network

MDPI

Article

Unified Distributed Machine Learning for 6G Intelligent Transportation Systems: A Hierarchical Approach for Terrestrial and Non-Terrestrial Networks

David Naseh [1,*], Arash Bozorgchenani [2], Swapnil Sadashiv Shinde [3] and Daniele Tarchi [4]

[1] Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy
[2] School of Computer Science, University of Leeds, Leeds LS2 9JT, UK; a.bozorgchenani@leeds.ac.uk
[3] School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 11428 Stockholm, Sweden; ssshinde@kth.se
[4] Department of Information Engineering, University of Florence, 50139 Florence, Italy; daniele.tarchi@unifi.it
* Correspondence: david.naseh2@unibo.it

Abstract

The successful integration of Terrestrial and Non-Terrestrial Networks (T/NTNs) in 6G is poised to revolutionize demanding domains like Earth Observation (EO) and Intelligent Transportation Systems (ITSs). Still, it requires Distributed Machine Learning (DML) frameworks that are scalable, private, and efficient. Existing methods, such as Federated Learning (FL) and Split Learning (SL), face critical limitations in terms of client computation burden and latency. To address these challenges, this paper proposes a novel hierarchical DML paradigm. We first introduce Federated Split Transfer Learning (FSTL), a foundational framework that synergizes FL, SL, and Transfer Learning (TL) to enable efficient, privacy-preserving learning within a single client group. We then extend this concept to the Generalized FSTL (GFSTL) framework, a scalable, multi-group architecture designed for complex and large-scale networks. GFSTL orchestrates parallel training across multiple client groups managed by intermediate servers (RSUs/HAPs) and aggregates them at a higher-level central server, significantly enhancing performance. We apply this framework to a unified T/NTN architecture that seamlessly integrates vehicular, aerial, and satellite assets, enabling advanced applications in 6G ITS and EO. Comprehensive simulations using the YOLOv5 model on the Cityscapes dataset validate our approach. The results show that GFSTL not only achieves faster convergence and higher detection accuracy but also substantially reduces communication overhead compared to baseline FL, and critically, both detection accuracy and end-to-end latency remain essentially invariant as the number of participating users grows, making GFSTL especially well suited for large-scale heterogeneous 6G ITS deployments. We also provide a formal latency decomposition and analysis that explains this scaling behavior. This work establishes GFSTL as a robust and practical solution for enabling the intelligent, connected, and resilient ecosystems required for next-generation transportation and environmental monitoring.

Keywords: intelligent transportation systems; federated learning; split learning; transfer learning; integrated terrestrial and non-terrestrial networks

Academic Editors: Harry Leib and Hong-Chuan Yang

Received: 29 July 2025
Revised: 3 September 2025
Accepted: 8 September 2025
Published: 17 September 2025

Citation: Naseh, D.; Bozorgchenani, A.; Shinde, S.S.; Tarchi, D. Unified Distributed Machine Learning for 6G Intelligent Transportation Systems: A Hierarchical Approach for Terrestrial and Non-Terrestrial Networks. *Network* **2025**, *5*, 41. https://doi.org/10.3390/network5030041

1. Introduction

With the 6G vision set to shape society in the 2030s into a more advanced, digitized, fully connected, and intelligent world, transportation networks are undergoing a significant

transformation, converging into Intelligent Transportation Systems (ITSs) [1]. The field of ITS is rapidly expanding, with the aim of improving the safety, efficiency and sustainability of traditional transportation systems by using technologies such as Machine Learning (ML), the Internet of Things (IoT) and advanced communication modes [2]. The presence of IoT subsystems facilitates the generation of large amounts of data from various components of the ITS infrastructure over time [3]. These data can be used to provide intelligent solutions, optimize traffic management, improve user experiences, and improve environmental outcomes. However, significant challenges arise in the acquisition and analysis of data from various sources, including vehicles, sensors, and traffic cameras [4].

Recent advances in Distributed Machine Learning (DML) techniques offer promising solutions to these challenges. Among them, Federated Learning (FL) has emerged as a particularly effective paradigm for training ML models in a distributed manner, enabling collaboration while maintaining data privacy [5]. FL facilitates shared model training without requiring the exchange of individual data among multiple devices. Each device trains a local model on its data and then updates a central server with its progress. The server aggregates the updates and applies them to a global shared model. However, a significant limitation of FL lies in the necessity for each client to independently train the entire ML model. This process demands substantial computational resources and becomes particularly challenging for clients with constrained resources, notably within ITS contexts, where the use of complex models such as Deep Neural Networks (DNNs) is common [6]. Furthermore, iterative transmission of local and global model parameters has raised new privacy concerns, including the risks of data poisoning and model inversions, highlighting the need for improved security mechanisms in FL frameworks [7].

To mitigate the limitations of FL, Split Learning (SL) presents an alternative approach. SL allows complex ML models to be trained by dividing them into two parts, with each part trained on a client or a server using local data from distributed clients [8]. This method significantly reduces the computational burden on resource-limited devices, as only a portion of the model is trained locally and communication is limited to the activation of the cut layer. Consequently, SL enhances model privacy by preventing direct access to the entire model on either side. Recent research has demonstrated the potential of SL-inspired frameworks to enhance the efficiency and scalability of FL approaches, allowing the training of more sophisticated DNNs while maintaining privacy and reducing costs [9].

In addition to these methods, Transfer Learning (TL) from the meta-learning family has gained attention for its ability to improve training efficiency by facilitating knowledge transfer from previous tasks to new related tasks [10]. TL can accelerate convergence rates, reduce reliance on labeled data, and enhance the robustness of ML models in various vehicular scenarios. The integration of TL with FL, particularly in resource-constrained ITS environments, presents a valuable opportunity to leverage previous knowledge and improve model performance [11]. However, existing works have yet to comprehensively address how these techniques can be optimally combined to maximize their benefits in dynamic and heterogeneous environments.

Earth Observation (EO) has emerged as a complementary technology that can significantly enhance the capabilities of ITS. EO involves the collection and analysis of satellite and aerial data, providing critical information on traffic conditions, weather patterns, and environmental changes [12]. This integration allows ITS to benefit from real-time data streams, facilitating improved traffic management, hazard detection, and overall situational awareness [13]. However, integrating EO with ITS presents its own set of challenges, including the need for efficient data fusion techniques from heterogeneous sources, real-time processing capabilities, and advanced ML methodologies to analyze vast amounts of EO data effectively [14].

In the context of the 6G vision, Non-Terrestrial Networks (NTNs) play a pivotal role in providing global coverage and capacity enhancements for traditional terrestrial networks. Various NTN platforms, including High-Altitude Platforms (HAPs), offer significant advantages to vehicular users (VUs), such as reduced transmission distances, improved coverage, and flexible deployment options [15]. These characteristics make NTNs particularly well-suited for supporting intelligent solutions in ITS by facilitating efficient DML methods and enhancing connectivity in remote or underserved areas [16].

In the space of ITS-oriented DML, several hybrid frameworks have been explored, each contributing valuable insights, yet leaving important gaps. For example, ref. [17] introduces a hybrid of FL and SL tailored to ITS, enabling personalized model training across vehicles while preserving data privacy; however, it lacks TL integration and does not consider the latency and resource constraints of multilayer network deployments. Another relevant study, ref. [18], evaluates the security and effectiveness of FL, SL, and TL in vehicular networks; while it clarifies which techniques are feasible for privacy preservation, it remains a feasibility study and does not propose a unified framework that combines these paradigms with scalability in multilayer NTN settings. A third work [19] develops a variant of Federated Split Learning (FSL) that addresses the challenge of vehicles lacking labeled data by splitting the model between onboard units and RSUs, preserving both data and label privacy; however, it does not address dynamic grouping, hierarchical aggregation, or real-time adaptation in 6G NTN-enhanced ITS.

1.1. Gaps in the Literature and Motivations

Table 1 summarizes representative prior works and highlights the main differences relative to our approach. Although existing studies have advanced FL, SL, TL and hybrid schemes for edge and vehicular settings, several important gaps remain that limit their applicability to multilayer 6G ITS and NTN-enabled Earth Observation. The most salient gaps are as follows:

1. **Under-utilization of multilayer NTN architectures:** Many prior frameworks assume single-tier or purely terrestrial deployments and therefore do not exploit the potential benefits of coordinated processing across RSUs, HAPs, and satellites. This omission reduces opportunities for latency-aware placement, load balancing across tiers, and resilience in coverage-challenged regions—all critical for large-scale ITS.
2. **Limited onboard and in-space data processing:** Existing works rarely address efficient processing of EO data within non-terrestrial platforms or the trade-offs introduced by performing model tasks in space (e.g., at HAPs/LEO nodes). Without concrete strategies for in-space inference/aggregation, systems face increased communication cost or delayed decision-making for time-sensitive ITS applications.
3. **Insufficient integration of DML paradigms for resource-constrained, heterogeneous clients:** While FL, SL, and TL have been studied individually (and some combinations proposed), there is a lack of comprehensive frameworks that jointly leverage model partitioning, transfer initialization, and hierarchical aggregation to simultaneously address privacy, compute constraints, and converge quickly in highly heterogeneous ITS fleets.

The emerging 6G paradigm, characterized by ubiquitous and seamless T/NT connectivity, extreme low-latency links, pervasive edge intelligence, and AI-native network management, directly addresses the limitations noted above. Native NTN integration in 6G enables hierarchical, multilayer orchestration (e.g., RSU/UAV/HAP/satellite) that supports latency-aware placement and multi-tier aggregation. Likewise, 6G's stronger edge computing and deterministic service capabilities make practical the notion of in-space or at-edge model processing for EO and real-time ITS tasks, reducing the need for bulky

raw-data transfers. Finally, the AI-native control plane and advanced resource-slicing features foreseen for 6G facilitate dynamic model partitioning and adaptive use of FL/SL/TL mechanisms across heterogeneous clients. These 6G capabilities therefore motivate and make feasible the design objectives of this paper: a multilayer joint T/NT, latency-aware DML framework that unifies Federated, Split, and Transfer Learning and maps model components to the most appropriate network tier, improving scalability, privacy, and real-time performance for ITS and NTN-based EO scenarios.

Table 1. Comparison of the most related works and gaps/differences relative to our work.

Ref.	Contributions	Differences With Our Work
[6]	Multitask FL with hybrid client selection/aggregation in edge networks.	Pure FL (no SL/TL); no multilayer hierarchy or NTN integration; not tailored to ITS.
[8]	Evaluates and optimizes DML techniques for IoT; benchmarking focus.	Not ITS-centric; not hierarchical FL/SL over T/NTNs nor real-time ITS latency study.
[9]	Wireless Distributed Learning: hybrid split+federated approach.	No T/NTN tiering, no EO/ITS integration, and no multilayer ITS latency/accuracy study.
[11]	Federated Transfer Learning for cross-domain sensing with resource efficiency.	No ITS; no SL or multilayer NTN; narrower modality than our EO/vehicular perception.
[15]	6G-enabled advanced transportation systems; network capabilities and use cases.	No DML pipeline or a multilayer T/NTN latency/accuracy quantification.
[16]	Explores the potential of NTNs in next-gen ITS; positioning paper.	Lacks concrete DML training/aggregation across layers and empirical ITS evaluation.
[17]	Personalized FSL for ITS-distributed training: single-layer training.	No hierarchy, no EO integration, and no unified FL/SL/TL orchestration across T/NTNs.
[18]	Feasibility of SL/TL/FL for security in ITS; SL outperforms FL/TL baselines; security focus.	No ITS; no hierarchical T/NTN training or EO-ITS fusion; no multi-tier aggregation.
[19]	FSL with data/label privacy in vehicular networks.	No multilayer architecture and no joint FL+SL+TL orchestration.

To address these gaps, we propose two comprehensive methodologies: Federated Split Transfer Learning (FSTL) and its generalized version, Generalized Federated Split Transfer Learning (GFSTL). Both methods are designed for deployment in T/NTNs (or joint air–ground networks) and will be applied to three specific applications: vehicular scenarios, EO, and then their combination as an ITS scenario. The FSTL framework integrates the advantages of FL, SL, and TL, providing a scalable solution to train ML models in resource-constrained environments [20]. Meanwhile, GFSTL introduces a flexible approach that allows the independent use of FL and SL servers, catering to diverse user requirements. This novel architecture not only reduces latency and enhances the number of participants in the federated training process but also improves overall model accuracy and privacy protection. We will evaluate the effectiveness of the proposed methodologies through simulations in typical 6G ITS scenarios using advanced architectures like ResNet and You Only Look Once (YOLO) [21], demonstrating their superiority over traditional FL methods in terms of convergence rates, training accuracy, and overall latency.

1.2. Key Contributions

This paper introduces GFSTL and FSTL, two novel DML frameworks designed for multilayer 6G ITS and NTN-enabled EO. The main contributions are summarized as follows:

1. **New hybrid learning frameworks (FSTL and GFSTL):** We propose FSTL and its generalized version (GFSTL). FSTL integrates FL, SL, and TL into a single pipeline

tailored for resource-constrained edge nodes. GFSTL extends FSTL by supporting multiple independent FL/SL server placements and hierarchical aggregation across Road-Side Units (RSUs) and HAPs, enabling flexible deployment across Terrestrial and Non-Terrestrial Network layers (see Sections 3 and 4).

2. **Exploitation of multilayer T/NTN architectures for scalable DML:** We explicitly design the frameworks to exploit the multilayer structure of NTNs (RSUs, UAVs, HAPs, Low/Medium Earth Orbit (LEO/MEO) satellites) so that computation and aggregation are performed at the most appropriate network tier (see the multilayer architectures in Section 5). This hierarchical design reduces wall-clock training time, improves scalability, and increases the number of feasible participants in federated training (see Section 6 and the latency analysis in Section 5.4).
3. **Efficient Data Processing for Real-Time EO/ITS Applications:** The paper proposes novel strategies for real-time data processing in space to optimize the utilization of EO/ITS data. By leveraging GFSTL's flexible architecture, we address the inefficiencies in current EO/ITS systems, enabling more effective data transmission and model updates, even in highly dynamic environments with limited resources (Section 5).
4. **Application to Diverse Use Cases and Comprehensive Evaluation:** The proposed methodologies, system architecture, training process, added benefits, and challenges are introduced and evaluated across three distinct use cases: vehicular scenarios (Section 5.1), EO (Section 5.2), and their integration into a unified ITS scenario (Section 5.3). This work also provides a comprehensive latency analysis for a multilayer 6G ITS environment (Section 5.4), showcasing the practical benefits of our approach in large-scale, real-world deployments.
5. **Enhanced Latency, Accuracy, and Privacy in DML:** Our proposed methods significantly reduce latency and enhance training accuracy compared to traditional FL-based frameworks. The introduction of flexible server configurations in GFSTL further improves the number of participants in the FL process while ensuring robust privacy protection. These advances are demonstrated through simulations in typical 6G ITS scenarios, where the proposed methodologies outperform traditional FL techniques in terms of convergence rates, model accuracy (Sections 6.2 and 6.3), and overall system latency (Section 6.4).
6. **Practical split/transfer parameterization for ITS and EO tasks:** We provide a concrete parameterization (model cut-points, smashed-representation sizing, and YOLO/ResNet choices) that maps model components to node capabilities (RSU/HAP/edge) (Section 6.1), balancing accuracy, computation, and communication. This parameterization is validated empirically in Section 6 and supports real-time ITS perception tasks using compact intermediate representations.
7. **Comprehensive performance and latency study under unified experimental settings:** We evaluate FL, SL, FSL, FSTL, and GFSTL on a representative ITS scenario using the same hardware/network assumptions to ensure a fair comparison. This study quantifies trade-offs in accuracy, convergence speed, per-round latency, and communication volume and demonstrates our proposed frameworks' improvements in these dimensions (results and discussion in Sections 6 and 7).

The above contributions jointly advance the state of the art by presenting scalable, privacy-preserving, and latency-aware DML solutions that are directly applicable to multilayer 6G ITSs and NTN-based EO systems.

1.3. Organization of the Paper

This paper is organized as follows: Section 2 provides an overview of existing DML frameworks, including FL, SL, FSL, and TL, discussing their advantages and limitations.

Section 3 details the proposed FSTL framework, outlining its architecture and training process and showcasing its application in ITSs. Building upon this, Section 4 introduces the GFSTL framework, presenting its architecture and training algorithm for scalable, multi-group network environments. Section 5 demonstrates the versatility of GFSTL through two distinct use cases: a vehicular Aerial–Ground Integrated Network (AGIN) and an NTN-based EO system, followed by a comprehensive latency analysis for a unified multi-layer 6G ITS scenario. Section 6 presents the simulation setup and parameters, evaluation metrics, and the results validating the proposed methodologies. Finally, Section 7 provides a discussion of the findings, limitations, and future directions, with Section 8 concluding the paper.

2. Distributed Machine Learning Frameworks

DML has become an essential approach to enable collaborative model training across multiple devices without requiring centralized data storage. This method is particularly valuable in scenarios like ITS, NTNs, and EO, where large volumes of data are distributed among many devices, and privacy concerns or bandwidth limitations make data centralization impractical. By allowing each device to process its data locally and share only model updates with a central server, DML frameworks ensure that privacy is maintained while enabling efficient large-scale model training. Some of the most widely adopted DML techniques include FL and SL, each offering unique benefits to address the challenges of distributed data environments [22].

2.1. Federated Learning

FL is one of the foundational methods for DML, allowing multiple clients (e.g., VUs in an ITS) to collaboratively train a shared ML model without sharing their raw data. FL ensures privacy and reduces the bandwidth needed for communication, making it well-suited for environments where data privacy is critical, such as ITSs [23]. Each client performs model training on its local data and only transmits model updates (such as gradients) to a central server for aggregation.

Consider a system with N distributed users defined through the set $\mathcal{U} = \{u_1, \ldots, u_N\}$, where each user i has a local dataset $\mathcal{D}_i = \{(\mathbf{x}_k, y_k)\}$ containing K_i labeled data samples. The feature vector $\mathbf{x}_k \in \mathbb{R}^n$ is the k-th sample, while y_k is its corresponding label. The task is to train a global model W^p by minimizing a global loss function L^p between all clients without sharing the raw data.

During each round of communication, users update their local models by minimizing their local loss functions $L_i^p(\mathcal{D}_i, W_t^p)$, where W_t^p represents the global model in round t. The updated local model parameters of each user are sent to the server, which aggregates them using a weighted averaging scheme, such as FedAvg [24]:

$$W_{t+1} = \frac{1}{N} \sum_{i=1}^{N} W_{i,t}^p \tag{1}$$

The process continues until the model converges or a stopping criterion is met, such as a target loss value. The advantages of FL are as follows:

- **Data Privacy:** FL allows users to keep their raw data local, making it ideal for privacy-sensitive scenarios.
- **Reduced Communication Overhead:** Only the model parameters are exchanged, not the raw data, saving bandwidth.
- **Adaptability:** FL can continuously improve models with real-time data from users, making it responsive to dynamic environments.

The disadvantages are the following:

- **Communication Latency:** Multiple rounds of communication are required, increasing the overall latency, especially in bandwidth-constrained networks.
- **Convergence Speed:** Achieving convergence may require many iterations, resulting in high computational and communication costs for resource-constrained users.

While FL addresses privacy and communication challenges, its iterative nature introduces certain limitations, particularly in environments where latency or computational resources are limited [25]. To overcome these challenges, more advanced frameworks such as SL have been introduced. The interested reader could refer to [26] for a comprehensive analysis of FL algorithms.

2.2. Split Learning

Although FL provides a solid foundation for DML, it can place significant computational and communication burdens on clients, especially when training deep models [27]. To mitigate these issues, SL divides the learning process between clients and servers. In SL, each client processes only the lower layers of the model, while the server handles the remaining computations. This reduces the computational load on clients and improves privacy by keeping raw data within the client device [28].

In the context of an ITS scenario, consider a model W^p that is divided into two components: (i) the client-side model $S(\cdot)$, which each user or client maintains locally, and (ii) the server-side model $M(\cdot)$, which is responsible for completing the forward and backward passes on the server. This split architecture allows resource-constrained devices, such as VUs, to perform only the initial stages of the computation, reducing their computational and storage requirements.

The SL process involves the following steps for a single forward–backward round:

1. Each user processes its local data $\mathcal{D}_i$ using the client-side model $S(\cdot)$, which generates intermediate activations:
$$H_i = S(\mathcal{D}_i) \tag{2}$$
These intermediate activations H_i represent compressed feature representations and are transmitted to the server.
2. Upon receiving H_i, the server performs the forward pass using its merge model $M(\cdot)$ to produce predictions $\hat{y}_i$:
$$\hat{y}_i = M(H_i) \tag{3}$$
This completes the forward pass for the entire model W^p using both client- and server-side components.
3. Next, the server begins the backward pass on the server-side model $M(\cdot)$ to compute the gradients of its model parameters based on the loss function, typically using a method such as stochastic gradient descent. The server then calculates the gradient of the intermediate activations H_i, denoted as ∇H_i, and transmits this gradient back to the user.
4. Finally, the user completes the backward pass locally on its split model $S(\cdot)$ using ∇H_i. This enables each client to update their local model parameters for $S(\cdot)$ based on its unique data, thus maintaining data privacy and security.

This split process enables each user to perform forward and backward propagation collaboratively with the server, forming a full round of training over the model W^p. By structuring the model in this manner, SL allows resource-constrained client devices to participate in training without handling the full computational load. The server's ability to process intermediate activations reduces client–server communication overhead while preserving data privacy by ensuring that raw data remains local to each client.

The advantages of SL are as follows:

- **Reduced Client-Side Load:** Clients only need to compute part of the model, reducing their computational overhead.
- **Privacy Preservation:** Only smashed data (intermediate activations) is transmitted, protecting the privacy of raw data.
- **Lower Communication Cost:** Fewer data need to be exchanged between clients and the server, as only intermediate activations are shared.

The disadvantages are the following:

- **Sequential Processing:** The split model introduces latency because clients and the server must alternate between steps.
- **Reduced Model Expressiveness:** Splitting the model might reduce its capacity, potentially leading to performance degradation.

Although SL reduces computational load and improves privacy, it can still suffer latency issues due to its sequential nature [29]. The interested reader could refer to [30] for a comprehensive analysis of SL algorithms.

2.3. Federated Split Learning

To overcome earlier limitations, a novel framework named FSL merges the benefits of FL and SL. FSL brings together the privacy safeguards characteristic of FL with the computational efficiency found in SL. This framework allows clients to execute a segment of the model processing locally, as per SL, followed by employing the federated averaging technique from FL to consolidate intermediate client outputs. Consequently, this synthesis taps into the strengths of both systems while mitigating their weaknesses [31].

In FSL, each user computes intermediate activations $H_i = S(\mathcal{D}_i)$ locally, which are sent to the server for aggregation. The server merges these activations using a federated averaging scheme:

$$H_{merged} = \frac{1}{N} \sum_{i=1}^{N} H_i \tag{4}$$

The merged activations are then used to update the global model, and the updated parameters are distributed back to the users. By combining the FL distributed training process with the SL efficient model split approach, FSL can achieve better scalability and improved privacy. The advantages of FSL are as follows:

- **Hybrid Approach:** FSL benefits from both FL and SL, enabling DML with lower communication costs and enhanced privacy.
- **Scalability:** The federated averaging mechanism allows the framework to scale effectively across many clients without increasing the communication burden.
- **Reduced Client-Side Load:** Like SL, FSL reduces the computational burden on clients by only requiring them to compute the lower layers of the model.

The disadvantages are the following:

- **Slow Convergence:** The nature of SL can result in slower convergence, particularly in large-scale networks.
- **High Local Training Requirements:** Clients still need sufficient local computational resources to handle their portion of the model, which may not always be feasible in resource-constrained environments.

Although FSL improves the benefits offered by both FL and SL, its implementation can suffer from delayed convergence rates, particularly in extensive deployment scenarios [32]. To further optimize the efficiency of the learning procedure, TL can be integrated with FSL,

giving rise to the FSTL methodology proposed later. Before analyzing FSTL, we introduce TL, focusing on its advantages.

2.4. Transfer Learning

TL is a powerful technique that improves model performance by leveraging knowledge gained from previous tasks, enabling efficient training on new but related tasks. TL is especially useful in scenarios with limited data or heterogeneous data distributions among clients, as it reduces the training required for convergence and enhances overall model accuracy [33]. By initializing clients with pre-trained models developed on large datasets, TL allows each client to start training from a more informed state, rather than from random initialization [34].

Formally, let $\mathcal{D}_{source} = \{(x_i, y_i)\}_{i=1}^{N_s}$ represent the source domain with N_s samples and associated labels y_i. The objective is to learn a model $W^{p'}$ that minimizes the loss function L^{source} in this domain, i.e.,

$$L^{source} = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}(y_i, f(x_i; W^{p'})) \tag{5}$$

where $\mathcal{L}$ is a chosen loss function (e.g., cross-entropy) and $f(x_i; W^{p'})$ is the model prediction for input x_i.

Once trained on the source domain, the model is fine-tuned on the target domain $\mathcal{D}_{target} = \{(x_j, y_j)\}_{j=1}^{N_t}$ with N_t samples to minimize the loss function L^{target}:

$$L^{target} = \frac{1}{N_t} \sum_{j=1}^{N_t} \mathcal{L}(y_j, f(x_j; W^{p'})) \tag{6}$$

TL offers the following advantages:

- **Accelerated Training:** TL reduces the training time required for model convergence, which is particularly advantageous in real-time or resource-constrained applications.
- **Improved Model Accuracy:** Leveraging learned knowledge from a related domain improves model performance, especially when training data is limited or heterogeneous.
- **Enhanced Generalization:** Pre-trained models offer better adaptability across diverse client datasets, as they can transfer features to new tasks effectively.

3. Federated Split Transfer Learning

In the realm of FSL, TL offers significant advantages. The allocation of a pre-trained model $W^{p'}$ to each client expedites the learning process and enhances the model's capacity to generalize across the diverse datasets of clients. This synergistic approach is especially beneficial when there are disparities in client data distributions, as the pre-trained features and patterns provide a solid foundation for adaptation. As FSL progresses to FSTL, TL further strengthens the learning architecture. The fusion of TL with FSL optimizes both efficiency and scalability while maintaining the privacy and computational benefits inherent to FSL. This collaboration facilitates quicker and more adaptable model training in distributed settings, which is particularly advantageous for a range of applications that feature distinct data properties.

In FSTL, the goal is to enhance DML capabilities by combining the benefits of FL, SL, and TL. The FSTL framework is particularly advantageous in scenarios where resource-constrained devices, such as VUs or unmanned aerial vehicles (UAVs), participate in collaborative learning while maintaining data privacy and minimizing communication overhead.

3.1. FSTL Architecture

The architecture of FSTL begins with a pre-trained neural network model $W^{p'}$, which consists of L layers. This model is divided into a specific cut layer k (where $1 < k < L$) with two parts:

1. $W_U^{p'}$ is the portion up to layer k, which is deployed on each user for local processing, and
2. $W_S^{p'}$ represents the remaining layers, which are managed by the server in the SL paradigm.

During training, the generic i-th VU processes its local dataset $\mathcal{D}_i$ using the client-side split model $W_{VU}^{p'}$, generating intermediate representations H_i. These representations are then transmitted to the SL server-side model $W_S^{p'}$, which processes the intermediate data and updates the global model parameters. The split structure allows users to handle computationally lighter tasks while offloading the most demanding parts to the server, thereby reducing local resource consumption.

By integrating TL into the FSL paradigm, the FSTL framework leverages the advantages of pre-trained models on the client side. The pre-trained layers up to the cut layer k capture generalizable features and patterns that are useful across different tasks. Meanwhile, the remaining layers on the server allow further refinement and collaborative learning among distributed users. As a result, FSTL benefits from more efficient knowledge transfer, improved model performance, and faster convergence compared to traditional SL or FL methods.

A key improvement in the FSTL framework over traditional SL is its capacity to enable parallel processing by allowing all clients to communicate simultaneously with both the SL and FL servers. This architecture, as illustrated in Figure 1, offers several distinct advantages over sequential methods:

- **Higher Convergence Rate**: By enabling parallel, simultaneous updates from all clients, the model can learn from a diverse set of data within each training round. This avoids the slow, one-by-one sequential processing inherent in standard SL, leading to a more efficient path to model convergence.
- **Reduced Communication Bottlenecks**: In a sequential learning setup, the server can only communicate with one client at a time, creating a significant bottleneck that scales with the number of users. The FSTL structure removes this limitation by allowing the central server to handle communications with all participating clients in parallel, thereby improving network efficiency.
- **Balanced Computational Loads**: The framework design inherently distributes the workload. While each client handles the initial, less intensive part of the model, the server manages the more computationally demanding tasks and the aggregation of updates from all clients. This parallel structure ensures a more balanced and efficient use of computational resources throughout the network.

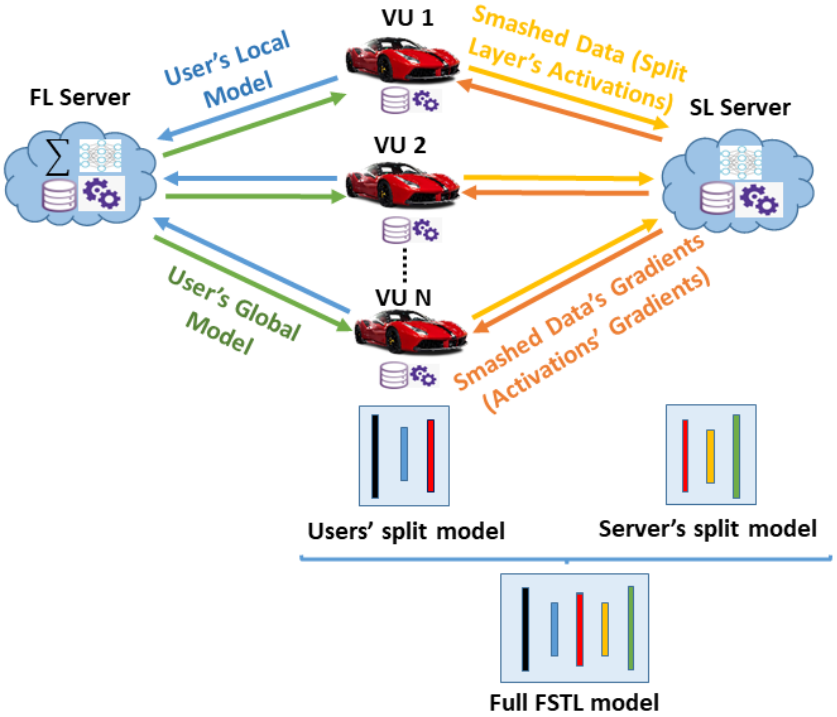

Figure 1. Overall structure of FSTL.

3.2. FSTL Training Process

The FSTL training procedure is an iterative, distributed process that allows multiple users to collaboratively train a model while managing data privacy and distributing computational load. The model is divided into two components: (i) the client-side model, which is hosted locally by each user, and (ii) the server-side model, which is hosted centrally.

To formalize the training process, let $W_U^{p'} = \{\theta_i\}$ denote the parameters of the client-side model and $W_S^{p'} = \{\theta_s\}$ denote the parameters of the server-side model. The procedure follows the following steps:

1. **Initialization:**
   (a) The global model parameters θ are initialized using pre-trained weights from a TL model.
   (b) A specific layer index k is chosen, determining the split between the client and server computations. The layers before k are hosted on the client side, while the remaining layers are hosted on the server.
2. **Iterative Training:** For each training iteration $t = 1, 2, \ldots, T$, the following steps occur:
   (a) The initial global model parameters θ are distributed to all participating users.
   (b) Each user i processes its local data $\mathcal{D}_i$ through its client-side split model $W_U^{p'}$ to generate intermediate representations:

$$H_i = W_U^{p'}(\mathcal{D}_i) \tag{7}$$

   (c) The user sends these intermediate representations H_i to the central server.
   (d) The central server aggregates the intermediate representations from all users by applying the following merge function $\text{Merge}(\cdot)$:

$$H_{\text{merged}} = \text{Merge}(H_1, H_2, \ldots, H_N) \tag{8}$$

(e) Using H_{merged}, the server performs a forward–backward pass to compute an updated version of the global model parameters:

$$\theta' = \theta - \eta \cdot \nabla_\theta L(H_{\text{merged}}, \theta) \tag{9}$$

Here, L denotes the loss function, while η is the learning rate. This forward–backward process, which leverages federated averaging, is conducted in a way that preserves privacy, since only aggregated updates are calculated.

(f) The central server updates θ by setting

$$\theta = \theta' \tag{10}$$

(g) Each user i then receives the updated global model parameters θ from the server and uses them to update their own server-side model parameters as follows:

$$\theta_i' = W_S^{p'}(H_i, \theta) \tag{11}$$

(h) Finally, each user i updates its client-side model parameters as follows:

$$\theta_i = \theta_i' \tag{12}$$

This iterative process repeats until the model reaches a stopping criterion, such as convergence of the loss function or a predetermined number of training rounds. In each iteration, users independently process their local data through the client-side model to produce intermediate representations that they then share with the central server. The server merges these representations and performs federated updates to enhance the global model, subsequently sending the updated parameters back to users. This cycle of local data processing, secure sharing, and federated aggregation enables DML while preserving data privacy and distributing computational demands across the network.

3.3. FSTL Advantages

FSTL provides a comprehensive and powerful framework for DML, integrating the advantages of FL, SL, and TL. This approach allows individuals to handle data locally while taking advantage of the global collaborative capabilities offered by TL-enhanced learning. This integration significantly improves model performance, accelerates convergence rates, and mitigates resource limitations, making it particularly suitable for dynamic and privacy-sensitive environments such as ITSs [35]. The benefits of FSTL can be elaborated upon as follows:

1. **Data Privacy and Security:** With the retention of raw data on the user's side, FSTL guarantees that sensitive personal information is not exposed or communicated outward. This addresses the fundamental privacy issues associated with collaborative learning frameworks.
2. **Efficient Communication:** The transmission involves only the intermediate data representations between clients and the central server, thus significantly reducing communication overhead in the process.
3. **Accelerated Convergence:** By employing models that have been pre-trained, convergence is more rapidly achieved since these models utilize existing knowledge instead of starting the learning process from the beginning.
4. **Improved Performance:** The server's collaborative learning approach fosters improvements in the global model across all users, facilitating the adaptability of the framework to ever-changing environments and diverse ITS scenarios.

3.4. FSTL in ITS Scenarios

In the context of a 6G-enabled ITS, achieving accurate and adaptive model learning requires various sources of data input to reflect the real-world conditions vehicles face on the roads. In ITS scenarios, FSTL is expanded to encompass not only VUs but also UAVs and strategically located cameras on streets and intersections. This variety of data sources provides more comprehensive environmental information, which is crucial for creating high-efficiency and responsive ITS applications.

VUs provide ground-based data, including real-time information on vehicle location, speed, and road conditions, factors crucial for direct traffic management and road safety. UAVs, on the other hand, contribute an aerial perspective, capturing broad-area observations that can help monitor larger traffic patterns, assess road congestion, and even support emergency responses through quick area scans. Cameras placed on streets and intersections provide fixed high-frequency snapshots of specific intersections or traffic points, enabling an accurate assessment of local traffic density, pedestrian movements, and potential hazards. Together, these users form a comprehensive multiperspective network that aligns with the 6G ITS goals by connecting each component of the ITS infrastructure.

In this ITS FSTL scenario, as illustrated in Figure 2, the Road-Side Unit (RSU) serves as the FSL server. Here, each type of user (i.e., VUs, UAVs, and cameras) retains its raw data locally, thus preserving privacy, and sends only intermediate representations, known as *smashed data*, to the RSU for further processing and model training. This approach significantly reduces communication overhead by limiting exchanged data to model updates, supporting low-latency interaction, and reducing bandwidth demands, actions important for real-time ITS applications.

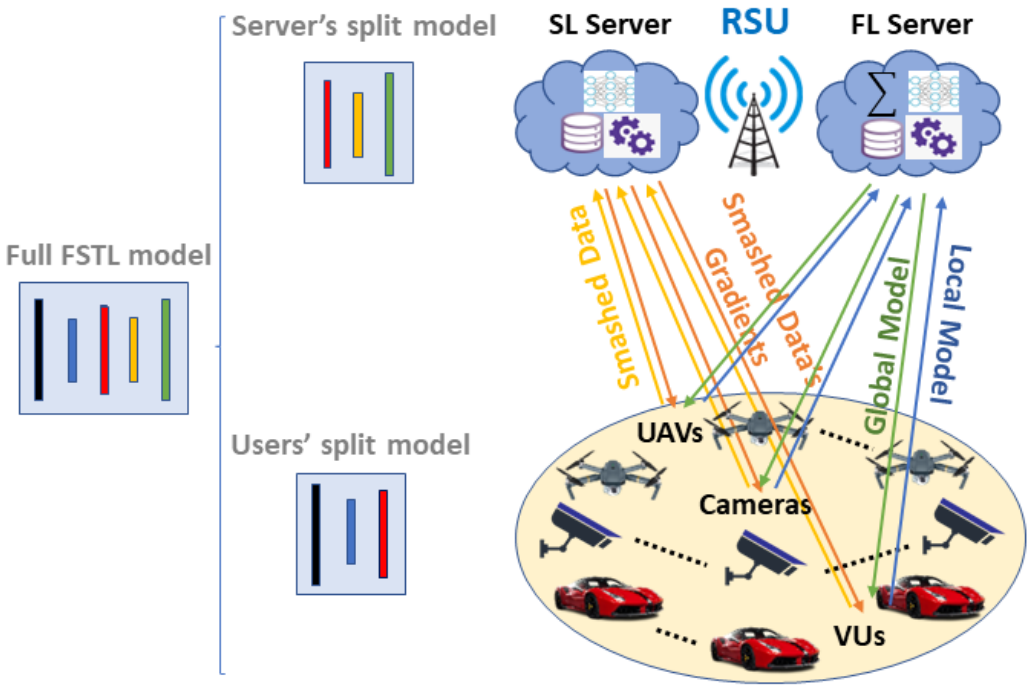

Figure 2. FSTL structure for ITSs with the RSU as the FSL Server.

3.5. Benefits of FSTL in ITSs

The RSU serves as the main FSL server, which allows collaborative training between VUs, UAVs, and cameras by integrating data from multiple perspectives, promoting a model that can adapt to varying road conditions. Positioned near traffic points, RSUs are strategically located to provide reliable connection points for each type of user device. They facilitate low-latency communication and rapid data processing, allowing the ITS

framework to respond promptly to traffic changes, weather impacts, and other dynamic factors. By aggregating insights from these diverse data sources, the FSTL framework in the ITS supports real-time adaptability and robust decision-making, which are critical for modern transportation systems.

3.6. FSTL Training Process

Each user, in coordination with the RSU, participates in iterative model updates that allow the server to refine the model collaboratively. The following iterative algorithm (Algorithm 1) outlines this training process. The algorithm begins by initializing the central server's (RSU) model parameters, denoted as θ_s, with an initial state $\theta_s(0)$ (Line 1). Simultaneously, each of the N individual users initializes their local model parameters, θ_i, with their respective initial states $\theta_i(0)$ (Lines 2–3). This sets up the initial state for both the global and local models before training commences. Each user then independently performs a forward propagation step on their local model using their private local data (Line 4). Instead of sending their raw data or full model, each user computes and transmits an intermediate representation (H_i) to the RSU (Line 5). This H_i likely encapsulates features or activations derived from the local data, without revealing the data itself, thus contributing to data privacy. Upon receiving the intermediate representations (H_i) from all active users, the RSU performs a forward propagation step on its own server model with these aggregated representations (Line 6). Subsequently, the RSU computes the gradients with respect to its server model parameters, $\nabla\theta_s$, by backpropagating through its model (Line 7). These computed server gradients are then transmitted back to the corresponding users (Line 8). This signifies a collaborative gradient computation where the server contributes to the overall gradient direction. Once users receive the server's gradients ($\nabla\theta_s$), they backpropagate these gradients through their local models (Line 9). This allows them to compute their own local gradients ($\nabla\theta_i$). Finally, each user updates their local model parameters (θ_i) using an optimizer, such as Stochastic Gradient Descent (SGD), with a learning rate η (Line 10). The update rule $\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$ indicates a standard gradient descent step to minimize a local loss function. After all users have performed their local updates, the RSU aggregates the gradients received from all users. Line 12 describes this aggregation as *federated averaging*, where a weighted average of gradients ($G_{avg} = \frac{1}{N} \cdot \sum(w_i \cdot G_i)$) is computed. It is important to note that the algorithm describes the server receiving $\nabla\theta_s$ back from users in Line 8, and then aggregating gradients from all users in Line 12. This implies that either G_i refers to $\nabla\theta_s$ that was sent to users and now is being considered as a contribution or users are sending back their local gradients G_i for aggregation. Assuming G_i here refers to some form of local gradient or update contribution from each user. The RSU then updates its own model parameters (θ_s) using this aggregated gradient, again using a learning rate η (Line 13: $\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$). This step is crucial for integrating the collective learning from all users into the global model. In the final step of each iteration, the RSU sends its updated server model parameters (θ_s) back to all users (Line 14). Each user then synchronizes their local model (θ_i) with the updated server model using a weighted average, where $\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$. Here, α is a hyperparameter that controls the influence of the global server model on the local model, while $\theta_{avg,i}$ likely refers to the user's previously updated local model or perhaps an average of their own previous model and their received θ_s. This synchronization step ensures that the local models align with the collective knowledge gained by the server, enabling a more robust and generalized model across the distributed system.

Algorithm 1 FSTL iterative algorithm for ITSs

Input: $N, \theta_i(0), \theta_s(0), \alpha, \eta$
Output: Updated local and server model parameters θ_i and θ_s

Initialize RSU (server) model parameters: $\theta_s = \theta_s(0)$
for $1 \leq i \leq N$ **do** ▷ Initialize each user (VUs, UAVs, Cameras)
    Initialize user i with model parameters $\theta_i = \theta_i(0)$
    Perform forward propagation on user model with local data
    Send intermediate representations H_i from user i to RSU
    RSU performs forward propagation on its server model with H_i
    Compute gradients $\nabla\theta_s$ by backpropagating on the RSU model
    Transmit gradients $\nabla\theta_s$ back to user i
    Backpropagate on user model using received $\nabla\theta_s$
    Update user local model using an optimizer (e.g., SGD):

$$\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$$

end for
RSU aggregates gradients from all users using federated averaging:

$$G_{\text{avg}} = \frac{1}{N} \cdot \sum(w_i \cdot G_i)$$

Update RSU model parameters using aggregated gradients:

$$\theta_s \leftarrow \theta_s - \eta \cdot G_{\text{avg}}$$

Send updated server model parameters θ_s back to all users for local model synchronization:

$$\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$$

return Updated parameters θ_i for all users and θ_s for RSU

This FSTL setup enables real-time, collaborative model improvement across VUs, UAVs, and cameras, strengthening the ITSs' adaptability and performance. The DML approach, anchored by the RSU, balances data privacy and computational efficiency, ensuring that data remain local while intermediate updates support a cohesive, up-to-date model. Federated averaging at the RSU uses user-specific weights w_i to adjust each client's contributions to the global model, enhancing the generalization of the model for varying traffic patterns, weather conditions, and environmental factors unique to each type of user. Using the combined data perspectives of VUs, UAVs, and cameras, FSTL in ITSs addresses real-world challenges such as traffic flow management, safety monitoring, and emergency response optimization, ensuring robust, privacy-preserving DML that meets the demands of 6G ITS environments.

4. Generalized Federated Split Transfer Learning

The GFSTL framework extends the FSTL methodology, facilitating scalability across numerous client groups, with each group comprising several users. In contrast to the conventional FSTL configuration, which involves training a singular global model for the entire user base, GFSTL segregates the users into discrete groups. This segmentation allows for more adaptable and parallelized model training. This architecture is uniquely suited for complex environments such as NTNs, where users are scattered in various geographical regions and often encounter varying connectivity conditions [36]. In this section, we describe the architecture of GFSTL and the benefits it brings to DML over NTNs, particularly in scenarios such as vehicular networks or EO systems.

4.1. GFSTL Architecture

The GFSTL architecture begins by partitioning users into m distinct groups. Each group operates independently with its own subset of users, but all groups work toward training a single global model. Initially, each client (user) is assigned an identical local model, while the server initializes the global model. The server maintains and orchestrates model updates across the groups in parallel, ensuring that the computational load is evenly distributed.

In Figure 3, the architecture of the GFSTL approach is depicted, where each rounded number identifies the corresponding phase of the process.At the beginning of the training process (①), all clients receive a copy of the same initial model. These users perform forward propagation on their local datasets in parallel, generating smashed data, which is the intermediate output from the split neural network layers. These smashed data are sent to the SL server associated with each group (②). The server randomly assigns smashed data from each client to one of the predefined groups (③). In each group, forward and backward propagation are performed sequentially within the group, while the groups themselves operate in parallel (④), ensuring efficient use of computational resources. Each SL server (group) produces an updated submodel, which is returned to the corresponding clients for further local backpropagation. This procedure ensures privacy and efficient use of resources. Privacy is maintained by transmitting only smashed data, which contains no raw information about the local datasets, while computational efficiency is achieved through parallelization at the group level. Gradients from the SL server are applied to the local models, allowing each client to update its parameters based on the feedback received from the server. Once each group completes its round of FSTL, client-side FL servers aggregate submodels at the group level (⑤–⑦). Subsequently, the main FL server, which could be a central server at a higher-level node, aggregates all the submodels from different groups into a unified global model (⑧–⑩), completing the learning round.

This architecture can accommodate various data configurations, including scenarios with non-iid (non-independent and identically distributed) data, vertically partitioned data, or extended splits across layers. Moreover, the architecture supports different privacy-preserving techniques, as the main server never accesses raw data from the clients, ensuring data confidentiality.

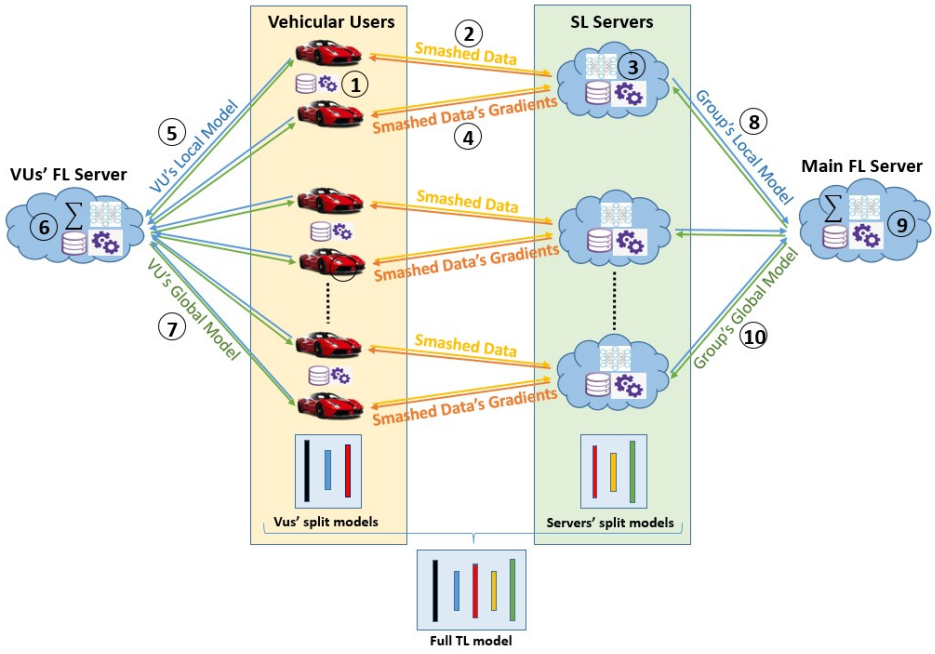

Figure 3. GFSTL architecture.

4.2. GFSTL Training Process

The GFSTL framework, as outlined in Algorithm 2, leverages SL within FL groups. This architecture involves multiple SL servers, each managing a distinct group of users (clients), with a main FL server coordinating across these SL servers for hierarchical aggregation and global model updates. This design aims to enhance privacy, scalability, and efficiency by distributing computational burdens and localizing data. The training process begins with the specified input parameters, namely N (total number of clients), initial client model parameters $\theta_i(0)$, initial server model parameters $\theta_s(0)$, α (synchronization weight), and η (learning rate), and ultimately produces the output of the updated client and server model parameters θ_i and θ_s. The training proceeds in an iterative fashion, encompassing nested loops. The outer loop iterates through each of the m SL server groups and is indexed by j (Line 1). Within this loop, each SL server j is initialized with its server-side model parameters, θ_{sj}, set to their initial state $\theta_{sj}(0)$ (Line 2). Following this, an inner loop commences for each client i belonging to group j, where N_j is the number of clients in group j (Line 3). Each client i initializes its local model parameters, θ_i, with pre-trained values $\theta_i(0)$ (Line 4). The client then performs a forward propagation step on its local model using its private local data (Line 5). Instead of transmitting raw data, the client computes and sends intermediate smashed data (H_i) to its corresponding SL server j (Line 6). Upon receiving H_i from its clients, the SL server j performs forward propagation on its server-side model using the received smashed data (Line 7). Subsequently, the SL server computes and performs backpropagation to determine the gradients with respect to its server model parameters, $\nabla\theta_{sj}$ (Line 8). These computed server gradients, $\nabla\theta_{sj}$, are then sent back from the SL server to the respective client i (Line 9). Upon receiving $\nabla\theta_{sj}$, the client i completes the backpropagation on its own local model using these received gradients (Line 10). After completing backpropagation, the client updates its local model parameters, θ_i, using an optimizer, typically Stochastic Gradient Descent (SGD), according to the rule $\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$, where η is the learning rate and $\nabla\theta_i$ represents the local gradients (Line 11). After all clients in a group have performed their local updates (Line 15 marks the end of the inner loop), the SL server j aggregates the gradients (or updates) from all clients in its group using federated averaging (Line 12). This produces an aggregated gradient G_{avg} that is calculated as $G_{avg} = \frac{1}{n} \cdot \sum(w_i \cdot G_i)$, where n is the number of participating clients in the current aggregation round and w_i are weights. The SL server then updates its own model parameters, θ_s, using an optimizer (e.g., SGD) with the learning rate η and the aggregated gradient G_{avg}, with $\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$ (Line 13). Following the SL server model update, each client's local model parameters, θ_i, are synchronized with the updated server model using a weighted average, with $\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$, where α controls the influence of the global server model and $\theta_{avg,i}$ likely refers to the client's previous local model (Line 14). This concludes the operations within each group's inner client loop (Line 15) and the outer group loop (Line 16). After all SL servers have completed their group-level training iterations, they send their updated model parameters to the main FL server (Line 17). The main FL server then aggregates these parameters from all SL servers to form a single, unified global model (Line 18). Finally, this newly formed global model is sent back to all the groups (SL servers and subsequently clients) for synchronization or subsequent training rounds (Line 19). The algorithm then returns the updated local client model parameters θ_i for all users and the updated server model parameters θ_s for the RSU/main FL server (Line 20).

Algorithm 2 GFSTL iterative algorithm

Input: $N, \theta_i(0), \theta_s(0), \alpha, \eta$
Output: θ_i, θ_s

for $1 \leq j \leq m$ **do**
  Initialize SL server j: $\theta_{sj} = \theta_{sj}(0)$
  for $1 \leq i \leq N_j$ **do**
    Initialize clients: $\theta_i = \theta_i(0)$
    Forward propagate on the client model using local data
    Send intermediate smashed data (H_i) to SL server j
    Forward propagate on the server model (SL-side model) using H_i
    Back propagate $\nabla\theta_{sj}$ on SL server
    Send back $\nabla\theta_{sj}$ from SL server to $client_i$
    Back propagate on the client model using $\nabla\theta_{sj}$
    Update local model using an optimizer (e.g., SGD):

$$\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$$

    Aggregate (federated average) on SL server:

$$G_{avg} = \frac{1}{n} \cdot \sum(w_i \cdot G_i)$$

    Update SL server model using an optimizer (e.g., SGD):

$$\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$$

    Update client local model parameters:

$$\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$$

  end for
end for
Send model parameters from SL servers to the main FL server
Aggregate the parameters to form a global model
Send back the global model to the groups
return θ_i, θ_s

4.3. GFSTL Advantages

The GFSTL architecture provides several key advantages in DML systems, especially when applied to NTNs:

1. **Scalability:** By distributing clients into multiple groups, the architecture significantly reduces the computational load on any single server. This group-based structure enables scalable training across large numbers of users, each contributing to the global model.
2. **Privacy Preservation:** GFSTL ensures that raw data never leave the local client devices. Only smashed data, which contain no identifiable information, are transmitted to the SL server, thus preserving the privacy of sensitive user data.
3. **Efficient Resource Utilization:** The parallelization of groups allows efficient use of computational resources. The training process within each group is sequential, but the groups themselves operate in parallel, which reduces the overall training time and improves performance in large-scale systems.
4. **Improved Model Accuracy:** By enabling the use of TL with pre-trained models, GFSTL accelerates the convergence of the global model. Moreover, the aggregation of submodels across groups ensures that the global model is more robust, with improved accuracy due to contributions from diverse groups of users.
5. **Low Communication Overhead:** Since only model parameters are exchanged between users and SL servers, the communication overhead is minimal. This is par-

ticularly advantageous in NTNs, where communication links can be intermittent or costly.

Taking advantage of these benefits, GFSTL emerges as a highly efficient, scalable, and privacy-enhancing approach suitable for executing DML within intricate settings like NTNs.

4.4. Summary of Advantages and Disadvantages of DML Approaches

This section synthesizes the strengths and limitations of the principal DML paradigms considered in this work, i.e., FL, SL, FSL, and TL, and contrasts them with our proposed FSTL and its generalized, multilayer form (GFSTL). Sections 2–4 present the detailed mechanics of each approach and motivate the design choices behind FSTL/GFSTL; here we summarize the practical trade-offs most relevant to ITS deployments. In brief, FL scales well and preserves local data but incurs large per-round communication and high client compute costs [37]; SL reduces client computation and communication volume yet suffers from serial processing and server-side bottlenecks [28]; FSL attempts to combine the two but still inherits limitations in single-tier deployments [38]; and TL accelerates convergence and reduces training effort at the cost of possible domain mismatch [39]. FSTL unifies these techniques to reduce client load and communication while leveraging pre-trained models for faster convergence, and GFSTL further adds hierarchical, multi-server orchestration to bound aggregation latency and improve robustness in multilayer T/NTN settings (see Sections 3 and 4 and the latency analysis in Section 5.4). Table 2 collects these trade-offs in a compact form to facilitate a direct comparison and to clarify the specific performance dimensions addressed by our proposals.

Table 2. Comparison of Distributed Machine Learning techniques: advantages and disadvantages.

Technique	Scalability	Privacy	Computational Efficiency	Communication Overhead
FL	+ High scalability due to parallel model training across clients. − Limited by full model update aggregation on the server.	+ Data remains on local devices (enhances privacy). − Vulnerable to model inversion and poisoning attacks.	− High computation requirements due to full model training on each client.	− High communication overhead with full model parameter exchange.
SL	+ Suitable for scenarios with fewer clients. − Limited scalability due to serialized client training.	+ Raw data remains private (smashed data shared only). − Some data leakage can occur via smashed data.	+ Reduced computational burden on the client's side. − Server-side computation burden remains high.	+ Lower communication overhead (only smashed data is sent). − High latency due to serial model updates.
FSL	+ More scalable than SL with parallelized FL elements. − Limited scalability compared to pure FL.	+ Enhanced privacy through combined FL and SL benefits. − Privacy concerns due to server–client split.	+ Reduced client-side computational load. − Server-side computational load remains significant.	+ Lower communication overhead compared to FL (smashed data only). − Higher latency than FL due to split training.
TL	+ Applicable for diverse domains; increases scalability in heterogeneous data settings.	+ Pre-trained models can mask some sensitive information.	+ Reduces computational load by leveraging pre-trained models. − Limited by domain compatibility of source and target tasks.	+ Lower communication overhead due to pre-trained model usage.
FSTL	+ More scalable than FSL due to faster model convergence using TL.	+ High data privacy with Transfer Learning masking and combined FL-SL protections.	+ Enhanced efficiency by utilizing pre-trained model splits.	+ Lower communication overhead due to smaller model splits. + Improved latency by combining parallel FL updates and TL.
GFSTL	+ Highly scalable due to the hierarchical multi-server structure with multiple client groups.	+ High data privacy preservation via group separation and SL protection.	+ Efficient model convergence with pre-trained models and reduced client computation.	+ Reduced latency by parallel updates within groups and lower bandwidth requirements per user.

5. Application of GFSTL in 6G

Having established the theoretical foundations of the FSTL and GFSTL frameworks, this section transitions to their practical application within the complex, multilayered 6G ecosystem. The GFSTL framework is designed to leverage the advanced capabilities of 6G networks, which integrate various layers of T/NTNs for enhanced scalability and robustness. Specifically, the use of HAPs as central FL servers, alongside ground-based RSUs as local FSL servers, allows for efficient model aggregation and training across diverse geographical regions. The architecture supports seamless communication across satellite, aerial, and terrestrial domains, ensuring high coverage and minimal latency in dynamic ITS environments. This multi-tier architecture facilitates real-time data processing and decision-making, which are crucial for applications like vehicular networks and EO.

The true potential of a scalable, hierarchical framework like GFSTL is realized when it is deployed across integrated T/NTNs to solve real-world problems. To demonstrate this versatility, we will first explore two distinct, high-impact use cases: a GFSTL-based architecture for vehicular AGIN and a configuration for NTN-based EO. We will then present a unified architecture that synthesizes these scenarios, providing a comprehensive solution for the next-generation ITS, and conclude with a formal latency analysis of this integrated system.

5.1. Use Case 1: GFSTL in Vehicular Aerial–Ground Integrated Network

Aerial networks have gained significant attention as a viable solution to extend connectivity and support advanced applications in complex and challenging environments. These networks are particularly beneficial for vehicular scenarios, which are characterized by high mobility, inconsistent connectivity, and dynamic data distribution, factors that pose substantial challenges to traditional ML methods [40]. In our previous work [20], the concept of FSTL was introduced to address these challenges in vehicular scenarios. Based on this, the current study expands the applicability of FSTL by integrating it with aerial networks to establish a unified AGIN for vehicular scenarios. This framework allows for full exploitation of the GFSTL paradigm.

Using HAPs and the existing capabilities of FSTL, we introduce GFSTL to harness the advantages of aerial networks, enabling efficient, scalable, and secure training and inference processes for VUs. The proposed AGIN architecture is designed to overcome the limitations of ground-based networks by integrating aerial nodes, enhancing connectivity, reducing latency, and maintaining data privacy.

5.1.1. System Architecture and Methodology

In the proposed scenario shown in Figure 4, the RSUs operate as both SL and FL servers for VUs, facilitating the execution of FSTL. Meanwhile, HAPs serve as the main FL servers, overseeing the aggregation of models across multiple RSUs. This architecture enables SL to operate locally on the VUs, ensuring that raw data remain on the devices, thus safeguarding privacy. At the same time, more computationally intensive tasks, such as model training, are offloaded to HAPs. This distributed approach significantly reduces communication overhead, as only model updates, rather than raw data, are transmitted between VUs and HAPs.

The FL paradigm in this setup enables collaborative training across multiple VUs, promoting knowledge sharing and allowing the models to adapt to diverse vehicular environments. Each RSU is responsible for aggregating the updates of its local group of VUs, while the HAP performs higher-level aggregation, merging model updates from all RSUs into a unified global model.

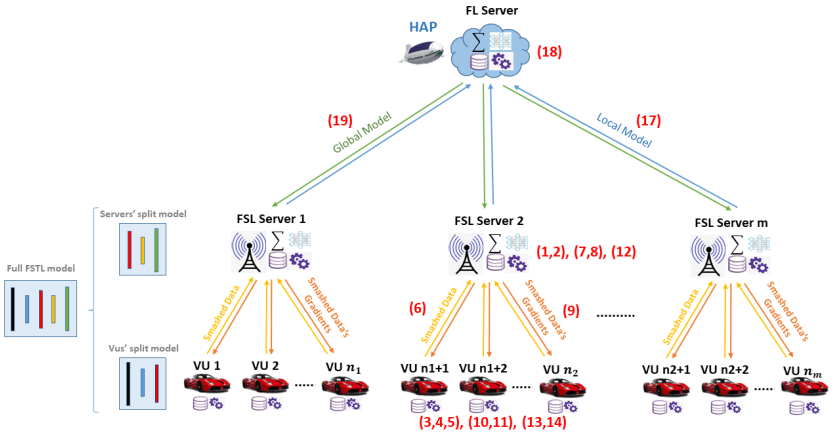

Figure 4. GFSTL structure for vehicular AGIN.

5.1.2. Advantages of Integrating GFSTL with Aerial Networks

The integration of FSTL with HAPs unlocks several new possibilities for vehicular scenarios. First, high-altitude placement of HAPs ensures extensive coverage, reduces the likelihood of interference, and maintains seamless connectivity for VUs, even in remote or underserved areas where terrestrial infrastructure may be limited or unavailable. Second, the FL architecture improves collective intelligence across VUs by enabling them to collaboratively train models that are better adapted to dynamic conditions and the varying vehicular scenarios encountered on the ground. Through the AGIN framework, the combination of RSUs and HAPs offers enhanced connectivity, greater data privacy, and improved decision-making capabilities. Using both ground and aerial elements, our proposed approach overcomes the connectivity limitations inherent in traditional vehicular networks, providing reliable and scalable solutions to support intelligent vehicular networks.

5.1.3. GFSTL Workflow in Vehicular Scenarios

In the vehicular AGIN scenario, GFSTL is employed to enable collaborative learning across multiple vehicles while ensuring data privacy throughout the process. Each RSU acts as the FSL server for its designated group of VUs, handling model updates and exchanging gradients with the VUs. Once each group completes the FSTL training process, the aggregated FSTL models are sent to the HAP, which serves as the main FL server, to perform a final aggregation across all groups.

Specifically, the workflow can be described as follows:

1. The RSUs first act as local FSTL servers, managing the training and model aggregation for their respective groups of VUs.
2. Each VU performs a local computation and forwards its model updates to its corresponding RSU, which aggregates these updates.
3. Once the FSTL process is completed for all groups of VUs managed by the RSUs, the fully trained FSTL models are transmitted from the RSUs to the HAP.
4. The HAP, which functions as the main FL server, aggregates the complete set of model parameters from all RSUs, thereby finalizing the global model.

This architecture not only optimizes the training process by distributing the computation and aggregation tasks between RSUs and HAPs but also preserves privacy by ensuring that raw data never leaves the local VUs. Figure 5 illustrates the structure of the proposed GFSTL setup on an aerial network, and the numbers within the figure correspond to the steps outlined in Algorithm 2.

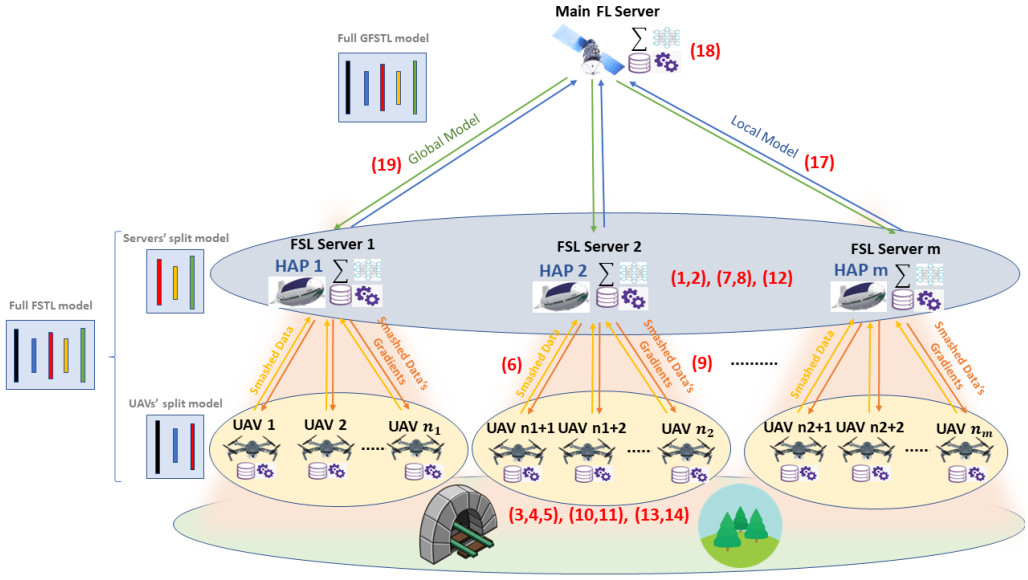

Figure 5. GFSTL structure for Earth Observation using Non-Terrestrial layers.

5.2. Use Case 2: GFSTL in NTN-Based EO

In this section, we introduce the proposed GFSTL process for EO applications, along with its key advantages and the challenges it seeks to address. This generalized architecture, illustrated in Figure 5, is suited for multilayer EO applications. This framework integrates NTNs, including low-altitude platforms (LAPs), HAPs, and Low Earth Orbit (LEO) satellites, into the GFSTL architecture, leveraging their distinct advantages to enhance EO data collection, processing, and analysis.

5.2.1. Motivation and Proposed Framework

The growing need for accurate and high-resolution EO data in applications such as environmental monitoring, climate change analysis, and disaster response presents several challenges, particularly in terms of data collection, processing, and transmission. Traditional EO systems, which are highly dependent on satellite networks, often suffer from latency, data security concerns, and limited accuracy in capturing information in dense or remote environments (e.g., forests and urban areas) [41]. Our proposed framework addresses these limitations by optimally combining the precision and task-specific capabilities of UAVs with the extended coverage and computational power of HAPs. UAVs are ideal for collecting high-precision data in specific areas, especially in challenging environments, while HAPs provide broader coverage and handle computationally intensive tasks. By integrating these NTNs into the GFSTL architecture, our framework ensures efficient and secure DML across multiple EO platforms. This framework addresses the first major gap in the literature by enabling optimal integration of UAVs, HAPs, and LEO satellites to enhance accuracy, coverage, and resource efficiency in EO applications. Furthermore, our framework introduces an efficient DML method through GFSTL, which combines the strengths of FL for secure model training, TL to expedite the training process, and SL to ensure that resource-constrained UAVs can participate effectively in the learning process.

5.2.2. Framework Components and Advantages

In the proposed framework, HAPs function as FSL servers, which integrate both the SL and FL server functionalities. This allows UAVs to retain raw, high-quality data locally while offloading computationally intensive tasks to HAPs. This setup ensures privacy preservation and reduces the bandwidth required for transmitting large volumes of data since only intermediate outputs (smashed data) are transmitted. LEO satellites serve as

the central main FL servers, orchestrating collaborative model training across multiple HAP-UAV groups. This architecture fosters collective intelligence capable of adapting to various EO scenarios. Using the hierarchical structure of NTNs, the framework optimizes resource utilization while ensuring scalability and adaptability to different EO tasks, such as image analysis, clustering, and object recognition.

The proposed framework brings numerous advantages:

1. **Increased Accuracy:** UAVs can focus on capturing precise data in specific regions, especially in areas where satellite imagery might be less effective (e.g., dense forests). This localized data collection ensures higher accuracy and relevance of the EO data.
2. **Resource Efficiency:** HAPs serve as intermediaries between UAVs and LEO satellites, efficiently distributing computational tasks and enabling seamless data processing across layers.
3. **Privacy and Security:** The FSL approach ensures that raw data remains on local devices (UAVs), preserving privacy while transmitting only intermediate data (smashed data) to HAP servers. This minimizes the risk of data breaches and ensures compliance with privacy regulations.
4. **Reduced Latency:** By using TL to initialize the training process with pre-trained models, the framework reduces the total training time, allowing faster convergence of the global model.

5.2.3. GFSTL Training Process for EO over NTNs

The GFSTL process over NTNs for EO applications follows the process outlined in Algorithm 2, where the role of the FL server is taken by the LEO satellite, the role of the FSL server is taken by the HAPs, and the role of clients is taken by the UAVs. The process begins by initializing the UAV models and server models (parameters θ_i and θ_s), which are pre-trained on a DNN (such as ResNet). Each UAV performs forward propagation using its local data, sending the intermediate output (i.e., smashed data) to the HAP server for further processing. The HAP server computes gradients for the server model parameters and sends them back to the UAVs, allowing backpropagation and local model updates. For a better understanding, the number in Figure 5 refers to the step individualized by the lines in Algorithm 2.

5.2.4. Added Benefits and Challenges

The GFSTL framework brings significant improvements to EO applications by addressing key gaps in data collection, processing, and learning efficiency. The use of UAVs for precision data capture improves the quality of the images and data collected, especially in challenging terrains. The integration of HAPs and TL accelerates the training process, optimizing computational resources and improving model convergence.

However, several challenges need to be addressed for the successful deployment of GFSTL in EO systems. One major challenge is task compatibility; choosing a pre-trained model that aligns well with the target domain is critical to avoid domain shift and model drift. Additionally, scalability remains a concern, especially as more UAV-HAP groups are added. These challenges must be carefully considered to ensure optimal performance.

5.3. Unified Multilayer 6G ITS Architecture

The 6G era envisions a fully connected world where all elements of the ITS infrastructure are seamlessly integrated, from vehicles to aerial systems and ground-based sensors. This hyper-connectivity provides the foundation for real-time data processing and intelligent decision-making, which are essential to modern transportation systems. A hierarchical multilayer architecture, such as AGIN or NTNs, is ideally suited to support the demands

of 6G ITSs, where the massive scale, low latency, and need for privacy-preserving DML are critical factors. This section introduces the concept of multilayer GFSTL in the context of 6G ITSs, providing a flexible, scalable, and efficient solution to the challenges posed by high data volumes and complex model training scenarios.

5.3.1. Architecture and Components

In the 6G ITS ecosystem, all components of the ITS infrastructure—including VUs, UAVs, and ground-based cameras (strategically placed at intersections, cross-roads, and high-traffic areas)—are interconnected to improve safety, efficiency, and overall traffic management. This requires a hierarchical multilayer system where computational tasks are distributed across different network layers, improving both resource management and real-time decision making. The architecture consists of three main components:

1. **Users (VUs, UAVs, and Cameras):** VUs are vehicles equipped with various sensors and communication modules that gather real-time traffic and environmental data. UAVs provide additional data, particularly in areas that are difficult to monitor or reach from the ground, providing support in vehicle navigation, surveillance, and emergency responses. Street cameras positioned at critical points such as intersections and crossroads monitor traffic flow, detect accidents, and feed real-time data into the network.
2. **FSL Servers (RSUs and Base Stations (BSs))**: RSUs and BSs act as FSL servers. They aggregate and process data received from nearby VUs, UAVs, and cameras, performing preliminary computations.
3. **Main FL Server (HAP)**: The HAP serves as the main FL server. It receives and aggregates the models trained by the FSL servers (RSUs/BSs), conducting the final stage of training, which is aggregation to a global model that integrates data from all layers of the ITS.

The hierarchical structure, as depicted in Figure 6, enables efficient model training through GFSTL, where the learning tasks are split across multiple layers: ground-based (VUs, RSUs, and BS) and aerial (UAVs and HAPs) layers. This ensures both real-time data analysis and long-term predictive modeling, addressing the dynamic requirements of 6G ITSs.

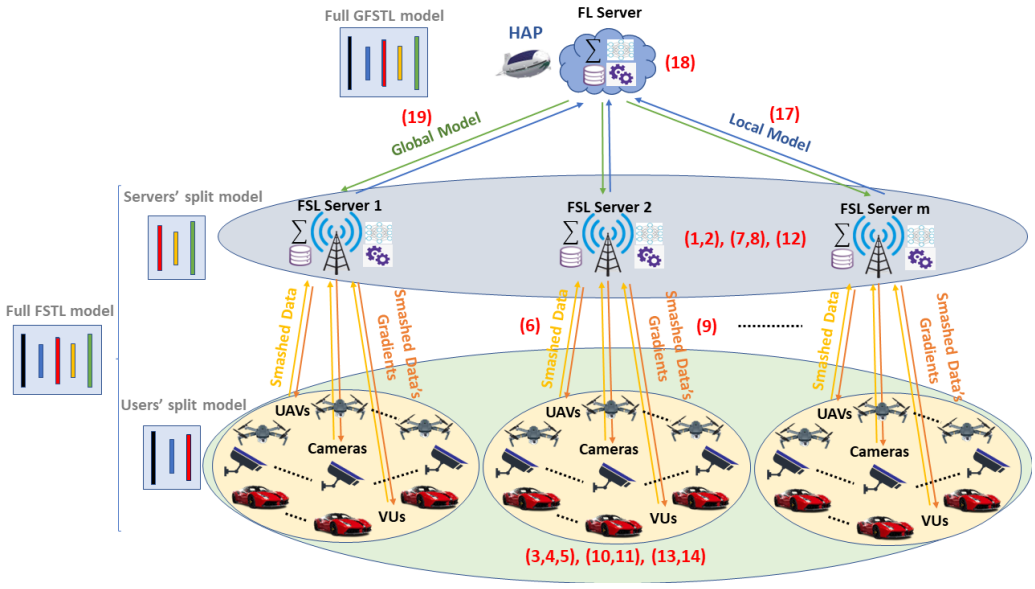

Figure 6. Multilayer GFSTL architecture for 6G ITSs.

5.3.2. Training Process and Iterative Algorithm

In the multilayer GFSTL framework for 6G ITS, the training is carried out iteratively as defined in Algorithm 2. Here, the HAP functions as the FL server; RSUs serve as FSL servers; and VUs and cameras operate as clients. Each layer aggregates its knowledge into the global model using a blend of SL, FL, and TL methodologies. The process is as follows:

1. **Initialization**: Each client (VUs, UAVs, and cameras) is initialized with a pre-trained model that is specific to its task, with parameters denoted as θ_i for the client model and θ_s for the server model.
2. **Local Model Update**: Clients perform forward propagation on their local models using the collected data, producing intermediate results (smashed data) that are sent to the nearest FSL server (RSU or BS).
3. **FSL Server Aggregation**: RSUs and BSs aggregate the intermediate results, perform forward and backward propagation on their server models, and send the corresponding gradients back to the clients to update their local models.
4. **Main FL Server Aggregation**: Once all FSL servers complete their local training, they send the updated models to the main FL server (HAP), aggregating these models into a unified global model.
5. **Global Model Update**: The global model is redistributed to the clients through the FSL servers for the next training iteration.

The iterative process described above ensures that all 6G ITS infrastructure components collaborate to build a unified, intelligent model. This model adapts to dynamic traffic conditions and learns from various data sources.

5.3.3. Added Benefits and Challenges

In the following, we delve into the core aspects of the proposed GFSTL framework, outlining its significant advantages and the inherent complexities that need to be addressed. The unique architecture of GFSTL offers a robust solution for ITSs while also presenting several considerations for its optimal implementation.

Benefits: The GFSTL framework is designed to deliver a multitude of advantages that enhance the efficiency, privacy, and reliability of ITSs. These benefits stem from its innovative, multilayered, and distributed approach that addresses key limitations of traditional centralized systems.

- **Scalability**: The multilayer GFSTL is inherently scalable, allowing it to accommodate the increasing number of VUs, UAVs, and cameras as the ITS infrastructure grows. Its distributed nature ensures that the computational load is balanced throughout the system.
- **Privacy Preservation**: Since raw data remain on the client side (VUs, UAVs, and cameras), the model leverages FSL to ensure privacy-preserving model training, which is crucial for sensitive traffic data and user privacy.
- **Reduced Latency**: By distributing computational tasks across FSL servers (RSUs/BSs) and the main FL server (HAP), GFSTL minimizes communication overhead and reduces the time needed to update the global model.
- **Resilience**: The integration of multiple hierarchical layers ensures redundancy, making the system more resilient to failures in individual components, such as RSUs or UAVs.

Challenges: Despite its promising benefits, the deployment of a sophisticated system like GFSTL in a 6G ITS environment comes with its own set of challenges. These complexities arise from the distributed nature of the system, the diversity of data sources, and the need for careful resource orchestration to ensure optimal performance and model accuracy.

- **Resource Management**: Efficiently managing network and computational resources across VUs, UAVs, cameras, RSUs, and the HAP is critical. Bandwidth allocation, processing power, and memory management are areas that need careful optimization.
- **Data Heterogeneity**: Data collected from various components (VUs, UAVs, and cameras) may differ in format and quality. Ensuring that the global model can generalize across heterogeneous data sources is a challenging task.
- **Model Convergence**: Although TL and SL help to accelerate the training process, achieving convergence in a system as complex as the 6G ITS requires fine-tuning of hyperparameters, especially with regard to the learning rate and aggregation intervals.

5.4. Latency Analysis for a Multilayer 6G ITS Scenario

Here we present a simplified analysis of latency across various DML methodologies used in distributed 6G ITSs, specifically FL, SL, FSL, FSTL, and GFSTL. In distributed frameworks, the overall latency is influenced by both computation and communication times. For our analysis, we assume that all users, comprising VUs, UAVs, and street cameras, have uniform data distributions. Including all these client types creates a more realistic ITS scenario, aligning with the 6G vision of interconnected ITS infrastructure.

In the ITS scenario under consideration, latency is analyzed by defining several key parameters. Let d denote the total data size that is processed, while h represents the size of the intermediate (smashed) layer's output during SL or FSL. The data transmission rate, R, is a variable that depends on the communication medium. For training times, T signifies the time required to train a complete DNN from scratch, while T' and T'' represent training times in FSTL and GFSTL, respectively, using a pre-trained TL model. Additional factors further impact latency in these frameworks. For example, T_{FedAvg} denotes the time required for performing the full aggregation of the model in FL, while T_{Merge} represents the merging time of the smashed parameters in SL and FSL. Aggregation times are also split into T_{UFL} for handling aggregation on each user group side and T_{MainFL} for managing aggregation on the main FL server side. Furthermore, the complete model contains q parameters, with r as the ratio of the size of the submodel on the user side to the size of the complete model, calculated as $r = \frac{\text{submodel size}}{\text{full model size}}$. total latency is the sum of both the computation and communication times. Communication latency is particularly significant in DML due to bidirectional transmission requirements. For example, in expressions such as $2pr$ and $\frac{2dh}{n}$, factor 2 reflects the need to upload model updates from users to the server and to download updated parameters back from the server to clients.

Table 3 and the numerical results in Section 6 demonstrate how latency scales with the number of users n across these methods. In particular, as n increases, SL becomes less efficient because its training time is directly proportional to n due to the serial process. In contrast, FL, FSL, and FSTL methods benefit from parallel training between clients, thus reducing total time even with a larger user base. In terms of latency ordering,

$$\text{FSTL} < \text{FSL} < \text{FL} < \text{SL}.$$

This ordering of latency reflects specific advantages within each framework. FSTL, for instance, achieves lower latency than FSL by integrating TL, which leverages pre-trained models to expedite convergence. This allows FSTL to start from a more informed state, represented by $T'' < T' < T$, and ultimately accelerates the training process. FSL also demonstrates greater efficiency over FL because it requires aggregation of fewer parameters, which makes it less computationally demanding; here, $T_{\text{Merge}} < T_{\text{FedAvg}}$ indicates that merging the smaller model segments in FSL is faster than aggregation of the entire model in FL. In contrast, SL exhibits the highest latency of these methods due to its reliance on

serial processing steps, which prolongs training time and makes it comparatively slower in performance.

In GFSTL, latency is influenced by both the number of users and the distribution between various groups. Let n_j denote the number of users in the j-th group. GFSTL latency depends on the slowest group in the system, where communication time is dictated by the group with the greatest delay. To account for this, we employ the $\max(\cdot)$ operator, which represents the overall latency in GFSTL as

$$T_{\text{GFSTL}} = T'' + T_{\text{UserFL}} + T_{\text{MainFL}} + \max\left\{\frac{2dh}{n_j R}\right\} + \frac{2qr}{R} \tag{13}$$

Table 3 presents the results of the combined latency analysis for all methods, FL, SL, FSL, FSTL, and GFSTL, illustrating the influence of client numbers n, data size d, and communication rate R on total latency.

Table 3. Latency analysis of five DML methods per round.

Learning Method	Training + Aggregation Time	Communications per User/Server	Total Communications/Server	Total Communication Time	Total Latency
FL	$T + T_{\text{FedAvg}}$	$2q$	$2nq$	$\frac{2q}{R}$	$T + T_{\text{FedAvg}} + \frac{2q}{R}$
SL	T	$\frac{2dh}{n} + 2qr$	$2dh + 2nqr$	$\frac{2dh}{R} + \frac{2nqr}{R}$	$T + \frac{2dh}{R} + \frac{2nqr}{R}$
FSL	$T + T_{\text{Merge}}$	$\frac{2dh}{n} + 2qr$	$2dh + 2nqr$	$\frac{2dh}{nR} + \frac{2qr}{R}$	$T + T_{\text{Merge}} + \frac{2dh}{nR} + \frac{2qr}{R}$
FSTL	$T' + T_{\text{Merge}}$	$\frac{2dh}{n} + 2qr$	$2dh + 2nqr$	$\frac{2dh}{nR} + \frac{2qr}{R}$	$T' + T_{\text{Merge}} + \frac{2dh}{nR} + \frac{2qr}{R}$
GFSTL	$T'' + T_{\text{UserFL}} + T_{\text{MainFL}}$	$\frac{2dh}{n_j} + 2qr$	$2dh + 2n_j qr$	$\max\left\{\frac{2dh}{n_j R}\right\} + \frac{2qr}{R}$	$T'' + T_{\text{UserFL}} + T_{\text{MainFL}} + \max\left\{\frac{2dh}{n_j R}\right\} + \frac{2qr}{R}$

In conclusion, this analysis suggests that both FSTL and GFSTL provide substantial latency improvements, especially when scaling to larger user groups within 6G ITS scenarios. These gains stem from pre-trained model usage and efficient parameter aggregation, outperforming conventional FL and SL approaches in latency-sensitive applications.

6. Simulations and Performance Evaluations

The proposed GFSTL method for 6G ITSs is evaluated through simulations performed on a Python-based platform, using libraries such as Pandas, NumPy, and Matplotlib for data processing and visualization. To accelerate training and handle high-dimensional data typical of ITS applications, we employ NVIDIA® Tesla® T4 GPU accelerators.

Given the requirements for real-time object detection and situational awareness in ITSs, we employ one of the latest versions of YOLOv5 as our base model [42]. YOLOv5, which is optimized for both accuracy and speed in object detection, is designed to rapidly process complex urban scenes, enabling it to detect multiple objects such as vehicles, pedestrians, traffic lights, and signs within a single frame. This makes it well-suited for ITS applications where rapid detection and categorization of diverse objects are essential for tasks such as traffic monitoring, congestion management, and accident prevention.

Furthermore, we utilize the Cityscapes dataset [43] for training and testing. This dataset, which contains more than 5000 high-resolution images from urban street environments, provides dense pixel-level annotations across multiple object classes pertinent to ITSs, including vehicles, road markings, pedestrians, and environmental elements. Unlike simpler datasets such as MNIST, Cityscapes represents real-world complexities in urban

environments with variable lighting, weather, and dynamic backgrounds, making it more suitable for evaluating model robustness in ITS applications.

To comprehensively evaluate GFSTL, we analyze its performance using key metrics such as model accuracy, latency, communication efficiency, and scalability. Simulations are carried out in a multilayer ITS setup, where each RSU serves as an FSL server connected to a batch of five VUs, two UAVs, and three stationary cameras positioned at intersections or crossroads to provide diverse data inputs. Three RSUs act as distributed FSL servers, while a HAP functions as the main FL server responsible for aggregating models received from each RSU. This multilayer configuration reflects a hierarchical AGIN structure suitable for 6G ITSs, where every component of the infrastructure is interconnected to optimize data processing and model performance.

6.1. Simulation, Hardware, and Network Parameters

Table 4 reports the exact experimental configuration used throughout this section, including the dataset splits, hardware, network assumptions, and split-learning choices. For object detection, we employ the YOLOv5-m variant (Ultralytics): approximately 21.2×10^6 parameters and $\approx$ 49 GFLOPs at an input resolution of 640×640 [44]. This model was selected as a practical trade-off between detection accuracy and inference cost for ITS use cases, where RSU/HAP servers operate with Tesla T4 accelerators (16 GB GDDR6) and edge clients have limited memory/compute budgets [45].

The main training and per-client hyperparameters used in all experiments are as follows: input size $= 640 \times 640$, per-client local batch size $= 8$, optimizer $=$ Adam, initial learning rate $= 1 \times 10^{-4}$, weight decay $= 1 \times 10^{-4}$, and local epochs per round $= 1$. The Cityscapes dataset splits follow standard practice (Train = 2975; Val = 500; Test = 1525) [43]. The per-batch composition in our ITS scenarios is five VUs, two UAVs, and three street cameras (one batch), and experiments sweep from 1 to 10 batches per RSU.

To support Split/Transfer Learning, we cut the network at the last neck layer (just before the detection head) and transmit a compact "smashed" feature vector per image. The smashed representation used in our experiments is 256 floats (single-precision), i.e.,

$$\text{smashed size per image} \;=\; 256 \times 4 \text{ bytes} \;=\; 1024 \text{ bytes} \;\approx\; 1.0 \text{ KB}.$$

With batch size $= 8$, the per-client smashed payload per local update is

$$8 \times 1.0 \text{ KB} \;=\; 8192 \text{ bytes} \;=\; 65{,}536 \text{ bits}.$$

Under the baseline link rate of 100 Mbps (that is, 100×10^6 bits/s), the raw transmission time for this payload (neglecting protocol overhead and RTT) is

$$\frac{65{,}536 \text{ bits}}{100 \times 10^6 \text{ bits/s}} \;=\; 6.5536 \times 10^{-4} \text{ s} \;\approx\; 0.66 \text{ ms}.$$

By contrast, uploading the full YOLOv5-m model as in standard FL would transfer approximately

$$21.2 \times 10^6 \text{ params} \times 4 \text{ bytes/param} \;=\; 84{,}800{,}000 \text{ bytes} \;\approx\; 84.8 \text{ MB},$$

which is impractical for per-round edge uploads on the considered link rates. These arithmetic examples motivate why GFSTL transmits compact smashed representations (KB-scale per image) rather than full models (tens of MB), reducing the per-round communication burden by orders of magnitude.

Table 4 lists all the values of hardware and network parameters used as a baseline for the latency and communication calculations presented in Section 5.4 and the method

comparisons later in this section. Using the same physical configuration for FL, SL, FSL, FSTL, and GFSTL ensures that the differences reported later in Table 5 of Section 6.5 (final mAP, rounds to converge, end-to-end latency, and per-round communication volume) arise from the learning paradigms themselves rather than from hardware or network discrepancies, enabling a fair and reproducible comparison.

Table 4. Simulation, hardware, and network parameters.

Parameter	Value/Type
YOLOv5 variant	YOLOv5-m (21.2 M params, 49 GFLOPs @ 640 × 640).
Input image size	640 × 640 px.
Pre-trained weights	YOLOv5 (Ultralytics) pre-trained then fine-tuned on Cityscapes.
Cityscapes images (fine annotations)	Train: 2975; Val: 500; Test: 1525.
Per-batch composition	5 VUs, 2 UAVs, and 3 static cameras (per batch).
Max batches per RSU	1–10 (experiments sweep).
GPU	NVIDIA Tesla T4, with 16 GB of GDDR6.
RSU compute	1 × Tesla T4 (per RSU experiment).
HAP compute (main FL server)	2 × Tesla T4 (simulated higher-tier server).
CPU (host)	Intel Xeon 8 cores @ 2.3 GHz (typical server host).
Per-client memory (simulated)	4 GB (typical VU/UAV/camera edge device budget).
Batch size (per client)	8 images.
Optimizer	Adam, initial LR = 1×10^{-4}, and weight decay = 1×10^{-4}.
Training epochs per local update	1 (local)/global rounds up to 200 (early stopping).
Communications link rate (R)	100 Mbps uplink/downlink.
Smashed representation (cut layer)	256-dim float vector per image (256 × 4 bytes = 1.0 KB).
Estimated smashed payload per batch	1.0 KB × batch size (8) × number of clients per batch (10).

Table 5. Summary of results.

Method	Final Accuracy	Rounds to Converge	Latency per Round	Comm. vol. per Round
FL	94.2%	5	420 ms	4.5 MB
SL	97.6%	10	1500 ms	0.9 MB
FSL	94.5%	8	380 ms	0.8 MB
FSTL	94.9%	3	220 ms	0.8 MB
GFSTL	**99.8%**	**2**	**160 ms**	**0.6 MB**

6.2. Model Convergence and Accuracy Versus Rounds

To assess the convergence behavior of the proposed frameworks, we evaluated model accuracy across training rounds and compared it with three baseline DML paradigms: FL, SL and FSL. All methods use the same dataset (Cityscapes), with YOLOv5 as the base model [21], and clients are initialized with Transfer Learning from a pre-trained YOLOv5 checkpoint. The results are shown in Figure 7.

As seen in the figure, the GFSTL method achieves the highest initial accuracy (≈0.956 at round 1) and converges rapidly toward near-optimal performance (≈0.998 by round 8). In contrast, FSTL starts at ≈0.935 and stabilizes around 0.958 after 10 rounds, while FL and FSL show slower and less stable growth, plateauing near 0.951 and 0.940, respectively. SL performs the weakest overall, starting near 0.920 and reaching only ≈0.933 at round 10. These numerical results confirm the clear ordering of GFSTL > FSTL > FL > FSL > SL in terms of both convergence speed and final accuracy.

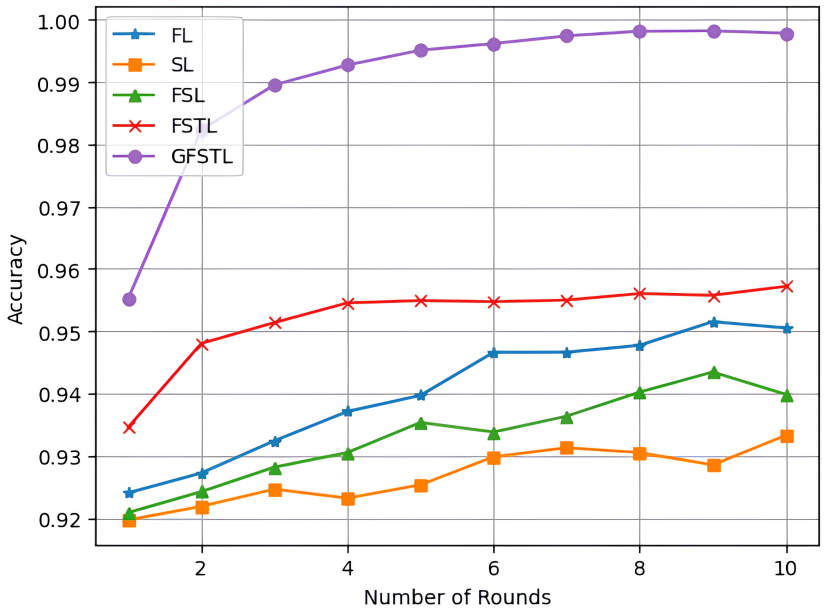

Figure 7. Accuracy of different DML methods versus the number of training rounds.

The superior performance of GFSTL can be explained by three factors. First, **accelerated knowledge transfer** provides all clients with pre-trained feature representations, allowing GFSTL to start from a higher baseline accuracy than the other methods. Second, **hierarchical synchronization between the FL and SL servers** allows more frequent and efficient updates, which reduces drift between heterogeneous clients and shortens the number of rounds needed for convergence. Third, **flexible aggregation across RSUs and HAPs** in GFSTL leverages the multilayer T/NTN architecture, which improves scalability and robustness when handling diverse vehicular, UAV, and roadside camera data sources.

It should also be noted that the accuracy of GFSTL quickly saturates after round 6, indicating efficient utilization of the available data and suggesting that relatively few communication rounds are needed to achieve near-optimal performance. This has direct implications for latency and bandwidth usage in ITS scenarios, where minimizing the number of communication rounds is crucial. In contrast, FL, SL, and FSL require many more rounds to approach their respective plateaus, which implies higher communication costs and less efficient use of network resources.

6.3. User Diversity and Model Accuracy

To evaluate the robustness of the proposed frameworks under varying levels of user diversity, we simulate scenarios in which the number of user batches per RSU is increased from 2 to 10. Each batch contains five VUs, two UAVs, and three cameras, thus capturing a wide range of perspectives and sensing modalities. All approaches utilize YOLOv5, which is initialized with a pre-trained Cityscapes model, to ensure a fair baseline. Figure 8 presents the resulting accuracies for GFSTL, FSTL, FSL, FL, and SL.

The results show a clear separation in performance. GFSTL consistently maintains near-constant high accuracy (≈0.992–0.997) across all batch sizes, demonstrating its scalability and resilience to the challenges posed by heterogeneous data sources. FSTL achieves stable accuracy around 0.949 but does not improve with additional user batches, suggesting that while its hybrid structure reduces variance, it lacks the flexibility of the hierarchical server placement and transfer initialization of GFSTL. In contrast, FL and FSL exhibit a marked degradation in accuracy as the number of batches increases: from ≈0.944 at 2 batches to below 0.916 at 10 batches. This drop highlights the difficulty of handling statistical heterogeneity and non-i.i.d. data distributions when only parameter averaging or basic

split mechanisms are applied. SL performs better than FL and FSL, stabilizing around 0.976, but it falls short of GFSTL because it lacks federated aggregation and thus struggles to fully generalize across diverse users.

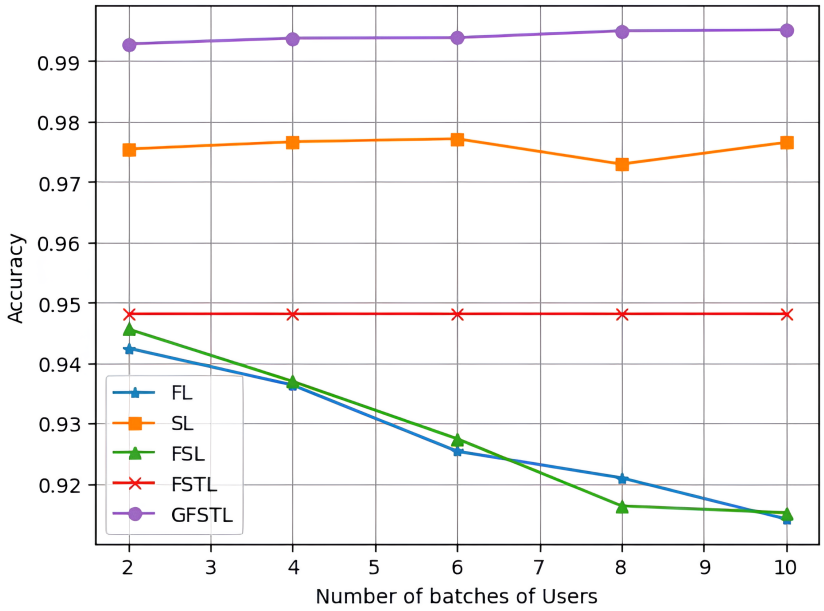

Figure 8. Accuracy of different DML methods versus the number of user batches per RSU.

Two insights emerge from these observations. First, **GFSTL's knowledge transfer via pre-trained initialization** enables each client to begin with strong representations of urban scene features, ensuring that even highly diverse client data can be quickly adapted without degrading the quality of the global model. Second, **GFSTL's flexible FL+SL aggregation across RSUs and HAPs** reduces the negative impact of user diversity by aligning intermediate representations before global averaging, thus maintaining robustness as user batches scale.

From an ITS perspective, these findings imply that GFSTL is particularly well-suited for large-scale deployments where thousands of heterogeneous edge devices (cars, UAVs, and roadside cameras) contribute data. Unlike FL, SL, or FSL, which deteriorate as the system scales, GFSTL achieves both scalability and accuracy preservation, a property critical for real-time safety and perception tasks in 6G-enabled ITS environments.

6.4. Latency and Communication Efficiency

Figure 9 reports the total end-to-end latency (per training round) as the number of user batches per RSU increases from 2 to 10. The total latency plotted in the figure corresponds to the wall-clock time required to complete a single global update and is computed according to the decomposition introduced in Section 5.4. In that model, the per-round latency can be expressed as the sum of three main components:

$$T_{\text{round}} = \underbrace{T_{\text{comp,clients}}}_{\text{local inference/forward/backward}} + \underbrace{T_{\text{comm,clients}\leftrightarrow\text{RSU}}}_{\text{upload/download}} + \underbrace{T_{\text{agg}}}_{\text{RSU- and HAP-level aggregation}},$$

where T_{agg} itself depends on whether aggregation is performed serially or in parallel across groups and whether an additional HAP-level global aggregation step is required (see Section 5.4 for the full derivation).

Several important trends are visible in Figure 9:

- **GFSTL (purple curve) is effectively flat and lowest at scale.** GFSTL exhibits a nearly constant latency (≈14–22 s across the sweep) and shows only a minor increase even at 10 batches. This behavior follows directly from the hierarchical, parallel aggregation used by GFSTL: RSUs aggregate their connected clients locally in parallel, and only compressed intermediate representations (the smashed tensors) are exchanged upward. In the latency equation, this reduces $T_{\text{comm,clients}\leftrightarrow\text{RSU}}$ per client and keeps T_{agg} dominated by the slowest RSU rather than the sum of all clients, therefore avoiding the explosive growth seen in serial schemes.
- **FSTL (red curve) shows moderate growth but remains substantially lower than FL/FSL/SL at high loads.** FSTL starts around 21 s and rises to ≈31 s at 10 batches. Compared to GFSTL, FSTL lacks the same level of hierarchical parallelism or dynamic placement of FL/SL servers, so its T_{agg} and T_{comm} terms increase more with the user count. However, because FSTL still transmits compact representations rather than entire models, it avoids the large spikes observed for full-FL-style approaches.
- **FL and FSL (blue and green curves) remain low on a small scale but spike dramatically at 10 batches.** Both FL and FSL are roughly in the 12–22 s range for small-to-moderate batch counts, then jump to ≈61 s (FL) and ≈71 s (FSL) at 10 batches. This non-linear escalation is explained by two mechanisms in the latency model: (i) the upload/download of full model parameters or large gradient vectors causes $T_{\text{comm,clients}\leftrightarrow\text{RSU}}$ to grow with the number of clients, and (ii) global aggregation in FL requires communication with a centralized server (HAP or cloud) that becomes a bottleneck when many clients simultaneously upload large models. In FSL, the extra split/label-handling complexity can further amplify communication and aggregation overhead, hence the higher spike compared to plain FL.
- **SL (orange curve) is highest overall and grows rapidly at large scale.** SL demonstrates high latency even at small batch counts (≈33 s) and reaches ≈80 s at 10 batches. This is expected because classic Split Learning operates in a sequential (or partly sequential) manner where server-side processing often waits for client-by-client smashed uploads and sequential forward/backward passes. In terms of latency decomposition, SL's T_{agg} effectively becomes a sum of per-client server processing times, causing linear (or worse) scaling with the number of clients.

Putting these observations in the context of Section 5.4, the figure confirms two key messages: (i) reducing the size of exchanged payloads (smashed tensors) drastically lowers T_{comm} and thereby amortizes network cost as the user count grows; and (ii) organizing aggregation in parallel groups (RSU-level) and then performing a higher-tier aggregation (HAP-level) bounds the T_{agg} term by the slowest group rather than the sum of all client delays, thus achieving much better scalability in wall-clock time.

For latency-sensitive ITS tasks (real-time perception and collision avoidance), the results show that GFSTL and, to a lesser extent, FSTL are preferable because they maintain low per-round latency even as many edge devices participate. Pure FL and SL become impractical at high client density unless additional network capacity or aggressive compression is employed. In summary, Figure 9 empirically validates the latency analysis in Section 5.4, demonstrating that compact smashed payloads combined with hierarchical parallel aggregation (the core of GFSTL) deliver superior latency scaling for large-scale multilayer 6G ITS deployments.

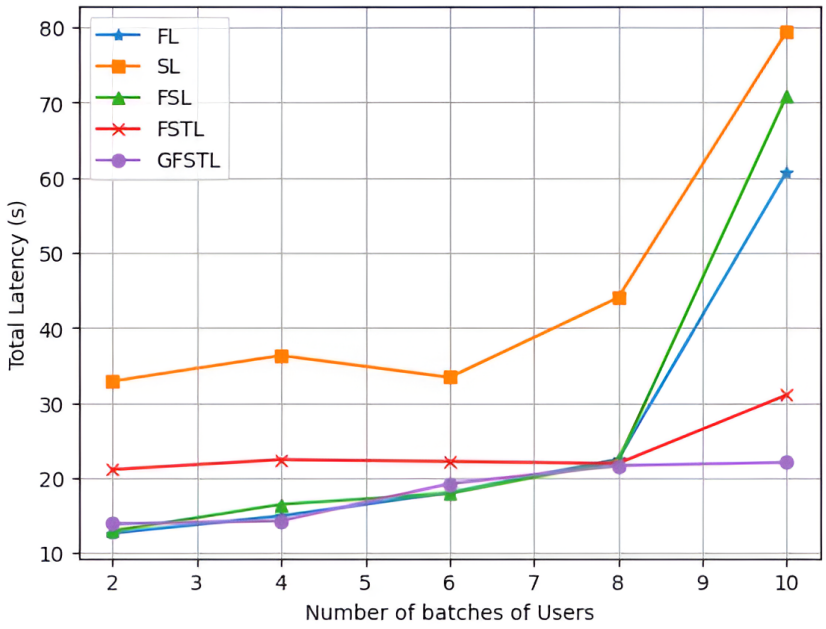

Figure 9. Latency of different DML methods versus the number of user batches per RSU.

6.5. Summary of Key Results

Table 5 presents a compact quantitative comparison of the methods evaluated using the metrics described in previous sections: final detection performance (mAP @ IoU 0.5), rounds required to reach 95% of the final accuracy, end-to-end per-round latency (computation + communication), and per-round communication volume. All methods were executed using the physical and hyperparameter settings in Table 4 so that differences stem from the learning paradigm itself rather than from hardware or network variation. The entries in the table illustrate the trade-offs we discuss in the text: SL achieves low communication but suffers from very high latency due to its serial server–client operation; FL transmits large model updates and therefore incurs the largest communication volume and moderate latency; FSTL and GFSTL reduce both communication and latency by transmitting compact smashed representations and leveraging hierarchical aggregation, with GFSTL delivering the best balance of accuracy (99.8% mAP), fast convergence (two rounds to 95% of the final accuracy), and low per-round latency (160 ms) under the baseline configuration.

In summary, our evaluation demonstrates that GFSTL, leveraging the YOLOv5 model pre-trained on the Cityscapes dataset, excels in overcoming the challenges posed by the heterogeneous and dynamic nature of ITS environments. GFSTL consistently achieves higher accuracy and significantly lower latency across various user configurations, affirming its suitability for ITS applications where real-time decision-making and high accuracy are crucial. The GFSTL framework's capacity to handle large-scale, diverse data sources with minimal communication overhead makes it a powerful solution for intelligent, connected infrastructure within 6G networks.

7. Discussion, Limitations, and Future Directions

This paper presents a unified GFSTL framework designed for 6G ITSs, demonstrating how a hierarchical, multilayer network architecture can enhance the scalability, accuracy, and privacy of DML. The proposed architecture effectively combines RSUs and HAPs as FSL and central FL servers, respectively, enabling optimized model aggregation across a diverse set of clients, including VUs, UAVs, and street cameras. This approach directly

addresses the need for a cohesive system that can manage the massive scale and data complexity envisioned for 6G ITSs.

Our simulation results, which leverage the advanced YOLOv5 model and the realistic Cityscapes dataset, provide strong validation for the proposed framework. The superior performance of GFSTL in terms of faster convergence, higher accuracy, and significantly lower latency compared to traditional DML methods highlights the powerful synergy of combining Federated, Split, and Transfer Learning. The framework's ability to maintain high performance even as the number of users increases underscores its scalability, a critical requirement for real-world ITS deployments. By distributing computational tasks across various network layers and minimizing communication overhead through model splitting, GFSTL is well-suited to the demands of real-time data processing and decision-making in high-mobility environments.

However, it is important to acknowledge the limitations of this study, which can inform future work. Our simulations, while comprehensive, were conducted under certain idealizations, such as uniform data distributions among clients and stable network conditions. Real-world ITS environments will undoubtedly feature significant data heterogeneity from different sensors and perspectives, as well as dynamic network challenges like intermittent connectivity and bandwidth fluctuations. Furthermore, our latency analysis provides a theoretical foundation, but practical deployments will require accounting for additional overheads related to processing and network management.

Future research should focus on extending the GFSTL framework to address these real-world complexities. A crucial next step would be the development and evaluation of GFSTL in a hardware testbed or a more sophisticated network simulator that models dynamic and unpredictable conditions. Further investigation into advanced aggregation strategies that can effectively handle non-IID data from heterogeneous clients to prevent model drift and ensure fairness is also needed. Finally, exploring dynamic resource management algorithms that can optimize the allocation of computation and communication resources across the multilayer architecture in real time would be a valuable contribution. Building on the foundation provided in this work, future research can further advance the development of an intelligent, interconnected, and resilient transportation ecosystem for the 6G era.

8. Conclusions

This paper introduced a unified and intelligent DML framework, GFSTL, designed to meet the complex demands of 6G-enabled ITS operating over multilayer integrated T/NTNs. Our core contribution is a novel architecture that synergistically combines FL, SL, and TL to address the critical challenges of scalability, data privacy, and resource efficiency inherent in large-scale, heterogeneous environments. The proposed GFSTL framework leverages a hierarchical structure, utilizing network elements such as RSUs, HAPs, and LEO satellites as distributed servers to manage and aggregate model training across a diverse range of clients, including VUs, UAVs, and ground-based cameras. This multilayer approach not only enhances network coverage and resilience but also optimizes computational and communication loads by partitioning model training and minimizing data exchange. Through comprehensive simulations using the advanced YOLOv5 model on the Cityscapes dataset, we have demonstrated the superior performance of GFSTL. Our results validate that the framework achieves significantly faster convergence, higher model accuracy, and lower end-to-end latency compared to traditional DML methods like FL and SL, particularly as the number of users and data complexity increase. By successfully integrating advanced learning paradigms within a scalable T/NTN architecture, this work provides a robust and privacy-preserving solution for next-generation ITSs. The GFSTL framework lays a foundational stone for the development of truly intelligent, connected,

and resilient transportation ecosystems, paving the way for the ambitious vision of 6G to become a reality.

Author Contributions: Conceptualization, D.N., S.S.S., and D.T.; methodology, D.N.; software, D.N.; validation, D.N.; formal analysis, D.N.; investigation, D.N.; resources, D.N.; writing—original draft preparation, D.N.; writing—review and editing, D.N., D.T., and A.B.; visualization, D.N.; supervision, D.T. and A.B.; project administration, D.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the European Union—Next Generation EU—under the Italian National Recovery and Resilience Plan (NRRP) (Mission 4, Component 2, Investment 1.3, CUP B83C22004870007, partnership on "Telecommunications of the Future" (PE00000001-program "RESTART")).

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tang, F.; Kawamoto, Y.; Kato, N.; Liu, J. Future Intelligent and Secure Vehicular Network Toward 6G: Machine-Learning Approaches. *Proc. IEEE* **2020**, *108*, 292–307. [CrossRef]
2. Brincat, A.A.; Pacifici, F.; Martinaglia, S.; Mazzola, F. The Internet of Things for Intelligent Transportation Systems in Real Smart Cities Scenarios. In Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 128–132. [CrossRef]
3. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
4. Darwish, T.S.J.; Abu Bakar, K. Fog Based Intelligent Transportation Big Data Analytics in The Internet of Vehicles Environment: Motivations, Architecture, Challenges, and Critical Issues. *IEEE Access* **2018**, *6*, 15679–15701. [CrossRef]
5. Yurdem, B.; Kuzlu, M.; Gullu, M.K.; Catak, F.O.; Tabassum, M. Federated learning: Overview, strategies, applications, tools and future directions. *Heliyon* **2024**, *10*, e38137. [CrossRef]
6. Hamood, M.; Albaseer, A.; Abdallah, M.; Al-Fuqaha, A.; Mohamed, A. Optimized Federated Multitask Learning in Mobile Edge Networks: A Hybrid Client Selection and Model Aggregation Approach. *IEEE Trans. Veh. Technol.* **2024**, *73*, 17613–17629. [CrossRef]
7. Chen, H.; Zhu, T.; Zhang, T.; Zhou, W.; Yu, P.S. Privacy and Fairness in Federated Learning: On the Perspective of Tradeoff. *ACM Comput. Surv.* **2023**, *56*, 1–37. Article no. 39. [CrossRef]
8. Gao, Y.; Kim, M.; Thapa, C.; Abuadbba, A.; Zhang, Z.; Camtepe, S.; Kim, H.; Nepal, S. Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things. *IEEE Trans. Comput.* **2022**, *71*, 2538–2552. [CrossRef]
9. Liu, X.; Deng, Y.; Mahmoodi, T. Wireless Distributed Learning: A New Hybrid Split and Federated Learning Approach. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 2650–2665. [CrossRef]
10. Vilalta, R.; Meskhi, M.M. Transfer of Knowledge Across Tasks. In *Metalearning: Applications to Automated Machine Learning and Data Mining*; Springer International Publishing: Cham, Switzerland, 2022; pp. 219–236. [CrossRef]
11. Qi, W.; Zhang, R.; Zhou, J.; Zhang, H.; Xie, Y.; Jing, X. A Resource-Efficient Cross-Domain Sensing Method for Device-Free Gesture Recognition With Federated Transfer Learning. *Trans. Green Commun. Netw.* **2023**, *7*, 393–400. [CrossRef]
12. Leo, M.D.; Minghini, M.; Kóňa, A.; Kotsev, A.; Dusart, J.; Lumnitz, S.; Ilie, C.M.; Kilsedar, C.E.; Tzotsos, A. Digital earth observation infrastructures and initiatives: A review framework based on open principles. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2023**, *48* , 33–40. [CrossRef]
13. Salcedo-Sanz, S.; Ghamisi, P.; Piles, M.; Werner, M.; Cuadra, L.; Moreno-Martínez, A.; Izquierdo-Verdiguier, E.; Muñoz-Marí, J.; Mosavi, A.; Camps-Valls, G. Machine learning information fusion in Earth observation: A comprehensive review of methods, applications and data sources. *Inf. Fusion* **2020**, *63*, 256–272. [CrossRef]
14. Blasch, E.; Pham, T.; Chong, C.Y.; Koch, W.; Leung, H.; Braines, D.; Abdelzaher, T. Machine Learning/Artificial Intelligence for Sensor Data Fusion–Opportunities and Challenges. *IEEE Aerosp. Electron. Syst. Mag.* **2021**, *36*, 80–93. [CrossRef]
15. Liu, R.; Hua, M.; Guan, K.; Wang, X.; Zhang, L.; Mao, T.; Zhang, D.; Wu, Q.; Jamalipour, A. 6G Enabled Advanced Transportation Systems. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 10564–10580. [CrossRef]

16. Djihane, M.; Abdelkrim, H.; Mohamed, B. Exploring the potential of Non Terrestrial Networks in next generation Intelligent Transport Systems. In Proceedings of the 2024 8th International Conference on Image and Signal Processing and Their Applications (ISPA), Biskra, Algeria, 21–22 April 2024; pp. 1–7. [CrossRef]
17. Dai, C.; Zhu, T.; Xiang, S.; Xie, L.; Garg, S.; Hossain, M.S. PSFL: Personalized Split Federated Learning Framework for Distributed Model Training in Intelligent Transportation Systems. *IEEE Trans. Intell. Transp. Syst.* **2025**, 1–10. *early access*. [CrossRef]
18. Otoum, S.; Guizani, N.; Mouftah, H. On the Feasibility of Split Learning, Transfer Learning and Federated Learning for Preserving Security in ITS Systems. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 7462–7470. [CrossRef]
19. Wu, M.; Cheng, G.; Ye, D.; Kang, J.; Yu, R.; Wu, Y.; Pan, M. Federated Split Learning With Data and Label Privacy Preservation in Vehicular Networks. *IEEE Trans. Veh. Technol.* **2024**, *73*, 1223–1238. [CrossRef]
20. Naseh, D.; Shinde, S.S.; Tarchi, D. Enabling Intelligent Vehicular Networks Through Distributed Learning in the Non-Terrestrial Networks 6G Vision. In Proceedings of the European Wireless 2023 (EW 2023), Rome, Italy, 2–4 October 2023. [CrossRef]
21. Kosasi, T.; Sihombing, Z.A.L.; Husein, A.M. YOLO-Based Vehicle Detection: Literature Review. *J. Comput. Networks, Archit. High Perform. Comput.* **2024**, *6*, 1384–1389. [CrossRef]
22. Naseh, D.; Shinde, S.S.; Tarchi, D. Network Sliced Distributed Learning-as-a-Service for Internet of Vehicles Applications in 6G Non-Terrestrial Network Scenarios. *J. Sens. Actuator Netw.* **2024**, *13*, 14. [CrossRef]
23. Ridolfi, L.; Naseh, D.; Shinde, S.S.; Tarchi, D. Implementation and Evaluation of a Federated Learning Framework on Raspberry PI Platforms for IoT 6G Applications. *Future Internet* **2023**, *15*, 358. [CrossRef]
24. Qi, P.; Chiaro, D.; Guzzo, A.; Ianni, M.; Fortino, G.; Piccialli, F. Model aggregation techniques in federated learning: A comprehensive survey. *Future Gener. Comput. Syst.* **2024**, *150*, 272–293. [CrossRef]
25. Alsharif, M.H.; Kannadasan, R.; Wei, W.; Nisar, K.S.; Abdel-Aty, A.H. A contemporary survey of recent advances in federated learning: Taxonomies, applications, and challenges. *Internet Things* **2024**, *27*, 101251. [CrossRef]
26. Jin, Y.; Zhu, H.; Xu, J.; Chen, Y. *Federated Learning: Fundamentals and Advances*; Machine Learning: Foundations, Methodologies, and Applications; Springer: Singapore, 2023. [CrossRef]
27. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Vincent Poor, H. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1622–1658. [CrossRef]
28. Lin, Z.; Qu, G.; Chen, X.; Huang, K. Split Learning in 6G Edge Networks. *IEEE Wirel. Commun.* **2024**, *31*, 170–176. [CrossRef]
29. Wu, W.; Li, M.; Qu, K.; Zhou, C.; Shen, X.; Zhuang, W.; Li, X.; Shi, W. Split Learning Over Wireless Networks: Parallel Design and Resource Management. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 1051–1066. [CrossRef]
30. Vepakomma, P.; Raskar, R. Split Learning: A Resource Efficient Model and Data Parallel Approach for Distributed Deep Learning. In *Federated Learning: A Comprehensive Overview of Methods and Applications*; Ludwig, H., Baracaldo, N., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 439–451. [CrossRef]
31. Turina, V.; Zhang, Z.; Esposito, F.; Matta, I. Federated or Split? A Performance and Privacy Analysis of Hybrid Split and Federated Learning Architectures. In Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 5–10 September 2021; pp. 250–260. [CrossRef]
32. Liang, Y.; Kortoçi, P.; Zhou, P.; Lee, L.H.; Mehrabi, A.; Hui, P.; Tarkoma, S.; Crowcroft, J. Federated split GANs for collaborative training with heterogeneous devices. *Softw. Impacts* **2022**, *14*, 100436. [CrossRef]
33. Naseh, D.; Abdollahpour, M.; Tarchi, D. Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications. *Information* **2024**, *15*, 190. [CrossRef]
34. Zhao, Z.; Alzubaidi, L.; Zhang, J.; Duan, Y.; Gu, Y. A comparison review of transfer learning and self-supervised learning: Definitions, applications, advantages and limitations. *Expert Syst. Appl.* **2024**, *242*, 122807. [CrossRef]
35. Naseh, D.; Shinde, S.S.; Tarchi, D. Multi-Layer Distributed Learning for Intelligent Transportation Systems in 6G Aerial-Ground Integrated Networks. In Proceedings of the 2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Antwerp, Belgium, 3–6 June 2024; pp. 711–716. [CrossRef]
36. Naseh, D.; Shinde, S.S.; Tarchi, D. Distributed Learning Framework for Earth Observation on Multilayer Non-Terrestrial Networks. In Proceedings of the 2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), Stockholm, Sweden, 5–8 May 2024; pp. 1–2. [CrossRef]
37. Price, E. Federated Learning for Privacy-Preserving Edge Intelligence: A Scalable Systems Perspective. *J. Comput. Sci. Softw. Appl.* **2025**, *5* . [CrossRef]
38. Jia, Y.; Liu, B.; Zhang, X.; Dai, F.; Khan, A.; Qi, L.; Dou, W. Model Pruning-enabled Federated Split Learning for Resource-constrained Devices in Artificial Intelligence Empowered Edge Computing Environment. *ACM Trans. Sens. Netw.* **2024**. [CrossRef]
39. Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; He, Q. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* **2021**, *109*, 43–76. [CrossRef]
40. Niu, Z.; Shen, X.S.; Zhang, Q.; Tang, Y. Space-air-ground integrated vehicular network for connected and automated vehicles: Challenges and solutions. *Intell. Converg. Netw.* **2020**, *1*, 142–169. [CrossRef]

41. Naseh, D.; Shinde, S.S.; Tarchi, D.; DeCola, T. Distributed Intelligent Framework for Remote Area Observation on Multilayer Non-Terrestrial Networks. In Proceedings of the 2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Madrid, Spain, 8–11 July 2024; pp. 1–6. [CrossRef]
42. Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; Kwon, Y.; Michael, K.; Fang, J.; Yifu, Z.; Wong, C.; Montes, D.; et al. ultralytics/yolov5: V7.0—YOLOv5 SOTA Realtime Instance Segmentation (v7.0). *Zenodo* **2022**. [CrossRef]
43. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3213–3223. [CrossRef]
44. Khanam, R.; Hussain, M. What is YOLOv5: A deep look into the internal features of the popular object detector. *arXiv* **2024**, arXiv:2407.20892. [CrossRef]
45. Desislavov, R.; Martínez-Plumed, F.; Hernández-Orallo, J. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustain. Comput. Inform. Syst.* **2023**, *38*, 100857. [CrossRef]