NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

FRANCISCO JOSÉ ROSA FREITAS

BSc in Computer Science and Engineering

VERTICAL FEDERATED LEARNING IN SATELLITE CONSTELLATIONS FOR LOWER EARTH ORBIT

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2025

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

VERTICAL FEDERATED LEARNING IN SATELLITE CONSTELLATIONS FOR LOWER EARTH ORBIT

FRANCISCO JOSÉ ROSA FREITAS

BSc in Computer Science and Engineering

Adviser: Cláudia Soares
Assistant Professor, NOVA University Lisbon

Co-adviser: Pedro Valdeira
PhD, Carnegie Mellon University & Instituto Superior Técnico

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2025

Vertical Federated Learning in Satellite Constellations for Lower Earth Orbit

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.2.1) (**novathesis-manual**).

Abstract

The rapid expansion of Low Earth Orbit (LEO) satellite constellations has unlocked new possibilities for applying Machine Learning (ML) to critical global challenges such as disaster management, environmental monitoring, and secure communications. However, traditional centralized ML approaches are impractical in this context due to severe bandwidth constraints and sparse satellite-ground connectivity. Federated Learning (FL), a distributed learning paradigm, allows multiple entities to collaboratively train models while keeping the raw data local. Among various FL paradigms, Vertical Federated Learning (VFL)—in which different participants hold complementary features of shared data samples—emerges as a particularly suitable approach for LEO systems, as different satellites often gather distinct types of data.

Despite its potential, VFL encounters significant communication bottlenecks, impeding efficient model convergence in satellite networks. To address this, a novel communication-efficient VFL framework is proposed, drawing inspiration from two state-of-the-art methodologies. For the purpose of minimizing communication overhead while ensuring convergence, Error Feedback compressed Vertical Federated Learning (EFVFL) employs error feedback compression, while the FedSpace framework, operating in the horizontal FL setting—where different participants share the same set of features but hold different data samples—dynamically schedules model aggregation by leveraging deterministic satellite connectivity patterns, effectively tackling idleness and staleness issues.

This thesis aims to develop and validate a unified framework, leveraging ideas from EFVFL and FedSpace for LEO satellite constellations, where the proposed solution will be evaluated based on realistic simulations, assessing communication efficiency, convergence speed and scalability. Ultimately, by overcoming existing limitations, this research will contribute to the advance of FL-based applications in LEO environments.

Keywords: Low Earth Orbit satellites, Vertical Federated Learning, Federated Learning, Communication efficiency, Space-based machine learning

Resumo

A crescente expansão de constelações de satélites de baixa órbita terrestre (LEO) tem impulsionado novas oportunidades para a aplicação de Machine Learning (ML) em desafios globais críticos, como a gestão de desastres, a monitorização ambiental e a melhoria das comunicações. No entanto, as abordagens tradicionais de ML centralizado revelam-se impraticáveis neste contexto, devido às limitações de largura de banda e à conectividade intermitente entre satélites e estações terrestres. Federated Learning (FL), um paradigma de ML distribuído, permite que várias entidades treinem modelos colaborativamente, mantendo os dados brutos localmente. Entre os vários paradigmas de FL, destaca-se Vertical Federated Learning (VFL), no qual diferentes participantes detêm características complementares de amostras de dados partilhadas, sendo uma abordagem particularmente adequada para sistemas LEO, onde diferentes satélites recolhem distintos tipos de dados.

Apesar do seu potencial, VFL enfrenta alguns desafios relacionados com a comunicação, dificultando a convergência eficiente dos modelos em redes de satélites. Para abordar este problema, propõe-se um novo framework de VFL eficiente em termos de comunicação, inspirado em duas metodologias recentemente desenvolvidas. Para minimizar a sobrecarga de comunicação, garantindo simultaneamente a convergência, Error Feedback compressed Vertical Federated Learning (EFVFL) adota técnicas de compressão de erro, enquanto o framework FedSpace, que opera num contexto de FL horizontal—onde diferentes participantes partilham o mesmo conjunto de características, mas possuem amostras de dados distintas—agenda dinamicamente a agregação de modelos, tirando partido dos padrões determinísticos de conectividade dos satélites para mitigar problemas de idleness e staleness dos modelos.

Esta tese tem como objetivo desenvolver e validar um framework unificado que aproveite as ideias de EFVFL e de FedSpace para constelações de satélites LEO. A solução proposta será avaliada através de simulações realistas, analisando a eficiência de comunicação, a velocidade de convergência e a escalabilidade. Em última análise, ao superar as limitações existentes, esta investigação contribuirá para o avanço de aplicações de FL em ambientes espaciais LEO.

Palavras-chave: Satélites de Baixa Órbita Terrestre, Vertical Federated Learning, Federated Learning, Comunicação Eficiente, Machine Learning em Ambiente Espacial

Contents

List of Figures

List of Algorithms

Glossary

idleness A state in which a satellite (or client) does not contribute to the updates of the global model despite being connected and available. Idleness reflects unused communication opportunities and can reduce the overall efficiency of the distributed system. *(pp. v, 10)*

staleness A measure of how outdated an update is in the context of federated learning. It represents the difference between the current time step and the last time step an update from a client was included in the global model. Staleness can negatively impact model convergence if not properly mitigated. *(pp. v, 7–10, 27)*

Acronyms

CVFL Compressed Vertical Federated Learning *(pp. 11, 13–15, 27, 28, 30)*

EFVFL Error Feedback compressed Vertical Federated Learning *(pp. 1, 11–15, 17, 18, 25, 27–30, 33)*

FL Federated Learning *(pp. 1, 7, 8, 10, 11)*

HFL Horizontal Federated Learning *(pp. 1, 3, 5, 10)*

IID Independent and Identically Distributed *(p. 10)*

LEO Low Earth Orbit *(pp. 1–3, 5, 7, 10, 15)*

ML Machine Learning *(p. 1)*

non-IID non-Independent and Identically Distributed *(p. 3)*

SGD Stochastic Gradient Descent *(p. 26)*

SVFL Standard Vertical Federated Learning *(pp. 13, 14, 30)*

VFL Vertical Federated Learning *(pp. 1–5, 7, 10, 11, 33)*

1

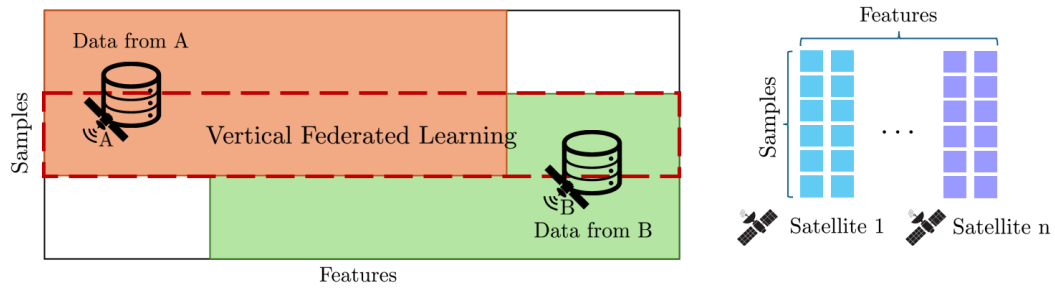
Introduction

Low Earth Orbit (LEO) systems generate large volumes of distributed data, such as imagery and sensor readings. Centralized Machine Learning (ML) approaches require transmitting raw data between satellites and ground stations for processing, but this is impractical in LEO due to bandwidth constraints and intermittent connectivity. Federated Learning (FL) offers a solution by enabling clients to collaboratively train models without sharing raw data.

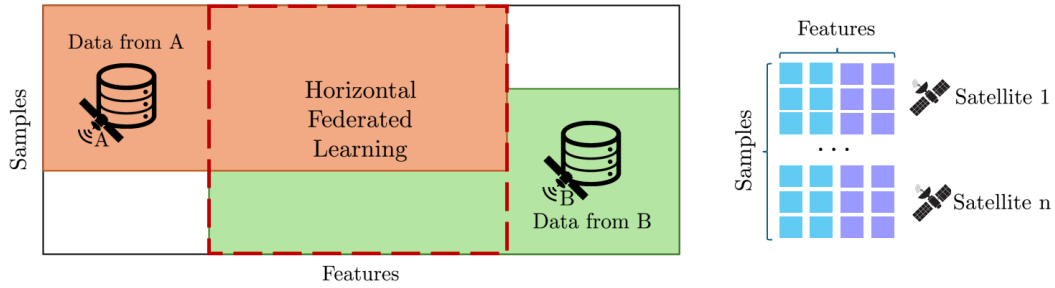
FL can be categorized into Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). HFL involves training models collaboratively across clients with different data samples but the same feature set, while VFL focuses on clients holding complementary features for shared data samples, as shown in Figure 1.1.

Although HFL has been widely studied and applied in different scenarios, VFL is less explored (L. Li et al. 2020). In satellite constellations, each satellite can collect specialized types of data based on its onboard sensors, such as spectral imaging, atmospheric measurements, or radar signals. Complementarity may also arise when satellites observe different geographic footprints within a shared time window, so that feature blocks reflect both sensing modality and spatial coverage. This results in a scenario where different satellites hold non-overlapping feature sets for the same geographical regions or events. Such heterogeneity makes VFL a natural fit for collaborative training in LEO networks, where combining diverse data sources can lead to more comprehensive and accurate models.

Despite its suitability, implementing VFL introduces significant communication overhead challenges. Frequent exchanges of intermediate information between clients and the server during training can slow down the process and even emerge as a key bottleneck (Dean et al. 2012; Lian et al. 2017). To address these challenges, state-of-the-art methods have been proposed. For instance, Error Feedback compressed Vertical Federated Learning (EFVFL) (Valdeira et al. 2025) enhances communication efficiency by employing lossy compression and error feedback techniques to reduce the volume of data exchanged during training.



(a) VFL: Clients share the same data samples but hold different sets of features.



(b) HFL: Clients share the same set of features but hold different data samples.

Figure 1.1: Illustration of VFL and HFL in distributed machine learning systems. (Q. Yang et al. 2019)

Beyond VFL-specific approaches, frameworks like FedSpace tackle another communication challenge, by optimizing aggregation schedules through leveraging the predictable and deterministic nature of satellite connectivity in LEO constellations. While not specific to VFL, FedSpace demonstrates the potential to improve communication efficiency across federated learning setups in satellite networks. Together, these advancements form the foundation for this thesis, which aims to design a communication-efficient VFL framework tailored to the unique challenges of LEO satellite networks.

1.1 Foundations of Federated Learning

1.1.1 Horizontal Federated Learning

For a horizontally partitioned dataset $\mathcal{D} = \{\xi_n\}_{n=1}^{N}$ with N samples, as in Figure 1.1b, all clients share the same feature space, while the data are partitioned by samples across clients. Each sample ξ_n is held by exactly one client, so client k holds a local subset $\mathcal{S}_k = \{\xi_n\}_{n\in\mathcal{I}_k}$, with $\mathcal{I}_i \cap \mathcal{I}_j = \varnothing$ for $i \neq j$ and $\bigcup_k \mathcal{I}_k = \{1, \ldots, N\}$. The seminal work FedAvg (McMahan et al. 2017) improves communication efficiency by averaging model updates at a central server after multiple local updates. This approach is widely used due to its simplicity and scalability.

Although HFL is applicable to various scenarios, many realistic use cases in LEO constellations are better aligned with the VFL paradigm, where satellites not only collect similar types of data from different regions but also gather complementary data types from the same regions, resulting in feature-based data partitioning. While FedAvg has been shown to achieve good performance, even on non-Independent and Identically Distributed (non-IID) data distributions (X. Li et al. 2019), HFL is fundamentally designed for settings with identical feature sets across clients.

1.1.2 Vertical Federated Learning

In VFL, for a global dataset $\mathcal{D} = \{\xi_n\}_{n=1}^N$ with N samples and distributed across K clients, each sample is partitioned into feature blocks $\xi_n = (\xi_{n1}, \ldots, \xi_{nK})$. Each client k holds the k-th feature block across all samples, $\mathcal{D}_k = \{\xi_{nk}\}_{n=1}^N$. For every sample n, client k computes a local embedding $\mathbf{h}_k(\mathbf{x}_k; \xi_{nk}) \in \mathbb{R}^{E_k}$, where $\mathbf{x}_k$ are the parameters of client k. Stacking across samples gives

$$\mathbf{H}_k(\mathbf{x}_k; \mathcal{D}_k) = \begin{bmatrix} \mathbf{h}_k(\mathbf{x}_k; \xi_{1k}) \\ \mathbf{h}_k(\mathbf{x}_k; \xi_{2k}) \\ \vdots \\ \mathbf{h}_k(\mathbf{x}_k; \xi_{Nk}) \end{bmatrix} \in \mathbb{R}^{N \times E_k}, \qquad k = 1, \ldots, K.$$

Here, E_k denotes the embedding dimension at client k.

When unambiguous, we emphasize optimization variables and write $\mathbf{H}_k(\mathbf{x}_k)$ for $\mathbf{H}_k(\mathbf{x}_k; \mathcal{D}_k)$ and $\mathbf{h}_{kn}(\mathbf{x}_k)$ for $\mathbf{h}_k(\mathbf{x}_k; \xi_{nk})$.

At the server, we define for $n = 1, \ldots, N$ the per-sample server block $\mathbf{h}_{0n}(\mathbf{x}_0) = \mathbf{x}_0 \in \mathbb{R}^{E_0}$ and

$$\mathbf{H}_0(\mathbf{x}_0) = \begin{bmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_0 \end{bmatrix} = \mathbf{1}_N \, \mathbf{x}_0^\top \in \mathbb{R}^{N \times E_0}.$$

Concatenating the blocks column-wise, we obtain the global embedding

$$\mathbf{H}(\mathbf{x}) := \big[\mathbf{H}_0(\mathbf{x}_0),\, \mathbf{H}_1(\mathbf{x}_1),\, \ldots,\, \mathbf{H}_K(\mathbf{x}_K)\big] \in \mathbb{R}^{N \times E}, \qquad E = \sum_{k=0}^{K} E_k.$$

Equivalently, the n-th row of $\mathbf{H}(\mathbf{x})$ is $\mathbf{h}_n(\mathbf{x})^\top$.

Let ℓ denote the task loss (e.g., cross-entropy for classification or squared error for regression), y_n the ground-truth label for sample n that is available at the server and to all clients, under the relaxed protocol (Valdeira et al. 2025), and $\psi : \mathbb{R}^E \to \mathbb{R}^{|\mathcal{Y}|}$, the server-side fusion model (e.g., a linear head). We write $\psi(\mathbf{x}_0, \mathbf{h}_{1n}(\mathbf{x}_1), \ldots, \mathbf{h}_{Kn}(\mathbf{x}_K))$ to emphasize its dependence on the client embeddings and server parameters.

We define the per-sample composite (loss $\circ$ fusion) as

$$\phi_n\big(\mathbf{x}_0, \mathbf{h}_{1n}(\mathbf{x}_1), \ldots, \mathbf{h}_{Kn}(\mathbf{x}_K)\big) := \ell\big(\psi(\mathbf{x}_0, \mathbf{h}_{1n}(\mathbf{x}_1), \ldots, \mathbf{h}_{Kn}(\mathbf{x}_K)),\, y_n\big).$$

Equivalently, this coincides with the notation $\phi_n\big(\mathbf{h}_n(\mathbf{x})\big) = \ell\big(\psi(\mathbf{h}_n(\mathbf{x})), y_n\big)$ used earlier.

The server-side map Φ applied to the stacked embedding is

$$\Phi\big(\mathbf{H}(\mathbf{x})\big) := \frac{1}{N}\sum_{n=1}^{N} \phi_n\big(\mathbf{x}_0, \mathbf{h}_{1n}(\mathbf{x}_1), \ldots, \mathbf{h}_{Kn}(\mathbf{x}_K)\big),$$

and the global objective is

$$f(\mathbf{x}) \;:=\; \Phi\big(\mathbf{H}(\mathbf{x})\big). \tag{1.1}$$

The server computes the gradient of $\Phi\big(\mathbf{H}(\mathbf{x})\big)$ with respect to its own parameters $\mathbf{x}_0$ but does not have access to the local parameters $\{\mathbf{x}_k\}_{k=1}^{K}$ of clients. Instead, it computes the gradient with respect to the aggregated global embedding $\mathbf{H}(\mathbf{x})$ and transmits the necessary block-wise activation gradients back to the clients.

Each client k then uses this information to compute its own gradient using the chain rule, with all derivatives evaluated at iteration t:

$$\nabla_{\mathbf{x}_k} \Phi\big(\mathbf{H}(\mathbf{x}^t)\big) \;=\; \frac{\partial \Phi}{\partial \mathbf{H}_k}\big(\mathbf{H}(\mathbf{x}^t)\big) \;\cdot\; \frac{\partial \mathbf{H}_k}{\partial \mathbf{x}_k}\big(\mathbf{x}_k^t\big), \qquad k = 1, \ldots, K. \tag{1.2}$$

Finally, each client performs a local update using gradient descent:

$$\mathbf{x}_k^{t+1} = \mathbf{x}_k^t - \eta\, \nabla_{\mathbf{x}_k} \Phi\big(\mathbf{H}(\mathbf{x}^t)\big),$$

where η is the learning rate.

The process of split learning in a standard VFL setting is illustrated in Figure 1.2. At each training step, the clients compute and transmit their local embeddings to the central server, which concatenates them into a global embedding. The server then computes the block-wise activation gradients $\partial\Phi/\partial\mathbf{H}_k$ and sends them back to the clients, enabling local updates of the parameters $\mathbf{x}_k$. This collaborative approach ensures that the global model benefits from the complementary feature sets held by the clients.

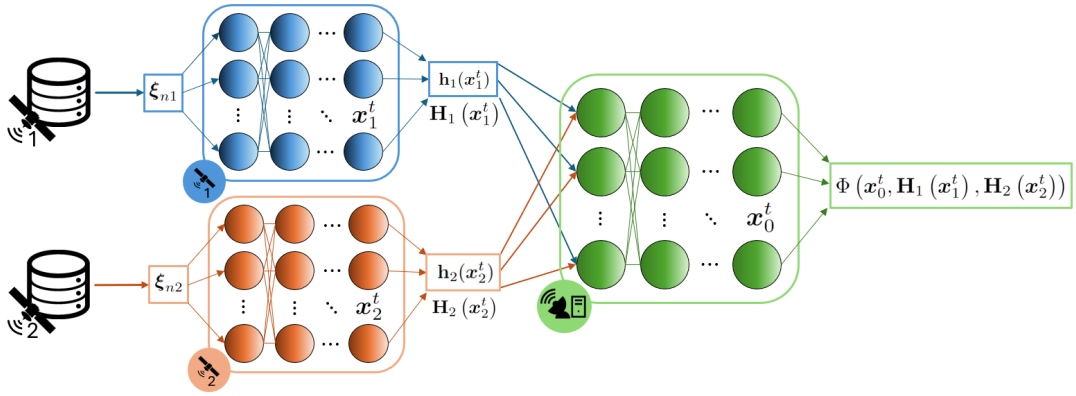


Figure 1.2: Split learning in a standard VFL setting. Example for two clients.

1.2 Contributions

This thesis advances the state of federated learning for LEO satellite constellations by uniting communication-efficient VFL with schedule-aware aggregation. The main contributions are:

- We design a framework that combines error-feedback compressed VFL (EFVFL) with a FedSpace-inspired planner. The framework exploits deterministic connectivity to schedule aggregations while using error-feedback with surrogate embeddings to reduce communication load, thereby addressing both idleness/staleness and payload size.
- We adapt schedule optimization—originally formulated for HFL—to the VFL split-learning pipeline. This includes: (i) time-slotted aggregation decisions aligned with contact windows, (ii) staleness tracking tailored to vertically partitioned embeddings and their compressed errors, and (iii) crediting rules that determine when client transmissions contribute to an aggregation step.
- We formalize a payload model for VFL communications that consistently accounts for embeddings, compressed errors, and server-to-client signals. Using this model, we report learning curves with accuracy as a function of communication (MB), enabling fair comparisons across compressors and schedulers.
- We implement a simulation workflow that integrates VFL training with deterministic LEO connectivity. The study spans multiple schedulers (synchronous, asynchronous, buffered asynchronous, and FedSpace-style planning) and compression families (direct compression and error-feedback), under partial participation.
- We evaluate communication efficiency, convergence behavior, and robustness under aggressive sparsification. The results show that error-feedback maintains stable optimization under strong compression and that integrating EFVFL with connectivity-aware planning substantially reduces the number of transmitted megabytes required to reach target accuracies, compared with unplanned or uncompressed baselines.

To the best of our knowledge, this is the first work to integrate error-feedback VFL with deterministic, connectivity-aware scheduling for LEO constellations, providing a unified, communication-efficient solution evaluated under realistic connectivity dynamics. The code supporting this work is available at `https://github.com/FranciscoJRFreitas/FedSpace-EFVFL`.

2

Background and Related Work

This chapter explores FL frameworks that address space-specific challenges and communication efficiency. We first examine frameworks that utilize deterministic satellite connectivity to optimize model aggregation in space-based networks. Subsequently, we explore communication-efficient methods, with a particular focus on their application in VFL.

2.1 FL in Space

FL in space presents unique challenges due to limited connectivity, constrained bandwidth, and the heterogeneous nature of LEO satellite networks, where different satellites often collect different types of data. Efficient scheduling and aggregation strategies are essential to ensure effective training of global models while minimizing model staleness. Several FL frameworks have been proposed to address these challenges, with FedSpace (So et al. 2022) emerging as a leading solution. In this section, we explore FedSpace, focusing on its dynamic scheduling mechanisms and their role in optimizing communication and training efficiency in satellite networks.

In this setting, each satellite k maintains a local dataset $\mathcal{D}_k$ and contributes to learning a shared global model $\mathbf{w}$ by minimizing a global objective function:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) = \sum_{k=1}^{K} \frac{m_k}{m} f_k(\mathbf{w}) \right\} \tag{2.1}$$

where $f_k(\mathbf{w})$ is the local objective function dependent on dataset $\mathcal{D}_k$, and $m_k = |\mathcal{D}_k|$ represents the number of local samples stored at satellite k. The total dataset size is $m = \sum_{k=1}^{K} m_k$, and training aims to minimize the aggregate loss across all satellites.

A central feature of FedSpace is its dynamic aggregation scheduling, which balances satellite activity through **staleness** $s_k^{(t)}$:

$$s_k^{(t)} = t - \tau_k^{(t)} \tag{2.2}$$

where t is the current time step and $\tau_k^{(t)}$ is the most recent time step when the update of satellite k was included in the global model.

2.1.1 Deterministic Connectivity

FedSpace utilizes the predictable nature of satellite orbits and the rotation of the Earth to precompute communication patterns between satellites and ground stations. In an Earth-centered inertial coordinate system, each satellite $k \in \mathcal{K} = \{1, \ldots, K\}$ and ground station $g \in \mathcal{G} = \{K+1, \ldots, K+G\}$ follows a fixed trajectory $\mathbf{r}_k(t)$ and $\mathbf{r}_g(t)$, respectively. A valid communication link between satellite k and ground station g is established when the elevation angle $\alpha_{k,g}(t)$ meets the condition:

$$\alpha_{k,g}(t) = \frac{\pi}{2} - \angle\big(\mathbf{r}_g(t), \mathbf{r}_k(t) - \mathbf{r}_g(t)\big) \geq \alpha_{\min},$$

where $\angle(\mathbf{a}, \mathbf{b})$ represents the angle between vectors $\mathbf{a}$ and $\mathbf{b}$, and $\alpha_{\min}$ is the minimum elevation angle required for connectivity.

To improve scheduling, FedSpace discretizes time into intervals of length T_0, where each interval is indexed by $i \in \{0, 1, \ldots, N\}$. The connectivity of satellites at time i is described by a **connectivity set**:

$$C_i = \{k \in \mathcal{K} \mid \text{satellite } k \text{ is connected to any ground station during } [iT_0, (i+1)T_0)\} \quad (2.3)$$

The sequence of connectivity sets $C = \{C_0, C_1, \ldots, C_N\}$ captures the deterministic and time-varying communication patterns. This allows FedSpace to precompute future connectivity sets C_i with high accuracy.

2.1.2 Optimization of Aggregation Schedules

In a FL setting involving LEO satellites, updates from different satellites arrive at varying time intervals due to inconsistent connectivity. Some updates are more relevant than others, depending on when they were generated and how they influence the global model. Consequently, an efficient aggregation schedule must determine which updates should be incorporated while balancing the trade-off between timely updates and communication efficiency.

To achieve this, FedSpace employs a dynamic scheduling strategy that selects updates for aggregation based on two primary factors. The first is **staleness** s_l, which quantifies the delay of an update by measuring the time elapsed since it was last incorporated into the global model. The second is the **training status** θ_l, which represents the current state of the global model at the time of aggregation, ensuring that only meaningful updates contribute to the learning process.

Over a planning horizon of L slots, an aggregation plan is a binary vector $\mathbf{a} = (a_1, a_2, \ldots, a_L)$ representing the aggregation schedule, with each entry $a_l \in \{0, 1\}$ indicating whether aggregation occurs at time step l ($a_l = 1$) or not ($a_l = 0$). The set of

selected aggregation time steps is

$$\mathcal{I}_{\text{agg}}(\mathbf{a}) = \{\, l \mid a_l = 1 \,\}. \tag{2.4}$$

With these considerations in mind, FedSpace determines the optimal aggregation schedule by maximizing a **utility function** that prioritizes updates based on their staleness and training status over a planning horizon of L slots:

$$\arg \max_{\mathbf{a} \in \{0,1\}^L} \sum_{l \in \mathcal{I}_{\text{agg}}(\mathbf{a})} u(s_l, \theta_l). \tag{2.5}$$

Here, the utility function $u(s_l, \theta_l)$ ensures that updates that contribute significantly to model convergence are prioritized, while outdated updates with minimal impact are discarded or weighted accordingly. By structuring aggregation in this way, FedSpace optimally selects updates to improve model performance while maintaining communication efficiency.

2.1.3 Staleness Compensation

The global model update is performed by aggregating the local gradients from the selected satellites at an aggregation slot. In FedSpace, staleness influences both when aggregation occurs (via the scheduling utility; Section 2.1.2) and how much each participating update contributes at aggregation time through a **staleness compensation** term (So et al. 2022):

$$c\left(s_k^{(t)}\right) \;=\; \left(s_k^{(t)} + 1\right)^{-\alpha}, \tag{2.6}$$

which satisfies $c(0) = 1$ and decreases monotonically with s (Xie, Koyejo, and Gupta 2019). The hyperparameter $\alpha > 0$ controls how aggressively stale contributions are downweighted.

At each time step t, the update direction from satellite k is the local gradient

$$\mathbf{g}_k^{(t)} \;=\; \nabla f_k\left(\mathbf{w}^{(t)}\right). \tag{2.7}$$

When the FedSpace schedule triggers aggregation, the server updates the global model using staleness-normalized weights over the participating set $\mathcal{S}^{(t)}$:

$$\mathbf{w}^{(t+1)} \;=\; \mathbf{w}^{(t)} \;-\; \eta \sum_{k \in \mathcal{S}^{(t)}} \gamma_k^{(t)}\, \mathbf{g}_k^{(t)}, \qquad \gamma_k^{(t)} \;:=\; \frac{c\left(s_k^{(t)}\right)}{\sum_{j \in \mathcal{S}^{(t)}} c\left(s_j^{(t)}\right)}. \tag{2.8}$$

Here, $\eta > 0$ is the learning rate. The coefficients $\{\gamma_k^{(t)}\}$ form a convex combination that depends only on staleness.

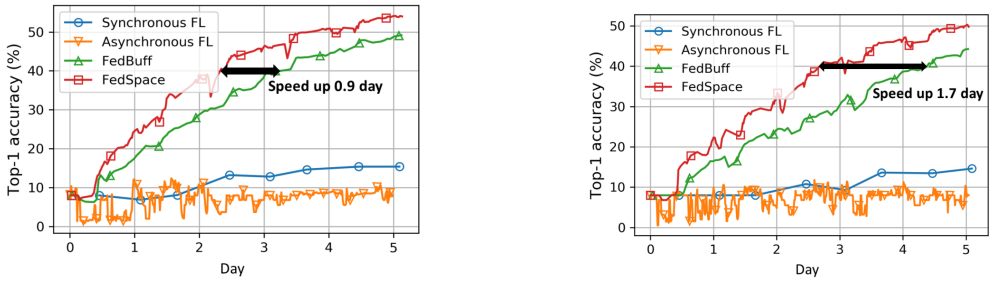
2.1.4 FL Frameworks Comparison

FedSpace and other existing federated learning frameworks, such as synchronous[1] FL (FedAvg) (McMahan et al. 2017), asynchronous FL (Xie, Koyejo, and Gupta 2019), and buffered asynchronous FL (FedBuff) (Nguyen et al. 2022), operate within an HFL paradigm.

While effective in scenarios with frequent and reliable communication, FedAvg faces challenges in LEO environments, where connectivity is sparse and intermittent (Kim et al. 2024). On the other hand, asynchronous FL improves upon this by allowing clients to update the global model without waiting for all participants, thus reducing idle time. However, pure asynchronous updates often lead to significant model staleness, which can degrade convergence.

FedBuff introduces a buffered asynchronous aggregation mechanism, where client updates are collected in a buffer and aggregated only when a sufficient number of updates are available. This approach strikes a balance between synchronous and asynchronous methods, maintaining reasonable staleness. However, it remains unclear whether it achieves the most effective trade-offs in scheduling decisions (So et al. 2022).

FedSpace, in contrast, dynamically schedules aggregation based on the deterministic and time-varying connectivity of satellites and ground stations. By leveraging satellite orbits and the rotation of the Earth, it accurately calculates future connectivity patterns to make informed trade-offs between idleness and staleness. This minimizes idle time while controlling staleness to levels that do not compromise convergence. Figure 2.1b shows empirical results demonstrating that FedSpace reduces training time by 1.7 days (38.6%) compared to FedBuff in non-Independent and Identically Distributed (IID) settings, achieving significant speedups while maintaining accuracy.



(a) IID setting.

(b) Non-IID setting.

Figure 2.1: Comparison of the Top-1 validation accuracy of DenseNet-161 on the fMoW dataset using realistic satellite networks with 12 ground stations and 191 satellites. (Figure from (So et al. 2022))

However, while FedSpace effectively handles sparse connectivity in HFL settings, it lacks the necessary adaptations for VFL, where clients hold different feature sets of the

[1] FedAvg performs synchronous aggregation, where clients periodically send their model updates to a central server for averaging.

same data samples. This limitation motivates the integration of FedSpace with state-of-the-art VFL methods, which we now discuss.

2.2 Communication-Efficient VFL

Communication efficiency in FL—and consequently in VFL—can be improved through several complementary methods, including increasing local computation to reduce the number of rounds (local updating / periodic averaging), resource-aware client selection and scheduling, reducing or deferring model updates, decentralized or peer-to-peer training to alleviate server bottlenecks, and compressing the communicated objects (model parameters, gradients, or embeddings) (Shahid et al. 2021). In this section, we focus specifically on compression for VFL and analyze two families: direct compression (Sec. 2.2.1) and error-feedback compression (Sec. 2.2.2).

2.2.1 Direct Compression Approaches

Recent advancements in VFL address challenges of communication efficiency. To reduce the communication load, lossy compression can be applied, representing the information in the matrix embeddings $\mathbf{H}_k(\mathbf{x}_k^t)$ using fewer bits.

A straightforward approach is to directly apply a compression function $C(\mathbf{H}_k)$, transmitting the compressed embeddings to the server, which is the compression mechanism followed by Compressed Vertical Federated Learning (CVFL) (Castiglia et al. 2022).

2.2.2 Error Feedback Compression in VFL

In contrast with direct compression approaches, EFVFL (Valdeira et al. 2025) transmits only a compressed error between clients and the server, instead of compressing the full matrix embeddings $\mathbf{H}_k(\mathbf{x}_k^t)$ directly. Each client k and the server maintain matching compression estimates $\{\mathbf{G}_\ell^t\}_{\ell=0}^K$ that act as surrogates for $\{\mathbf{H}_\ell(\mathbf{x}_\ell^t)\}_{\ell=0}^K$. At round t, client k uploads a compressed error $\mathbf{C}_k^t$, so the per-round uplink (client-to-server) budget matches direct compression (e.g., top-k sends k entries). The server then forms the collection $\{\mathbf{C}_\ell^t\}_{\ell=0}^K$ (including its own $\mathbf{C}_0^t$) and broadcasts this full set to all clients, so every machine can synchronize its matching compression estimates $\{\mathbf{G}_\ell^t\}$. Note that, typically, the communication bottleneck is the uplink communication, rather than the server-side broadcasting (Haddadpour et al. 2021). Formal definitions follow in Sec. 2.2.2.1.

2.2.2.1 Surrogate Embeddings and Updates

The surrogate embedding is recursively updated as:

$$\mathbf{G}_k^t := \mathbf{G}_k^{t-1} + \mathbf{C}_k^t \tag{2.9}$$

where the compressed error is defined as:

$$\mathbf{C}_k^t := C\left(\mathbf{H}_k(\mathbf{x}_k^t) - \mathbf{G}_k^{t-1}\right) \tag{2.10}$$

where C is an α-contractive compressor, i.e., there exists $\alpha \in (0, 1]$ such that, for all $v \in \mathbb{R}^d$,

$$\mathbb{E}\left\|C(v) - v\right\|^2 \leq (1 - \alpha)\|v\|^2,$$

with the expectation taken with respect to the (possible) randomness in C. In the uncompressed case ($\alpha = 1$), C is the identity map, so $\mathbf{C}_k^t = \mathbf{H}_k(\mathbf{x}_k^t) - \mathbf{G}_k^{t-1}$ and the update $\mathbf{G}_k^t = \mathbf{G}_k^{t-1} + \mathbf{C}_k^t$ yields $\mathbf{G}_k^t = \mathbf{H}_k(\mathbf{x}_k^t)$.

2.2.2.2 Gradient Computation with Surrogate Functions

EFVFL thus computes a surrogate function $\Phi(\mathbf{x}_0^t, \mathbf{G}_1^t, \ldots, \mathbf{G}_K^t)$ at the server, instead of directly computing the original objective function, as illustrated in Figure 2.2.

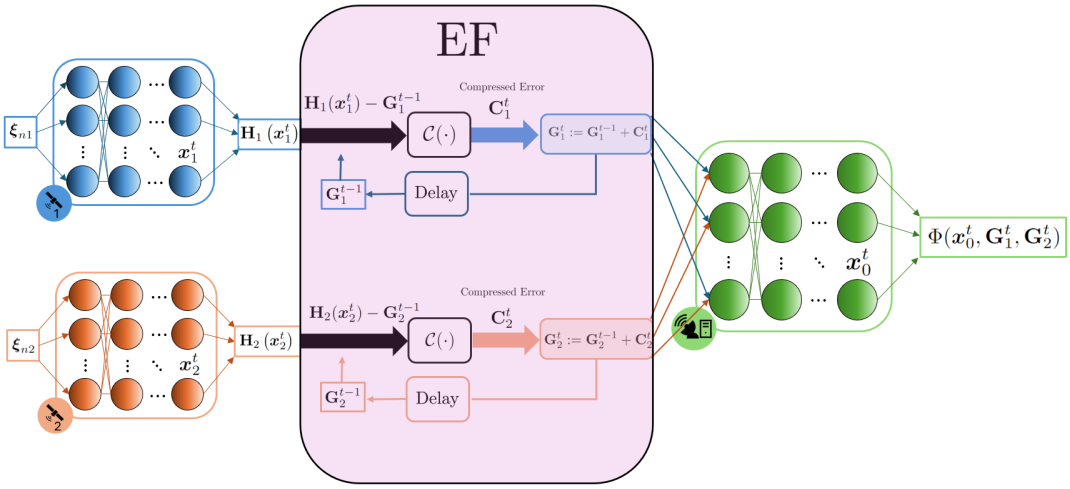


Figure 2.2: Compressed error feedback applied to split learning in a VFL setting. Example with two clients.

2.2.2.3 Backward Pass and Gradient Update

During the backward pass, clients use the following surrogate for the gradient:

$$\mathbf{g}_k^t = \sum_{i=1}^{N} \sum_{j=1}^{E_k} \left[\widetilde{\nabla}_k^t \Phi\right]_{ij} \left[\nabla \mathbf{H}_k(\mathbf{x}_k^t)\right]_{ij:} \tag{2.11}$$

where $\widetilde{\nabla}_k^t \Phi := \nabla_k \Phi(\ldots, \mathbf{G}_{k-1}^t, \mathbf{H}_k(\mathbf{x}_k^t), \mathbf{G}_{k+1}^t, \ldots)$ represents the gradient surrogate and depends on the compressed surrogates $\mathbf{G}_\ell^t$ for $\ell \neq k$, while $\nabla \mathbf{H}_k(\mathbf{x}_k^t)$ is the derivative of the embedding matrix with respect to the local parameters. In this equation, $\left[\widetilde{\nabla}_k^t \Phi\right]_{ij}$ represents the scalar element at the i-th row and j-th column of the surrogate gradient matrix $\widetilde{\nabla}_k^t \Phi$. This scalar indicates the contribution of the embedding of the i-th sample at the j-th feature dimension to the overall global gradient. The term $\left[\nabla \mathbf{H}_k(\mathbf{x}_k^t)\right]_{ij:}$ refers

to the vector along the third dimension of the gradient matrix $\nabla \mathbf{H}_k(\mathbf{x}_k^t)$, specifically corresponding to the i-th sample and the j-th embedding feature. This vector contains the partial derivatives of the j-th feature of the embedding of the i-th sample with respect to the parameters $\mathbf{x}_k^t$ of the client model. Together, $\left[\widetilde{\nabla}_k^t \Phi\right]_{ij}$ and $\left[\nabla \mathbf{H}_k(\mathbf{x}_k^t)\right]_{ij:}$ are combined in the summation over all samples i and features j to compute the gradient contribution for client k.

If we stack all parameters into $\mathbf{x}$ and the block gradients into $\mathbf{g}^t$, the joint update can be compactly written as

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \mathbf{g}^t, \tag{2.12}$$

where $\mathbf{g}^t := (\mathbf{g}_0^t, \mathbf{g}_1^t, \ldots, \mathbf{g}_K^t)$ aggregates the contributions from the server and all clients, and η is the step size. Operationally, each block update is performed locally and in parallel.

After the local parameter updates, each client $k \in \{1, \ldots, K\}$ forms a new compressed error with the updated parameters and the previous surrogate,

$$\mathbf{C}_k^{t+1} := C\left(\mathbf{H}_k(\mathbf{x}_k^{t+1}) - \mathbf{G}_k^t\right),$$

and sends $\mathbf{C}_k^{t+1}$ to the server. The server ($k = 0$) similarly computes its own $\mathbf{C}_0^{t+1}$ locally. It then broadcasts the full set $\{\mathbf{C}_\ell^{t+1}\}_{\ell=0}^K$, after which every machine (server and clients) refreshes all surrogates:

$$\mathbf{G}_\ell^{t+1} = \mathbf{G}_\ell^t + \mathbf{C}_\ell^{t+1}, \qquad \ell = 0, \ldots, K.$$

(Recall that, when forming the next-round gradient surrogate $\widetilde{\nabla}_k^{t+1}\Phi$ for client k, the local block uses $\mathbf{H}_k(\mathbf{x}_k^{t+1})$, while the other blocks use $\mathbf{G}_\ell^{t+1}$ for all $\ell \neq k$.)

2.2.3 Communication-Efficient VFL Methods Comparison

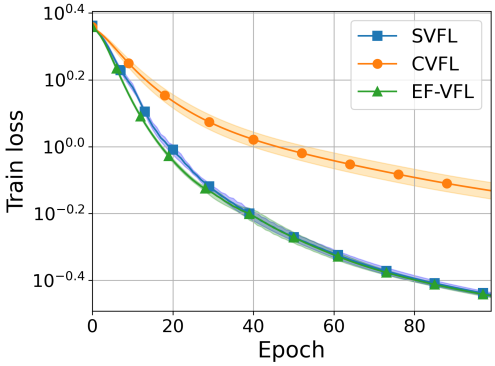
Standard Vertical Federated Learning (SVFL) (Liu et al. 2022) incurs a communication cost of $O(N \cdot E \cdot T)$ due to transmitting the full embeddings $\mathbf{H}_k(\mathbf{x}_k^t)$ at each iteration over T training rounds. Concretely, at each round every client k uploads a matrix of size $N \times E_k$, and the server broadcasts quantities of the same order. Summing across clients yields a per-round payload proportional to $N \cdot E$.

The prior state-of-the-art CVFL (Castiglia et al. 2022) method relies on direct compression but requires the compression error to vanish over time to achieve convergence, resulting in either increased communication per iteration or a suboptimal convergence rate of $O(1/\sqrt{T})$.

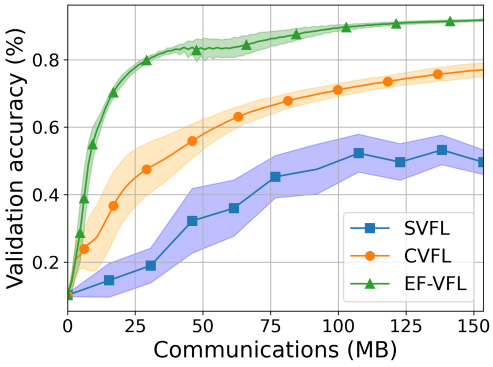
EFVFL achieves a significant improvement over prior methods in both communication efficiency and convergence rates. Unlike SVFL (Liu et al. 2022), which incurs a communication cost of $O(N \cdot E \cdot T)$ due to transmitting the full embeddings $\mathbf{H}_k(\mathbf{x}_k^t)$ at each iteration, EFVFL reduces this overhead by sending only compressed errors $\mathbf{C}_k^t$. This significantly lowers the per-iteration communication cost, while matching the $O(1/T)$ convergence rate of uncompressed methods, even with non-vanishing compression errors. The $O(1/T)$

convergence rate means that the error decreases inversely proportional to the number of iterations T.

The series of the following experiments were conducted using the settings described in (Valdeira et al. 2025) and implemented with the available codebase from the authors. For the next examples, we train a shallow neural network with only one hidden layer on the MNIST digit recognition dataset (LeCun et al. 1998). For the compression method, CVFL and EFVFL employ a top-k sparsification keeping 10% (Figure 2.3), 1% (Figure 2.4), and 0.1% (Figure 2.5) of the entries, while SVFL is the same throughout.

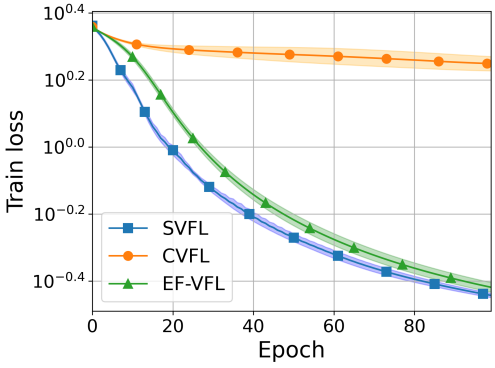


(a) Train loss per epoch.

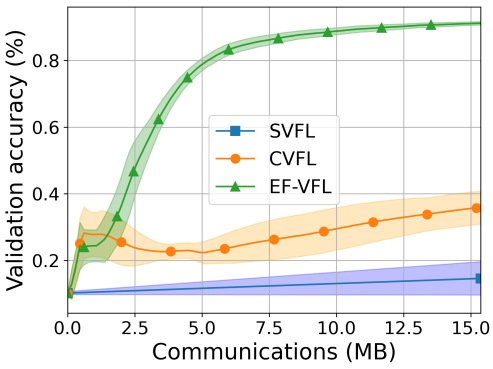


(b) Validation accuracy per communication cost.

Figure 2.3: Top-k keeping 10% of the entries.

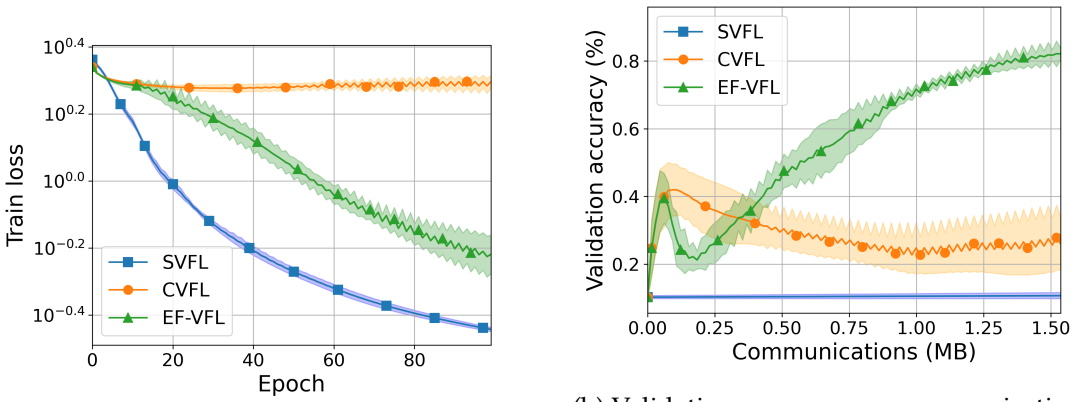


(a) Train loss per epoch.



(b) Validation accuracy per communication cost.

Figure 2.4: Top-k keeping 1% of the entries.



(a) Train loss per epoch.

(b) Validation accuracy per communication cost.

Figure 2.5: Top-k keeping 0.1% of the entries.

These results show that, as compression becomes more aggressive in CVFL, the performance deteriorates, as seen by the increasing train loss and stagnant validation accuracy, despite lower communication costs. This outcome is attributed to the direct sparsification approach used by CVFL, which can lead to considerable information loss, particularly at higher compression rates.

On the other hand, EFVFL tends to converge faster for a given communication budget, consistently achieving lower training loss and higher validation accuracy with minimal performance decline due to its error-feedback strategy. This balance of reduced communication overhead and fast convergence makes EFVFL a promising choice for LEO networks.

3

The FedSpace-EFVFL Framework

This chapter describes how we integrate the communication-efficient VFL scheme with a FedSpace-style aggregation scheduler. Rather than re-deriving the theory, we instantiate it using the notation and results introduced in Chapters 1–2.

3.1 Assumptions

We consider training under the relaxed label protocol (labels available to all clients) on a dataset of size N, using mini-batches of size B. Throughout, $\|\cdot\|$ denotes the Euclidean norm. The analysis in this chapter proceeds under the following standing assumptions, aligned with the EFVFL convergence conditions.

- **Contractive compressor.** There exists $\alpha \in (0, 1]$ such that for all $v \in \mathbb{R}^d$,

  $$\mathbb{E}\,\|C(v) - v\|^2 \;\le\; (1 - \alpha)\,\|v\|^2,$$

  where the expectation is over the (possible) randomness of the compressor, taken independent across its applications and independent of mini-batch sampling. In the uncompressed case ($\alpha = 1$), C is the identity. In our experiments the uplink compressor is Top-k sparsification, which is known to satisfy the above contractive property under the Euclidean norm, with a contraction factor proportional to k/d for d-dimensional vectors (and exactly $\alpha = k/d$ for random-k). (Valdeira et al. 2025)

- **Smoothness.** We instantiate Φ as a differentiable server-side map whose gradient is Lipschitz on the parameter region explored during training (e.g., a linear head paired with a smooth loss). On bounded inputs and with weight decay constraining parameters to a compact set, the resulting empirical objective f is L-smooth on the iterates we visit. (Valdeira et al. 2025)

- **Bounded representation derivatives.** Each client map $\mathbf{H}_k$ has a uniformly bounded Jacobian with respect to its local parameters x_k:

  $$\sup_{x_k} \left\| \nabla_{x_k} \mathbf{H}_k(x_k) \right\| \;\le\; H, \qquad \text{where } \|\cdot\| \text{ is the operator norm induced by the Euclidean norm.}$$

For a single-layer representation $\mathbf{H}_k(x_k) = \mathrm{ReLU}(W_k Z_k)$ applied row-wise to a batch (with local features Z_k), the parameter-Jacobian $\nabla_{x_k}\mathbf{H}_k(x_k)$ equals a ReLU mask (entries in $\{0,1\}$) times the linear map induced by Z_k, so $\|\nabla_{x_k}\mathbf{H}_k(x_k)\| \le c\,\|Z_k\|$ for some layer-dependent constant c. With input normalization (hence $\|Z_k\| \le Z_{\max}$) and regularization/early stopping keeping parameters in a compact set, there is a uniform $H < \infty$ such that $\sup_{x_k} \|\nabla_{x_k}\mathbf{H}_k(x_k)\| \le H$. (Valdeira et al. 2025)

- **Mini-batch stochasticity.** Let $\mathcal{F}_t := \sigma\big(G^0, x^1, G^1, \ldots, x^t, G^t\big)$, where $G^t := \{G_0^t, \ldots, G_K^t\}$, denote the sigma-algebra generated by all information available at the start of round t (past iterates, surrogates, and past randomness). We write $\mathbf{g}^t$ for the surrogate full-batch update used in EFVFL at round t (i.e., the block-gradient built by combining $\mathbf{H}_k(x_k^t)$ for the local block and $\{G_\ell^t\}_{\ell\neq k}$ for the other blocks; cf. Sec. 2.2.2.1). Let $\tilde{\mathbf{g}}^t$ be the stochastic update computed from a mini-batch B^t of size B. We assume:

$$\mathbb{E}\big[\tilde{\mathbf{g}}^t \mid \mathcal{F}_t\big] \;=\; \mathbf{g}^t, \qquad \mathbb{E}\big\|\tilde{\mathbf{g}}^t - \mathbf{g}^t\big\|^2 \;\le\; \frac{\sigma^2}{B},$$

  where the conditional expectation is taken over the fresh randomness at round t (mini-batch sampling and, when applicable, compressor randomness), independent of past history $\mathcal{F}_t$. Mini-batches are sampled uniformly at random, yielding an unbiased estimator of the surrogate update; bounded inputs and controlled parameter norms give finite second moments and the stated $O(1/B)$ variance bound. The surrogate bias relative to the true full-batch gradient $\nabla f(x^t)$ is handled in the EFVFL analysis via a separate stability term. (Valdeira et al. 2025)

- **Step size.** The learning rate satisfies $0 < \eta \le \eta_{\max}(L, H, \alpha, K)$, as required by the EFVFL analysis. We choose η within the stability range established by the EFVFL convergence proof (on the order of $1/L$, with dependence on H, α, K) and verify an initial monotone decrease of the training loss, indicating operation within the safe regime. (Valdeira et al. 2025)

3.2 Model Architecture (Split Network)

We adopt a split client-server architecture, as in LEO constellations, satellites act as clients that collect and process data onboard, while the ground station serves as the server that aggregates their updates. This mirrors a vertically partitioned setting, where each satellite holds a distinct subset of features and communicates with the GS (server) over short, bandwidth-limited uplink (client to server) windows. Sending compact client embeddings, and under EFVFL only compressed errors, reduces uplink load and improves robustness to intermittent connectivity.

Client side. Each client $k \in \mathcal{K}$ implements a representation model $\mathbf{h}_k(\cdot)$ that maps its feature block to a vector in $\mathbb{R}^{E_k}$ (we denote E_k as the embedding (*cut*) dimension of client

k, as used in Chapter 2). To reduce uplink cost we apply the EFVFL surrogate mechanism, where clients maintain $\mathbf{G}_k$ and transmit the compressed error $\mathbf{C}_k$, as explained in 2.2.2.1.

Server side. The server concatenates the K client embeddings into $\mathbf{h}_n(\mathbf{x}) \in \mathbb{R}^{\sum_{k=1}^{K} E_k}$ and applies a linear head to $\mathbb{R}^{|\mathcal{Y}|}$. Training minimizes the same empirical objective $f(\mathbf{x})$, introduced in 1.1.

3.3 Connectivity, Slots, and Aggregation

We adopt the time discretization and connectivity sequence $C = \{C_0, \ldots, C_{I_0-1}\}$ from Chapter 2. A plan for one window is a binary vector $\mathbf{a} \in \{0,1\}^{I_0}$ with aggregation set $\mathcal{I}_{\text{agg}}(\mathbf{a})$. When an aggregation occurs at slot $\ell \in \mathcal{I}_{\text{agg}}(\mathbf{a})$, we compute the staleness vector $\mathbf{s}_\ell$.

Union buffer. To reduce idleness while respecting connectivity, we maintain a union upload buffer $\mathcal{U}$ that accumulates clients observed online since the last aggregation; only clients in $\mathcal{U}$ are *credited* when aggregation happens.

3.4 Error-Feedback with Surrogates and Slot Execution

We follow the EFVFL surrogate mechanism from Chapter 2 (Section 2.2.2): each client k and the server keep synchronized surrogates $\mathbf{G}_k \in \mathbb{R}^{N \times E_k}$ and exchange compressed residuals $\mathbf{C}_k = C(\mathbf{H}_k - \mathbf{G}_k)$.

Training proceeds in slots with mini-batches of size B (the last batch may be smaller). For slot t, let $\mathcal{B}_t$ be the set of batch indices processed in that slot. For client k and batch $b \in \mathcal{B}_t$, let $\mathcal{I}_{t,b} \subset \{1, \ldots, N\}$ denote the sample indices of batch (t, b). We define the batch embedding slice and the corresponding surrogate slice as

$$\mathbf{H}_k^{(t,b)} \;:=\; \left[\mathbf{h}_{ki}(\mathbf{x}_k^t)\right]_{i \in \mathcal{I}_{t,b}} \in \mathbb{R}^{|\mathcal{I}_{t,b}| \times E_k}, \qquad \mathbf{G}_k^{(t,b)} \;:=\; \mathbf{G}_k[\mathcal{I}_{t,b}, :] \in \mathbb{R}^{|\mathcal{I}_{t,b}| \times E_k},$$

and the per-batch compressed residual as

$$\mathbf{C}_k^{(t,b)} \;:=\; C\big(\mathbf{H}_k^{(t,b)} - \mathbf{G}_k^{(t,b)}\big).$$

Let $t_0 < t$ be the index of the previous aggregation slot. For each client k, we buffer pairs $\big(\mathcal{I}_{t,b}, \mathbf{C}_k^{(t,b)}\big)$ during non-aggregation slots. Formally,

$$\mathcal{Q}_k \;:=\; \{(r,b) : r \in \{t_0+1, \ldots, t\},\; b \in \mathcal{B}_r\}$$

collects the mini-batches buffered for client k since the last aggregation.

If $a_t = 1$ (aggregation at the end of slot t), we transmit the buffered pairs $\big(\mathcal{I}_{r,b}, \mathbf{C}_k^{(r,b)}\big)$ for the credited clients $k \in \mathcal{U}$, and update surrogates on all machines via the row-wise operation

$$\mathbf{G}_k[\mathcal{I}_{r,b}, :] \;:=\; \mathbf{G}_k[\mathcal{I}_{r,b}, :] \;+\; \mathbf{C}_k^{(r,b)} \quad \text{for all } (r,b) \in \mathcal{Q}_k \text{ and } k \in \mathcal{U}.$$

No server-side surrogate update occurs when $a_t = 0$; only local buffering of the pairs continues.

Per-slot execution and accounting. At the start of slot t, observe the connectivity set C_t and update the union buffer

$$\mathcal{U} := \mathcal{U} \cup C_t,$$

which accumulates the clients observed online since the last aggregation. The scheduler then:

- **Skip** ($a_t = 0$): perform forward/backward passes, buffer $\left(\mathcal{I}_{t,b}, \mathbf{C}_k^{(t,b)}\right)$ locally; do not take an optimizer step and do not modify server-side surrogates.
- **Aggregate** ($a_t = 1$): accumulate gradients within the slot, perform one optimizer step at its end, then push the buffered pairs for $k \in \mathcal{U}$, apply the surrogate updates above, account communication by the chosen compressor, clear $\mathcal{U}$, and update client staleness as in Eq. 2.2 (Chapter 2) with credited clients receiving $s_k = \rho - \tau_k - 1$ and non-credited clients marked $s_k = -1$.

3.5 Planning on Deterministic Connectivity

Plan selection follows FedSpace utility maximization. Given deterministic connectivity $C = \{C_0, \ldots, C_{I_0-1}\}$, a binary plan $\mathbf{a} \in \{0,1\}^{I_0}$ induces aggregation indices $\mathcal{I}_{\text{agg}}(\mathbf{a})$ and credited sets (via the union buffer), which determine the staleness vectors $\{\mathbf{s}_\ell\}_{\ell \in \mathcal{I}_{\text{agg}}(\mathbf{a})}$. We score a plan by

$$U(\mathbf{a} \mid \zeta) = \sum_{\ell \in \mathcal{I}_{\text{agg}}(\mathbf{a})} \hat{u}(\mathbf{s}_\ell, \zeta),$$

where ζ is the validation loss at the start of the window.

At each window boundary we sample an initial plan and refine it with a budget-limited random search (n_{rand} evaluations). We constrain the number of aggregations per window via $\|\mathbf{a}\|_0 \in [n_{\min}, n_{\max}]$ and only consider slots with nonempty connectivity ($|C_t| > 0$), sampling them with probability proportional to $|C_t|$. Each candidate is scored by simulating the union buffer and staleness evolution from the current last-credited rounds $\{\tau_k\}$ to obtain the $\{\mathbf{s}_\ell\}$ and then computing $U(\mathbf{a} \mid \zeta)$.

3.6 Learning the Utility Regressor from Logs

To approximate the utility function $u(s, \theta)$ (Chapter 2), we build a supervised dataset of triplets $(\mathbf{s}, \theta, \Delta f)$ from past training runs:

- $\mathbf{s} \in \{-1, 0, 1, \ldots\}^K$: staleness vector at an aggregation round.
- $\theta \in \mathbb{R}$: training status (we use the most recent validation loss before the update).

- $\Delta f \in \mathbb{R}$: *immediate* loss reduction across that update.

We then fit a regression model $\hat{u} : \mathbb{R}^{K+1} \to \mathbb{R}$ that predicts $\widehat{\Delta f}$ from the input feature vector $[\mathbf{s}; \theta]$. At run time, candidate plans $\mathbf{a}$ are scored via $\sum_{\ell \in \mathcal{I}_{\text{agg}}(\mathbf{a})} \hat{u}(\mathbf{s}_\ell, \theta)$.

3.7 Algorithms

Algorithm 1 and Algorithm 2 provide a complete, structured specification of the proposed FedSpace-EFVFL training procedure, integrating the assumptions (Sec. 3.1), split-network model (Sec. 3.2), slot-wise error-feedback with surrogate buffering (Sec. 3.4), and deterministic-connectivity planning with the learned utility (Secs. 3.5-3.6).

Scope and contributions. The core EFVFL update rules (surrogate maintenance and compressed-error exchange) follow (Valdeira et al. 2025). Building on an available EFVFL implementation, our contributions in this chapter are:

(i) a slot-based integration of EFVFL with deterministic connectivity;

(ii) a union-buffer/credited-set mechanism that accumulates arrivals across slots before aggregation;

(iii) staleness-aware weighting with an exponent α applied at aggregation time;

(iv) per-batch buffering of compressed residual slices and a row-wise surrogate update at aggregation;

(v) explicit communication accounting compatible with sparsification payloads;

(vi) an offline utility regressor trained from logs and a random-search planner that selects aggregation slots within windows;

These pieces together constitute the proposed FedSpace–EFVFL procedure presented below.

Algorithm 1 FedSpace–EFVFL. Markers (i)–(vi) reference the contributions listed above.

Input: initial parameters $\mathbf{x}^0 = (x_0^0, \ldots, x_K^0)$, step size η, cut size d_{cut}, compressor C, initial surrogates $G_k^0 = \mathbf{H}_k(x_k^0)$, window size I_0, connectivity $\{C_0, \ldots, C_{I_0-1}\}$, utility regressor $\hat{u}(\mathbf{s}, \theta)$, staleness exponent $\alpha \geq 0$.
State: union buffer $\mathcal{U} := \varnothing$; last credited round $\tau_k := -1$ for all k; aggregation round $\rho := 0$.
for epoch $e = 0, 1, \ldots$ **do**
  obtain current training status θ (e.g., latest validation loss)
  initialize a window plan $\mathbf{a} \in \{0,1\}^{I_0}$ by a random search that maximizes $\sum_{\ell \in I_{\text{agg}}(\mathbf{a})} \hat{u}(\mathbf{s}_\ell, \theta)$ under the deterministic $\{C_t\}$ ▷ **(vi)** planner using $\hat{u}$
  for slot $t = 0, \ldots, I_0 - 1$ **do** ▷ **(i)** slot-based integration with deterministic $\{C_t\}$
    $\mathcal{U} := \mathcal{U} \cup C_t$ ▷ **(ii)** union buffer / credited set accumulation
    if $a_t = 0$ **then** ▷ skip this slot
      continue
    ▷ Aggregate at the end of the slot
    define credited set for weighting $S_t := \mathcal{U}$ ▷ **(ii)** credited set at aggregation
    if $\alpha > 0$ and $S_t \neq \varnothing$ **then** ▷ **(iii)** staleness-aware weighting
      for $k \in S_t$: $s_k := \big(0$ if $\tau_k < 0$ else $\rho - \tau_k - 1\big)$
      $\lambda_k := \dfrac{(s_k + 1)^{-\alpha}}{\sum_{j \in S_t} (s_j + 1)^{-\alpha}}$
    Note: We omit data-size weights m_k because credited clients use equal batch sizes in VFL, so contributions are modulated by staleness only.
    else
      set $\lambda_k := 1$ for $k \in S_t$
    for each mini-batch b in slot t **do**
      build representations $\mathbf{H}_k$; only $k \in C_t$ are fresh in this slot
      accumulate gradients for clients $i \in C_t$ with loss weight λ_i; also accumulate one fusion-only pass ▷ **(iv)** per-batch processing; residuals can be buffered per-batch
    apply one optimizer step with step size η
    Communication & EF update (credited for $k \in \mathcal{U}$):
    for each $k \in \mathcal{U}$ in parallel **do**
      $C_k^{\rho+1} := C\big(\mathbf{H}_k(x_k^{\rho+1}) - G_k^\rho\big)$; send $C_k^{\rho+1}$ to the server ▷ **(iv)** per-batch slices may be concatenated; **(v)** payloads accounted
    server broadcasts $\{C_k^{\rho+1}\}_{k \in \mathcal{U}}$ to all clients;
    for each $k \in \mathcal{U}$ **do** ▷ server and all clients apply
      $G_k^{\rho+1} := G_k^\rho + C_k^{\rho+1}$ ▷ **(iv)** row-wise surrogate update
    for each $k \notin \mathcal{U}$ **do**
      $G_k^{\rho+1} := G_k^\rho$ ▷ unchanged
    compute staleness vector $\mathbf{s}_\rho$ with $s_k = -1$ if $k \notin \mathcal{U}$, else $s_k = \rho - \tau_k - 1$; set $\tau_k := \rho$ for $k \in \mathcal{U}$
    log $(\mathbf{s}_\rho, \theta, \Delta f_\rho)$ and the bytes for this round; $\rho := \rho + 1$; clear $\mathcal{U}$ ▷ **(v)** explicit communication accounting

Algorithm 2 Offline learning of the FedSpace utility $\hat{u}(\mathbf{s}, \theta)$ (implements contribution **(vi)**)

1: collect logs from past runs: staleness rows $\mathbf{s}_r$, training status θ_r, per-round losses f_r
2: align f_r to aggregation rounds; define $\Delta f_r = f_r - f_{r+1}$
3: fit a regression model on pairs $\big([\mathbf{s}_r; \theta_r],\ \Delta f_r\big)$ (random forest); deploy $\hat{u}$ ▷ **(vi)**

4

Experiments

In this section we empirically evaluate the proposed system that combines EFVFL with a FedSpace-style aggregation scheduler. We measure (i) optimization progress (train loss vs. epochs) and (ii) communication efficiency (validation accuracy vs. transmitted MB) under deterministic, slot-based connectivity. We follow the notation introduced in Chapters 1–2.

4.1 Setup

4.1.1 Architecture and feature partitioning

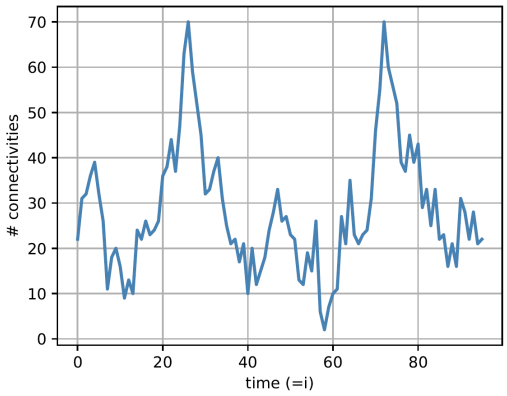
We use the MNIST digit-recognition dataset as our primary testbed because it offers a well-understood, low-variance benchmark that lets us isolate the effects of vertical partitioning and scheduling on the accuracy-communication trade-off, while keeping model capacity and training dynamics simple and reproducible across schedulers. Each client uses a single `Linear` layer $\mathbb{R}^{d_k} \to \mathbb{R}^{d_{\text{cut}}}$, a `ReLU`, and an EF compressor (`top-k sparsification`, configured per run). For the fusion model ψ, we concatenate the K client representations into $\mathbb{R}^{Kd_{\text{cut}}}$ and apply a linear head $\mathbb{R}^{Kd_{\text{cut}}} \to \mathbb{R}^{|\mathcal{Y}|}$.

MNIST feature partition. For MNIST (28×28 flattened to 784), we use $K = 149$ clients: 39 clients hold 6 pixels each and 110 clients hold 5 pixels each (summing to 784).

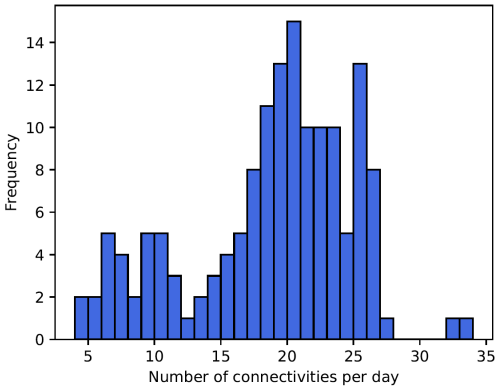
4.1.2 Data and connectivity schedule

We use real data of a LEO constellation from Planet Labs to derive connectivity. Clients represent 149 satellites in a constellation; 12 real ground stations are abstracted as a single FL server. Following the FedSpace methodology, we run the **COTE** simulator (Denby and Lucia 2020) to generate per-second line-of-sight logs over 24 hours. We discretize into $S = 96$ contiguous 15-minute slots and define the slot-level connectivity sequence $C = \{C_0, \ldots, C_{95}\}$, where $C_t \subseteq \mathcal{K}$ lists satellites deemed online in slot t.

A satellite is marked *connected* in a slot if and only if it was visible for at least 42.5% of that slot (at least 383 of 900 seconds). This threshold matches the qualitative connectivity structure used in FedSpace.



(a) Number of connected satellites ($|C_i|$).



(b) Histogram of $n_k = \sum_i \mathbb{1}\{k \in C_i\}$, $k \in \mathcal{K}$.

Figure 4.1: (a) Connected satellites per slot and (b) per-satellite connections across the day.

We observe two forms of heterogeneity:

- **Over time.** Figure 4.1(a) shows pronounced variability in the number of online satellites $|C_i|$, ranging from 4 to 70 within the day. Capacity arrives in *bursts*: extended low-availability stretches (< 15) are punctuated by short peaks (> 60).
- **Across satellites.** Figure 4.1(b) plots the per-satellite contact count n_k and reveals a broad distribution (5–33 per day), with most satellites in the 18-24 range and a long tail of rarely connected nodes.

Implications for FL. This pattern makes simplistic scheduling inefficient: strictly *synchronous* aggregation is throttled by the least-connected satellites and wastes peak slots, while fully *asynchronous* aggregation over-weights frequently connected satellites and accumulates staleness from infrequent ones. These plots motivate planning-based policies (e.g., FedSpace) that time aggregations near connectivity peaks and credit the union of clients across slots, balancing idleness and staleness.

4.1.3 Aggregation schedule

A *planning window* comprises I_0 consecutive slots, and an *aggregation plan* for one window is a binary vector

$$\boldsymbol{a} \;=\; (a_0, \ldots, a_{I_0-1}) \in \{0,1\}^{I_0},$$

with aggregation set $I_{\text{agg}}(\boldsymbol{a}) = \{\ell : a_\ell = 1\}$.

4.1.4 Model and optimization

Each client uses a single `Linear` layer $\mathbb{R}^{d_k} \to \mathbb{R}^{d_{\text{cut}}}$ with ReLU. We set the cut size $d_{\text{cut}} = 64$ (i.e., $E_k \equiv d_{\text{cut}}$ for all k). The server concatenates K representations (`conc`) and applies a linear head $\mathbb{R}^{Kd_{\text{cut}}} \to \mathbb{R}^{10}$ to match the number of output classes in MNIST for the digit recognition task. Loss is cross-entropy. We use Stochastic Gradient Descent (SGD),

learning rate 0.02, weight decay 10^{-4} and minibatch size 128, up to 100 epochs. One optimizer step is taken *at most once per slot* (at aggregation).

Client representations $\boldsymbol{h}_{nk}$ are compressed either by sending compressed embeddings (CVFL) or by maintaining surrogates $\boldsymbol{G}_k$ and transmitting the compressed error (EFVFL).

Schedulers (aggregation policies). We compare four policies on the same $\{C_i\}$:

- **Sync:** aggregate only when all K clients have been credited since the last aggregation.
- **FedBuff:** aggregate when the union buffer reaches $M = 96$ unique clients.
- **Async:** aggregate in any slot that has at least one online client (no unioning across slots).
- **FedSpace:** plan over windows of $I_0 = 96$ slots using a learned utility $u(\mathbf{s}, \theta)$ and an anytime search. The union buffer $\mathcal{U}$ credits all clients seen since the previous aggregation; communication is *charged only at aggregation* for the credited set.

From training logs we build triplets $(\mathbf{s}, \theta, \Delta f)$: per-round staleness vector $\mathbf{s}$, training status θ (current validation loss), and immediate loss drop Δf. A random-forest regressor maps $[\mathbf{s}; \theta] \mapsto \widehat{\Delta f}$ and is used online to score candidate plans.

4.1.5 Logging and evaluation metrics

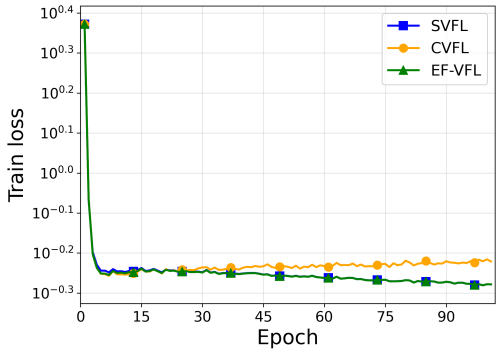
We report (i) the training loss over epochs and (ii) the validation accuracy as a function of the cumulative MB transmitted by credited clients at aggregation rounds (according to the payload model of the compressor).

4.2 Results

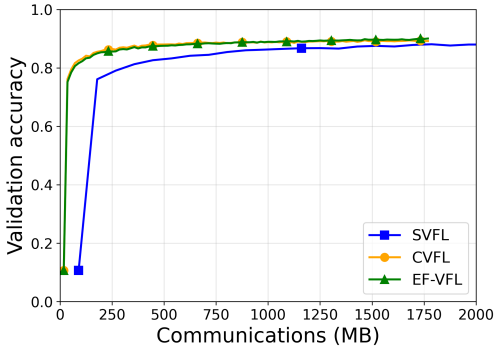
4.2.1 CVFL and EFVFL under Partial Participation

We start by fixing the aggregation policy to asynchronous partial participation—aggregating in any slot with arrivals—to emulate irregular satellite connectivity and the resulting staleness, while holding the scheduler fixed so that any performance gap can be attributed solely to the compression mechanism. We compare EFVFL against CVFL, both employing top-k sparsification, while keeping $p \in \{0.20, 0.05\}$ of the entries. (Figures. 4.2a–4.3b).

With **20%** compression keep rate, the two methods behave very similarly, as the accuracy-MB curves almost overlap, as observed in Figure 4.2b: at around 290 MB used, we see approximately **87.0%** for CVFL compared with **86.4%** for EFVFL. Even at the largest budget in this experiment (approximately 1.77 GB), EFVFL only edges ahead by <1% (about **89.9%** compared with **89.2%**). In short, for this compression the benefit of error feedback is muted with the staleness induced by partial participation, and the two methods are effectively tied across budgets.

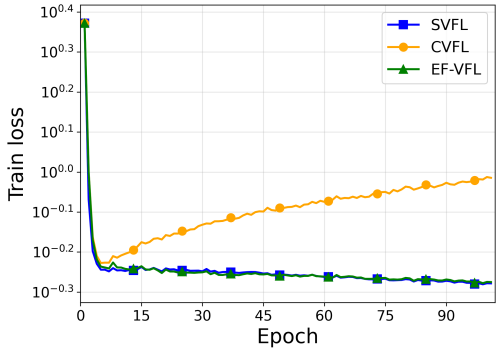


(a) Train loss per epoch (keep 20%).

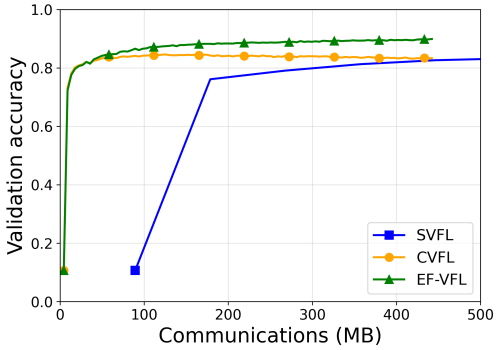


(b) Validation accuracy per MB (keep 20%).

Figure 4.2: Baseline comparison: SVFL, CVFL (direct), and EFVFL on MNIST with $K = 149$.



(a) Train loss per epoch (keep 5%).



(b) Validation accuracy per MB (keep 5%).

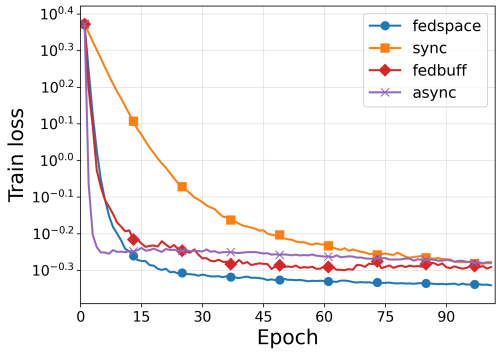
Figure 4.3: Aggressive compression: EFVFL maintains stable optimization and a steeper accuracy-MB slope than direct compression.

For **5%** of the entries kept, EFVFL consistently outperforms the direct baseline, but absolute accuracies are modest. As observed in Figure 4.3b, at approximately 71.5 MB EFVFL attains **85.5%** compared with **83.8%** for CVFL; around 170 MB the gap widens to **88.2%** compared with **84.5%**. By the end of the training (442 MB) EFVFL reaches **89.9%** while CVFL is at **83.1%**. Thus EFVFL provides a steady advantage across budgets, approaching, but not exceeding 90% within this range. We can further notice the divergence of CVFL under aggressive compression, with training loss blowing up and accuracy deteriorating.

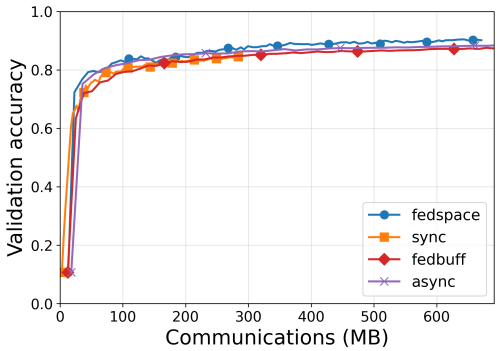
4.2.2 Scheduler comparison under EFVFL

In order to analyze scheduler effects in isolation, we keep the compression mechanism fixed (EFVFL) and vary only the aggregation policy. This reveals how scheduling alone trades idleness against staleness under deterministic connectivity. The four schedulers achieve similar peak accuracies while consuming very different communication budgets—visible in the clustered accuracy traces but widely separated positions along the MB axis.

For top-k sparsification keeping 20% of the entries, the best validation accuracies and



(a) Train loss per epoch (EFVFL, 20% top-k sparsification).



(b) Validation accuracy per MB (EFVFL, 20% top-k sparsification).

Figure 4.4: EFVFL with Sync, FedBuff (M=96), Async, and FedSpace (I_0=96).

total communication at the end of training are:

FedSpace: **90.6**% at **671 MB**,
Async: **89.9**% at **1,769 MB**,
FedBuff: **88.7**% at **1,269 MB**,
Sync: **84.5**% at **290 MB**.

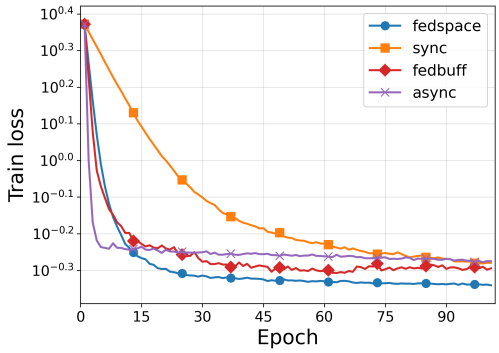
Reading the curves at a fixed target in the high-eighty percent to approximately ninety percent band (Fig. 4.4b), FedSpace reaches these accuracy values hundreds of MB earlier than the other schedulers: roughly a 2.6× reduction relative to Async and about 1.9× relative to FedBuff. Sync communicates the least but underperforms in accuracy within the same training horizon (plateauing mid-eighty percent), which aligns with its slower loss decay in Fig. 4.4a. In short, for near-equal accuracy (90%), FedSpace delivers large communication savings while avoiding the staleness of Async and the idleness of Sync, while adapting better than FedBuff.

Keeping only 5% of the entries and under stronger compression (Fig. 4.5), the previously observed behavior still remains, with the four curves staying close in accuracy, yet the communication needed to get there still separates the schedulers. Thus, the scheduling effect persists even when gradients are heavily compressed.

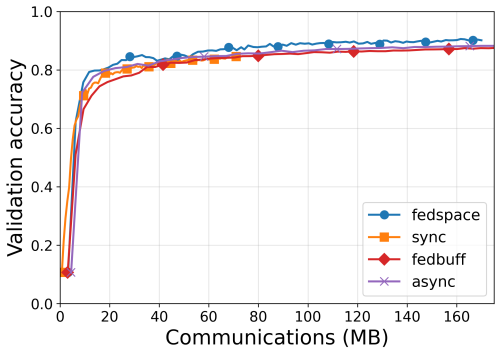
Scheduler Takeaway. Across both compression rates, the scheduler mostly shifts *how many MB* are needed to reach a given accuracy rather than the ultimate accuracy itself; FedSpace consistently moves the curve left (fewer MB for the same accuracy), while Async and FedBuff pay a large communication penalty to catch up, and Sync lags in accuracy for a fixed budget.

4.2.3 FedSpace-EFVFL Results

Figures 4.6a–4.7b isolate the effect of the FedSpace planner when paired with EFVFL. We combine FedSpace with EFVFL to test whether connectivity-aware planning amplifies the



(a) Train loss per epoch (EFVFL, 5% top-k sparsification).



(b) Validation accuracy per MB (EFVFL, 5% top-k sparsification).

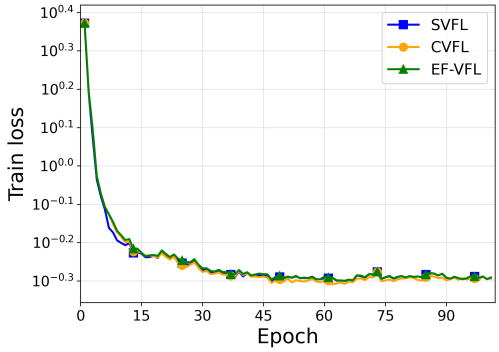
Figure 4.5: Under stronger compression, FedSpace still retains its advantage over Sync, FedBuff, and Async.

benefits of error feedback end-to-end, quantifying communication savings at matched accuracy against both uncompressed and compressed, but unscheduled baselines.

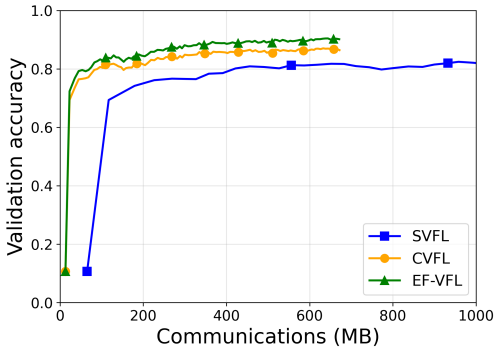
With the moderate compression rate of **20%**, FedSpace reaches **90.6%** at 671 MB. In contrast, the uncompressed async curve in Fig. 4.2b requires about 7.32 GB to reach the same **90.6%** (approximately 11× more bytes), while the compressed async baselines (CVFL/EFVFL) remain far to the right, hovering around approximately 90% only at multi-GB budgets.

With the more aggressive **5%** compression rate, FedSpace reaches **84.5%** with only 72.5 MB, crosses **90%** by 170 MB, and attains **94.7%** at 317 MB. By comparison, the uncompressed async baseline (SVFL; Fig. 4.3b) requires about 0.80 GB to reach **84.5%** (approximately 11× more) and roughly 5.27 GB to hit approximately **90%** (approximately 31× more), and it never attains **94.7%** even after 8.84 *GB*.

Reading the two FedSpace curves at fixed accuracy, the *5%* configuration saves approximately **75%** of the bytes required to hit **90%** (170 *MB* compared with 671 *MB*). These results indicate that the deterministic planning of FedSpace allows EFVFL to leverage sparsification effectively: for the same accuracy target, the required communication is substantially reduced relative to the uncompressed asynchronous baseline, while the training dynamics remain stable and fast.

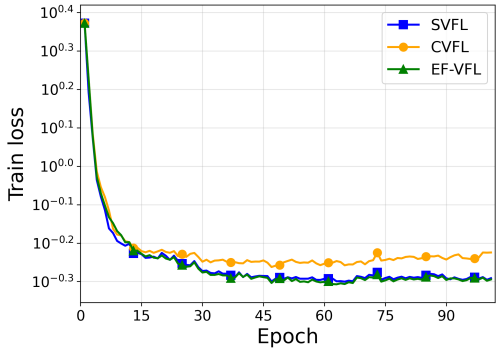


(a) Train loss per epoch.

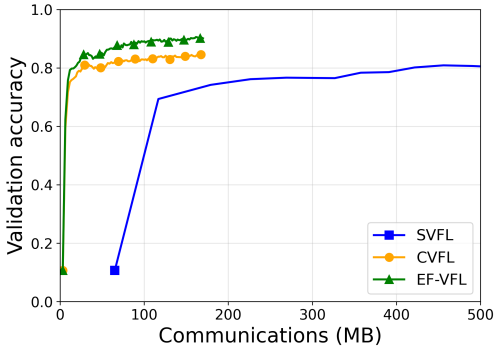


(b) Validation accuracy per unit of communication.

Figure 4.6: FedSpace on MNIST with $K = 149$ and 20% top-k sparsification.



(a) Train loss per epoch.



(b) Validation accuracy per unit of communication.

Figure 4.7: FedSpace on MNIST with $K = 149$ and 5% top-k sparsification.

5

Conclusions

In this work, we investigated VFL under deterministic, time-varying connectivity and showed that pairing EFVFL with a planner, such as FedSpace, yields substantial communication savings without sacrificing optimization stability. Error feedback recovers information lost to compression, enabling aggressive sparsification without degrading convergence (Valdeira et al. 2025), while planning over deterministic connectivity resolves the idleness-staleness trade-off by timing aggregations and crediting the union of clients across slots. As demonstrated empirically, these two methods combined move the accuracy-communication curve decisively leftward, reaching the same accuracy with far fewer bytes compared to uncompressed asynchronous training, and outperforming or matching alternative schedulers at much lower budgets. Overall, the results show that leveraging deterministic connectivity patterns is a powerful lever for bandwidth-constrained settings such as satellite constellations, making communication-efficient VFL feasible in practice.

Bibliography

Castiglia, T. J. et al. (2022). "Compressed-VFL: Communication-efficient learning with vertically partitioned data". In: *International Conference on Machine Learning*. PMLR, pp. 2738–2766 (cit. on pp. 11, 13).

Dean, J. et al. (2012). "Advances in neural information processing systems". In: *Large Scale Distributed Deep Networks; Association for Computing Machinery: New York, NY, USA*, pp. 1223–1231 (cit. on p. 1).

Denby, B. and B. Lucia (2020). "Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, pp. 939–954. ISBN: 9781450371025. DOI: 10.1145/3373376.3378473 (cit. on p. 25).

Haddadpour, F. et al. (2021-13–15 Apr). "Federated Learning with Compression: Unified Analysis and Sharp Guarantees". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 2350–2358 (cit. on p. 11).

Kim, G. et al. (2024). *Space for Improvement: Navigating the Design Space for Federated Learning in Satellite Constellations*. arXiv: 2411.00263 [cs.LG] (cit. on p. 10).

LeCun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 14).

Li, L. et al. (2020). "A review of applications in federated learning". In: *Computers & Industrial Engineering* 149, p. 106854 (cit. on p. 1).

Li, X. et al. (2019). "On the convergence of fedavg on non-iid data". In: *arXiv preprint arXiv:1907.02189* (cit. on p. 3).

Lian, X. et al. (2017). "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent". In: *Advances in neural information processing systems* 30 (cit. on p. 1).

Liu, Y. et al. (2022). "FedBCD: A communication-efficient collaborative learning framework for distributed features". In: *IEEE Transactions on Signal Processing* 70, pp. 4277–4290 (cit. on p. 13).

McMahan, B. et al. (2017). "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR, pp. 1273–1282 (cit. on pp. 2, 10).

Nguyen, J. et al. (2022). "Federated learning with buffered asynchronous aggregation". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3581–3607 (cit. on p. 10).

Shahid, O. et al. (2021). *Communication Efficiency in Federated Learning: Achievements and Challenges*. arXiv: 2107.10996 [cs.LG] (cit. on p. 11).

So, J. et al. (2022). "FedSpace: An Efficient Federated Learning Framework at Satellites and Ground Stations." In: *CoRR* abs/2202.01267 (cit. on pp. 7, 9, 10).

Valdeira, P. et al. (2025). "Communication-Efficient Vertical Federated Learning via Compressed Error Feedback". In: *IEEE Transactions on Signal Processing* 73, pp. 1065–1080. DOI: 10.1109/TSP.2025.3540655 (cit. on pp. 1, 3, 11, 14, 17, 18, 21, 33).

Xie, C., S. Koyejo, and I. Gupta (2019). "Asynchronous federated optimization". In: *arXiv preprint arXiv:1903.03934* (cit. on pp. 9, 10).

Yang, Q. et al. (2019). "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2, pp. 1–19 (cit. on p. 2).