according to Eq. equation 5;

    Upload $(\mathbf{x}_f, \mathbf{y})$ and the associated training metadata to the server;

    Compute local SSM task loss and cut-layer gradient according to Eq. equation 6;

    Update the θ_c^i and θ_{SSM}^i according to Eqs. equation 7– equation 8;

    if *the local update iteration reaches the synchronization interval* K **then**

      Synchronize θ_{SSM}^i with the server-side SSM $\theta_{\text{SSM}}^{\text{server}}$ according to Eq. equation 9;

    if *the local update iteration reaches the aggregation interval* I **then**

      Aggregate client-side models via FedAvg: $\theta_c \leftarrow \text{FedAvg}(\{\theta_c^i\}_{i=1}^N)$;

  Server-Side Optimization

  Receive incoming client features asynchronously;

  if *a new feature* f_i *from client* i *is received* **then**

    Update $\mathcal{T}$ according to Eq. equation 11;

    Initialize the metadata of f_i;

  if $\mathcal{T} \neq \emptyset$ **then**

    Evaluate the utility of every feature in $\mathcal{T}$ according to Eqs. equation 13– equation 16;

    Select feature f_t^* according to Eq. equation 12;

    Optimize server-side model with f_t^* according to Eq. equation 17;

    Optimize and align server-side SSM with f_t^* according to Eqs. equation 2– equation 4;

    Update usage, server-side loss, gradient inconsistency of f_t^* in $\mathcal{T}$;

    Increment the server iteration:$t \leftarrow t + 1$;

odically aggregated by the fed server using FedAvg. The complete training procedure is summarized in Algorithm 1. Theoretical analysis of StreamSplit, including the surrogate gradient approximation, SSM architecture and alignment, and optimization-aware feature selection, is provided in Appendix.

3.1 Shadow Server-side Model (SSM)

The SSM serves as a lightweight surrogate of the evolving server-side model, providing surrogate gradients for continuous client-side optimization. SSM is deployed to maintain consistency between client-side and server-side optimization by ensuring that the gradients used to update the client-side model approximate those that would otherwise be propagated from the server-side model, while the client-side model is encouraged to generate cut-layer features that remain suitable for further deep semantic extraction by the server-side model, rather than features optimized solely for immediate prediction by AN in previous works. To balance learning capacity and computation overhead, we design the SSM with a lightweight depth-decoupled architecture and continuously align it with the evolving server-side model through a server-guided alignment mechanism.

3.1.1 SSM Architecture

To approximate the evolving server-side model, the SSM needs to capture representations at different depths and generate a cut-layer gradient close to that of the server-side model. However, given the limited client-side computation budget, directly reproducing the full sequential architecture of the server-side model is impractical. Moreover, in a sequential design, alignment errors at different depths are propagated toward the cut layer through successive transformations, causing errors from deep depths to accumulate or amplify. We therefore adopt an identical initial transformation followed by depth-decoupled branches, enabling independent alignment of multi-depth features while

avoiding the sequential propagation of alignment errors across depths. The detailed SSM architecture is shown in the lower block beneath the StreamSplit framework in Fig. 1.

Specifically, the SSM consists of four convolutional blocks, denoted as Blocks A–D, followed by a fusion block and a classification head. Each block contains less than two convolutional layers with batch normalization and nonlinear activation. Block A serves as the initial transformation of the cut-layer feature, with its convolutional layer identical to the corresponding layer of the server-side model. This correspondence facilitates feature alignment at the first layer. The output of Block A is shared by three depth-decoupled branches, Blocks B–D, which approximate the *shallow*, *medium*, and *deep* features of the server-side model, respectively. Different convolutional kernels are used to obtain different feature depths. Their outputs are then fused into a unified representation and passed to the classification head to produce the prediction logits. Formally, the SSM is defined as

$$\begin{aligned} \mathbf{h}^A = F_A(\mathbf{x}_f), \qquad \mathbf{h}^k = F_k(\mathbf{h}^A), k \in B, C, D, \\ \mathbf{h}^{\text{fuse}} = F_{\text{fuse}}\left(\mathbf{h}^B, \mathbf{h}^C, \mathbf{h}^D\right), \qquad \mathbf{z}_{\text{SSM}} = F_{\text{cls}}\left(\mathbf{h}^{\text{fuse}}\right), \end{aligned} \tag{1}$$

where $\mathbf{x}_f$ is the cut-layer feature, $\mathbf{h}^A$ is the output of block A, and $\mathbf{h}^B$, $\mathbf{h}^C$, and $\mathbf{h}^D$ are the parallel multi-depth representations. F_{fuse} and F_{cls} denote the fusion block and classification head.

3.1.2 SSM Alignment

To align the SSM with the evolving server-side optimization, we employ two alignment strategies. *1st-layer alignment* aligns the first-layer feature and prediction logits, encouraging the SSM to produce a surrogate cut-layer gradient closer to that of the server-side model, while *hierarchical alignment* aligns the SSM's multi-depth features with the corresponding features of the server-side model, providing independent supervision at different depths without accumulating alignment errors. The two strategies are alternated during SSM optimization.

The 1st-layer alignment matches the first-layer feature and prediction logits of the SSM with those of the server-side model. Let $\mathbf{h}_s^{\text{1st}}$ and $\mathbf{z}_s$ denote the corresponding server-side feature and logits. The alignment loss is

$$\mathcal{L}_{\text{align}}^{\text{1st}} = \left\|\mathbf{h}^A - \mathbf{h}_s^{\text{1st}}\right\|_2^2 + \left\|\mathbf{z}_{\text{SSM}} - \mathbf{z}_s\right\|_2^2. \tag{2}$$

The hierarchical alignment matches the outputs of Blocks B–D with the corresponding *shallow*, *medium*, and *deep* features during the server-side model forward propagation. Let $\mathbf{h}_s^{\text{shallow}}$, $\mathbf{h}_s^{\text{medium}}$, and $\mathbf{h}_s^{\text{deep}}$ denote these features. The hierarchical alignment loss is

$$\mathcal{L}_{\text{align}}^{\text{hier}} = \left\|\mathbf{h}^B - \mathbf{h}_s^{\text{shallow}}\right\|_2^2 + \left\|\mathbf{h}^C - \mathbf{h}_s^{\text{medium}}\right\|_2^2 + \left\|\mathbf{h}^D - \mathbf{h}_s^{\text{deep}}\right\|_2^2. \tag{3}$$

For each SSM update, $\mathcal{L}_{\text{align}}$ is instantiated as either $\mathcal{L}_{\text{align}}^{\text{1st}}$ or $\mathcal{L}_{\text{align}}^{\text{hier}}$, with the two strategies alternated throughout optimization. The server-side SSM is optimized as

$$\mathcal{L}_{\text{SSM}}^{\text{server}} = \mathcal{L}_{\text{task}} + \lambda\mathcal{L}_{\text{align}}, \mathcal{L}_{\text{align}} \in \left\{\mathcal{L}_{\text{align}}^{\text{1st}}, \mathcal{L}_{\text{align}}^{\text{hier}}\right\}, \qquad \boldsymbol{\theta}_{\text{SSM}}^{\text{server}} \leftarrow \boldsymbol{\theta}_{\text{SSM}}^{\text{server}} - \eta_{\text{SSM}} \nabla_{\boldsymbol{\theta}_{\text{SSM}}^{\text{server}}} \mathcal{L}_{\text{SSM}}^{\text{server}}, \tag{4}$$

where η_{SSM} is the learning rate of the server-side SSM. Since the server-side model and SSM are optimized on the same selected feature, this alignment actively guides the server-side SSM to follow the evolving server-side model optimization behavior.

3.1.3 Client-side optimization with local SSM

To remove the real-time dependency of client-side optimization on server-side gradients, each client maintains a local copy of the SSM as a surrogate of the server-side model, allowing continuous client-side optimization. For a mini-batch $(\mathbf{x}, \mathbf{y})$, the client-side model generates cut-layer feature:

$$\mathbf{x}_f = F_c(\mathbf{x}; \boldsymbol{\theta}_c). \tag{5}$$

The feature $\mathbf{x}_f$ is transmitted to the server, as well as fed to the local SSM. Local SSM computes task loss $\mathcal{L}_{\text{task}}^{\text{local}}$ and performs backward propagation for the surrogate gradient $\mathbf{g}_f^{\text{SSM}}$ at the cut layer:

$$\mathcal{L}_{\text{task}}^{\text{local}} = \ell\left(F_{\text{SSM}}\left(\mathbf{x}_f; \boldsymbol{\theta}_{\text{SSM}}^{\text{local}}\right), \mathbf{y}\right), \quad \mathbf{g}_f^{\text{SSM}} = \nabla_{\mathbf{x}_f} \mathcal{L}_{\text{task}}^{\text{local}}. \tag{6}$$

The surrogate gradient is then propagated through the client-side model to update its parameters:

$$\boldsymbol{\theta}_c \leftarrow \boldsymbol{\theta}_c - \eta_c \frac{\partial \mathbf{x}_f}{\partial \boldsymbol{\theta}_c} \mathbf{g}_f^{\text{SSM}}. \tag{7}$$

Meanwhile, the local SSM is updated using the same task loss:

$$\boldsymbol{\theta}_{\text{SSM}}^{\text{local}} \leftarrow \boldsymbol{\theta}_{\text{SSM}}^{\text{local}} - \eta_{\text{SSM}} \nabla_{\boldsymbol{\theta}_{\text{SSM}}^{\text{local}}} \mathcal{L}_{\text{task}}^{\text{local}}. \tag{8}$$

Thus, the client-side model and the local SSM are jointly updated at each local iteration.

3.1.4 PERIODIC SSM SYNCHRONIZATION

Prolonged independent updates may cause the local SSM to deviate from the evolving optimization trajectory of the server-side model. We therefore periodically synchronize each local SSM with the latest server-side SSM, which is jointly optimized with the server-side model as its optimization proxy. This synchronization allows local SSMs to retain their local adaptations while following the latest server-side optimization state, thereby maintaining optimization consistency as the server-side model evolves. After every K local iterations, the local SSM parameters are updated as

$$\boldsymbol{\theta}_{\text{SSM}}^{\text{local}} \leftarrow (1 - \alpha_{SSM})\boldsymbol{\theta}_{\text{SSM}}^{\text{local}} + \alpha_{SSM}\boldsymbol{\theta}_{\text{SSM}}^{\text{server}}, \tag{9}$$

where K denotes the synchronization interval and α_{SSM} controls the contribution of the latest server-side SSM. Since the SSM is lightweight and synchronization is performed only periodically, the additional communication overhead is limited and does not introduce a real-time dependency on server-side gradients.

3.2 CONTINUOUS SERVER-SIDE OPTIMIZATON

Under asynchronous training, due to differences in computing speeds, the server may become idle when it waits for newly generated client features to arrive. StreamSplit eliminates this dependency by maintaining the latest available feature from each client in a *Latest Feature Table (LFT)* and continuously evaluating and selecting valuable features from this pool for training.

3.2.1 LATEST FEATURE TABLE (LFT)

To enable continuous server-side optimization without waiting for newly generated client features, StreamSplit maintains a *Latest Feature Table (LFT)* for server-side training, denoted by $\mathcal{T}_t$. The key idea is to retain only the latest available cut-layer feature from each client, such that the LFT contains one feature per client and its size is bounded by the number of clients. This converts the asynchronously arriving feature stream into a bounded pool of up-to-date client features, allowing the server to continuously select features for training while avoiding the accumulation of outdated features. Each entry in the LFT is defined as

$$f_i = (\mathbf{x}_f^i, y_f^i, m_f^i), \tag{10}$$

where $\mathbf{x}_f^i$ and y_f^i denote the feature and its corresponding label, respectively, and m_f^i records the associated metadata, including the feature generation iteration, server-side loss, usage count, and gradient discrepancy. When a new feature from client i arrives, it replaces the previous entry:

$$\mathcal{T}_{t+1} = \mathcal{T}_t \cup f_i^{\text{new}} \setminus f_i^{\text{old}}. \tag{11}$$

By retaining only one feature from each client, the LFT keeps the feature-selection space proportional to the number of clients and bounds the storage of stale representations. The server can therefore continuously perform optimization over the available feature pool rather than waiting for the latest feature from any particular client. Although the LFT retains only the latest feature from each client, the server processes features substantially faster than clients generate and transmit new ones, allowing each feature to be processed at least once before replacement.

3.2.2 OPTIMIZATION-AWARE FEATURE SELECTION

The latest available features in LFT do not necessarily have the same optimization value for the server-side model, as they are generated at different stages of local training and may have been selected by the server for previous updates. To address this, StreamSplit introduces an *optimization-aware feature evaluation and selection strategy* that evaluates the optimization value of available features and selects the feature most beneficial to the current server-side update, while also considering its potential to improve consistency between the server-side model and the SSM:

$$f_t^* = \arg\max_{f \in \mathcal{T}_t} U_t(f), \tag{12}$$

where f_t^* denotes the feature selected for server-side optimization at time step t, and $U_t(f)$ denotes its optimization utility. We define the utility as

$$U_t(f) = \alpha P_t(f) + \beta H_t(f) + \gamma G_t(f), \tag{13}$$

where $P_t(f)$ measures the learning potential of the feature, $H_t(f)$ captures its usage history, and $G_t(f)$ measures the *gradient discrepancy* between the server-side model and the SSM under the same feature. The coefficients α, β, and γ control the relative importance of the three factors.

Learning potential. The learning potential of a feature reflects both the training maturity of the client representation and its remaining optimization potential under the current server-side model. A feature generated after more local updates comes from a more effectively trained client-side model and is therefore expected to provide a more useful representation for server-side optimization. Meanwhile, a larger server-side loss when the feature was most recently used indicates that the current server-side model has not yet sufficiently optimized this representation, leaving greater potential for further improvement. We therefore combine these two signals as

$$P_t(f) = \lambda_p I_t(f) + (1 - \lambda_p) L_t(f), \tag{14}$$

where $I_t(f)$ denotes the normalized client-side training iteration at which feature f was generated, and $L_t(f)$ denotes the loss incurred when f was most recently used by the server-side model. A larger $P_t(f)$ indicates greater potential benefit for the next server-side optimization step.

Usage regularization. Since each feature remains available in the LFT until a newer representation from the same client arrives, a feature can be selected repeatedly across multiple server-side optimization steps. Without regularization, this may cause the server to repeatedly rely on a small subset of client representations, especially when heterogeneous client training rates make the learning-potential term favor features generated by faster clients. We therefore introduce a usage-based regularization term to encourage more balanced feature selection:

$$H_t(f) = \frac{1}{1 + n_t(f)}, \tag{15}$$

where $n_t(f)$ denotes the number of times feature f has been selected for server-side optimization. Thus, frequently used features receive lower utility, while underutilized features receive higher utility, promoting more balanced utilization across features.

Gradient discrepancy. The gradient discrepancy measures the mismatch between the server-side model and the SSM at the cut layer for a given feature. Since the SSM is intended to approximate the server-side optimization, a larger discrepancy indicates that its cut-layer gradient is less consistent with that of the server-side model. We therefore define

$$G_t(f) = 1 - \cos\left(\mathbf{g}_s^t(f), \mathbf{g}_{\mathrm{SSM}}^t(f)\right), \tag{16}$$

where $\mathbf{g}_s^t(f)$ and $\mathbf{g}_{\mathrm{SSM}}^t(f)$ denote the gradients with respect to the cut-layer feature f generated by the server-side model and the SSM, respectively. A larger $G_t(f)$ indicates a larger cut-layer gradient mismatch and assigns higher priority to the feature, as it provides a stronger signal for improving the alignment between the SSM and the server-side model.

3.2.3 Server-side Optimization

Once a feature f_t^* is selected, the server uses it to optimize the server-side model. Specifically, the server performs forward and backward propagation on f_t^* and its corresponding label:

$$\mathcal{L}_{\mathrm{task}}^s(f_t^*) = \ell\left(F_s(f_t^*; \boldsymbol{\theta}_s), y_t^*\right), \quad \boldsymbol{\theta}_s \leftarrow \boldsymbol{\theta}_s - \eta_s \nabla_{\boldsymbol{\theta}_s} \mathcal{L}_{\mathrm{task}}^s(f_t^*), \tag{17}$$

where $\boldsymbol{\theta}_s$ denotes the server-side model parameters, η_s is the server-side learning rate, and y_t^* is the label associated with f_t^*. The backward propagation also provides the cut-layer gradient $\mathbf{g}_s^t(f_t^*)$, which is used to evaluate the gradient discrepancy with the SSM. The same selected feature is then used to optimize the server-side SSM following the alignment procedure described in Section 3.1.2. By jointly optimizing the server-side model and SSM on the same feature, the SSM continuously follows the evolving server-side model and improves the consistency of its surrogate gradients.

After each optimization step, the corresponding LFT metadata is updated, including the usage count, most recent server-side loss, and gradient discrepancy. These updated statistics are then used for subsequent optimization utility evaluation and feature selection. Thus, the LFT continuously feeds selected available features into both server-side model optimization and SSM alignment without requiring the server to wait for newly generated client features.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and Datasets. We evaluate StreamSplit using ResNet18 (He et al., 2016) on CIFAR100 (Krizhevsky, 2009) and ViT-Tiny (Dosovitskiy et al., 2021) on Tiny-ImageNet (Le & Yang, 2015). For ResNet18, the model is partitioned after the first residual block in Layer1, while for ViT-Tiny, the first three transformer blocks are deployed on the client and the remaining nine blocks are deployed on the server. All datasets are partitioned among 40 clients using a Dirichlet non-IID distribution (Zhao et al., 2018) with concentration parameter $\alpha = 0.7$. The SSM follows the parallel convolutional or transformer-block architecture described in last section. Local SSMs are synchronized with the server-side SSM every 100 local iterations.

Baselines. We compare StreamSplit with four representative SFL methods. All baselines are implemented following their original papers. Para-SFL (Thapa et al., 2020) follows the standard SFL pipeline. Pipe-SFL (Gao et al., 2025) extends Para-SFL with scheduling by prioritizing features from longer-waiting clients for server-side training. SFL with Auxiliary Network (AN-SFL) (Han et al., 2021) uses an AN for client-side training. FSL-SAGE (Nair et al., 2025) extends AN-SFL by selecting the AN whose gradients are more similar to those of the server-side model.

Client Heterogeneity. We consider three client groups with different levels of heterogeneity, denoted as Cluster-1, Cluster-2, and Cluster-3. Cluster-1 represents a homogeneous setting, where all clients have the same computation and feature transmission speeds. Cluster-2 represents a moderately heterogeneous setting with four client types, each containing 10 clients. The computation and feature transmission speeds of the four types follow a ratio of $1{:}2{:}3{:}4$, resulting in corresponding feature arrival intervals of $1{:}2{:}3{:}4$. Cluster-3 follows the same setting with four equally sized client types, but uses a more heterogeneous speed ratio of $1{:}2{:}4{:}8$, and a feature arrival intervals of $1{:}2{:}4{:}8$.

Detailed experimental configurations are provided in the Appendix. The Appendix further presents additional experiments on scalability, SSM architecture design, and local SSM synchronization frequency. In addition, we include the complete accuracy–time and loss–time curves for the SSM alignment and feature selection studies to provide a more comprehensive analysis of their training dynamics.

4.2 TRAINING EFFICIENCY UNDER CLIENT HETEROGENEITY

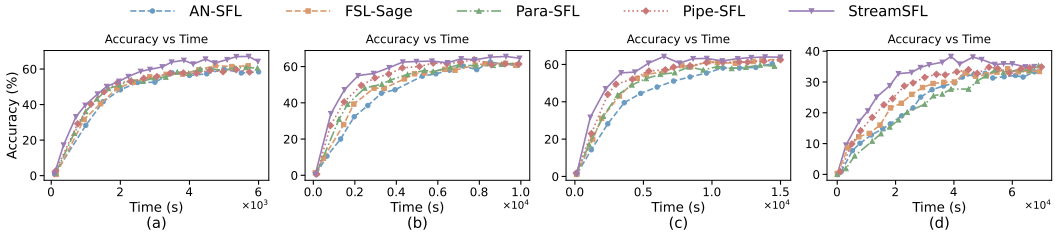


Figure 2: Training accuracy over time across different model–dataset pairs and levels of client heterogeneity. (a)–(c) show the results for ResNet-18 on CIFAR-100 under Cluster-1, Cluster-2, and Cluster-3, respectively; (d) shows the results for ViT-Tiny on Tiny-ImageNet under Cluster-2.

We first evaluate StreamSplit under different levels of client heterogeneity. We train ResNet18 on CIFAR-100 under Cluster-1, Cluster-2, and Cluster-3, and ViT-Tiny on Tiny-ImageNet under Cluster-2, with all other settings following the previous subsection. Figure 2 shows the test accuracy as training progresses. StreamSplit consistently converges faster than the baselines across different models and heterogeneity levels. Under the most heterogeneous Cluster-3 setting, for example, existing methods achieve approximately 47%–56% accuracy at 6000 s, whereas StreamSplit reaches 62%. This improvement comes from decoupling client-side and server-side optimization, which allows both sides to proceed asynchronously and reduces execution stalls in conventional SFL. Overall, these results demonstrate that StreamSplit consistently improves training efficiency and accelerates convergence across different levels of client heterogeneity, while the consistent gains

Table 1: Accuracy and gradient similarity under different SSM alignment strategies.

Accuracy and Grad. Sim. @ 2000 s		
Method	**Accuracy(%)**	**Grad Sim**
StreamSplit	52.80	0.2193
Logit	45.30	0.2021
Cut-layer	44.10	0.1958
Multi-depth	47.00	0.2165
Cut-layer+Logit	48.50	0.2122

Table 2: Accuracy and loss under different feature selection strategies.

Accuracy and Loss @ 3000 s		
Method	**Accuracy(%)**	**Loss**
StreamSplit	58.60	2.1423
FIFO	56.70	2.2768
Random	57.30	2.1627
Freshness	58.10	2.2268

on both ResNet18 and ViT-Tiny further show that the proposed decoupled execution mechanism is effective for both convolutional and transformer-based models.

4.3 Effectiveness of SSM Alignment

We further evaluate the effectiveness of the SSM alignment strategy by comparing it with four alternatives: 1st-layer alignment, logit alignment, cut-layer alignment, and multi-depth alignment. We use ResNet18 on CIFAR-100 under Cluster-2 with 40 clients, with other settings following the previous subsection. Each strategy is evaluated by test accuracy and the cosine similarity between the cut-layer gradients of the SSM and the server-side model. As shown in Table 1, StreamSplit achieves the highest accuracy (52.80%) and gradient similarity (0.2193) at 2000 s, outperforming all alternative strategies. These results demonstrate that our alignment strategy more effectively aligns the behavior of the SSM with that of the server-side model, particularly for the cut-layer gradients that directly guide client-side model updates. By better aligning the SSM with server-side model with our alignment strategy, StreamSplit maintains a more consistent optimization direction between the client-side and server-side models during decoupled asynchronous training, thereby preserving training performance.

4.4 Effectiveness of Feature Selection Strategy

We evaluate the server-side feature selection strategy by comparing our optimization-aware selection with FIFO, Freshness, and Random. We use ResNet18 on CIFAR-100 under Cluster-2, with other settings following the previous subsection. We evaluate server-side loss and test accuracy at 3000 s. As shown in Table 2, StreamSplit achieves the lowest loss (2.1423) and highest accuracy (58.60%), outperforming all three alternatives. This result shows that optimization-aware feature evaluate and selection strategy can identify features that are more beneficial to the current server-side optimization, improving training performance.

5 Conclusion

In this paper, we presented *StreamSplit*, a decoupled SFL framework that enables asynchronous and continuous optimization on clients and the server. By removing the bidirectional waiting dependency, StreamSplit alleviates execution stalls on clients and server. In StreamSplit. a lightweight Shadow Server-side Model (SSM) provides surrogate optimization guidance for local client updates, while the Latest Feature Table (LFT) enables continuous server-side optimization through optimization-aware feature selection. Extensive experiments on convolutional and transformer-based models demonstrate that StreamSplit improves training efficiency under heterogeneous environments while maintaining competitive accuracy and scalability. Overall, our results show that the proposed StreamSplit can improve SFL efficiency while preserving consistent optimization behavior. Future work will explore SSM performance across different split points and client-side model scales.

REFERENCES

Shengchao Chen, Guodong Long, Jing Jiang, and Chengqi Zhang. Federated foundation models on heterogeneous time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 15839–15847, 2025a. doi: 10.1609/AAAI.V39I15.33739.

Xing Chen, Jingtao Li, Deliang Fan, and Chaitali Chakrabarti. Heterosfl: Split federated learning with heterogeneous clients and non-iid data. *IEEE Internet of Things Journal*, 12(15):30460–30474, 2025b. doi: 10.1109/JIOT.2025.3572393.

Yishan Chen, Xiangwei Zeng, Xiansong Luo, and Zhiquan Liu. Joint client–server selection and resource allocation based on split federated learning in edge-to-cloud computing environments. *Computer Networks*, 270:111502, 2025c. doi: 10.1016/j.comnet.2025.111502.

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.

Yunqi Gao, Bing Hu, Mahdi Boloursaz Mashhadi, Wei Wang, and Mehdi Bennis. Pipesfl: A fine-grained parallelization framework for split federated learning on heterogeneous clients. *IEEE Transactions on Mobile Computing*, 24(3):1774–1791, 2025. doi: 10.1109/TMC.2024.3489642.

Dong-Jun Han, Hira Irfan Bhatti, Jaewoo Lee, and Joonhyuk Moon. Accelerating federated learning with split learning on locally generated losses. In *ICML 2021 Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2021.

Pengchao Han, Chao Huang, Geng Tian, Ming Tang, and Xin Liu. Convergence analysis of split federated learning on heterogeneous data. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 38, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Qing Hu, Tianchi Liao, Shuyi Wu, Lei Yang, and Chuan Chen. Fedlagc: Towards high performance system-heterogeneous federated learning via layer-adaptive submodel extraction and gradient correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 21858–21866, 2026.

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

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Yichen Li, Hang Su, Huifa Li, Haolin Yang, Xinlin Zhuang, Haochen Xue, Haozhao Wang, and Imran Razzak. Fedcd: Towards consolidated distillation for heterogeneous federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23256–23264, 2026. doi: 10.1609/aaai.v40i28.39494.

Dandan Liang, Jianing Zhang, Evan Chen, Zhe Li, Rui Li, and Haibo Yang. Towards straggler-resilient split federated learning: An unbalanced update approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1062.

Yunming Liao, Yang Xu, Hongli Xu, Zhiwei Yao, Liusheng Huang, and Chunming Qiao. Parallelsfl: A novel split federated learning framework tackling heterogeneity issues. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 845–860. ACM, 2024. doi: 10.1145/3636534.3690665.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

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. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, pp. 45507–45528. PMLR, 2025.

Boris Radovic, Marco Canini, Samuel Horvath, Veljko Pejovic, and Praneeth Vepakomma. Towards a unified framework for split learning. In *Proceedings of the 5th Workshop on Machine Learning and Systems (EuroMLSys)*, 2025.

Yuwei Sun and Hideya Ochiai. Bidirectional contrastive split learning for visual question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21602–21609, 2024.

Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, and Seyit Ahmet Camtepe. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence Workshops*, 2020. doi: 10.1609/aaai.v36i8.20825.

Joana Tirana, Dimitra Tsigkari, David Solans Noguero, and Nicolas Kourtellis. Data heterogeneity and forgotten labels in split federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 25940–25948, 2026. doi: 10.1609/aaai.v40i31.39794.

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.

Haoming Wang and Wei Gao. Tackling intertwined data and device heterogeneities in federated learning with unlimited staleness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 21080–21089, 2025. doi: 10.1609/aaai.v39i20.35405.

Lixu Wang, Bingqi Shang, Yi Li, Payal Mohapatra, Wei Dong, Xiao Wang, and Qi Zhu. Split adaptation for pre-trained vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Philipp Wiesner, Ramin Khalili, Dennis Grinwald, Pratik Agrawal, Lauritz Thamsen, and Odej Kao. Fedzero: Leveraging renewable excess energy in federated learning. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems (e-Energy '24)*, pp. 373–385. ACM, 2024. doi: 10.1145/3632775.3639589.

Zeke Xia, Ming Hu, Dengke Yan, Ruixuan Liu, Anran Li, Xiaofei Xie, and Mingsong Chen. Multisfl: Towards accurate split federated learning via multi-model aggregation and knowledge replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 914–922, 2025. doi: 10.1609/aaai.v39i1.32076.

Chunfeng Xie, Zhixiong Chen, Wenqiang Yi, Hyundong Shin, and Arumugam Nallanathan. Tackling class imbalance and client heterogeneity for split federated learning in wireless networks. *IEEE Transactions on Wireless Communications*, 24(6):4920–4936, 2025. doi: 10.1109/TWC.2025.3545236.

Zuan Xie, Yang Xu, Yunming Liao, Junhao Cheng, Jingjing Yang, and Ying Zhu. Optimizing split federated learning through adaptive pipeline parallelism. In *Proceedings of the IEEE INFOCOM 2026 - IEEE Conference on Computer Communications*, pp. 1–10. IEEE, 2026. doi: 10.1109/INFOCOM59046.2026.11571502.

Dengke Yan, Ming Hu, Xiaofei Xie, Yanxin Yang, and Mingsong Chen. $S^2$2fl: Toward efficient and accurate heterogeneous split federated learning. *IEEE Transactions on Computers*, 75(1):320–334, 2026.

Jiarong Yang and Yuan Liu. Gas: Generative activation-aided asynchronous split federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 21956–21964, 2025. doi: 10.1609/aaai.v39i20.35503.

Zhiyuan Yao et al. Towards privacy-preserving split learning for controlnet. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2025.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. In *arXiv preprint arXiv:1806.00582*, 2018.

AI Assistance Disclosure

We used generative AI tools to assist with improving the readability and clarity of the manuscript, as well as for literature retrieval and discovery during the research process. All AI-assisted content was carefully reviewed and verified by the authors. The authors take full responsibility for the final content of this submission, including all claims, analyses, and results.

Ethics Statement

This work does not involve human subjects, personally identifiable information, or sensitive personal data. All experiments are conducted on publicly available benchmark datasets for research purposes. We do not identify any additional ethical concerns associated with the methodology or experimental setup.

A Appendix

A.1 Additional Experiments

A.1.1 Additional Implementation Details

Experimental Environment. We provide additional implementation details omitted from the main paper for reproducibility. All experiments are conducted by simulating the split federated learning (SFL) environment, where multiple clients and one server are instantiated on a multi-GPU computing platform. The simulation platform is equipped with two Intel Xeon Gold 5218R CPUs (40 cores in total), 256 GB RAM, and four NVIDIA RTX 4090 GPUs. The experiments are implemented with PyTorch 2.4 and CUDA 12.1.

Training and Optimization Settings. For ResNet18 and ViT-Tiny, we use batch sizes of 32 and 128, respectively. For ResNet18, the server-side SSM is optimized using Adam with a learning rate of 0.001, while ViT-Tiny adopts AdamW with a learning rate of 1×10^{-4} and weight decay of 0.05. The federated aggregation interval is set to 500 and 1,000 local iterations for ResNet18 and ViT-Tiny, respectively. For feature selection strategy, we set $\alpha_f = 1.0$, $\beta = 0.2$, and $\gamma = 0.7$, and initialize $\lambda_p = 0.001$, with these values determined based on empirical experience and preliminary experiments. The learning-potential coefficient is updated as

$$\lambda_p(t) = 0.001 \left(1 - 0.2 \left\lfloor \frac{t}{2000} \right\rfloor\right), \tag{18}$$

where t denotes the current server iteration. This coefficient is gradually decayed to mitigate the increasing discrepancy in local update counts across heterogeneous clients, thereby preventing the feature selection strategy from becoming overly biased toward clients with faster local progress as training proceeds.

SSM Architecture Details. We use architecture-specific SSMs for ResNet18 and ViT-Tiny. For ResNet18, each SSM block consists of two 3×3 convolutional layers, each followed by batch normalization and ReLU activation. The first block (A) follows the corresponding shallow structure of the server-side model, while blocks B, C, and D are parallel branches that generate shallow, medium, and deep features, respectively. Their first convolutional layers use strides of 2, 4, and 8 to match the spatial resolutions of the corresponding server-side layers, while the second convolutional layers further refine the features. For ViT-Tiny, the outputs of the 3rd, 6th, and 9th Transformer blocks of server-side model are treated as the shallow, medium, and deep features, respectively. The SSM first transforms the client-side feature through block A, and then uses three parallel Transformer blocks (B, C, and D) to approximate the three corresponding depth features. Each Transformer block uses 3-head self-attention, an MLP with a $4\times$ expansion ratio and GELU activation,

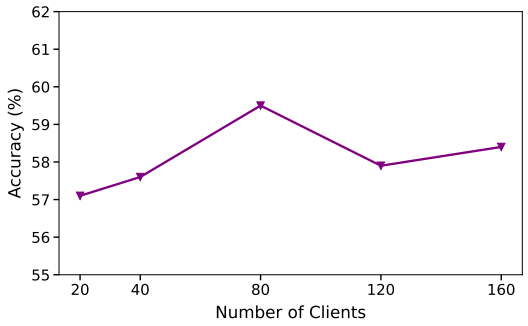


Figure 3: Scalability of StreamSplit with increasing numbers of clients.

layer normalization, and residual connections. The three representations are then fused for the final prediction.

SSM Overhead. To quantify the client-side overhead introduced by the SSM, we measure its additional computation, parameter size, and memory consumption. For ResNet18, the SSM introduces approximately 221.8M FLOPs in a forward pass, corresponding to a 53.0% increase over the 418.8M FLOPs of the client-side model. For ViT-Tiny, the SSM requires approximately 67.6M FLOPs, corresponding to a 66.7% increase over the 101.4M FLOPs of the client-side model.

A.1.2 Scalability with Increasing Clients

We next evaluate the scalability of StreamSplit with different numbers of clients. We conduct experiments using ResNet18 on CIFAR-100 under Cluster-2, where clients are evenly distributed across four groups with relative delays of $1\times$, $2\times$, $3\times$, and $4\times$, respectively. The number of clients is varied from 20 to 160 while maintaining the same client heterogeneity distribution. Other experimental settings are kept consistent with those described in the previous subsection.

As shown in Figure 3, StreamSplit maintains stable performance as the number of clients increases. At 3000 s, the test accuracy remains within a narrow range of 57.10%–59.50% across 20, 40, 80, 120, and 160 clients. This indicates that the performance of StreamSplit is robust to changes in the client population and demonstrates its scalability to larger-scale client environments.

A.1.3 Effect of SSM Synchronization Frequency

We further evaluate the impact of the synchronization frequency between the local SSM and the server-side SSM. We vary the synchronization interval across 20, 50, 100, 200, and 500 local client iterations. We use ResNet18 on CIFAR-100 while other experimental settings remain consistent with those described in the previous subsection.

As shown in Figure 4, a synchronization interval of 100 iterations achieves slightly better performance than the other settings under the same communication cost, while overly frequent synchronization of 20-iteration performs the worst. For example, at a communication cost of 500,000 MB, the 100-iteration setting reaches 48.10% accuracy, compared with 40.10% for the 20-iteration setting, while the other intervals achieve accuracies within 40%–48%. These results indicate that overly frequent synchronization interrupts the continuous learning of the local SSM, causing its optimization direction to be disturbed before it can sufficiently improve its modeling capability, while also introducing additional communication overhead. An appropriate synchronization interval allows the local SSM to learn more effectively while maintaining consistency with the server-side SSM, thereby improving overall training performance.

A.1.4 Effect of SSM Architecture and Size

We further investigate the impact of the SSM architecture and model size on training performance. We compare the parallel SSM architecture described in last section with a sequential architecture, where Blocks A–D are arranged sequentially and each block takes the output of the preceding block

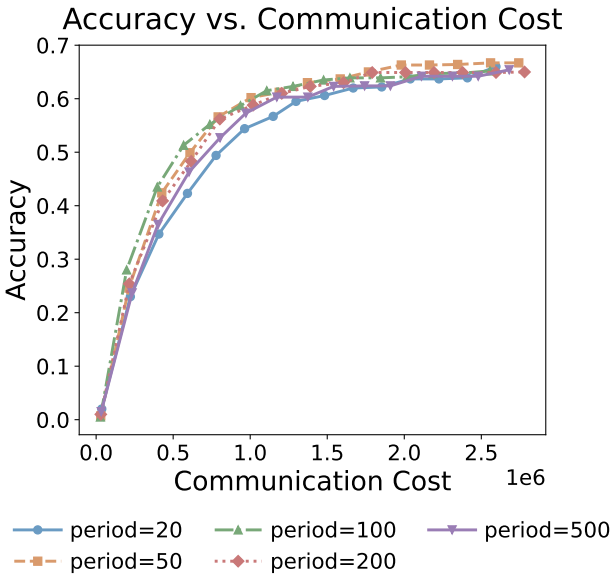


Figure 4: Impact of different SSM synchronization intervals on performance.

Table 3: Accuracy at 4000 s under different SSM structures.

SSM Structure	Small	Medium	Large
Sequential	59.20%	60.40%	56.50%
Parallel	61.30%	62.30%	59.60%

as its input. For each architecture, we consider three SSM sizes, namely small, medium, and large, by varying the hidden dimension while keeping the internal structure of each block unchanged. Other experimental settings remain consistent with those described in the previous subsection.

As shown in Table 3, the parallel SSM consistently achieves higher accuracy than the sequential SSM across all three model sizes. At 4000 s, the parallel architecture outperforms the sequential architecture by 2.10, 1.90, and 3.10 percentage points for the small, medium, and large SSMs, respectively. This indicates that the parallel architecture, together with the proposed alignment strategy, can more effectively capture the behavior of the server-side model and maintain optimization consistency. We also observe that larger SSMs generally provide greater capacity to approximate the server-side optimization behavior and guide client-side updates, while we adopt the medium-sized SSM as a balance between modeling capacity and client-side computational overhead.

A.1.5 Additional Results of SSM Alignment

Figure 5 provides additional results of different SSM alignment strategies during training. Besides the final accuracy and gradient similarity reported in Table 1, we visualize their evolution over training time. The results show that StreamSplit consistently achieves faster accuracy improvement compared with other alignment strategies. This indicates that proposed server-guided alignment provides a more effective approximation of the server-side optimization direction, thereby improving optimization consistency between the client-side and server-side models.

A.1.6 Additional Results of Feature Selection Strategy

Figure 6 further illustrates the training dynamics of different feature selection strategies. Compared with FIFO, Freshness, and Random selection, StreamSplit achieves higher accuracy improvement and lower server loss throughout training. These results demonstrate that optimization-aware fea-

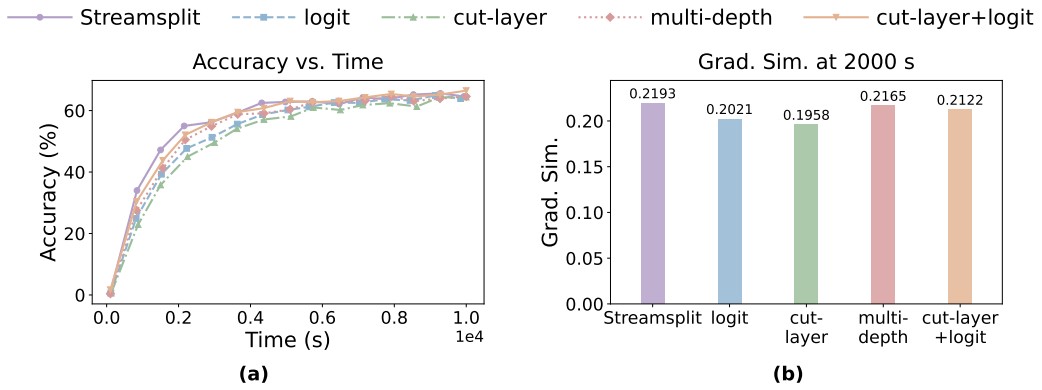


Figure 5: Effect of different SSM alignment strategies on training performance. (a) Accuracy convergence over time. (b) Cut-layer gradient similarity comparison at 2000 s.

ture selection can prioritize features that are more beneficial to the current server-side optimization, leading to more efficient utilization of server computation and improved convergence performance.

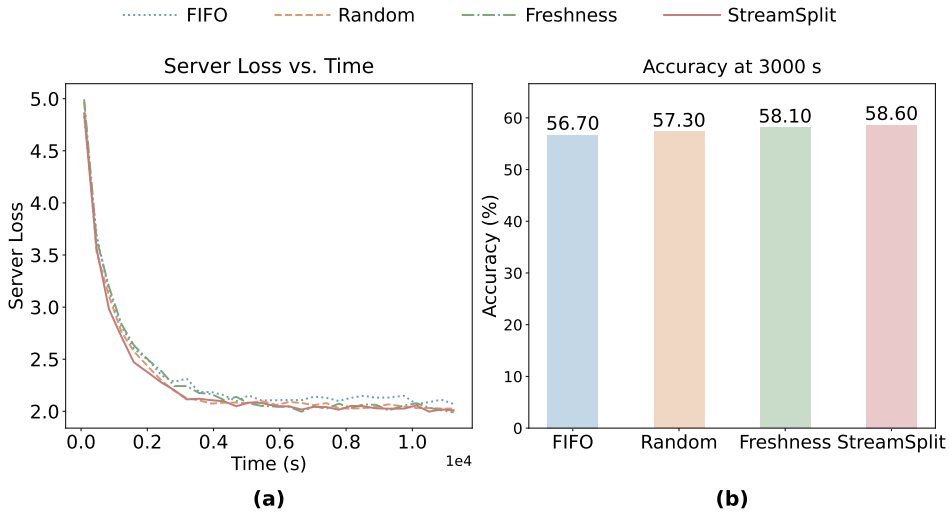


Figure 6: Comparison of different feature selection strategies. (a) Server loss convergence and (b) test accuracy at 3000 s.

A.2 Theoretical Analysis

The key objective of the SSM is to provide a cut-layer optimization signal that approximates the gradient produced by the evolving server-side model, while its parallel architecture is designed to avoid the accumulation of approximation errors across different feature depths. We therefore analyze two properties of StreamSplit: (i) the approximation error between the SSM surrogate gradient and the server-side gradient, and (ii) the effect of the parallel architecture on cross-depth error propagation. The analysis first establishes the assumptions and preliminary results required for these two properties.

A.2.1 ASSUMPTIONS

We first state the regularity conditions used in the analysis. These conditions characterize the local smoothness of the server-side model and SSM and the relationship between feature alignment and local Jacobian variation.

Assumption 1 (Differentiability and Bounded Jacobians). The server-side model and the SSM are differentiable with respect to their input feature $\mathbf{x}_f$. Moreover, their Jacobians with respect to the input and intermediate features are bounded in the neighborhood considered in the analysis. In particular, there exist finite constants M_s and M_{SSM} such that

$$\|\mathbf{J}_s\|_2 \le M_s, \qquad \|\mathbf{J}_{\mathrm{SSM}}\|_2 \le M_{\mathrm{SSM}}. \tag{19}$$

Assumption 2 (Locally Lipschitz Loss Gradient). For a fixed label y, the gradient of the loss with respect to the logits is locally Lipschitz. That is, there exists a finite constant L_ℓ such that

$$\|\nabla_{\mathbf{z}}\ell(\mathbf{z}_1, y) - \nabla_{\mathbf{z}}\ell(\mathbf{z}_2, y)\|_2 \le L_\ell \|\mathbf{z}_1 - \mathbf{z}_2\|_2 \tag{20}$$

for any $\mathbf{z}_1$ and $\mathbf{z}_2$ in the neighborhood of interest.

Assumption 3 (Local Jacobian Regularity). The Jacobians of the differentiable transformations involved in the SSM and server-side models are locally Lipschitz continuous with respect to their input features. Specifically, for any two features $\mathbf{h}_1$ and $\mathbf{h}_2$ within the local neighborhood considered in the analysis, there exists a finite constant $L_J > 0$ such that

$$\|\mathbf{J}(\mathbf{h}_1) - \mathbf{J}(\mathbf{h}_2)\|_2 \le L_J \|\mathbf{h}_1 - \mathbf{h}_2\|_2 . \tag{21}$$

Here, $\mathbf{J}(\mathbf{h})$ denotes the Jacobian of the corresponding differentiable transformation evaluated at feature $\mathbf{h}$, and L_J characterizes the local sensitivity of the Jacobian to perturbations in the input feature.

A.2.2 PRELIMINARIES

Let $\mathbf{x}_f$ denote a cut-layer feature and y its corresponding label. The server-side model produces logits

$$\mathbf{z}_s = F_s(\mathbf{x}_f; \boldsymbol{\theta}_s), \tag{22}$$

while the SSM produces

$$\mathbf{z}_{\mathrm{SSM}} = F_{\mathrm{SSM}}(\mathbf{x}_f; \boldsymbol{\theta}_{\mathrm{SSM}}). \tag{23}$$

Their corresponding cut-layer gradients are

$$\mathbf{g}_s^t(f) = \nabla_{\mathbf{x}_f}\ell(\mathbf{z}_s, y), \qquad \mathbf{g}_f^{\mathrm{SSM}} = \nabla_{\mathbf{x}_f}\ell(\mathbf{z}_{\mathrm{SSM}}, y). \tag{24}$$

By the chain rule, define the input-to-logit Jacobians

$$\mathbf{J}_s = \frac{\partial \mathbf{z}_s}{\partial \mathbf{x}_f}, \qquad \mathbf{J}_{\mathrm{SSM}} = \frac{\partial \mathbf{z}_{\mathrm{SSM}}}{\partial \mathbf{x}_f}. \tag{25}$$

Then the two cut-layer gradients can be written as

$$\mathbf{g}_s^t(f) = \mathbf{J}_s^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y), \qquad \mathbf{g}_f^{\mathrm{SSM}} = \mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}}, y). \tag{26}$$

The following decomposition separates the gradient discrepancy into two terms. Subtracting the two expressions in Eq. equation 26 and adding and subtracting $\mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y)$ gives

$$\begin{aligned}
\mathbf{g}_s^t(f) - \mathbf{g}_f^{\mathrm{SSM}} &= \mathbf{J}_s^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) - \mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}}, y) \\
&= \mathbf{J}_s^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) - \mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) \\
&\quad + \mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) - \mathbf{J}_{\mathrm{SSM}}^\top \nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}}, y) \\
&= (\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}})^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) \\
&\quad + \mathbf{J}_{\mathrm{SSM}}^\top \left[\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s, y) - \nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}}, y)\right].
\end{aligned} \tag{27}$$

Applying the triangle inequality and the induced matrix norm inequality

$$\|\mathbf{a} + \mathbf{b}\|_2 \le \|\mathbf{a}\|_2 + \|\mathbf{b}\|_2 , \qquad \left\|\mathbf{A}^\top \mathbf{v}\right\|_2 \le \|\mathbf{A}\|_2 \|\mathbf{v}\|_2 , \tag{28}$$

to Eq. equation 27 yields

$$\begin{aligned}\left\|\mathbf{g}_s^t(f)-\mathbf{g}_f^{\mathrm{SSM}}\right\|_2 &= \Big\| \left(\mathbf{J}_s-\mathbf{J}_{\mathrm{SSM}}\right)^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y) \\ &\qquad + \mathbf{J}_{\mathrm{SSM}}^\top\left[\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)-\nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}},y)\right]\Big\|_2 \\ &\le \left\|\left(\mathbf{J}_s-\mathbf{J}_{\mathrm{SSM}}\right)^\top \nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)\right\|_2 \\ &\qquad + \left\|\mathbf{J}_{\mathrm{SSM}}^\top\left[\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)-\nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}},y)\right]\right\|_2 \\ &\le \left\|\mathbf{J}_s-\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)\right\|_2 \\ &\qquad + \left\|\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)-\nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}},y)\right\|_2 .\end{aligned} \tag{29}$$

For the softmax cross-entropy loss used in our experiments,

$$\ell(\mathbf{z},y) = -\log\frac{\exp(z_y)}{\sum_j \exp(z_j)}, \tag{30}$$

and its logit gradient is

$$\nabla_{\mathbf{z}}\ell(\mathbf{z},y) = \mathbf{p}-\mathbf{e}_y, \qquad p_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}. \tag{31}$$

Since $\mathbf{p}$ is a probability vector,

$$\left\|\nabla_{\mathbf{z}}\ell(\mathbf{z},y)\right\|_2^2 = (1-p_y)^2 + \sum_{j\neq y} p_j^2 \le (1-p_y)^2 + \left(\sum_{j\neq y} p_j\right)^2 = 2(1-p_y)^2 \le 2, \tag{32}$$

where the first inequality follows from $\sum_j p_j^2 \le (\sum_j p_j)^2$ for nonnegative p_j. Therefore,

$$\left\|\nabla_{\mathbf{z}}\ell(\mathbf{z},y)\right\|_2 \le \sqrt{2}. \tag{33}$$

Furthermore, applying the Lipschitz condition in Assumption 2 to the second term of Eq. equation 29 gives

$$\left\|\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)-\nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}},y)\right\|_2 \le L_\ell \left\|\mathbf{z}_s-\mathbf{z}_{\mathrm{SSM}}\right\|_2 . \tag{34}$$

Combining Eqs. equation 29, equation 33, and equation 34, we obtain

$$\begin{aligned}\left\|\mathbf{g}_s^t(f)-\mathbf{g}_f^{\mathrm{SSM}}\right\|_2 \le\ & \left\|\mathbf{J}_s-\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)\right\|_2 \\ & + \left\|\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\nabla_{\mathbf{z}_s}\ell(\mathbf{z}_s,y)-\nabla_{\mathbf{z}_{\mathrm{SSM}}}\ell(\mathbf{z}_{\mathrm{SSM}},y)\right\|_2 \\ \le & \sqrt{2}\left\|\mathbf{J}_s-\mathbf{J}_{\mathrm{SSM}}\right\|_2 + L_\ell \left\|\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\mathbf{z}_s-\mathbf{z}_{\mathrm{SSM}}\right\|_2 .\end{aligned} \tag{35}$$

Equation equation 35 shows that the surrogate-gradient discrepancy is controlled by two quantities: the mismatch between the input-to-logit Jacobians of the server-side model and the SSM, and the discrepancy between their output logits. The following subsection therefore derives the Jacobian structures of the sequential server-side model and the parallel SSM and then relates their mismatch to the feature alignment objectives of StreamSplit.

A.2.3 Surrogate Gradient Approximation

We first analyze how accurately the SSM approximates the server-side cut-layer gradient used to update the client-side model. For a given feature-label pair $(\mathbf{x}_f, y_f)$, the SSM computes

$$\mathbf{h}^A = F_A(\mathbf{x}_f), \qquad \mathbf{h}^B = F_B(\mathbf{h}^A), \qquad \mathbf{h}^C = F_C(\mathbf{h}^A), \qquad \mathbf{h}^D = F_D(\mathbf{h}^A), \tag{36}$$

followed by

$$\mathbf{h}^{\mathrm{fuse}} = F_{\mathrm{fuse}}(\mathbf{h}^B,\mathbf{h}^C,\mathbf{h}^D), \qquad \mathbf{z}_{\mathrm{SSM}} = F_{\mathrm{cls}}(\mathbf{h}^{\mathrm{fuse}}). \tag{37}$$

The corresponding server-side model features are denoted by $\mathbf{h}_s^{\mathrm{1st}}$, $\mathbf{h}_s^{\mathrm{shallow}}$, $\mathbf{h}_s^{\mathrm{medium}}$, $\mathbf{h}_s^{\mathrm{deep}}$, and $\mathbf{z}_s$.

The preceding subsection has already established the general surrogate-gradient discrepancy bound in Eq. equation 35. Therefore, it remains to characterize the Jacobian mismatch $\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\|_2$ for the specific sequential server-side model and parallel SSM structures. In particular, we derive the Jacobians of the two computational structures and then relate their mismatch to the feature alignment objectives of StreamSplit.

For the parallel SSM, define

$$\mathbf{J}_A = \frac{\partial \mathbf{h}^A}{\partial \mathbf{x}_f}, \qquad \mathbf{J}_B = \frac{\partial \mathbf{h}^B}{\partial \mathbf{h}^A}, \qquad \mathbf{J}_C = \frac{\partial \mathbf{h}^C}{\partial \mathbf{h}^A}, \qquad \mathbf{J}_D = \frac{\partial \mathbf{h}^D}{\partial \mathbf{h}^A}. \tag{38}$$

Also define the partial Jacobians of the fusion mapping and the classification mapping as

$$\mathbf{J}^B_{\mathrm{fuse}} = \frac{\partial \mathbf{h}^{\mathrm{fuse}}}{\partial \mathbf{h}^B}, \qquad \mathbf{J}^C_{\mathrm{fuse}} = \frac{\partial \mathbf{h}^{\mathrm{fuse}}}{\partial \mathbf{h}^C}, \qquad \mathbf{J}^D_{\mathrm{fuse}} = \frac{\partial \mathbf{h}^{\mathrm{fuse}}}{\partial \mathbf{h}^D}, \qquad \mathbf{J}_{\mathrm{cls}} = \frac{\partial \mathbf{z}_{\mathrm{SSM}}}{\partial \mathbf{h}^{\mathrm{fuse}}}. \tag{39}$$

For an infinitesimal perturbation $d\mathbf{x}_f$, Block A gives

$$d\mathbf{h}^A = \mathbf{J}_A d\mathbf{x}_f. \tag{40}$$

Because Blocks B–D are parallel and all receive $\mathbf{h}^A$, their differentials are

$$d\mathbf{h}^B = \mathbf{J}_B \mathbf{J}_A d\mathbf{x}_f, \qquad d\mathbf{h}^C = \mathbf{J}_C \mathbf{J}_A d\mathbf{x}_f, \qquad d\mathbf{h}^D = \mathbf{J}_D \mathbf{J}_A d\mathbf{x}_f. \tag{41}$$

The fusion mapping depends on all three branch outputs. Therefore, by its total differential,

$$d\mathbf{h}^{\mathrm{fuse}} = \mathbf{J}^B_{\mathrm{fuse}} d\mathbf{h}^B + \mathbf{J}^C_{\mathrm{fuse}} d\mathbf{h}^C + \mathbf{J}^D_{\mathrm{fuse}} d\mathbf{h}^D. \tag{42}$$

Substituting Eq. equation 41 into Eq. equation 42 gives

$$d\mathbf{h}^{\mathrm{fuse}} = \left(\mathbf{J}^B_{\mathrm{fuse}} \mathbf{J}_B + \mathbf{J}^C_{\mathrm{fuse}} \mathbf{J}_C + \mathbf{J}^D_{\mathrm{fuse}} \mathbf{J}_D\right) \mathbf{J}_A d\mathbf{x}_f. \tag{43}$$

Since

$$d\mathbf{z}_{\mathrm{SSM}} = \mathbf{J}_{\mathrm{cls}} d\mathbf{h}^{\mathrm{fuse}}, \tag{44}$$

we obtain

$$\mathbf{J}_{\mathrm{SSM}} = \mathbf{J}_{\mathrm{cls}} \left(\mathbf{J}^B_{\mathrm{fuse}} \mathbf{J}_B + \mathbf{J}^C_{\mathrm{fuse}} \mathbf{J}_C + \mathbf{J}^D_{\mathrm{fuse}} \mathbf{J}_D\right) \mathbf{J}_A. \tag{45}$$

We now write the corresponding Jacobian of the sequential server-side model. Define

$$\mathbf{J}_{1\mathrm{st},s} = \frac{\partial \mathbf{h}^{1\mathrm{st}}_s}{\partial \mathbf{x}_f}, \qquad \mathbf{J}_{\mathrm{shallow},s} = \frac{\partial \mathbf{h}^{\mathrm{shallow}}_s}{\partial \mathbf{h}^{1\mathrm{st}}_s}, \qquad \mathbf{J}_{\mathrm{medium},s} = \frac{\partial \mathbf{h}^{\mathrm{medium}}_s}{\partial \mathbf{h}^{\mathrm{shallow}}_s},$$

$$\mathbf{J}_{\mathrm{deep},s} = \frac{\partial \mathbf{h}^{\mathrm{deep}}_s}{\partial \mathbf{h}^{\mathrm{medium}}_s}, \qquad \mathbf{J}_{\mathrm{cls},s} = \frac{\partial \mathbf{z}_s}{\partial \mathbf{h}^{\mathrm{deep}}_s}. \tag{46}$$

Because the server-side features are generated sequentially,

$$d\mathbf{h}^{1\mathrm{st}}_s = \mathbf{J}_{1\mathrm{st},s} d\mathbf{x}_f, \qquad d\mathbf{h}^{\mathrm{shallow}}_s = \mathbf{J}_{\mathrm{shallow},s} \mathbf{J}_{1\mathrm{st},s} d\mathbf{x}_f,$$

$$d\mathbf{h}^{\mathrm{medium}}_s = \mathbf{J}_{\mathrm{medium},s} \mathbf{J}_{\mathrm{shallow},s} \mathbf{J}_{1\mathrm{st},s} d\mathbf{x}_f,$$

$$d\mathbf{h}^{\mathrm{deep}}_s = \mathbf{J}_{\mathrm{deep},s} \mathbf{J}_{\mathrm{medium},s} \mathbf{J}_{\mathrm{shallow},s} \mathbf{J}_{1\mathrm{st},s} d\mathbf{x}_f. \tag{47}$$

Applying the final classification Jacobian therefore gives

$$\mathbf{J}_s = \mathbf{J}_{\mathrm{cls},s} \mathbf{J}_{\mathrm{deep},s} \mathbf{J}_{\mathrm{medium},s} \mathbf{J}_{\mathrm{shallow},s} \mathbf{J}_{1\mathrm{st},s}. \tag{48}$$

Equations equation 45 and equation 48 show that the two Jacobians have different computational structures: the server-side Jacobian is a sequential product, whereas the SSM Jacobian is a sum of three parallel paths. Therefore, even if all corresponding features are perfectly aligned, the two Jacobians need not be identical. We explicitly separate this structural difference from the error caused by imperfect feature alignment.

Define the structurally aligned reference Jacobian of the SSM as

$$\widetilde{\mathbf{J}}_{\mathrm{SSM}} = \mathbf{J}_{\mathrm{cls}} \left(\mathbf{J}^B_{\mathrm{fuse}} \mathbf{J}_{\mathrm{shallow},s} + \mathbf{J}^C_{\mathrm{fuse}} \mathbf{J}_{\mathrm{medium},s} + \mathbf{J}^D_{\mathrm{fuse}} \mathbf{J}_{\mathrm{deep},s}\right) \mathbf{J}_{1\mathrm{st},s}. \tag{49}$$

This reference replaces the SSM branch Jacobians by the corresponding server-side transformations while retaining the actual parallel fusion and classification mappings of the SSM. It therefore isolates the effect of the different computational structures.

We define the structural approximation residual as

$$R_{\text{struct}} = \left\| \mathbf{J}_s - \widetilde{\mathbf{J}}_{\text{SSM}} \right\|_2 \Big|_{\mathcal{A}}, \tag{50}$$

where $\mathcal{A}$ denotes the local alignment condition

$$\mathcal{A}: \quad \mathbf{h}^A = \mathbf{h}_s^{\text{1st}}, \quad \mathbf{h}^B = \mathbf{h}_s^{\text{shallow}}, \quad \mathbf{h}^C = \mathbf{h}_s^{\text{medium}}, \quad \mathbf{h}^D = \mathbf{h}_s^{\text{deep}}, \quad \mathbf{z}_{\text{SSM}} = \mathbf{z}_s. \tag{51}$$

Thus, R_{struct} measures the Jacobian mismatch that remains when the corresponding features are aligned and is caused only by the different sequential and parallel computational structures.

We next bound the feature-induced part $\left\| \widetilde{\mathbf{J}}_{\text{SSM}} - \mathbf{J}_{\text{SSM}} \right\|_2$. From Eqs. equation 45 and equation 49,

$$\begin{aligned}
\widetilde{\mathbf{J}}_{\text{SSM}} - \mathbf{J}_{\text{SSM}} =& \mathbf{J}_{\text{cls}} \left(\mathbf{J}_{\text{fuse}}^B \mathbf{J}_{\text{shallow},s} + \mathbf{J}_{\text{fuse}}^C \mathbf{J}_{\text{medium},s} + \mathbf{J}_{\text{fuse}}^D \mathbf{J}_{\text{deep},s} \right) \mathbf{J}_{\text{1st},s} \\
& - \mathbf{J}_{\text{cls}} \left(\mathbf{J}_{\text{fuse}}^B \mathbf{J}_B + \mathbf{J}_{\text{fuse}}^C \mathbf{J}_C + \mathbf{J}_{\text{fuse}}^D \mathbf{J}_D \right) \mathbf{J}_A \\
=& \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^B \mathbf{J}_{\text{shallow},s} \mathbf{J}_{\text{1st},s} + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^C \mathbf{J}_{\text{medium},s} \mathbf{J}_{\text{1st},s} + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^D \mathbf{J}_{\text{deep},s} \mathbf{J}_{\text{1st},s} \\
& - \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^B \mathbf{J}_B \mathbf{J}_A - \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^C \mathbf{J}_C \mathbf{J}_A - \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^D \mathbf{J}_D \mathbf{J}_A \\
=& \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^B \left(\mathbf{J}_{\text{shallow},s} - \mathbf{J}_B \right) \mathbf{J}_{\text{1st},s} + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^C \left(\mathbf{J}_{\text{medium},s} - \mathbf{J}_C \right) \mathbf{J}_{\text{1st},s} \\
& + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^D \left(\mathbf{J}_{\text{deep},s} - \mathbf{J}_D \right) \mathbf{J}_{\text{1st},s} + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^B \mathbf{J}_B \left(\mathbf{J}_{\text{1st},s} - \mathbf{J}_A \right) \\
& + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^C \mathbf{J}_C \left(\mathbf{J}_{\text{1st},s} - \mathbf{J}_A \right) + \mathbf{J}_{\text{cls}} \mathbf{J}_{\text{fuse}}^D \mathbf{J}_D \left(\mathbf{J}_{\text{1st},s} - \mathbf{J}_A \right) \\
=& \mathbf{J}_{\text{cls}} \Big[\mathbf{J}_{\text{fuse}}^B (\mathbf{J}_{\text{shallow},s} - \mathbf{J}_B) \\
& + \mathbf{J}_{\text{fuse}}^C (\mathbf{J}_{\text{medium},s} - \mathbf{J}_C) + \mathbf{J}_{\text{fuse}}^D (\mathbf{J}_{\text{deep},s} - \mathbf{J}_D) \Big] \mathbf{J}_{\text{1st},s} \\
& + \mathbf{J}_{\text{cls}} \left(\mathbf{J}_{\text{fuse}}^B + \mathbf{J}_{\text{fuse}}^C + \mathbf{J}_{\text{fuse}}^D \right) (\mathbf{J}_{\text{1st},s} - \mathbf{J}_A).
\end{aligned} \tag{52}$$

Applying the triangle inequality and the submultiplicativity of the induced 2-norm to Eq. equation 52, we obtain

$$\begin{aligned}
\left\| \widetilde{\mathbf{J}}_{\text{SSM}} - \mathbf{J}_{\text{SSM}} \right\|_2 \leq & \left\| \mathbf{J}_{\text{cls}} \right\|_2 \left\| \mathbf{J}_{\text{fuse}}^B \right\|_2 \left\| \mathbf{J}_{\text{shallow},s} - \mathbf{J}_B \right\|_2 \left\| \mathbf{J}_{\text{1st},s} \right\|_2 \\
& + \left\| \mathbf{J}_{\text{cls}} \right\|_2 \left\| \mathbf{J}_{\text{fuse}}^C \right\|_2 \left\| \mathbf{J}_{\text{medium},s} - \mathbf{J}_C \right\|_2 \left\| \mathbf{J}_{\text{1st},s} \right\|_2 \\
& + \left\| \mathbf{J}_{\text{cls}} \right\|_2 \left\| \mathbf{J}_{\text{fuse}}^D \right\|_2 \left\| \mathbf{J}_{\text{deep},s} - \mathbf{J}_D \right\|_2 \left\| \mathbf{J}_{\text{1st},s} \right\|_2 \\
& + \left\| \mathbf{J}_{\text{cls}} \right\|_2 \left(\left\| \mathbf{J}_{\text{fuse}}^B \right\|_2 + \left\| \mathbf{J}_{\text{fuse}}^C \right\|_2 + \left\| \mathbf{J}_{\text{fuse}}^D \right\|_2 \right) \left\| \mathbf{J}_{\text{1st},s} - \mathbf{J}_A \right\|_2 .
\end{aligned} \tag{53}$$

To further characterize the above Jacobian discrepancy in terms of the feature alignment errors between the SSM and the server-side model, we establish the following lemma based on Assumption 3.

Lemma 1 (feature-Induced Jacobian Error Bound). Under Assumption 3, for each corresponding SSM and server-side transformation, the Jacobian discrepancy can be decomposed into a feature-induced term and a local transformation mismatch. Specifically, define

$$\begin{aligned}
\delta_{\text{1st}} &= \left\| \mathbf{J}_A(\mathbf{h}_s^{\text{1st}}) - \mathbf{J}_{\text{1st},s}(\mathbf{h}_s^{\text{1st}}) \right\|_2 , \\
\delta_{\text{shallow}} &= \left\| \mathbf{J}_B(\mathbf{h}_s^{\text{shallow}}) - \mathbf{J}_{\text{shallow},s}(\mathbf{h}_s^{\text{shallow}}) \right\|_2 , \\
\delta_{\text{medium}} &= \left\| \mathbf{J}_C(\mathbf{h}_s^{\text{medium}}) - \mathbf{J}_{\text{medium},s}(\mathbf{h}_s^{\text{medium}}) \right\|_2 , \\
\delta_{\text{deep}} &= \left\| \mathbf{J}_D(\mathbf{h}_s^{\text{deep}}) - \mathbf{J}_{\text{deep},s}(\mathbf{h}_s^{\text{deep}}) \right\|_2 .
\end{aligned} \tag{54}$$

Then,

$$\begin{aligned}
\left\| \mathbf{J}_A - \mathbf{J}_{\text{1st},s} \right\|_2 &\leq L_J \left\| \mathbf{h}^A - \mathbf{h}_s^{\text{1st}} \right\|_2 + \delta_{\text{1st}}, \\
\left\| \mathbf{J}_B - \mathbf{J}_{\text{shallow},s} \right\|_2 &\leq L_J \left\| \mathbf{h}^B - \mathbf{h}_s^{\text{shallow}} \right\|_2 + \delta_{\text{shallow}}, \\
\left\| \mathbf{J}_C - \mathbf{J}_{\text{medium},s} \right\|_2 &\leq L_J \left\| \mathbf{h}^C - \mathbf{h}_s^{\text{medium}} \right\|_2 + \delta_{\text{medium}}, \\
\left\| \mathbf{J}_D - \mathbf{J}_{\text{deep},s} \right\|_2 &\leq L_J \left\| \mathbf{h}^D - \mathbf{h}_s^{\text{deep}} \right\|_2 + \delta_{\text{deep}}.
\end{aligned} \tag{55}$$

Proof. We prove the first inequality; the other three follow identically. By adding and subtracting $\mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}})$ and applying the triangle inequality, we obtain

$$\begin{aligned}\left\|\mathbf{J}_A(\mathbf{h}^A) - \mathbf{J}_{1\mathrm{st},s}(\mathbf{h}_s^{1\mathrm{st}})\right\|_2 &= \left\|\mathbf{J}_A(\mathbf{h}^A) - \mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}}) + \mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}}) - \mathbf{J}_{1\mathrm{st},s}(\mathbf{h}_s^{1\mathrm{st}})\right\|_2 \\ &\le \left\|\mathbf{J}_A(\mathbf{h}^A) - \mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}})\right\|_2 + \left\|\mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}}) - \mathbf{J}_{1\mathrm{st},s}(\mathbf{h}_s^{1\mathrm{st}})\right\|_2 .\end{aligned} \tag{56}$$

By Assumption 3, the first term satisfies

$$\left\|\mathbf{J}_A(\mathbf{h}^A) - \mathbf{J}_A(\mathbf{h}_s^{1\mathrm{st}})\right\|_2 \le L_J \left\|\mathbf{h}^A - \mathbf{h}_s^{1\mathrm{st}}\right\|_2 , \tag{57}$$

while the second term is exactly $\delta_{1\mathrm{st}}$ by Eq. equation 54. Therefore,

$$\|\mathbf{J}_A - \mathbf{J}_{1\mathrm{st},s}\|_2 \le L_J \left\|\mathbf{h}^A - \mathbf{h}_s^{1\mathrm{st}}\right\|_2 + \delta_{1\mathrm{st}}. \tag{58}$$

The same argument applied to Blocks B–D proves the remaining three inequalities. □

The lemma separates the effect of feature misalignment from the intrinsic mismatch between the corresponding SSM and server-side transformations. In particular, the first terms in Eq. equation 55 are directly controlled by the feature alignment objectives, whereas the δ terms remain when the corresponding features are perfectly aligned.

We now apply Lemma 1 to the Jacobian bound in Eq. equation 53. For notational simplicity, let the Jacobian norms appearing in Eq. equation 53 be bounded by finite constants. Substituting Eq. equation 55 into Eq. equation 53 yields

$$\begin{aligned}\left\|\widetilde{\mathbf{J}}_{\mathrm{SSM}} - \mathbf{J}_{\mathrm{SSM}}\right\|_2 \le\ & C'_{1\mathrm{st}} \left\|\mathbf{h}^A - \mathbf{h}_s^{1\mathrm{st}}\right\|_2 \\ & + C'_{\mathrm{hier}} \left(\left\|\mathbf{h}^B - \mathbf{h}_s^{\mathrm{shallow}}\right\|_2 + \left\|\mathbf{h}^C - \mathbf{h}_s^{\mathrm{medium}}\right\|_2 + \left\|\mathbf{h}^D - \mathbf{h}_s^{\mathrm{deep}}\right\|_2\right) \\ & + R_{\mathrm{J}},\end{aligned} \tag{59}$$

where $C'_{1\mathrm{st}}$ and C'_{hier} collect the bounded Jacobian norms and the common Lipschitz constant L_J, while

$$R_{\mathrm{J}} = C'_{1\mathrm{st}}\delta_{1\mathrm{st}} + C'_{\mathrm{hier}} \left(\delta_{\mathrm{shallow}} + \delta_{\mathrm{medium}} + \delta_{\mathrm{deep}}\right) \tag{60}$$

where $C'_{1\mathrm{st}}$ and C'_{hier} collect the bounded Jacobian norms and the common Lipschitz constant L_J, while R_J collects the residual mismatch between the corresponding SSM and server-side transformations at aligned features, which cannot be eliminated solely by feature alignment.

We can now bound the actual Jacobian mismatch. By the triangle inequality,

$$\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\|_2 \le \left\|\mathbf{J}_s - \widetilde{\mathbf{J}}_{\mathrm{SSM}}\right\|_2 + \left\|\widetilde{\mathbf{J}}_{\mathrm{SSM}} - \mathbf{J}_{\mathrm{SSM}}\right\|_2 . \tag{61}$$

By the definition of R_{struct} and Eq. equation 59,

$$\begin{aligned}\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\|_2 \le\ & C'_{1\mathrm{st}} \left\|\mathbf{h}^A - \mathbf{h}_s^{1\mathrm{st}}\right\|_2 \\ & + C'_{\mathrm{hier}} \left(\left\|\mathbf{h}^B - \mathbf{h}_s^{\mathrm{shallow}}\right\|_2 + \left\|\mathbf{h}^C - \mathbf{h}_s^{\mathrm{medium}}\right\|_2 + \left\|\mathbf{h}^D - \mathbf{h}_s^{\mathrm{deep}}\right\|_2\right) \\ & + \sqrt{2}(R_{\mathrm{J}} + R_{\mathrm{struct}}).\end{aligned} \tag{62}$$

Substituting Eq. equation 62 into Eq. equation 35 gives

$$\begin{aligned}\left\|\mathbf{g}_s^t(f) - \mathbf{g}_f^{\mathrm{SSM}}\right\|_2 \le\ & \sqrt{2}C'_{1\mathrm{st}} \left\|\mathbf{h}^A - \mathbf{h}_s^{1\mathrm{st}}\right\|_2 \\ & + \sqrt{2}C'_{\mathrm{hier}} \left(\left\|\mathbf{h}^B - \mathbf{h}_s^{\mathrm{shallow}}\right\|_2 + \left\|\mathbf{h}^C - \mathbf{h}_s^{\mathrm{medium}}\right\|_2 + \left\|\mathbf{h}^D - \mathbf{h}_s^{\mathrm{deep}}\right\|_2\right) \\ & + M_J L_\ell \left\|\mathbf{z}_s - \mathbf{z}_{\mathrm{SSM}}\right\|_2 \\ & + \sqrt{2}(R_{\mathrm{J}} + R_{\mathrm{struct}}).\end{aligned} \tag{63}$$

For notational simplicity, define

$$C_{1\mathrm{st}} := \sqrt{2}C'_{1\mathrm{st}}, \qquad C_{\mathrm{hier}} := \sqrt{2}C'_{\mathrm{hier}}, \qquad C_{\mathrm{logit}} := M_J L_\ell. \tag{64}$$

Then Eq. equation 63 becomes

$$\begin{aligned}\left\|\mathbf{g}_s^t(f)-\mathbf{g}_f^{\mathrm{SSM}}\right\|_2 \le\ & C_{\mathrm{1st}}\left\|\mathbf{h}^A-\mathbf{h}_s^{\mathrm{1st}}\right\|_2 \\ & + C_{\mathrm{hier}}\left(\left\|\mathbf{h}^B-\mathbf{h}_s^{\mathrm{shallow}}\right\|_2+\left\|\mathbf{h}^C-\mathbf{h}_s^{\mathrm{medium}}\right\|_2+\left\|\mathbf{h}^D-\mathbf{h}_s^{\mathrm{deep}}\right\|_2\right) \\ & + C_{\mathrm{logit}}\left\|\mathbf{z}_{\mathrm{SSM}}-\mathbf{z}_s\right\|_2 \\ & + \sqrt{2}(R_{\mathrm{J}}+R_{\mathrm{struct}}).\end{aligned} \tag{65}$$

Theorem 1 (SSM Surrogate Gradient Approximation). Under the bounded-Jacobian condition, Assumption 3, and the local transformation mismatch decomposition in Lemma 1, the SSM surrogate gradient satisfies Eq. equation 65, where the approximation error is characterized by the feature alignment terms, the structural residual R_{struct}, and the local transformation mismatch residual R_J.

Proof. The result follows sequentially from the chain-rule gradient expressions in Eq. equation 26, the gradient decomposition in Eq. equation 27, the induced-norm inequality in Eq. equation 28, the cross-entropy gradient bound in Eq. equation 30, the loss-gradient Lipschitz condition in Eq. equation 34, and the Jacobian mismatch bound in Eq. equation 62. Substituting these bounds yields Eq. equation 63, and the definitions in Eq. equation 64 give Eq. equation 65. □

Corollary 1 (Gradient Error Bound under Alignment Loss). We next express Theorem 1 directly in terms of the alignment losses used to train the SSM. Define the five nonnegative feature errors

$$\begin{aligned}& e_{\mathrm{1st}}=\left\|\mathbf{h}^A-\mathbf{h}_s^{\mathrm{1st}}\right\|_2, \qquad e_{\mathrm{shallow}}=\left\|\mathbf{h}^B-\mathbf{h}_s^{\mathrm{shallow}}\right\|_2, \\ & e_{\mathrm{medium}}=\left\|\mathbf{h}^C-\mathbf{h}_s^{\mathrm{medium}}\right\|_2, \qquad e_{\mathrm{deep}}=\left\|\mathbf{h}^D-\mathbf{h}_s^{\mathrm{deep}}\right\|_2, \qquad e_{\mathrm{logit}}=\left\|\mathbf{z}_{\mathrm{SSM}}-\mathbf{z}_s\right\|_2.\end{aligned} \tag{66}$$

Using these definitions, Eq. equation 65 becomes

$$\begin{aligned}\left|\mathbf{g}_s^t(f)-\mathbf{g}_f^{\mathrm{SSM}}\right|_2 \le\ & C_{\mathrm{1st}}e_{\mathrm{1st}} \\ & + C_{\mathrm{hier}}(e_{\mathrm{shallow}}+e_{\mathrm{medium}}+e_{\mathrm{deep}}) \\ & + C_{\mathrm{logit}}e_{\mathrm{logit}} \\ & + \sqrt{2}(R_J+R_{\mathrm{struct}}).\end{aligned} \tag{67}$$

From the 1st-layer alignment loss defined in the main text,

$$\mathcal{L}_{\mathrm{align}}^{\mathrm{1st}}=\left\|\mathbf{h}^A-\mathbf{h}_s^{\mathrm{1st}}\right\|_2^2+\left\|\mathbf{z}_{\mathrm{SSM}}-\mathbf{z}_s\right\|_2^2, \tag{68}$$

we have

$$\mathcal{L}_{\mathrm{align}}^{\mathrm{1st}}=e_{\mathrm{1st}}^2+e_{\mathrm{logit}}^2. \tag{69}$$

Similarly, from the hierarchical alignment loss,

$$\mathcal{L}_{\mathrm{align}}^{\mathrm{hier}}=\left\|\mathbf{h}^B-\mathbf{h}_s^{\mathrm{shallow}}\right\|_2^2+\left\|\mathbf{h}^C-\mathbf{h}_s^{\mathrm{medium}}\right\|_2^2+\left\|\mathbf{h}^D-\mathbf{h}_s^{\mathrm{deep}}\right\|_2^2, \tag{70}$$

we have

$$\mathcal{L}_{\mathrm{align}}^{\mathrm{hier}}=e_{\mathrm{shallow}}^2+e_{\mathrm{medium}}^2+e_{\mathrm{deep}}^2. \tag{71}$$

Adding Eqs. equation 69 and equation 71 therefore gives

$$\mathcal{L}_{\mathrm{align}}^{\mathrm{1st}}+\mathcal{L}_{\mathrm{align}}^{\mathrm{hier}}=e_{\mathrm{1st}}^2+e_{\mathrm{shallow}}^2+e_{\mathrm{medium}}^2+e_{\mathrm{deep}}^2+e_{\mathrm{logit}}^2. \tag{72}$$

We now bound the first five terms of Eq. equation 67. Define

$$\mathbf{a}=(C_{\mathrm{1st}},C_{\mathrm{hier}},C_{\mathrm{hier}},C_{\mathrm{hier}},C_{\mathrm{logit}}), \qquad \mathbf{e}=(e_{\mathrm{1st}},e_{\mathrm{shallow}},e_{\mathrm{medium}},e_{\mathrm{deep}},e_{\mathrm{logit}}). \tag{73}$$

The non-structural terms in Eq. equation 67 are exactly $\mathbf{a}^\top\mathbf{e}$. By the Cauchy–Schwarz inequality,

$$\mathbf{a}^\top\mathbf{e}\le\|\mathbf{a}\|_2\|\mathbf{e}\|_2. \tag{74}$$

The coefficient norm is

$$\|\mathbf{a}\|_2=\sqrt{C_{\mathrm{1st}}^2+3C_{\mathrm{hier}}^2+C_{\mathrm{logit}}^2}, \tag{75}$$

while Eq. equation 72 gives

$$\|\mathbf{e}\|_2 = \sqrt{\mathcal{L}_{\text{align}}^{\text{1st}} + \mathcal{L}_{\text{align}}^{\text{hier}}}. \tag{76}$$

Define

$$C_{\text{align}} = \sqrt{C_{\text{1st}}^2 + 3C_{\text{hier}}^2 + C_{\text{logit}}^2}. \tag{77}$$

Substituting Eqs. equation 75–equation 77 into Eq. equation 67 gives

$$\boxed{\left\|\mathbf{g}_s^t(f) - \mathbf{g}_f^{\text{SSM}}\right\|_2 \leq C_{\text{align}}\sqrt{\mathcal{L}_{\text{align}}^{\text{1st}} + \mathcal{L}_{\text{align}}^{\text{hier}}} + \sqrt{2}(R_J + R_{\text{struct}})} \tag{78}$$

Therefore, the alignment losses directly control the deviation between the SSM surrogate gradient and the true server-side cut-layer gradient, up to the structural approximation residual R_{struct} and the local transformation mismatch residual R_J. In the ideal case where the parallel SSM structure is locally compatible with the sequential server-side mapping, $R_{\text{struct}} = 0$, and the surrogate-gradient deviation is controlled entirely by the alignment losses. □

A.2.4 Optimization Consistency of Client Updates

The previous analysis establishes that the cut-layer gradient generated by the SSM can approximate the gradient of the actual server-side model. We next analyze how this gradient approximation affects client-side optimization. In particular, we show that replacing the unavailable server gradient with the SSM surrogate gradient results in a bounded deviation from the standard SFL client update. Thus, the decoupled client optimization in StreamSplit remains consistent with the optimization direction induced by the server-side model.

Let $\boldsymbol{\theta}_c^i$ denote the parameters of client i, and let $\mathbf{x}_f^i$ denote the cut-layer feature produced by the client model for a local sample (x, y). Under standard SFL, the client update at iteration t using the actual server-side gradient is

$$\boldsymbol{\theta}_{c,s}^{i,t+1} = \boldsymbol{\theta}_c^{i,t}\eta_c - \nabla_{\boldsymbol{\theta}_c^i}\ell\left(F_s(\mathbf{x}_f^i), y\right), \tag{79}$$

where η_c is the client learning rate. Since the gradient with respect to the client parameters is obtained through the cut-layer feature, the chain rule gives

$$\nabla_{\boldsymbol{\theta}_c^i}\ell\left(F_s(\mathbf{x}_f^i), y\right) = \left(\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right)^\top \mathbf{g}_s^t(f_i). \tag{80}$$

In StreamSplit, the server-side gradient $\mathbf{g}_s^t(f_i)$ is unavailable during the asynchronous client update. The client therefore uses the SSM surrogate gradient $\mathbf{g}_{f_i}^{\text{SSM}}$, yielding

$$\boldsymbol{\theta}_{c,\text{SSM}}^{i,t+1} = \boldsymbol{\theta}_c^{i,t} - \eta_c\left(\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right)^\top \mathbf{g}_{f_i}^{\text{SSM}}. \tag{81}$$

Define the client update deviation between the standard SFL using gradient generated by server-side model and the StreamSplit as

$$\Delta\boldsymbol{\theta}_c^{i,t+1} = \boldsymbol{\theta}_{c,\text{SSM}}^{i,t+1} - \boldsymbol{\theta}_{c,s}^{i,t+1}. \tag{82}$$

Substituting Eqs. equation 79 and equation 81 into Eq. equation 82 gives

$$\begin{aligned}\Delta\boldsymbol{\theta}_c^{i,t+1} &= -\eta_c\left(\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right)^\top \mathbf{g}_{f_i}^{\text{SSM}} + \eta_c\left(\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right)^\top \mathbf{g}_s^t(f_i) \\ &= \eta_c\left(\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right)^\top \left[\mathbf{g}_s^t(f_i) - \mathbf{g}_{f_i}^{\text{SSM}}\right].\end{aligned} \tag{83}$$

Assume that the client-side feature mapping has a bounded parameter Jacobian. That is, there exists a finite constant M_c such that

$$\left\|\frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i}\right\|_2 \leq M_c. \tag{84}$$

Applying the induced matrix norm inequality to Eq. equation 83 gives

$$\left\|\Delta\boldsymbol{\theta}_c^{i,t+1}\right\| \leq \eta_c \left\| \frac{\partial \mathbf{x}_f^i}{\partial \boldsymbol{\theta}_c^i} \right\|_2 \left\|\mathbf{g}_s^t(f_i) - \mathbf{g}_{f_i}^{\mathrm{SSM}}\right\|_2 \leq \eta_c M_c \left\|\mathbf{g}_s^t(f_i) - \mathbf{g}_{f_i}^{\mathrm{SSM}}\right\|_2 . \tag{85}$$

From the surrogate-gradient approximation established in Eq. equation 35,

$$\left\|\mathbf{g}_s^t(f_i) - \mathbf{g}_{f_i}^{\mathrm{SSM}}\right\|_2 \leq \sqrt{2}\left\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\right\|_2 + L_\ell \left\|\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\mathbf{z}_s - \mathbf{z}_{\mathrm{SSM}}\right\|_2 . \tag{86}$$

Substituting Eq. equation 86 into Eq. equation 85 yields

$$\left\|\Delta\boldsymbol{\theta}_c^{i,t+1}\right\|_2 \leq \eta_c M_c \Big[\sqrt{2}\left\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\right\|_2 + L_\ell \left\|\mathbf{J}_{\mathrm{SSM}}\right\|_2 \left\|\mathbf{z}_s - \mathbf{z}_{\mathrm{SSM}}\right\|_2 \Big]. \tag{87}$$

Therefore, when the SSM is well aligned with the server-side model, both the Jacobian discrepancy and the output discrepancy become small, and the resulting client update remains close to the update that would be obtained from the actual server-side gradient. In particular, if

$$\left\|\mathbf{J}_s - \mathbf{J}_{\mathrm{SSM}}\right\|_2 \leq \varepsilon_J, \qquad \left\|\mathbf{z}_s - \mathbf{z}_{\mathrm{SSM}}\right\|_2 \leq \varepsilon_z, \tag{88}$$

then Eq. equation 87 becomes

$$\left\|\Delta\boldsymbol{\theta}_c^{i,t+1}\right\|_2 \leq \eta_c M_c \left(\sqrt{2}\varepsilon_J + L_\ell M_{\mathrm{SSM}} \varepsilon_z\right). \tag{89}$$

The bound in Eq. equation 89 shows that the deviation introduced by asynchronous surrogate optimization is controlled by the approximation quality of the SSM rather than by the absence of a real-time server gradient itself. As the SSM becomes better aligned with the evolving server-side model, the deviation of each client update decreases accordingly.

We further consider multiple local updates performed between two SSM synchronization points. Let K denote the number of local client updates between two consecutive synchronizations, and let $\Delta\boldsymbol{\theta}_c^{i,t+k}$ denote the update deviation at local step k. The accumulated deviation after K local updates satisfies

$$\left\|\boldsymbol{\theta}_{c,\mathrm{SSM}}^{i,t+K} - \boldsymbol{\theta}_{c,s}^{i,t+K}\right\|_2 \leq \sum_{k=0}^{K-1} \left\|\Delta\boldsymbol{\theta}_c^{i,t+k+1}\right\|_2 \leq K\eta_c M_c \left(\sqrt{2}\varepsilon_J + L_\ell M_{\mathrm{SSM}}\varepsilon_z\right), \tag{90}$$

where the first inequality follows from the triangle inequality and the second follows from Eq. equation 89. Hence, for a fixed synchronization interval K, the accumulated discrepancy remains bounded. Moreover, reducing the SSM approximation error or the local synchronization interval directly reduces the deviation between asynchronous client optimization and the corresponding server-guided optimization.

This establishes the optimization consistency of StreamSplit: the client does not require an instantaneous server-side gradient to follow an optimization direction close to that of standard SFL. Instead, the SSM provides a bounded surrogate direction whose induced client-parameter update remains close to the server-guided update. The result also explains the role of periodic SSM synchronization: by keeping the local surrogate model aligned with the evolving server-side model, StreamSplit prevents the surrogate-gradient error from growing without bound during continuous client-side optimization.

A.2.5 Parallel Architecture Prevents Cross-Depth Error Accumulation

The parallel architecture of the SSM is designed to approximate features at different depths without propagating approximation errors through a sequential chain. We compare the error propagation of sequential and parallel architectures under the same local approximation quality.

Let the Jacobian approximation errors at the first, shallow, medium, and deep depths be

$$E_{1st} = J_A - J_{1st,s}, \quad E_{shallow} = J_B - J_{shallow,s},$$
$$E_{medium} = J_C - J_{medium,s}, \quad E_{deep} = J_D - J_{deep,s}. \tag{91}$$

From the local Jacobian compatibility assumption, these errors are bounded by the corresponding feature alignment errors. To isolate the effect of the architecture, we further assume that all depth-specific Jacobian errors satisfy

$$\|E_{1st}\|_2, \|E_{shallow}\|_2, \|E_{medium}\|_2, \|E_{deep}\|_2 \le \epsilon_J, \tag{92}$$

and that the Jacobians of the corresponding server-side transformations are bounded by

$$\|J_{1st,s}\|_2, \|J_{shallow,s}\|_2, \|J_{medium,s}\|_2, \|J_{deep,s}\|_2 \le M. \tag{93}$$

These conditions allow us to compare the error propagation caused solely by the computational structure.

Sequential architecture. Consider a sequential SSM in which the depth-specific features are generated successively. The Jacobian from the cut-layer feature x_f to the deep feature is

$$J_{seq}^{deep} = J_D J_C J_B J_A, \tag{94}$$

whereas the corresponding server-side Jacobian is

$$J_s^{deep} = J_{deep,s} J_{medium,s} J_{shallow,s} J_{1st,s}. \tag{95}$$

Substituting Eq. equation 91 into the sequential Jacobian gives

$$J_{seq}^{deep} = (J_{deep,s} + E_{deep})(J_{medium,s} + E_{medium})(J_{shallow,s} + E_{shallow})(J_{1st,s} + E_{1st}). \tag{96}$$

Expanding the product and subtracting J_s^{deep}, the first-order Jacobian error is

$$\begin{aligned}
J_{seq}^{deep} - J_s^{deep} = {} & J_{deep,s}J_{medium,s}J_{shallow,s}E_{1st} + J_{deep,s}J_{medium,s}E_{shallow}J_{1st,s} \\
& + J_{deep,s}E_{medium}J_{shallow,s}J_{1st,s} + E_{deep}J_{medium,s}J_{shallow,s}J_{1st,s} \\
& + J_{deep,s}J_{medium,s}E_{shallow}E_{1st} + J_{deep,s}E_{medium}J_{shallow,s}E_{1st} \\
& + J_{deep,s}E_{medium}E_{shallow}J_{1st,s} + E_{deep}J_{medium,s}J_{shallow,s}E_{1st} \\
& + E_{deep}J_{medium,s}E_{shallow}J_{1st,s} + E_{deep}E_{medium}J_{shallow,s}J_{1st,s} \\
& + J_{deep,s}E_{medium}E_{shallow}E_{1st} + E_{deep}J_{medium,s}E_{shallow}E_{1st} \\
& + E_{deep}E_{medium}J_{shallow,s}E_{1st} + E_{deep}E_{medium}E_{shallow}J_{1st,s} \\
& + E_{deep}E_{medium}E_{shallow}E_{1st} \\
= {} & J_{deep,s}J_{medium,s}J_{shallow,s}E_{1st} + J_{deep,s}J_{medium,s}E_{shallow}J_{1st,s} \\
& + J_{deep,s}E_{medium}J_{shallow,s}J_{1st,s} + E_{deep}J_{medium,s}J_{shallow,s}J_{1st,s} \\
& + O(\epsilon_J^2).
\end{aligned} \tag{97}$$

Applying the triangle inequality and the submultiplicativity of the induced 2-norm,

$$|AB|_2 \le |A|_2 |B|_2, \tag{98}$$

to Eq. equation 97, and using Eq. equation 93, gives

$$\begin{aligned}
\left\|J_{seq}^{deep} - J_s^{deep}\right\|_2 \le {} & M^3 \|E_{1st}\|_2 + M^3 \|E_{shallow}\|_2 \\
& + M^3 \|E_{medium}\|_2 + M^3 \|E_{deep}\|_2 \\
& + O(\epsilon_J^2).
\end{aligned} \tag{99}$$

Applying Eq. equation 92 yields

$$\boxed{\left\|J_{seq}^{deep} - J_s^{deep}\right\|_2 \le 4M^3\epsilon_J + O(\epsilon_J^2)} \tag{100}$$

Thus, for the four-depth sequential composition, an error introduced at an earlier depth is multiplied by the Jacobians of all subsequent transformations. For example, the first-layer error contributes

$$\|J_{deep,s}J_{medium,s}J_{shallow,s}E_{1st}\|_2 \le M^3\epsilon_J. \tag{101}$$

Therefore, the approximation error is subject to multiplicative cross-depth propagation.

More generally, for a sequential architecture with D depth-specific transformations, the first-order error consists of D terms, and the error introduced at depth k is multiplied by the Jacobians of the remaining $D-k$ transformations. Under the same bound in Eq. equation 93, the resulting error satisfies

$$\|J_{seq} - J_s\|_2 \le DM^{D-1}\epsilon_J + O(\epsilon_J^2). \tag{102}$$

Parallel architecture. In the proposed SSM, the shallow, medium, and deep features are generated independently from the shared first feature,

$$h^B = F_B(h^A), \qquad h^C = F_C(h^A), \qquad h^D = F_D(h^A). \tag{103}$$

Consequently, the corresponding branch Jacobians are

$$\frac{\partial h^B}{\partial x_f} = J_B J_A, \qquad \frac{\partial h^C}{\partial x_f} = J_C J_A, \qquad \frac{\partial h^D}{\partial x_f} = J_D J_A. \tag{104}$$

For the shallow branch, substituting Eq. equation 91 gives

$$J_B J_A - J_{shallow,s} J_{1st,s} = J_{shallow,s} E_{1st} + E_{shallow} J_{1st,s} + E_{shallow} E_{1st}. \tag{105}$$

Applying the triangle inequality and submultiplicativity gives

$$\|J_B J_A - J_{shallow,s} J_{1st,s}\|_2 \le M \|E_{1st}\|_2 + M \|E_{shallow}\|_2 + \|E_{shallow}\|_2 \|E_{1st}\|_2 . \tag{106}$$

Using Eq. equation 92, we obtain

$$\|J_B J_A - J_{shallow,s} J_{1st,s}\|_2 \le 2M\epsilon_J + \epsilon_J^2. \tag{107}$$

The same derivation applies independently to the medium and deep branches:

$$\begin{aligned} \|J_C J_A - J_{medium,s} J_{1st,s}\|_2 =& J_{medium,s} E_{1st} + E_{medium} J_{1st,s} + E_{medium} E_{1st} \\ \le& M_{medium} \|E_{1st}\|_2 + M_{1st} \|E_{medium}\|_2 \\ &+ \|E_{medium}\|_2 \|E_{1st}\|_2 \\ =& 2M\epsilon_J + \epsilon_J^2, \end{aligned} \tag{108}$$

$$\begin{aligned} \|J_D J_A - J_{deep,s} J_{1st,s}\|_2 &= J_{deep,s} E_{1st} + E_{deep} J_{1st,s} + E_{deep} E_{1st} \\ &\le M_{deep} \|E_{1st}\|_2 + M_{1st} \|E_{deep}\|_2 + \|E_{deep}\|_2 \|E_{1st}\|_2 \\ &= 2M\epsilon_J + \epsilon_J^2. \end{aligned} \tag{109}$$

The key difference is that the shallow-branch error $E_{shallow}$ does not enter the medium or deep branch, and E_{medium} does not enter the deep branch. Each branch therefore contains only the shared first-layer error and its own depth-specific error.

The three branches are subsequently combined by the fusion operation. From Eq. equation 45,

$$J_{SSM} = J_{cls} \left(J_{fuse}^B J_B + J_{fuse}^C J_C + J_{fuse}^D J_D \right) J_A. \tag{110}$$

Assume

$$\|J_{cls}\|_2 \le M_{cls}, \qquad \left\|J_{fuse}^B\right\|_2, \left\|J_{fuse}^C\right\|_2, \left\|J_{fuse}^D\right\|_2 \le M_{fuse}. \tag{111}$$

Applying the triangle inequality to the three independent branch errors in Eqs. equation 107–equation 109 gives

$$\|J_{SSM} - J_s\|_2 \le M_{cls} M_{fuse} \left[6M\epsilon_J + 3\epsilon_J^2 \right], \tag{112}$$

where $J_{SSM,s}$ denotes the corresponding Jacobian obtained by replacing each SSM depth-specific Jacobian with its server-side counterpart. The important point is that the depth-specific errors enter this bound additively rather than through products of Jacobians from subsequent depths.

More generally, for D independently generated branches, the first-order parallel error satisfies

$$\|J_{par} - J_s\|_2 \le C_{cls,fuse} D M \epsilon_J + O(\epsilon_J^2), \tag{113}$$

where $C_{cls,fuse}$ is a constant determined by the Jacobian norms of the shared classification and fusion operations. Unlike Eq. equation 102, this bound contains only a single depth-specific Jacobian factor M.

Comparing Eqs. equation 102 and equation 113, the first-order error propagation scales as

$$\boxed{\mathcal{E}_{seq} = O\left(D M^{D-1} \epsilon_J\right), \qquad \mathcal{E}_{par} = O\left(D M \epsilon_J\right).} \tag{114}$$

Hence, when the Jacobian bound $M > 1$, the sequential architecture introduces a depth-dependent multiplicative amplification factor of M^{D-2} relative to the corresponding linear-in-depth parallel bound, up to the shared fusion and classification constants. The parallel architecture therefore removes the cross-depth Jacobian error accumulation inherent in sequential composition.

Proposition. Under the uniform Jacobian error bound in Eq. equation 92 and the bounded server-side Jacobians in Eq. equation 93, the first-order Jacobian approximation error of a D-depth sequential architecture scales as $O(DM^{D-1}\epsilon_J)$, whereas that of the proposed parallel architecture scales as $O(DM\epsilon_J)$, up to the shared fusion and classification constants. Therefore, the parallel architecture avoids multiplicative error accumulation across depths and provides a more stable Jacobian approximation for surrogate gradient estimation.

A.2.6 A Convergence Bound for Asynchronous Server Optimization

We next characterize the convergence behavior of the asynchronous server-side optimization in StreamSplit under bounded feature staleness and bounded utility-based selection error.

Let

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_s} \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) \tag{115}$$

denote the ideal server-side gradient at server iteration t. Since StreamSplit reuses the latest retained feature in the LFT rather than requiring a newly generated feature at every server update, the feature used at iteration t may have been generated at an earlier client iteration. We first state the assumptions needed to bound the resulting error.

We assume that the server-side gradient is Lipschitz continuous with respect to the input feature. For any two features f_1 and f_2,

$$\|\nabla_{\boldsymbol{\theta}_s} \mathcal{L}^s_{\text{task}}(f_1) - \nabla_{\boldsymbol{\theta}_s} \mathcal{L}^s_{\text{task}}(f_2)\|_2 \le L_f \|f_1 - f_2\|_2 . \tag{116}$$

We further assume that the client-generated feature changes by at most B_f between two consecutive client-side updates:

$$\left\| f_i^{k+1} - f_i^k \right\|_2 \le B_f. \tag{117}$$

Finally, let τ_i^t denote the age of the feature from client i when it is used by the server at iteration t. We assume bounded LFT staleness:

$$0 \le \tau_i^t \le \Delta. \tag{118}$$

[Bounded LFT Staleness Error] Under Eqs. equation 116–equation 118, the gradient deviation caused by using a retained feature satisfies

$$\mathbb{E}\left[\left\| \mathbf{e}^t_{\text{stale}} \right\|_2^2 \right] \le L_f^2 B_f^2 \Delta^2. \tag{119}$$

Proof. Suppose that the feature used by the server at iteration t was generated at iteration $t - \tau_i^t$. By the triangle inequality,

$$\left\| f_i^t - f_i^{t-\tau_i^t} \right\|_2 \le \sum_{k=t-\tau_i^t}^{t-1} \left\| f_i^{k+1} - f_i^k \right\|_2 . \tag{120}$$

Applying the bounded feature-drift assumption in Eq. equation 117 to each term on the right-hand side gives

$$\left\| f_i^t - f_i^{t-\tau_i^t} \right\|_2 \le \sum_{k=t-\tau_i^t}^{t-1} B_f \ = \tau_i^t B_f. \tag{121}$$

Since $\tau_i^t \le \Delta$ by Eq. equation 118,

$$\left\| f_i^t - f_i^{t-\tau_i^t} \right\|_2 \le B_f \Delta. \tag{122}$$

The stale gradient error is the difference between the gradient induced by the retained feature and that induced by the corresponding current feature. Therefore, applying the gradient-feature Lipschitz condition in Eq. equation 116 gives

$$\left\| \mathbf{e}^t_{\text{stale}} \right\|_2 \le L_f \left\| f_i^t - f_i^{t-\tau_i^t} \right\|_2 \ \le L_f B_f \Delta. \tag{123}$$

Squaring both sides of Eq. equation 123 yields

$$\left\| \mathbf{e}^t_{\text{stale}} \right\|_2^2 \le L_f^2 B_f^2 \Delta^2. \tag{124}$$

Taking expectation on both sides gives Eq. equation 119. □

The utility-based selection may additionally select a feature whose server-side optimization effect differs from that of an ideal feature. We therefore assume that this selection-induced gradient deviation is bounded:

$$\mathbb{E}\left[\left\|\mathbf{e}^t_{\text{selection}}\right\|_2^2\right] \le \epsilon^2_{\text{selection}}. \tag{125}$$

This assumption does not require the utility rule to select the globally optimal feature. Instead, $\epsilon_{\text{selection}}$ characterizes the gradient-level deviation introduced by utility-based selection within the current LFT.

We further assume bounded stochastic gradient variance. Let $\widehat{\mathbf{g}}_t$ denote the stochastic estimate of the ideal server-side gradient. Then

$$\mathbb{E}\left[\widehat{\mathbf{g}}_t \mid \boldsymbol{\theta}^t_s\right] = \mathbf{g}_t \tag{126}$$

and

$$\mathbb{E}\left[|\widehat{\mathbf{g}}_t - \mathbf{g}_t|_2^2 \mid \boldsymbol{\theta}^t_s\right] \le \sigma^2. \tag{127}$$

The resulting server update can be expressed by separating stochastic noise, LFT staleness, and feature-selection error. Define the total gradient error as

$$\boldsymbol{\delta}_t = \widehat{\mathbf{g}}_t - \mathbf{g}_t + \mathbf{e}^t_{\text{stale}} + \mathbf{e}^t_{\text{selection}}. \tag{128}$$

Then the server-side update is

$$\boldsymbol{\theta}^{t+1}_s = \boldsymbol{\theta}^t_s - \eta_s\left(\mathbf{g}_t + \boldsymbol{\delta}_t\right). \tag{129}$$

[Non-convex Convergence under Bounded Asynchrony] Suppose that $\mathcal{L}^s_{\text{task}}$ is lower bounded by $\mathcal{L}^{s,\text{inf}}_{\text{task}}$ and has L_s-Lipschitz gradients. Under Eqs. equation 126–equation 127, the assumptions of Lemma A.2.6, and the bounded selection error in Eq. equation 125, for a learning rate satisfying $\eta_s \le 1/(4L_s)$, the server-side optimization satisfies

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla_{\boldsymbol{\theta}_s}\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s)\right\|_2^2\right] \le & \frac{4\left(\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^0_s) - \mathcal{L}^{s,\text{inf}}_{\text{task}}\right)}{\eta_s T} \\ & + 9\sigma^2 + 9L_f^2 B_f^2 \Delta^2 + 9\epsilon^2_{\text{selection}}.\end{aligned} \tag{130}$$

Proof. Since $\mathcal{L}^s_{\text{task}}$ has L_s-Lipschitz gradients, the standard smoothness inequality gives

$$\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s) \le \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) + \left\langle \mathbf{g}_t, \boldsymbol{\theta}^{t+1}_s - \boldsymbol{\theta}^t_s\right\rangle + \frac{L_s}{2}\left\|\boldsymbol{\theta}^{t+1}_s - \boldsymbol{\theta}^t_s\right\|_2^2. \tag{131}$$

Substituting the update in Eq. equation 129 into Eq. equation 131 gives

$$\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s) \le \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) - \eta_s\left\langle \mathbf{g}_t, \mathbf{g}_t + \boldsymbol{\delta}_t\right\rangle + \frac{L_s\eta_s^2}{2}\left\|\mathbf{g}_t + \boldsymbol{\delta}_t\right\|_2^2. \tag{132}$$

Using

$$\langle \mathbf{a}, \mathbf{b}\rangle \ge -\frac{1}{2}\|\mathbf{a}\|_2^2 - \frac{1}{2}\|\mathbf{b}\|_2^2, \tag{133}$$

with $\mathbf{a} = \mathbf{g}_t$ and $\mathbf{b} = \boldsymbol{\delta}_t$, we obtain

$$\langle \mathbf{g}_t, \boldsymbol{\delta}_t\rangle \ge -\frac{1}{2}\|\mathbf{g}_t\|_2^2 - \frac{1}{2}\|\boldsymbol{\delta}_t\|_2^2. \tag{134}$$

Therefore,

$$\langle \mathbf{g}_t, \mathbf{g}_t + \boldsymbol{\delta}_t\rangle \ge \frac{1}{2}\|\mathbf{g}_t\|_2^2 - \frac{1}{2}\|\boldsymbol{\delta}_t\|_2^2. \tag{135}$$

Similarly, by

$$\|\mathbf{a} + \mathbf{b}\|_2^2 \le 2\|\mathbf{a}\|_2^2 + 2\|\mathbf{b}\|_2^2, \tag{136}$$

we have

$$\|\mathbf{g}_t + \boldsymbol{\delta}_t\|_2^2 \le 2\|\mathbf{g}_t\|_2^2 + 2\|\boldsymbol{\delta}_t\|_2^2. \tag{137}$$

Substituting Eqs. equation 135 and equation 137 into Eq. equation 132 gives

$$\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s) \le \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) - \frac{\eta_s}{2}\left\|\mathbf{g}_t\right\|_2^2 + \frac{\eta_s}{2}\left\|\boldsymbol{\delta}_t\right\|_2^2 + L_s\eta_s^2\left\|\mathbf{g}_t\right\|_2^2 + L_s\eta_s^2\left\|\boldsymbol{\delta}_t\right\|_2^2. \tag{138}$$

Since $\eta_s \le 1/(4L_s)$,

$$L_s\eta_s^2 \le \frac{\eta_s}{4}. \tag{139}$$

Applying this inequality to Eq. equation 138 yields

$$\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s) \le \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) - \frac{\eta_s}{4}\left\|\mathbf{g}_t\right\|_2^2 + \frac{3\eta_s}{4}\left\|\boldsymbol{\delta}_t\right\|_2^2. \tag{140}$$

Taking expectation on both sides gives

$$\mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s)\right] \le \mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s)\right] - \frac{\eta_s}{4}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] + \frac{3\eta_s}{4}\mathbb{E}\left[\left\|\boldsymbol{\delta}_t\right\|_2^2\right]. \tag{141}$$

By the definition of $\boldsymbol{\delta}_t$ in Eq. equation 128 and the inequality

$$\left\|\mathbf{a}+\mathbf{b}+\mathbf{c}\right\|_2^2 \le 3\left\|\mathbf{a}\right\|_2^2 + 3\left\|\mathbf{b}\right\|_2^2 + 3\left\|\mathbf{c}\right\|_2^2, \tag{142}$$

we obtain

$$\mathbb{E}\left[\left\|\boldsymbol{\delta}_t\right\|_2^2\right] \le 3\mathbb{E}\left[\left\|\widehat{\mathbf{g}}_t - \mathbf{g}_t\right\|_2^2\right] + 3\mathbb{E}\left[\left\|\mathbf{e}^t_{\text{stale}}\right\|_2^2\right] + 3\mathbb{E}\left[\left\|\mathbf{e}^t_{\text{selection}}\right\|_2^2\right]. \tag{143}$$

Applying the variance bound in Eq. equation 127, the staleness bound from Lemma A.2.6, and the selection-error bound in Eq. equation 125 gives

$$\mathbb{E}\left[\left\|\boldsymbol{\delta}_t\right\|_2^2\right] \le 3\sigma^2 + 3L_f^2B_f^2\Delta^2 + 3\epsilon^2_{\text{selection}}. \tag{144}$$

Substituting Eq. equation 144 into Eq. equation 141 gives

$$\begin{aligned}\mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s)\right] \le{}& \mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s)\right] - \frac{\eta_s}{4}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] \\ &+ \frac{9\eta_s}{4}\sigma^2 + \frac{9\eta_s}{4}L_f^2B_f^2\Delta^2 + \frac{9\eta_s}{4}\epsilon^2_{\text{selection}}.\end{aligned} \tag{145}$$

Rearranging Eq. equation 145 gives

$$\begin{aligned}\frac{\eta_s}{4}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] \le{}& \mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s)\right] - \mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^{t+1}_s)\right] \\ &+ \frac{9\eta_s}{4}\sigma^2 + \frac{9\eta_s}{4}L_f^2B_f^2\Delta^2 + \frac{9\eta_s}{4}\epsilon^2_{\text{selection}}.\end{aligned} \tag{146}$$

Summing Eq. equation 146 from $t=0$ to $T-1$ and using the telescoping property gives

$$\begin{aligned}\frac{\eta_s}{4}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] \le{}& \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^0_s) - \mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^T_s)\right] \\ &+ \frac{9T\eta_s}{4}\sigma^2 + \frac{9T\eta_s}{4}L_f^2B_f^2\Delta^2 + \frac{9T\eta_s}{4}\epsilon^2_{\text{selection}}.\end{aligned} \tag{147}$$

Since $\mathcal{L}^s_{\text{task}}$ is lower bounded,

$$\mathbb{E}\left[\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^T_s)\right] \ge \mathcal{L}^{s,\text{inf}}_{\text{task}}. \tag{148}$$

Substituting this lower bound into Eq. equation 147 gives

$$\frac{\eta_s}{4}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] \le \mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^0_s) - \mathcal{L}^{s,\text{inf}}_{\text{task}} + \frac{9T\eta_s}{4}\sigma^2 + \frac{9T\eta_s}{4}L_f^2B_f^2\Delta^2 + \frac{9T\eta_s}{4}\epsilon^2_{\text{selection}}. \tag{149}$$

Dividing both sides by $\eta_s T/4$ yields

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\mathbf{g}_t\right\|_2^2\right] \le \frac{4\left(\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^0_s) - \mathcal{L}^{s,\text{inf}}_{\text{task}}\right)}{\eta_s T} + 9\sigma^2 + 9L_f^2B_f^2\Delta^2 + 9\epsilon^2_{\text{selection}}. \tag{150}$$

Finally, substituting

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_s}\mathcal{L}^s_{\text{task}}(\boldsymbol{\theta}^t_s) \tag{151}$$

into Eq. equation 150 gives Eq. equation 130. □

Discussion. Theorem A.2.6 separates three sources of deviation from ideal server-side optimization: stochastic gradient noise, feature staleness caused by asynchronous LFT reuse, and utility-based feature selection. The staleness term is explicitly controlled by Lemma A.2.6. In particular, the resulting contribution grows quadratically with the maximum feature age Δ, reflecting the accumulation of bounded feature drift over stale iterations.

When $\Delta = 0$ and $\epsilon_{\text{selection}} = 0$, the additional errors caused by asynchronous feature reuse and feature selection vanish, leaving the standard stochastic non-convex convergence characterization. For bounded Δ and $\epsilon_{\text{selection}}$, the server-side optimization converges to a neighborhood whose size is controlled by these error sources. Thus, retaining the latest feature from each client does not require exact synchronization with the current server state; instead, bounded feature age yields a bounded perturbation to the server-side gradient.

B END