Split Federated Learning-Driven Resource-Efficient MEC Framework for UAV-based Networks

Houda Hafi, Bouziane Brik, *Senior Member, IEEE*, Zakaria Abou El Houda, Adlen Ksentini, *Senior Member, IEEE*

***Abstract*—Distributed collaborative machine learning techniques enable the training of intelligent models while preserving user data privacy. However, in reality, training a large-scale and intricate model on resource-constrained devices such as Unmanned Aerial Vehicles (UAVs) is unfeasible. In this context, lightweight and resource-efficient deep learning techniques are required. This work first suggests a new resource-aware distributed framework, SFMec, in the context of a UAV power consumption scenario. The framework is evaluated and compared with other distributed frameworks, including FedMec, a federated learning-based approach, to assess its performance across different system architectures and resource management strategies. The results obtained demonstrate that SFMec has the potential to conserve more than 50% of the storage space occupied by FedMec, making it more attractive for devices with limited resources. Then, a novel architecture, denoted as SFMecLite, is introduced to minimize the interactions between SFMec entities. Furthermore, an enhanced version of SFMecLite is also presented that greatly outperforms FedMec and reduces the computational and communication costs in SFMec without compromising learning performance.**

***Index Terms*—Unmanned Aerial Vehicle (UAV), Resource-constrained Devices, Resource Optimization, Split Federated Learning (SFL), Mobile Edge Computing (MEC).**

I. Introduction

DUE to their ease and rapid deployment, low acquisition and maintenance costs, high flexibility, and mobility, Unmanned Aerial Vehicles (UAVs) are becoming increasingly indispensable for a wide range of socio-economic applications. For example, in the realm of environmental issues, sensor-equipped UAVs can be used in air pollution and emissions tracking, guiding industries to environmental sustainability [1]. In addition, UAVs equipped with sensors, cameras, and RFID can be used for smart warehouse management. Advanced analytics of the collected data enables predictive restocking, guaranteeing product availability for consumers [2]. Moreover, incorporating UAVs into logistics and supply chain management will significantly enhance the efficiency, speed, cost, and reliability of supply chain operations, aligning with the evolving expectations of consumers [3]. Furthermore, they can be utilized for predictive maintenance in smart factories by monitoring various machine parameters, such as temperature, voltage, and current. Analysis of collected data, leveraging advanced Machine Learning (ML) and Deep Learning (DL) algorithms, enables the identification of defects and anomalies in real time. This approach enhances production efficiency, prevents breakdowns, and ensures both material resilience and industrial operations sustainability. Additionally, UAVs can have a great impact on commercial activities by gathering location data to analyze customer behavior, enhance marketing, and improve satisfaction [4]. UAVs can also serve as a wireless infrastructure, facilitating seamless connectivity for data transfer between IoT devices and the control center [5]. Therefore, the integration of AI and UAV technologies in various industries will facilitate data-driven decision-making, personalized consumer experiences, sustainability practices, and efficient operations in several sectors.

However, despite the promising prospects, this transformative synergy also brings forth a set of challenges that must be addressed to fully leverage its potential. One of the primary challenges involves determining the appropriate methodology to train ML/DL models on resource-limited and hardware-constrained UAVs. Traditional training approaches, which involve transmitting UAV data to a central server for further analysis, require a large amount of communication bandwidth. Additionally, relying on a single entity can lead to system-wide failure, known as the single-point-of-failure. Another major concern is related to the security and privacy of sensitive data captured by UAVs (military and industrial secrets, geolocation data, etc.) [6].

Federated learning is a prominent distributed algorithm designed to precisely address these issues. In FL, training data are not managed centrally, but instead held by data owners who participate in the FL process [7]. Recent studies [8], indicate that advancements in computational power of onboard engines such as enhanced CPUs, GPUs, and digital signal processors, enable UAVs to locally train ML models on sub-datasets and share only model parameters with base stations instead of the entire raw data. In the literature, several studies have already applied federated learning to UAVs [9], [10], [11]. Although this technique reduces bandwidth consumption and enhances data privacy, the efficiency of such training remains context-dependent and sensitive to several factors, including hardware specifications, as well as the model size and complexity [12]. Therefore, the successful deployment of FL in UAVs cannot be generalized and requires case-specific evaluation. For example,

Houda Hafi is with the Faculty of New Information and Communication Technologies, Abdelhamid Mehri University, Constantine, Algeria, e-mail: (houda.hafi@univ-constantine2.dz).

Bouziane Brik is with the Computer Science Department, College of Computing and Informatics, Sharjah University, Sharjah, UAE, e-mail: (bbrik@sharjah.ac.ae).

Zakaria Abou El Houda is with Institut National de la Recherche Scientifique Centre Énergie Matériaux et Télécommunications (INRS-EMT), Varennes, QC J3X 1S2, Canada, Canada, e-mail: (zakaria.abouelhouda@inrs.ca).

Adlen Ksentini is with EURECOM, Sophia-Antipolis, France, e-mail: (adlen.ksentini@eurecom.fr).

training a full FL model with a large number of trainable parameters on UAVs with limited onboard resources rapidly depletes their energy, leading to premature termination of the training process.

To effectively optimize drone resource utilization, alternative lightweight deep learning algorithms are desideratum. Inspired by this, our study leverages a more sophisticated distributed technique that has recently been proposed, namely split federated learning [13], by dividing the neural network among the different clients/learners. Each learner trains exclusively a section of the entire model, while the remaining section would be trained by more resourceful devices (e.g., edge servers). This collaboration during the training process would extend the lifespan of drone batteries and reduce the task completion time.

The contribution of this work is three-fold:

1) **Resource-optimized framework for MEC-enhanced UAV networks**: First, we propose a collaborative resource-aware framework, called SFMec, that optimizes resource utilization for resource-constrained UAV computing devices while maintaining learning performance.
2) **An enhanced architecture for seamless connectivity and performance**: Building on the analysis conducted in the first part of this work, we identified that split federated learning involves frequent communication between clients and servers (main and fed servers). Therefore, a more efficient data exchange mechanism, optimizing the bidirectional communication between the involved entities, is provided. Thanks to the proposed technique, known as SFMecLite, both the learning performance and training efficiency of SFMec are significantly enhanced.
3) **A high-performance framework for optimized client-server communication**: Additionally, we are considering the reduction of split federated learning communication costs and computation time, all while ensuring there is no compromise in learning performance. To achieve that, a biologically inspired method using the Particle Swarm Optimization (PSO) algorithm is applied. Numerical results demonstrate that the new scheme surpasses FedMec, SFMec and SFMecLite in terms of all evaluated metrics.

The details about the organization of this work are provided as follows. We first present a review of the literature in Section II. Section III outlines our research methodology and our designed approach. In Section IV, the implementation settings and performance evaluation are discussed. SectionV introduces enhancements in communication efficiency for interactions between clients and servers in split federated learning. Section VI concludes and summarizes the key ideas of this study.

II. Literature Review

The emergence of federated learning has prompted researchers to delve into a comprehensive examination of its application in the air [14], [15]. In the literature, several federated learning solutions have been proposed to address various issues related to UAV-enabled networks. For example, in October 2021, the authors of [9] presented an FL approach based on CNN-LSTM for an accurate and timely prediction of the air quality index using a fleet of UAVs. In the same year, the authors of [11] proposed a federated learning-based drone authentication, where drones locally train the authentication model using their data in a federated manner. In [16], the authors introduced a secure federated learning framework for UAV-assisted mobile crowdsensing (SFAC). By integrating blockchain, local differential privacy, and a reinforcement learning-based incentive mechanism, SFAC enables decentralized and verifiable model sharing, protects UAV updates, preserves global accuracy, and motivates high-quality model contributions under uncertain networks.

The researchers in [10], designed a novel fair and robust federated learning (FRFL) technique for UAV-assisted crowdsensing. The contributions of this work are multifold. (1) the authors integrate 5G edge networks to provide efficient FL services with high data rate and low latency for UAVs. (2) Under knowledge asymmetry, a contract-theory-based incentive mechanism is used to guarantee truthful and equitable participation of UAVs. (3) The framework incorporates Byzantine-resilient aggregation and equitable profit distribution according to the contribution of UAVs to improve robustness. (4) To discourage free-riders and enhance trust, a reputation system is also included.

In [17], a cooperative tracking framework, FedTrack, using adaptive FL to improve tracking efficiency while reducing transmission costs and time, is proposed. FedTrack incorporates a dual reputation mechanism and an adaptive client selection algorithm for optimized participation, and a capability-based node selection strategy for efficiency aggregation. Experimental results show competitive accuracy with lower resource demands compared to existing methods. The researchers in [19], design a hybrid federated and centralized learning (HFCL) framework for wireless traffic prediction in UAV-aided multi-access edge computing (MEC) servers. The approach balances latency and energy consumption while ensuring compliance with 3GPP 5G standards. By formulating an optimization problem and incorporating a Deep Reinforcement Learning (DRL)-based solution, the framework achieves lower costs and better efficiency than benchmarks.

While studies [9], [10], [11], [17], [19] provide several advantages over centralized solutions, they fall short in addressing the restricted computational power and battery capacities of UAVs. The main challenge of these implementations is to train the entire model in a limited, low-resource environment. That is why other works try to optimize the federated training process in UAV-assisted networks.

The research conducted in [20] presents a DRL-based algorithm to maximize long-term FL performance. To reduce the complexity of the problem, the authors introduce the use of the Lyapunov optimization technique. This approach helps transform the long-term energy constraint into a deterministic problem. Afterwards, the optimization problem is reformulated as a Markov decision process (MDP). The MDP is solved with DRL, where the agent learns the optimal UAV placement and resource allocation, ensuring sustainable and energy-efficient UAV-assisted networks. Similarly, the authors in [21] explore the optimization of the federated edge learning process in UAV-enabled IoT for B5G/6G networks. The presented frame-

TABLE I: Comparison of distributed training approaches on UAVs in the literature

Reference	Approach	Solve Data Privacy	Solve Model Privacy	Full Model Training	Generalization via Aggregation
[9], [10], [11], [17], [16], [18]	Standard FL	Yes	No	Yes	Yes
[19]	Hybrid (Centralized + FL)	Yes	No	Yes	Yes
[20], [21], [22], [23], [24], [25]	Optimized FL	Yes	No	Yes	Yes
[26], [27]	Hybrid FL-SL	Yes	No	Yes	Yes
[28], [29], [30]	SL	Yes	Yes	No	No
Our Work	SFL + Optimized SFL	Yes	Yes	No	Yes

work allows devices to adjust their operating CPU frequency to prolong the UAVs battery life and avoid withdrawing from training untimely, through managing resource allocation in changing environments. To solve the optimization problem, the authors employ the Deep Deterministic Policy Gradient (DDPG) strategy. In a parallel way, authors in [22], developed a joint training and resource allocation method to minimize the energy consumption for the multi-UAV-assisted FL scheme. The proposed solution uses an optimization algorithm that addresses the minimization of overall training energy consumption of UAV swarms as well as the minimization of maximum energy consumption of UAV swarms. Likewise, the work in [25], presented an energy-efficient framework for federated learning called E2FL. By leveraging UAVs equipped with edge computing and wireless power transfer, the system acts as both an aerial server and energy source. E2FL jointly optimizes UAV placement, power control, bandwidth allocation, and computing resources to minimize total energy consumption. Simulation results demonstrate the effectiveness of joint optimization for sustainable FL.

The paper [23] addresses the high energy consumption in FL for wireless devices by using MEC-enabled UAVs for data collection and training. To handle UAV location uncertainties that affect data transmission and network performance, the authors model UAV deviations as Gaussian distributions and introduce probabilistic constraints on data offloading. Using Bernstein-Type Inequality (BTI), they convert these constraints into deterministic ones, making the optimization problem solvable. They then apply Block Coordinate Descent (BCD) to optimize UAV energy consumption while ensuring robust FL training. The work done in [24] suggests a two-tier hierarchical FL scheme assisted by a UAV swarm to address connectivity challenges in wireless FL. UAVs act as data collectors and relays. To optimize FL convergence and UAV data transmission, the authors formulate a joint optimization problem involving UAV-client matching, time allocation, and local training iterations. They propose an efficient solution combining a subgradient-based method with a cross entropy-based genetic algorithm. Numerical results demonstrate the effectiveness of this approach in improving FL performance and communication efficiency. In [18], the authors designed a decentralized, energy-efficient FL framework for UAV swarms using Spiking Neural Networks (SNNs) to reduce redundant computations and energy consumption during local training. In addition, to handle UAV mobility, the authors introduced an intelligent leader selection scheme (based on the Bayes theorem) to accelerate the aggregation of model parameters and reduce communication time.

Recently, a new idea was proposed in [26], where the authors suggest a hybrid approach, leveraging both federated learning for local training by a portion of clients and split learning for collaborative training with the base station. A related idea was implemented in [27], where UAVs are divided into two groups, namely FL-mode and SL-mode, based on their computational capacity and dataset size. FL-mode UAVs perform local model training on their entire datasets and transmit model parameters to the server for FedAvg aggregation. SL-mode UAVs train only a partial sub-model locally up to a cut layer and send the resulting activations to the same server, which completes the training and returns cut-layer gradients for backpropagation.

To minimize energy consumption in multitasking split inference in UAV networks, the authors in [28], propose a two-timescale optimization approach called OPETRL. The technique combines Tiny Reinforcement Learning (TRL) for transmission mode selection, and Optimization Programming (OP) for transmit power adjustment. The simulation results demonstrated that OPETRL can effectively reduce computational complexity while ensuring energy efficiency and higher task success rates during aerial AI operations. In [29], the authors suggest a split learning-based technique for image classification in a multi-UAV system to support applications such as area exploration and object detection. The study aims to demonstrate that SL can effectively offload computation from UAVs to a base station, reducing local processing time and improving training performance, especially under non-IID data conditions, compared to FL and centralized learning. The work in [30], presented a novel approach called Stitch-able Split Learning (SSL). The technique combine split learning with Stitch-able Neural Networks (SN-NET) to overcome challenges in multi-UAV environments, such as device instability, model heterogeneity, privacy concerns, and limited computational resources. The simulation results showed that SSL achieves reduced learning time, better accuracy, and adaptability against centralized learning, FL, traditional SL, and SplitFed V1 (SFLV1). TABLE I compares different distributed ML/DL training approaches on UAVs in the literature.

From the above, it is evident that most existing works leverage federated learning for distributed model training. However, training the complete FL model directly on UAVs poses a significant risk of training interruptions, due to their limited battery capacity, which can severely impact model convergence and overall system reliability. While some studies attempt to optimize the FL process, they still require UAVs to process the full model, which remains impractical. This highlights the necessity of an alternative approach where UAVs devices train only a lightweight model rather than the entire FL model. To fill this gap, through this work, we propose a lightweight deep-learning approach for UAVs that accommodates their storage and processing capabilities while maintaining favorable learning outcomes.

III. Proposed Framework

In this section, we first present an overview of the proposed framework. Then, we provide additional details on each step.

A. Overview

SFMec is a distributed collaborative framework designed to predict instantaneous power consumption in MEC-enabled UAV networks. By leveraging split federated learning, an emerging decentralized privacy-protection training technology, SFMec enables multiple clients, each with its own sensor data, to collaboratively train a global model. The framework incorporates Long Short-Term Memory (LSTM) for sequential data modeling. As illustrated in Fig. 1, the proposed framework is divided into two sides, namely: the front end, and the back end. This architectural division not only preserves data privacy but also ensures model privacy, as neither the front end nor the back end has complete access to the full model [31]. In SFMec, three types of entities are involved: a main server $\mathcal{M}$, a fed server $\mathcal{F}$, and distributed clients $\mathcal{K}$ (UAVs). Each client has a dataset $D_i = \{X_i, Y_i\}$, $i \in \{1, 2, ...K\}$. The global model is divided into two subsections, the first sub-model is maintained on F, while the second sub-model is hosted on the main server M.

B. Components and Operations

As mentioned earlier, the proposed framework is divided into two principal parts: a front end and a back end. The following presents a detailed description of each part.

- **Front end**: It represents the module operating on the client-side, tasked with carrying out lightweight computational operations. It consists of two layers, specifically the input layer and the first LSTM layer. At first, each client (UAV) downloads the initial weights from the fed server. Then, it performs the forward propagation by training only a segment of the global model (up to the cut layer), using its own input data and the associated client sub-model. Afterwards, each client (UAV) sends the intermediate outputs and target values to the main server. When the client receives the gradients of the cut layer (after the main server assignment completes successfully), it back-propagates the received gradients and updates its own weights. The updated weights of all the clients are then passed to the fed server for aggregation and the same training process will continue.
- **Back end**: At the edge level of the framework, this module is responsible for completing the training tasks on the server-side. In the server part, the network structure consists of the remaining layers, namely: two LSTM layers, the dense layer, and the output layer. After obtaining the output data from a client, the server resumes the feedforward pass and calculates the loss function L. Moreover, the gradients for both the server and client are computed, respectively. Subsequently, the gradients are sent back to the client and the forecasting precision can be measured. Algorithm 1 outlines the methodological steps constituting the framework in a systematic manner.

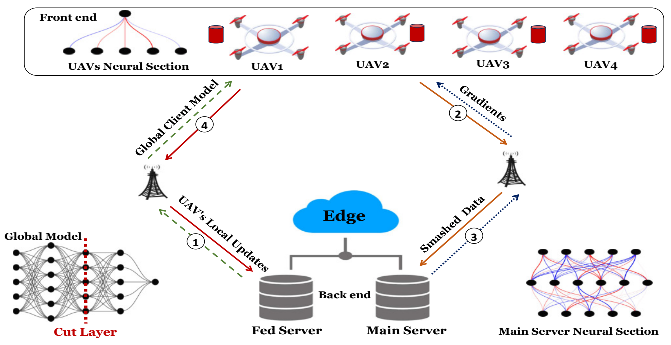


Fig. 1: Split Federated Learning MEC-enabled Framework for UAV Networks

Algorithm 1: SFMec Algorithm

```
/* UAV Side */ Input: Training data $(X, Y)$
for each round do
    for each UAV $U_i$ in parallel do
        for each Batch Size do
            $A_b = ForwardPass(X_b, W_{u_i})$;
            $MainServerProcess(A_b, Y_b)$;
            procedure UAVBackProp($G_b$);
            Update Weights $W'_{u_i}$;
        end
        $W^g_u \leftarrow FedServerProcess(W_{u_i})$;
    end
end

/* Main Server Side */ ;
Initialize Server Side Sub-model Weights $W_s$;

procedure MainServerProcess($A_b, Y_b$);
(1) $ForwardPass(A_b, W_s)$;
(2) Compute Cost Function $J(\theta)$;
(3) Compute Gradients $G_b = \nabla J(\theta)$;
(4) Update Weights $W'_s$;
(5) UAVBackProp($G_b$);

/* Fed Server Side */;
Initialize UAV Side Sub-model Weights $W^g_u$;
Forward $W^g_u$ to all participating UAVs;

function FedServerProcess($W_c$);
(1) Wait until receiving all UAVs' updates;
(2) Aggregate($W_u$);
(3) Update UAV Side Sub-model Weights $W^g_u$;
(4) Return $W^g_u$ to all participating UAVs;
```

TABLE II: Implementation Parameters

Parameters	Values
Seed, Window Step	42, 10
Data Split	80% Training / 20% Test
Activation Function	Tanh (hidden layers), Linear (output)
Loss Function / Optimizer	Huber (delta = 1) / Adam
Learning Rate / Batch Size	0.001 / 64
Normalization Alg / Rounds	MinMax Scalar / 200
FedMec	Client side: 5 layers, Server side: /
SplitMec and SFMec	Client side: 2 layers, Server side: 3 layers

IV. Implementation and Evaluation

In this section, we assess the performance of SFMec by comparing it against a federated learning-enabled (FedMec) and split learning-enabled (SplitMec) frameworks for prediction power consumption in UAV networks. We first discuss the dataset used and the experimental settings. Then we present the experimental findings and conduct a performance analysis.

A. Dataset

We chose a recent real-world dataset collected under non-i.i.d conditions using a hexacopter drone [32] that has six 18-inch propellers, weighs $6\,\mathrm{kg}$, and a maximum takeoff mass of $13\,\mathrm{kg}$. The goal is to empirically measure the power consumption of electric UAVs. The experimenters collect data during automatic and manual missions, without any payload weight and with additional payload weights of $2\,\mathrm{kg}$, $4\,\mathrm{kg}$, and $6\,\mathrm{kg}$. Measurements were taken for the hourglass-shaped trajectory with a velocity equal to $4\,\mathrm{m/s}$. The dataset contains 27 variables (altitude, velocities, orientation, total mass, etc.). To carry out our experimental study, we selected the most relevant features.

B. Parameter Settings

Our framework operates with 5 clients over 200 rounds. 20% of the dataset is separated as the testing data for all the clients while the rest 80% of the dataset is equitably divided among the five clients. The model is implemented in Python 3.10.12 with TensorFlow 2.12.0. The Hyperband tuning algorithm [33] is used to determine the best hyper-parameters of the neural network. The experiments were conducted in Google Colab using a supervised LSTM regression model with four layers: input (features), 3 hidden layers, and output (energy prediction). The first two hidden layers consist of 128 neurons, whereas the third hidden layer contains 32 neurons. Except for the last layer of the model that uses a linear activation function, the other layers are followed by the Tanh function. The model employs the Huber loss function and Adam optimizer. For a fair comparison, all previously mentioned training parameters are consistently maintained for FedMec and SplitMec without any modifications. TABLE II outlines the parametric settings used in the evaluation process.

C. Performance Metrics

1) Learning Performance:

- Loss: It is a standard metric that quantifies the error between model prediction and the actual ground truth values in the training data. In our case, we selected the Huber as a loss function that can be derived using the equation 1.

$$\mathrm{L}(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \le \delta \\ \delta(|y-\hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{1}$$

- Mean Absolute Error (MAE): is the average of the absolute differences between the predicted and actual values.

2) Computing Performance:

- Model storage complexity: It denotes the client's necessary memory to store its related segment for each model, calculated as the total number of client-side parameters multiplied by the size of each parameter.
- Memory usage (during training): denotes the amount of memory consumed throughout the training process.
- Communication overhead: It represents the data size sent between clients and servers (weights, activation, gradients, etc.).
- Total training time: It represents the time necessary to complete the training process between the clients and the server.
- Bandwidth: It refers to the quantity of data conveyed between the clients and the server within a specific amount of time.

D. Numerical Results and Discussion

From Fig. 2, we can observe that initially SplitMec records a low level of loss compared with FedMec and SFMec. This is mainly due to the synchronous training mode of SplitMec, where the current participating client does not start the training with randomly initialized weights but rather downloads the model's pre-existing weights from its previous neighbor. This knowledge transfer among neighbors accelerates the learning process. After 40 rounds, the curve appears in a decreasing trend for the three models. Thereafter, a minor decrease in FedMec loss is observed, whereas SplitMec and SFMec exhibit a high degree of similarity throughout the 200 rounds.

Fig. 3 demonstrates the mean absolute error of FedMec, SplitMec and SFMec for the dataset used. As can be seen, the MAE values of FedMec and SFMec remain consistent and illustrate a close correlation as the number of rounds increases, indicating stability in the results. However, the MAE of the SplitMec model is slightly higher than FedMec and SFMec with small fluctuations. This discrepancy can be attributed to the fact that FedMec and SFMec share the same aggregation rule, which plays a crucial role in smoothing the results, reducing fluctuations, and enhancing the model generalization. However, SplitMec does not use any aggregation strategy that could reduce the risk of variations in the results obtained.

Next, we proceed with a comparative analysis of the computational performance across the different models within the UAV system. The comparison is made in terms of model storage complexity, memory usage, average training time,

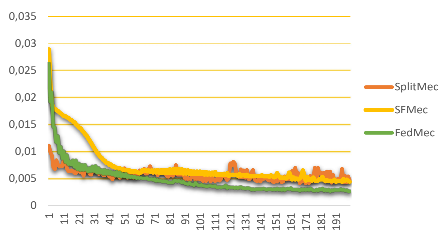


Fig. 2: Loss vs Number of rounds

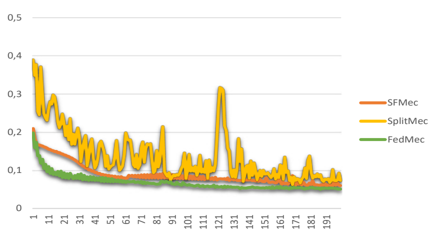


Fig. 3: MAE vs Number of rounds

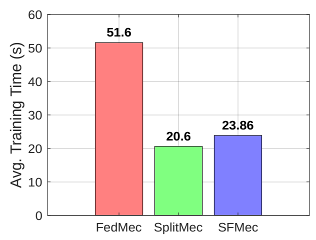


Fig. 4: Average Training Time

communication overhead and bandwidth. For the total training time (see Fig. 4), SplitMec and SFMec gave the best results, with SFMec coming second out of three and being slightly outperformed by SplitMec. In addition, we see that both SplitMec and SFMec offered a remarkable performance compared to FedMec in terms of model footprint (see Fig. 5). They could save more than half of the FedMec storage space, with only $0.27\,\mathrm{MB}$ compared to FedMec which requires $0.85\,\mathrm{MB}$. This is due to the reduction in the number of trainable parameters to only 7168, which is more than three times less than the 223393 parameters in FedMec (see Fig. 6). It is also important to note that SFMec and SplitMec outperform FedMec in optimizing memory utilization during training. As can be seen in Fig. 7, SplitMec and SFMec take up only $68.6\,\mathrm{MB}$ and $81\,\mathrm{MB}$ of memory during training, while FedMec occupies $306.7\,\mathrm{MB}$ (the reported results pertain to one round). In terms of communication overhead and bandwidth, as highlighted in Figures 8 and 9, it is evident that FedMec outperforms both SplitMec and SFMec. This is because SFMec (as well as SplitMec) involves additional data to manage communication between learners and the server, including smashed data and gradients. Consequently, this leads to an increased overhead and higher bandwidth consumption on the network. In contrast, in the FedMec approach, only the weights are transmitted between the UAVs and the server.

Based on these findings, we can deduce that, except for the communication overhead that directly impacts the consumed bandwidth, SFMec framework has proved its superiority in terms of computing performance compared to FedMec, and it could cope with the characteristics of aerial vehicles. Furthermore, as seen above, SFMec offers better results than SplitMec in terms of learning performance. All in all, we could say that SFMec achieves a good trade-off between learning performance and computing performance.

This first part of the study constituted an essential step in identifying the key limitations of split federated learning in resource-constrained UAV environments. Specifically, the high number of request-response exchanges between UAVs and servers, leading to significant communication overhead. The next section introduces the SFMecLite framework, an enhancement of SFMec that addresses this issue by optimizing the communication process.

V. Communication Optimization for Clients-Servers Interactions

A. SFMecLite: Novel Architecture for Optimal Connectivity and Performance

To model the system, suppose that $\mathcal{M}$ is the SFL model to train, p denotes the number of training rounds, n is the number of learners (drones), m represents the number of samples in the dataset, k indicates the number of samples for each batch. NR_{main} and NR_{fed} are the number of requests between the clients and the main/fed server respectively. The task is to train $\mathcal{M}$ for p rounds among n drones using a dataset of m samples.

In the traditional architecture, the total number of requests between the drones and the main server is typically

$$NR_{main} = 2 * i * p * m/k, i \in \{1, n\} \tag{2}$$

If we consider that UAVs forward their local weights towards the fed server after each round, then, the total number of requests between the drones and the fed server is

$$NR_{fed} = p * (i + 1)\,, i \in \{1, n\} \tag{3}$$

To minimize clients-servers communication, consider the proposed architecture design illustrated in Fig. 10. In the proposed system, existing base stations in the network are leveraged as central relay points for communication between drones and the main/fed server. Each participating node submits its individual activations/weights, identified by a client-specific ID, to the base station. The latter aggregates the received data into a batch of smashed data or weights, which are then transmitted to the main/fed server. Upon receiving the data batch, the main server proceeds with the backpropagation phase for each learner (drone). Subsequently, the main server forwards the vector of gradients back to the relay point (BS). The BS acts as an intermediary, efficiently propagating the computed gradients to the respective learners involved in the learning process. In our design, we have opted for a fixed base station for several reasons. First, a fixed base station provides high transmission power, ensuring stable and reliable communication between drones and servers. Furthermore, it has more computational resources and storage capacity. In addition, it is more scalable, supporting a higher number of UAVs and larger data flows. We recognize that there are scenarios, such as in remote areas, where the deployment of a fixed base station is impossible. In this situation, a temporary mobile base station, such as one of the UAVs, can be deployed. However, selecting the appropriate drone

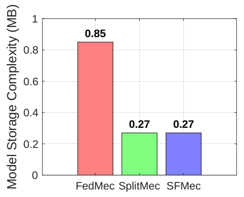


Fig. 5: Storage Complexity

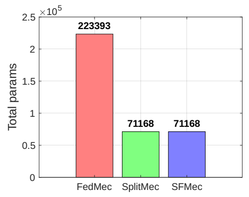


Fig. 6: Total Parameters

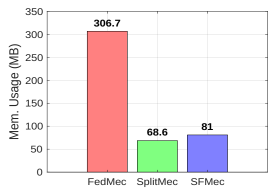


Fig. 7: Training Memory Usage

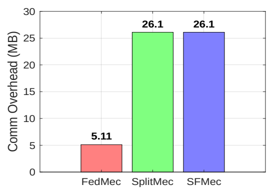


Fig. 8: Communication Overhead

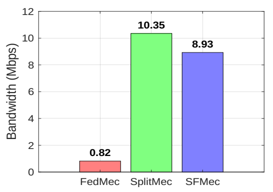


Fig. 9: Bandwidth

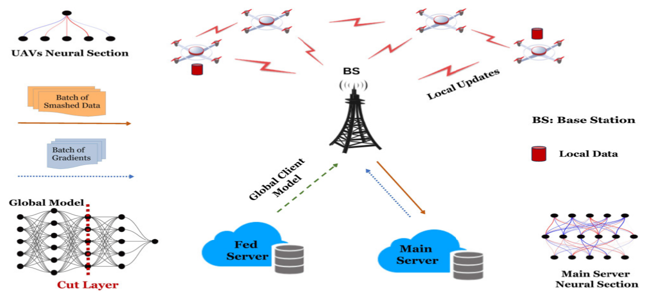


Fig. 10: New design for reduced client-server exchanges

to serve as a BS involves some challenges that must be carefully managed. These include the high mobility of UAVs that causes communication interruptions, frequent handover, increased overhead, and latency. Furthermore, the limited battery life and computational resources of UAVs restrict their operational capacities and time. It is important to note that while these challenges are significant, they are beyond the scope of this research and require a separate study to fully explore and address the issues related to mobile base station deployment.

With the proposed design, the total requests between the base station and the main/fed server will be minimized to

$$NR_{\text{main}} = 2*p*m/k \quad \text{and} \quad NR_{\text{fed}} = 2*p \tag{4}$$

Here, it is important to mention that SFMecLite serves to mitigate the communication costs in terms of additional data (headers and control information) between the clients and the server, but does not directly contribute to the reduction of the SFMec data (smashed data and gradients). For instance, if we consider that the original SFMec payload is sent by the standard TCP protocol that has a minimum header size of 20 bytes, each client that establishes a connection with the server will incur an overhead associated with the TCP header size and the three-way handshake process. Forwarding a batch of data through a single connection minimizes overhead by reducing connection establishment costs for each learner. Furthermore, this helps optimize resource utilization on both the client and server sides. The subsequent section addresses the minimization of the original data associated with split federated learning.

B. OptSFMec: Particle Swarm Approach for Balancing Multi-Objectives in Split Federated Learning

Traditional hyperparameter optimization algorithms involve iterative adjustment of model parameters to optimize (minimize or maximize) the cost function (accuracy, loss, etc.). However, in the context of distributed learning (e.g., split federated learning), where multiple entities are considered, additional metrics beyond the traditional ones, such as communication cost and computation time, become a significant consideration, and extended optimization is required. Our objective is to consider trade-offs between communication cost, computational time, and learning performance (loss). There is no known polynomial-time algorithm that guarantees an optimal solution for all these factors. Furthermore, the decision-parameter space is high-dimensional and non-convex, making it computationally challenging, even impractical, to explore exhaustively, especially when dealing with complex models. In the literature, metaheuristics have been successfully applied to address similar optimization problems in Deep Neural Networks (DNN) and federated learning [34] [35].

In this section, we determine an optimal set of model parameters using an evolutionary technique, namely particle swarm optimization (PSO) [36] and prove that it is an efficient way to acquire satisfactory results.

1) Overview of Particle Swarm Optimization (PSO): It is a population-based metaheuristic inspired by bird flocking behavior during food search. It is widely used to solve complex optimization problems, including constrained, multiobjective, multimodal and discrete problems. The algorithm starts with a population (aka swarm) of particles with random locations and velocities in the N-dimensional search space. Each particle i represents a potential solution. During each iteration, the population is updated based on two best values, namely: $pbest_i^t$ which represents the best position found by particle i so far, and $gbest^t$ which is the best position obtained by any particle in the swarm so far. The process continues until the iteration limit or a satisfactory solution is reached.

2) Problem Formulation: Let us denote the set of clients as $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, the main server as M, and the fed server as F. First, we formulate the terms that contribute to the overall value of the objective function $f(X)$.

- Communication cost $C(X)$: It is the total data size transmitted between the clients and the servers. Taking $A_{i,p,b}$ as the input matrix for batch b of client $d_i \in \mathcal{D}$ during round p, W is the weight matrix, z is the bias vector, σ represents the activation function, ∇ refers to the gradient operator and θ denotes the model parameters. The total data transmitted of an SFL training is represented by

$$C(X) = \sum_{i\in\mathcal{N}} \sum_{p\in\mathcal{N}} \sum_{b\in\mathcal{N}} (\sigma(A_{i,p,b} \cdot W + z) + \nabla_\theta L_{i,p,b}(\theta)) \tag{5}$$

In practice, the real-world communication cost is measured using network monitoring for each client in each round. For clarity, if we assume that $S^{\text{c}\to\text{s}}_{c,r}$ is the smashed data size sent from client c to the main server in round r. $G^{\text{s}\to\text{c}}_{c,r}$ is the gradients size from the main server to the client c in round r. $W^{\text{c}\to\text{f}}_{c,r}$ is the weights size sent from the client c to the fed server in round r and F_r is the size of the new submodel broadcasted from the fed server to all clients. Formally, the communication cost can be expressed as

$$\text{C(X)} = \sum_{c\in\mathcal{C}} \sum_{r=1}^{R} \left(S^{\text{c}\to\text{s}}_{c,r} + G^{\text{s}\to\text{c}}_{c,r} + W^{\text{c}\to\text{f}}_{c,r}\right) + \sum_{r=1}^{R} F_r$$

- Computation time $T(X)$: It quantifies the duration that a split federated learning system invests in model training. If we consider that $CFPTj$ is the time complexity for the forwarding pass during round j, and $SFPTj$ is the time complexity of the main server for the same. Similarly, $CBPTj$ and $SBPTj$ represent the client and main server time complexities for backward pass during round j. $T^j_{F,regen}$ is the time required for the fed server to regenerate the global client sub model in round j. Then, the computation time of an SFL model training can be formulated as

$$T(X) = \sum_{j=1}^{p} [(\mathcal{CFPT}_j + \mathcal{SFPT}_j + \mathcal{SBPT}_j + \mathcal{CBPT}_j)] + \sum_{j=1}^{p} \text{T}^j_{F,regen} \tag{6}$$

For clarity and without loss of generality, in what follows, we provide further details on how each term of equation 6 can be computed.

In equation 7, we show that the time complexity for performing the forward pass depends on the computational power of the client, the batch size, the architecture of the client-side neural network, and the network bandwidth. Therefore, $CFPTj$ can be expressed as

$$\begin{aligned} CFPT_j &= f(S_{b,i}, \mathcal{O}(NN_i), P_i) + T_{\text{trans},i,j,b} \\ &= \sum_{i=1}^{n}\sum_{b=1}^{k} \frac{S_{b,i}\cdot\mathcal{O}(NN_i)}{P_i} + \frac{S_{b,i}}{BW} \end{aligned} \tag{7}$$

Where: $S_{b,i}$ is the batch size of client $d_i \in \mathcal{D}$. $\mathcal{O}(NN_\text{i})$ is the computational complexity of the client side neural network. P_i is the processing rate of the client d_i (UAV). $T_{\text{trans},i,j,b}$ denotes the transmission time of batch b between client d_i and the main server $\mathcal{M}$ during round j and BW is the network bandwidth.

Similarly, equation 8 describes that the increase/decrease in the time required for the main server to execute the forward pass is also influenced by the size of the smashed data, the architecture of the server-side neural network, and the computational capacities of the server. Therefore, $SFPT_j$ can be represented as

$$SFPT_j = f(S_{a,i}, \mathcal{O}(NN_M), P_M) = \sum_{i=1}^{n}\sum_{a=1}^{k} \frac{S_{a,i}\cdot\mathcal{O}(NN_M)}{P_M} \tag{8}$$

Where: $S_{a,i}$ is the activation size sent by client $d_i \in \mathcal{D}$, $\mathcal{O}(NN_\text{M})$ is the computational complexity of the server side neural network for processing the smashed data and P_M is the processing rate of the main server $\mathcal{M}$.

The total time required by the main server to calculate the gradients for itself and its associated clients, represented by $SBPT_j$, consists of the time needed to compute both gradients (T^{server}_{grad} and T^{client}_{grad} respectively), update the server weights (T^{server}_{update}), and forward the client's gradients to the respective client (T^{client}_{send}). This depends on the number of parameters of the server side sub-model, denoted as n_s, and the client side sub-model, denoted as n_c, as well as the processing rate of the main server P_M. Therefore, it can be formulated as

$$\begin{aligned} SBPT_j &= T^{server}_{grad} + T^{client}_{grad} + T^{server}_{update} + T^{client}_{send} \\ &= \sum_{i=1}^{n}\sum_{b=1}^{k} \Bigg(\frac{n_s\cdot\mathcal{O}(\text{grad}_{\textit{server, b}})}{P_M} + \frac{n_c\cdot\mathcal{O}(\text{grad}_{\textit{client, i, b}})}{P_M} \\ &\quad + \frac{n_s\cdot\mathcal{O}(\text{update}_{\textit{server, b}})}{P_M} + \frac{S^{client,i}_{grad}}{BW} \Bigg) \end{aligned} \tag{9}$$

If we assume that $\mathcal{O}(\text{grad}_{\text{server, b}}) = \mathcal{O}(\text{grad}_{\text{client, i, b}}) = \mathcal{O}(\text{update}_{\text{server, b}}) = \mathcal{O}(\text{grad}_{\textit{i, b}})$, then equation 9 will be simplified to

$$SBPT_j = \sum_{i=1}^{n}\sum_{b=1}^{k} \left(\frac{(2n_s + n_c)\cdot\mathcal{O}(\text{grad}_{\textit{i, b}})}{P_M} + \frac{S^{client,i}_{grad}}{BW} \right)$$

Where $S^{client,i}_{grad}$ represents the size of the gradients for client $d_i \in \mathcal{D}$.

Equation 10 defines the amount of time it takes the client to update its weights using the gradients received from the main server M

$$CBPT_j = \sum_{i=1}^{n}\sum_{b=1}^{k} \left(\frac{n_c\cdot\mathcal{O}(\text{update}_{\textit{client, i, b}})}{P_i} \right) \tag{10}$$

To express the time required by the fed server F to regenerate the client-side submodel, we need to consider the time to upload the weights of each client $d_i \in \mathcal{D}$ to F, represented by $T_{upload}(i, F)$. Furthermore, the time required by the fed server to aggregate all client weights, denoted as T_{agg} and the time required to broadcast the new global client submodel to all clients, $T_{broadcast}$. Equation 11 models the calculation of $T^j_{F,regen}$

$$\begin{aligned} T^j_{F,regen} &= \sum_{i=1}^{n} T_{upload}(i, F) + T_{agg} + T_{broadcast} \\ &= \sum_{i=1}^{n} \frac{S_{w,i}}{BW} + \frac{n\cdot\mathcal{O}(\text{update}_{weights})\cdot\overline{S}_{\text{weights}}}{P_F} \\ &\quad + \frac{S_{client-submod}}{BW} \end{aligned} \tag{11}$$

Where: $S_{w,i}$ is the size of the weights of client $d_i \in \mathcal{D}$, n is the number of participating clients, $\overline{S}_{\text{update}}$ represents the average size of the clients weights, P_F is the processing rate of the fed server, and $S_{client-submod}$ is the size of the new global client side submodel.

Note that $\mathcal{O}$ in all equations refers to the computational complexity of each of the mentioned terms.

- Loss $L(x)$: It quantifies the extent to which the model's predictions deviate from the true values (refer to equation 1).

Additionally, we introduce the following decision variables

$$\mathbf{X} = [N_\text{s}, C_\text{units}, M_{l1,\text{units}}, M_{l2,\text{units}}, \delta, \text{b}]$$

Where N_s represents the sequence length, C_units is the number of neurons on the client side, $M_{l1,\text{units}}$ and $M_{l2,\text{units}}$ are the numbers of neurons for the first and second layers on the server side, respectively, and b expresses the batch size. We consider a mono-objective minimization problem defined as follows

$$\begin{cases} \min \;\; f(X) = \alpha \cdot C(X) + \beta \cdot T(X) + \gamma \cdot L(X) & (12) \\ \text{S.t,} & \\ \alpha + \beta + \gamma = 1, \quad \alpha, \beta, \gamma \in \mathbb{R}^+ & (13) \\ C_\text{units} \leq \dfrac{\sum_{i=1}^{2} M_{li,units}}{2} & (14) \\ \forall d_i \in \mathcal{D} : C_{d_i}(X) \leq \tau \wedge \displaystyle\sum_{i=1}^{n} C_{d_i}(X) \leq \tau & (15) \\ \forall j \in p : T(X)_j \leq \rho & (16) \\ \textit{If (p = j), then} \quad L(X)_j \leq \varphi & (17) \end{cases}$$

In the objective function (12), our primary goal is to minimize the computation time, communication costs concerning the original data associated with SFL, and improvement in learning performance, measured by reduction in loss. These objectives are pursued while adhering to the following constraints. Constraint (13) ensures that the sum of the coefficients α, β and γ adds up to 1. Each coefficient indicates the relative importance of the associated term. In our experiments, we assign equal importance to all terms; therefore, $\alpha = \beta = \gamma = \frac{1}{3}$. The primary objective of constraint (14) is to ensure that clients always have fewer parameters than the server, respecting the limited resources of UAVs. The number of parameters for the client must be less than or equal to 50% of that of the server. Constraint (15) aims to minimize communication overhead while avoiding exceeding the threshold τ. The value of τ can be set by the network operator based on the link conditions or fixed to ∞ in the absence of restrictions. Constraint (16) guarantees that the round execution time must not exceed a predefined threshold ρ. Finally, constraint (17) ensures that in round j the loss reaches or goes below φ. We set τ, ρ on the basis of the best values obtained from our previous experiments. We specified τ to be 5.11, representing the minimum overhead achieved by FedMec (see Fig. 8). We configured ρ to 14.47, reflecting the minimum average training time obtained with SFMecLite. For φ, the optimization algorithm is expected to achieve a target value better than SFMecLite (equal or lower than 0.004 within 30 rounds).

TABLE III: Side Constraints

Parameters	N_s	b	C_units	$M_{l1,\text{units}}$	$M_{l2,\text{units}}$	δ
Initial Min	5	8	32	32	32	0.1
Initial Max	20	128	128	128	128	1
Selected Min	5	100	32	96	96	0.1
Selected Max	20	130	128	256	256	1

TABLE IV: PSO Parameters

Parameter	ω	ϕ_p	ϕ_g	minstep	minfunc	swarmsize
Value	0.5	0.5	0.5	1e-8	1e-8	4

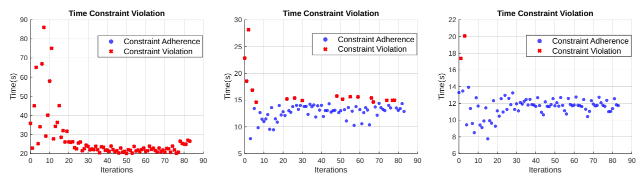
3) OptSFMec Implementation and Optimization Strategy: To implement the proposed OptSFMec, we have used pyswarm, a Python package for particle swarm optimization (PSO) with constraint support [37]. The PSO parameters are set to the values mentioned in TABLE IV. The optimization is terminated when two conditions are met: (1) if all constraints are satisfied and (2) the difference between the values of the objective function from three successive iterations is less than a defined tolerance of 0.01. This criterion is implemented to prevent unnecessary iterations.

The selection of decision variables is a crucial step in the optimization process. Properly chosen decision variables reduce the computational burden associated with searching in unfeasible regions and lead to faster convergence. In addition, it contributes to the generation of high-quality solutions. To achieve this, we carefully considered the lower and upper bounds of each decision variable. We started by setting the decision variables with values commonly utilized in ML/DL algorithms. We have chosen a sequence length ranging from 5 to 20. This selection aims to strike a balance, avoiding the computational expense and potential memory limitations associated with processing longer sequences while ensuring the model captures relevant patterns within a reasonable time. For the batch size, Fig. 11 explores how varying this decision variable influences the satisfaction of our problem constraints and then the convergence process. As can be seen, when batch sizes of 8 to 50 are used, the time constraint is consistently not met. Thus, opting for a batch size value outside this range is imperative to ensure better alignment with the specified time constraint. As depicted in Fig. 11b, with batch sizes between 51 and 128, the time constraint is more often adhered to than breached while the algorithm violates the constraint only twice when employing a batch size from 100 to 130, as illustrated in Fig. 11c. This limited occurrence of violations served as a motivating factor in opting for this specific range. Furthermore, given the expected higher computational capacities on the server side compared to the client side, it is more optimal to opt for an expanded range of the number of neurons on the server side. This permits effective utilization of computational power while enabling the server-deployed neural network to capture complex data patterns, enhancing its convergence and generalization. In addition, this increases the likelihood of satisfying constraint (14) and achieving the desired outcomes. Finally, we opted for a δ range between 0.1 and 1 to strike

TABLE V: PSO Solution

Parameter	N_s	b	C_{units}	$M_{l1,\text{units}}$	$M_{l2,\text{units}}$	δ
Solution	5	123	33	129	176	0.1

a balance between robustness to outliers and sensitivity to data variations. Figures 12 and 13 illustrate that, with the adopted parameters, the client and overhead constraints are less violated compared to the initial parameters.



a [8-50] b [51, 128] c [100, 130]

Fig. 11: Time-constraint violation across varied batch sizes

C. Results and Analysis

To the best of our knowledge, no existing literature has applied a metaheuristic approach to SFL for comparative analysis. To demonstrate the effectiveness of OptSFMec, we conducted comparisons with FedMec, SFMec and SFMecLite. Comparison is always made in terms of learning performance and computing performance. It is worth mentioning that for a fair comparison, we have used the PSO-derived solution listed in TABLE V to train FedMec anew.

We have two different sets of results to explore. The first part reveals the effectiveness of the proposed SFMecLite.

Fig. 14a and 14b demonstrate that SFMecLite has faster updates and prevents the spikes observed in SFMec, demonstrating a more consistent and stable learning curve. In contrast to SFMec setup where the server receives all activations from a single client, which can be exposed to a limited perspective of the data, training in batches allows multiple clients to package their activations into a single batch where each client may have different data characteristics, allowing the server to capture a more comprehensive set of patterns present in the overall dataset. This diversity provides a richer variety of information, helps the model generalize better, and enables it to adapt to different aspects of the data, leading to faster convergence and improved overall model performance. Furthermore, Fig. 14c shows that reducing the number of trips between clients and servers significantly contributes to reducing training time. The mean training time of SFMecLite has been reduced to $14.47\,\text{s}$, in contrast to the SFMec that required $23.86\,\text{s}$.

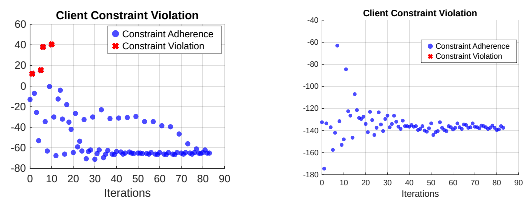
The second part of the results provides insights into the results of the optimization process, starting with an analysis of the learning performance. From Fig. 14a and 15a, one can easily observe the superiority of the proposed OptSFMec over the other three models in terms of the loss. It is even better than FedMec, which slightly outperformed SFMec in our first experiments. The same observations hold true for the MAE (Fig. 14b and 15b). OptSFMec reported a remarkable better performance compared to the baseline SFMec. Also, it exhibited better behavior than FedMec and SFMecLite variants, as it converges faster. This confirms the efficiency of OptSFMec in terms of learning performance over the two variants, SFMec and SFMecLite, as well as FedMec.

We now compare the different variants of SFMec and FedMec in terms of computing performance between the four models. Fig. 16 indicates that OptSFMec is significantly smaller than FedMec, SFMec and SFMecLite. The trainable parameters on the client side for OptSFMec is more than ten times better than the basic SFMec (as well as SFMecLite) and negligible compared to the number of parameters of FedMec. This directly influences the storage space necessary to save the model. As illustrated in Fig. 17, under OptSFMec, clients only need $0.022\,\text{MB}$ to store the model, making it ultra-lightweight and highly suitable for resource-constrained UAVs, where storing and running large models is infeasible. Moreover, OptSFMec significantly optimizes the communication overhead between the trainers and the servers. As shown in Fig. 18, compared to FedMec, it achieves a 71.24% reduction in overhead, and compared to SFMec and SFMecLite, it reduces overhead by 92.29% respectively, indicating a major improvement. This low communication overhead makes the model ideal for resource-constrained environments, as it decreases energy consumption, and extends the operational time of resource-limited clients. Moreover, the model scales more effectively in networks with multiple clients, as reduced data transmission helps minimize network congestion. Fig. 19 outlines the average training time (per round) for the four models. As can be seen, OptSFMec achieves the fastest training time ($9.65\,\text{s}$) outperforming FedMec ($27.39\,\text{s}$), SFMec ($23.86\,\text{s}$) and SFMecLite ($14.47\,\text{s}$). This leads to lower energy consumption, while also reducing memory and processing requirements. Furthermore, Fig. 20 demonstrates that OptSFMec exhibits significantly higher bandwidth efficiency compared to the three models and performs even better than FedMec.

Through the presented results, we have experimentally demonstrated that, in contrast to Hyperband, the PSO metaheuristic could be successfully used to deal with multi-criteria and high-dimensional machine learning optimization problems. The PSO solution has substantially decreased the overhead and training time, leading to significant improvements in the efficiency of the training process. The solution has also provided adaptable parameters for clients with resource limitations, enabling effective participation in the learning process despite their resource constraints. More importantly, these enhancements in efficiency and adaptability do not come at the expense of the model's learning performance. In contrast, numerically speaking, OptSFMec has shown superior results across all metrics compared to other proposed models.

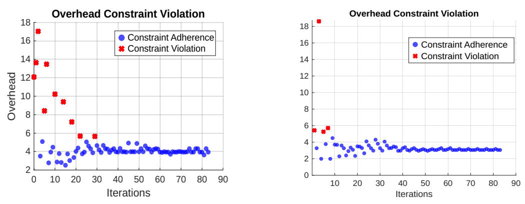
VI. Conclusion

This work introduces three distributed, collaborative, privacy-preserving and lightweight frameworks for resource optimization in MEC-enabled UAV networks. The proposed frameworks highlight a clear progression of enhancement. The first variant, SFMec, achieves comparable learning performance with FedMec, while significantly outperforming it in terms of computational efficiency. SFMec optimizes the model



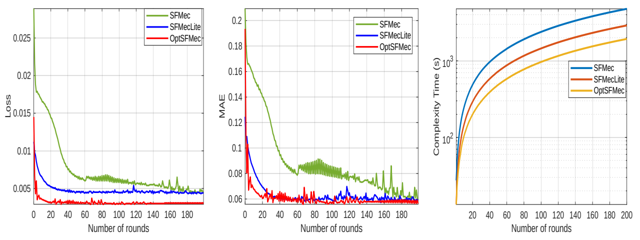
a Initial Parameters b Adopted Parameters

Fig. 12: Client-constraint violation



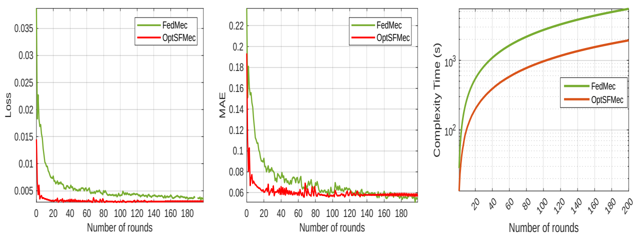
a Initial Parameters b Adopted Parameters

Fig. 13: Overhead constraint violation



a Loss b MAE c Training Time

Fig. 14: Learning performance and training time for SFMec, SFMecLite, and OptSFMec



a Loss b MAE c Training Time

Fig. 15: Comparative analysis of learning performance and training time complexity for OptSFMec and FedMec

size that each drone should train, enhancing both training efficiency and computational performance. This makes it an efficient baseline for high-performing UAV-based networks. Building on SFMec, we proposed a second variant, SFMecLite, which introduces significant enhancements to optimize the iterative exchanges between UAVs and the main server. Through empirical validation, we have demonstrated its efficiency compared to the baseline SFMec. Finally, we proposed a third variant, OptSFMec, that extends the enhancements of both SFMec and SFMecLite, by integrating additional optimization of the communication overhead between UAVs and the main server, as well as the training time, while concurrently enhancing the learning performance in comparison to FedMec, SFMec, and SFMecLite. Future work will consider unreliable and challenging network conditions between UAVs, base stations and edge servers. We will also incorporate additional parameters into the optimization process, evaluate the impact of various PSO parameters on OptSFMec, and explore alternative metaheuristics.

References

[1] V. Lambey and A. Prasad, "A review on air quality measurement using an unmanned aerial vehicle," *Water, Air, & Soil Pollution*, vol. 232, pp. 1–32, 2021.

[2] C. Malang, P. Charoenkwan, and R. Wudhikarn, "Implementation and critical factors of unmanned aerial vehicle (uav) in warehouse management: A systematic literature review," *Drones*, vol. 7, no. 2, p. 80, 2023.

[3] A. Rejeb, K. Rejeb, S. J. Simske, and H. Treiblmaier, "Drones for supply chain management and logistics: a review and research agenda," *International Journal of Logistics Research and Applications*, vol. 26, no. 6, pp. 708–731, 2023.

[4] S. Sharma, P. Kulkarni, and P. Pathak, "Applications of unmanned aerial vehicles (uavs) for improved business management," in *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)*. IEEE, 2022, pp. 53–57.

[5] N. H. Mahmood, N. Marchenko, M. Gidlund, and P. Popovski, *Wireless Networks and Industrial IoT*. Springer, 2020.

[6] H. J. Hadi, Y. Cao, K. U. Nisa, A. M. Jamil, and Q. Ni, "A comprehensive survey on security, privacy issues and emerging defence technologies for uavs," *Journal of Network and Computer Applications*, vol. 213, p. 103607, 2023.

[7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[8] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *Trans. Wireless. Comm.*, vol. 22, no. 4, p. 2650–2665, Apr. 2023.

[9] P. Chhikara, R. Tekchandani, N. Kumar, M. Guizani, and M. M. Hassan, "Federated learning and autonomous uavs for hazardous zone detection and aqi prediction in iot environment," *IEEE Internet of Things Journal*, vol. 8, no. 20, pp. 15 456–15 467, 2021.

[10] Y. Wang, Z. Su, T. H. Luan, R. Li, and K. Zhang, "Federated learning with fair incentives and robust aggregation for uav-aided crowdsensing," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3179–3196, 2022.

[11] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and H. Karimipour, "Federated learning for drone authentication," *Ad Hoc Networks*, vol. 120, p. 102574, 2021.

[12] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2021.

[13] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proc. Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.

[14] B. Brik, A. Ksentini, and M. Bouaziz, "Federated learning for uavs-enabled wireless networks: Use cases, challenges, and open problems," *IEEE Access*, vol. 8, pp. 53 841–53 849, 2020.

[15] H. Zhang and L. Hanzo, "Federated learning assisted multi-uav networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 14 104–14 109, 2020.

[16] Y. Wang, Z. Su, N. Zhang, and A. Benslimane, "Learning in the air: Secure federated learning for uav-assisted crowdsensing," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1055–1069, 2021.

[17] Y. Pan, C. Zhu, L. Luo, Y. Liu, and Z. Cheng, "Fedtrack: A collaborative target tracking framework based on adaptive federated learning," *IEEE Transactions on Vehicular Technology*, 2024.

[18] C. Shang, D. Thai Hoang, M. Hao, D. Niyato, and J. Yu, "Energy-efficient decentralized federated learning for uav swarm with spiking neural networks and leader election mechanism," *IEEE Wireless Communications Letters*, vol. 13, no. 10, pp. 2742–2746, 2024.

[19] M. Na, S. Cho, F. Solat, T. Na, and J. Lee, "Energy-efficient hybrid federated and centralized learning for edge-based wireless traffic prediction in aerial networks," *IEEE Access*, 2024.

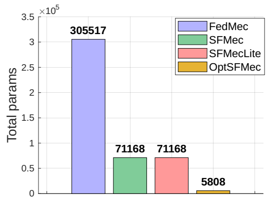


Fig. 16: Totals of parameters (client side)

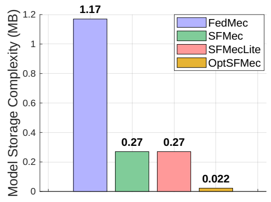


Fig. 17: Model Storage Complexity

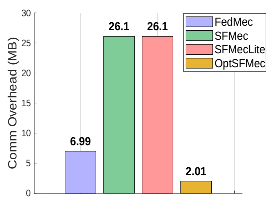


Fig. 18: Communication Overhead (per round)

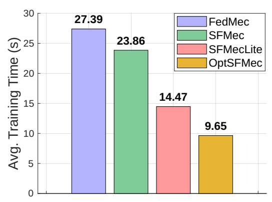


Fig. 19: Average Training Time (per round)

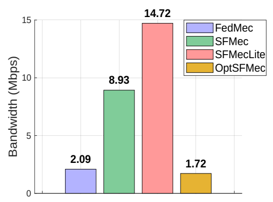


Fig. 20: Bandwidth

[20] Q. V. Do, Q.-V. Pham, and W.-J. Hwang, "Deep reinforcement learning for energy-efficient federated learning in uav-enabled wireless powered networks," *IEEE Communications Letters*, vol. 26, no. 1, pp. 99–103, 2021.

[21] S. Tang, W. Zhou, L. Chen, L. Lai, J. Xia, and L. Fan, "Battery-constrained federated edge learning in uav-enabled iot for b5g/6g networks," *Physical Communication*, vol. 47, p. 101381, 2021.

[22] Y. Shen, Y. Qu, C. Dong, F. Zhou, and Q. Wu, "Joint training and resource allocation optimization for federated learning in uav swarm," *IEEE Internet of Things Journal*, vol. 10, no. 3, pp. 2272–2284, 2022.

[23] C. Wang, X. Tang, D. Zhai, R. Zhang, N. Ussipov, and Y. Zhang, "Energy-efficient federated learning through uav edge under location uncertainties," *IEEE Transactions on Network Science and Engineering*, vol. 12, no. 1, pp. 223–236, 2025.

[24] T. Wang, X. Huang, Y. Wu, L. Qian, B. Lin, and Z. Su, "Uav swarm-assisted two-tier hierarchical federated learning," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 943–956, 2024.

[25] Q.-V. Pham, M. Le, T. Huynh-The, Z. Han, and W.-J. Hwang, "Energy-efficient federated learning over uav-enabled wireless powered communications," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 4977–4990, 2022.

[26] X. Liu, Y. Deng, and T. Mahmoodi, "A novel hybrid split and federated learning architecture in wireless uav networks," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 1–6.

[27] W. He, H. Yao, F. Wang, Z. Wang, and Z. Xiong, "Enhancing the efficiency of uav swarms communication in 5g networks through a hybrid split and federated learning approach," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*, 2023, pp. 1371–1376.

[28] C. Zhao, M. Sheng, J. Liu, T. Chu, and J. Li, "Energy-efficient power control for multiple-task split inference in uavs: A tiny learning-based approach," *IEEE Internet of Things Journal*, vol. 11, no. 12, pp. 21 146–21 157, 2024.

[29] T. Sun, X. Wang, M. Umehira, and Y. Ji, "Split learning assisted multi-uav system for image classification task," in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*. IEEE, 2023, pp. 1–6.

[30] T. Sun, X. Wang, X. Ye, and B. Han, "Stitch-able split learning assisted multi-uav systems," *IEEE Open Journal of the Computer Society*, vol. 5, pp. 418–429, 2024.

[31] H. Hafi, B. Brik, P. A. Frangoudis, A. Ksentini, and M. Bagaa, "Split federated learning for 6g enabled-networks: Requirements, challenges and future directions," *IEEE Access*, 2024.

[32] K. Góra, P. Smyczyński, M. Kujawiński, and G. Granosik, "Machine learning in creating energy consumption model for uav," *Energies*, vol. 15, no. 18, 2022.

[33] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[34] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 481–488.

[35] Z. Li, H. Li, and M. Zhang, "Hyper-parameter tuning of federated learning based on particle swarm optimization," in *2021 IEEE 7th International Conference on Cloud Computing and Intelligent Systems (CCIS)*. IEEE, 2021, pp. 99–103.

[36] F. Marini and B. Walczak, "Particle swarm optimization (pso). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[37] P. Contributors. (2022) Welcome to pyswarms's documentation! [Online]. Available: https://pyswarms.readthedocs.io/en/latest/index.html

PLACE PHOTO HERE

Houda HAFI conducted her Ph.D. studies in computer science at the University of Abdelhamid Mehri, Constantine, Algeria, and the Engineering School Polytech, Dijon, France. She obtained her Ph.D. from the University of Abdelhamid Mehri in 2019 and currently works as an Assistant Professor at the same University. Her ongoing research centers on wireless communications, vehicular and mobile networks, and the application of AI, machine learning, and distributed learning techniques in networking.

PLACE PHOTO HERE

Bouziane BRIK received his Engineer degree (Ranked First) and his Ph.D. degree in computer science from Laghouat University, Algeria, in 2010 and 2017, respectively. He is currently working as Assistant Professor at Computer Science department at University of Sharjah, UAE. He has also worked as an Assistant Professor at DRIVE department of Bourgogne university in France. He has been (still) working on resources management and security challenges of 5G networks and beyond. His research interests also include Explainable AI, and machine/deep learning for wireless networks.

PLACE PHOTO HERE

Zakaria Abou El Houda received the Ph.D. degree in computer science from the University of Montreal in 2021. He is currently an Assistant Professor with the National Institute of Scientific Research (INRS), within the Energy, Materials and Telecommunications Center (INRS-EMT), where he leads the Resilient Cybersecurity Research Laboratory. He is a member of the joint INRS-UQO Research Unit on Cybersecurity and Digital Trust. His current research interests include applied AI for intrusion detection systems, security in distributed/federated machine learning, and blockchain for network security.

PLACE PHOTO HERE

ADLEN KSENTINI received the Ph.D. degree in computer science from the University of Cergy-Pontoise. Since 2016, he has been a Professor with the Communication Systems Department, EURECOM. He is currently an IEEE COMSOC Distinguished Lecturer on topics related to 5G and Network Softwarization. His current research focuses on architectural enhancements to mobile core networks, mobile cloud networking, NFV, and SDN. He received the Best Paper Award from the IEEE WCNC 2018, IWCMC 2016, ICC 2012, ACM MSWiM 2005, and the IEEE Fred W. Ellersik Prize for the Best IEEE Communications Magazine for 2017.