Impact of Neural Network Depth on Split Federated Learning Performance in Low-Resource UAV Networks

Houda Hafi*, Bouziane Brik†, Miloud Bagaa‡, and Adlen Ksentini §
*NTIC Faculty, Abdelhamid Mehri University, Constantine, Algeria, houda.hafi@univ-constantine2.dz
†Computer Science Department, College of Computing and Informatics, Sharjah University, Sharjah, UAE, bbrik@sharjah.ac.ae,
‡Universite du Quebec a Trois-Rivieres, Trois-Rivieres, QC, Canada, miloud.bagaa@uqtr.ca,
§EURECOM, Sophia-Antipolis, France, adlen.ksentini@eurecom.fr

***Abstract*—Training without sharing data is one of the drivers that makes Federated Learning (FL) more attractive, compared to centralized approaches. However, requiring each learner to train the full model may not be efficient, particularly for devices with restricted resources, such as those available in Unmanned Aerial Vehicles (UAVs). To address this issue, a variation of FL technique, specifically Split Federated Learning (SFL), has recently been proposed. Unlike FL, the key concept of SFL is to divide the layers of the neural network among the involved learners. Therefore, each individual client will train only a segment of the model (submodel) rather than the entire model. Clearly, this technique, besides data privacy, optimizes the utilization of computational resources, reduces client-side training time, and enhances model privacy. However, there are questions that require answers: How should we split the model? Shall we systematically divide it in half, or is there a more optimal approach? In this line of thought, this paper provides a detailed analysis of possible splitting schemes of a power consumption prediction model for UAVs. First, the SFL-enabled model is presented. Second, an experimental analysis is conducted in which different splitting alternatives are made and numerically analyzed to examine the influence of network layering on split federated learning performance.**

***Index Terms*—Low-Resource Devices, Unmanned Aerial Vehicle (UAV), Split Federated Learning (SFL), Splitting Strategy.**

I. Introduction

Distributed Learning (DL) in general and Federated Learning (FL) in particular have garnered remarkable interest in the research community in recent years. Many state-of-the-art papers have suggested robust FL-based solutions in many applications, such as healthcare, smart grid, industrial engineering, and the Internet of Things (IoT) [1]. Notwithstanding the importance of federated learning in model training when data sharing is not possible, it is worth noting that the application of FL may not always be the best-performing policy. FL demands substantial computational capabilities, especially for large-scale models, which can pose challenges for less powerful or limited resource devices to effectively train the entire model. In real-world scenarios, numerous clients, such as UAVs, face limitations in computational power, memory, and energy compared to a server [2]. Therefore, training the full FL model on UAVs devices can deplete their resources, resulting in premature stoppage of the training process. Furthermore, in FL, both the server and clients have the ability to access the complete local and global models, making the system vulnerable to potential model attacks. To overcome these issues, SL [3] and SFL [4] are two recent algorithms that consist of splitting a complex neural network into smaller parts (subsections), each subsection is trained at a separate location (client or server-side). The participant learners handle the data up to an intermediate layer, referred to as the cut layer, while the remaining layers are maintained on the server side. It is apparent that both algorithms reduce the computation burden on the clients side since they train only a few layers instead of the entire model. In addition, SL and SFL ensure model privacy achieved through the architectural division of the model between clients and the server, as the client-side cannot access to the server-side model and vice versa [5]. However, the major inconvenience of SL lies in its training nature. In SL, each client i must wait for the previous client $i-1$ to finish before starting the training. This has a negative impact on the model's efficiency, particularly in terms of training time. Split Federated Learning (or SFL) is an enhanced distributed learning algorithm in which clients conduct their local training concurrently (in parallel), effectively overcoming the limitations of SL. Therefore, our focus in this study will be on SFL, which avoids the disadvantages associated with both FL and SL approaches by providing less computational complexity than FL and less temporal complexity than SL [6]. As will be seen in the upcoming sections, diverse set of SFL split scenarios are applicable, selecting the best strategy involves taking into account a range of criteria. The objective of this work is to investigate the impact of neural network depth on SFL performance in resource-constrained environments.

This will help to select the best configuration that strikes a good balance between model performance and resource efficiency. To this end, we design an SFL-based model for UAVs power consumption forecasting and investigate every possible splitting strategy between the clients and the main server. The remainder of the paper is organized as follows.

In Section II, a succinct review of the existing literature is presented. Afterward, our methodology and the designed approach are introduced in Section III. We comprehensively detail the experimental design and discuss the results obtained in Section IV. Finally, we conclude the paper in Section V.

II. Literature Review

Uploading UAV data to a central node for training can reveal scalability and privacy concerns. With the increasing prominence of federated learning as a distributed and privacy-preserving solution, numerous FL-supported methods have been proposed to tackle multiple problems in wireless UAV networks. For example, UAV path control [7], air quality sensing [8] and security issues [9]. Nevertheless, the applicability of FL in UAV networks remains a matter of contention due to the constrained resources of UAV devices, especially for complex models with a high number of trainable parameters. Training large models requires substantial computational resources and a long-lasting battery, which are not typically inherent in UAV devices. To further improve the federated training process in UAV-assisted networks, other initiatives focus on optimization. The study [10] introduces a deep reinforcement learning (DRL)-based algorithm for improving long-term FL performance. It employs Lyapunov optimization to simplify energy constraints, transforms the problem into a Markov decision process (MDP), and uses DRL to optimize UAV placement and resource allocation, promoting sustainable UAV-assisted networks and energy conservation. Similarly, the authors in the paper [11] investigate the optimization of federated edge learning in the context of UAV-enabled IoT for B5G/6G networks. They propose a framework that enables devices to adapt their CPU-frequency settings to extend the battery life of UAVs and prevent premature dropout from the training process, particularly in dynamic environments. To address this optimization challenge, the authors employ the deep-deterministic policy gradient strategy (DDPG) as their approach. In a parallel way, authors in [12], 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. In a recent publication by [13], authors outlines a hybrid methodology. This approach combines FL for a portion of clients (with high resources) and SL for collaborative training in conjunction with a central base station.

III. Methodology and Study Design

A. Overview

In the context of an SFL model $\mathcal{M}$ of ℓ hidden layers, it is pertinent to note that the total number of potential splitting possibilities is then $(\ell - 1)$. For each possibility $\mathcal{P}_i$, $i \in \{1, \dots (\ell - 1)\}$, the client and server sides would have $i+1$ and $(\ell - i + 1)$ layers, respectively. The aim of this work is to examine the influence of the splitting strategy on both the performance of SFL model and the utilization of devices' resources. For that, we investigate each probability and analyze its impact on the entities (the clients and the server) involved in the neural network framework for the given application. The objective is to determine the best decomposition scheme that yields good performance on low-resource devices. In this vein, we opt for a UAV battery power consumption application. First, we define a centralized model using Long Short-Term Memory (LSTM) [14] to forecast the instantaneous power consumption of UAVs using their sensed data. After selecting the optimal network hyperparameters, the training of the SFL model is carried out using the same hyperparameter settings.

B. LSTM Model Architecture

LSTM plays a crucial role in capturing sequential information and dependencies within time series data, making them suitable for tasks involving sequences, such as energy forecasting. In our study, we employed a supervised regression model comprising eight LSTM layers. The model takes a set of numerical features as input, the core of the model consists of six LSTM layers stacked on top of each other. The final LSTM layer's output is passed through an output layer (energy prediction). The first two hidden layers consist of 128 neurons, whereas the third hidden layer contains 64 neurons. The fourth and fifth hidden layers each of which consists of 32 neurons, while the final hidden layer contains 64 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 Huber loss function is used with $\delta = 1$. The model is optimized by Adam optimizer with a learning rate of 0.001.

C. LSTM Model Splitting

As stated earlier, SFL adopts the same principle of SL by dividing the neural network into two parts (see Fig. 1). The first part (client side) is trained by the participating nodes (UAVs), while the second part (server side) is trained by the *main server*.

- **LSTM Client**: On this side, each UAV retrieves the initial weights from the fed server (Step 1 in Fig. 1). Then, it performs the forward propagation using the input data and transmits the interim results (*smashed data*) along with target values to the main server. When the server ends its associated tasks (see the LSTM server), the client obtains the gradients from the server, conducts the backpropagation process, and adjusts its weights. After some rounds, all participating UAVs forward their local updates to the fed server for aggregation (step 4 in Fig. 1).
- **LSTM Server**: Upon obtaining the output data from a client (Step 2 in Fig. 1), the server resumes the feedforward pass. Subsequently, the loss value, denoted as L, is computed using formula 1. Then, the gradients for both the server and client are computed. Lastly, the gradients are sent back to the the client's *cut layer* (the layer at which the neural network is split) (step 3 in Fig. 1) and the precision of the prediction is measured.

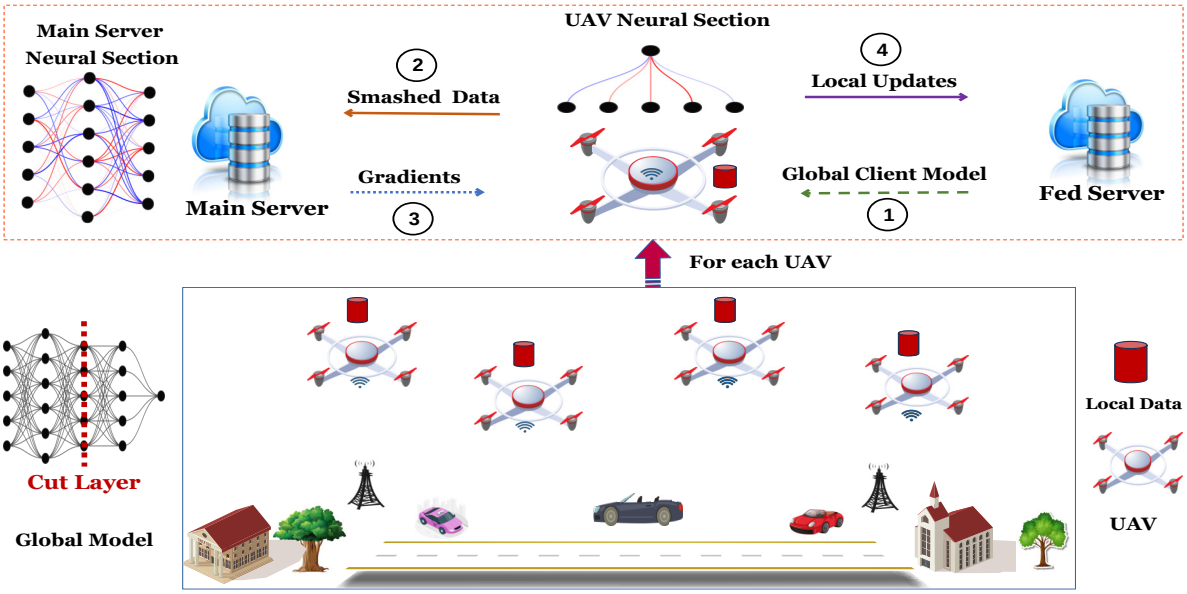


Fig. 1: Split federated learning framework architecture for UAV-enabled IoT networks

TABLE I: LSTM Model parameters.

Parameters	Values
Window Step	10
Loss Function	Huber (delta = 1)
Activation Function	Tanh (Input and Hidder Layers) Linear (Dense Layer)
Optimizer	Adam
Learning Rate	0.001
Batch Size	64
Normalization Alg	MinMax Scalar
Number of Rounds	200

TABLE II: Possible Split Variants.

Model	M1	M2	M3	M4	M5
(client,server)	(2,6)	(3,5)	(4,4)	(5,3)	(6,2)

IV. Experimental Setup, Results and Discussion

A. Dataset

We chose a recent dataset collected with a hexacopter drone [15] that has six 18-inch propellers, weighs 6 kg, and a Maximum Takeoff Mass (MTOM) of 13 kg. The goal is to empirically measure the power consumption of electric UAVs. The experimenters collect data during automatic and manual missions executed by UAV, both without any payload weight and with additional payload weights of 2 kg, 4 kg, and 6 kg. Measurements were collected for the hourglass shape trajectory with a velocity equal to 4 m/s. The dataset contains 27 variables (altitude (m), linear drone's velocities (m/s), angular drone's velocities (rad/s), orientation of the drone, total drone mass (kg), etc.). To carry out our experimental study, we selected the most relevant features.

B. Parameter Settings

The proposed model 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 divided equitably among the five clients. The algorithm is implemented in Python 3.10.12 with TensorFlow 2.12.0. The experiments were conducted within the Google Colab environment. Table I outlines the parametric settings used in the evaluation.

C. Performance Metrics

In this subsection, we present the metrics that we considered to validate our study in terms of computing and learning performance.

1) Computing Performance:

- Model's footprint: It denotes the client's necessary memory to store its related segment for each model. To carry this out, we calculate the product of the total number of parameters within the client's sub-model and the size of each parameter.
- Memory usage (during training): Here, we monitor the used memory during training. To accomplish this, we use the python module *memory-profiler*.
- Communication overhead: It represents the data size sent between the participating entities, namely: client and server (weights, activation, gradients, etc.).
- Training Complexity: In this context, we calculate the duration taken by the clients and the server to complete the forward and backward pass phases during one round of the process. Consequently, we will have two temporal parameters, namely:
  - Forwarding Pass Time (FPT): It represents the amount of time taken by the clients/server to execute the

feedforward phase. In our figures, we represent this as CFPT for clients and SFPT for servers.

– Backward Pass Time (BPT): This signifies the required time for the clients/server to perform the Back propagation phase. We designate this as CBPT for clients and SBPT for servers.

2) Learning Performance:

- Loss: It is a well-known learning metric that quantifies the error between the predicted outputs of the model 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 equation 1.

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

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

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (2)$$

Where:

- $\hat{y}$ is the predicted value
- y is the actual value
- n is the number of data points
- δ is a tuning parameter

D. Numerical Results and Discussion

To comprehensively explore all possible split scenarios, we vary the depth of LSTM on both the client and server sides. As mentioned in section III, with a model of six hidden layers, we will have five schemes from $M1$ to $M5$. Table II summarizes the different partitions. Each partition is represented by a pair denoted as (i, j), where i indicates the layers assigned to the client (including the input layer) and j represents the server layers (including the output layer).

Before delving further, it is worth noting that the client must have at least one hidden layer; otherwise, the configuration is unreliable from a learning perspective. Basically, it might be perceived as a centralized learning approach, where the clients pass their input data to the server hosting the ensemble of hidden layers, without performing any further transformation. The adoption of such a configuration would raise security and privacy concerns, which runs counter to the core objectives of split federated learning. Similarly, the server should include a minimum of one hidden layer, since hosting all the layers on the clients while providing only the output layer to the server (which is expected to have greater computational power than the clients) does not align with the requirements of poor-constrained devices.

Table III compares the five potential methods to split the proposed model in terms of various computing metrics, namely: total parameters, model footprint (kB), training memory (MB) and communication overhead (MB). We initially explore the correlation between the depth of the model in terms layers and its dimension in terms of parameters. From empirical results, it is obvious that when the number of layers increases on the client side, the overall number of parameters also increases. For instance, in model $M1$, which consists of only two client-hosted layers, the total number of parameters is 71168, roughly constituting 24% of the entire model, and equating to a storage size of $278\,\mathrm{kB}$. However, with three layers (model $M2$), the total number of parameters increased to 202752, representing 68% of the whole model and corresponding to $792\,\mathrm{kB}$ of size. This immense gap in terms of model size and parameter count between $M1$ and $M2$ is due to the increase in the number of computational units within $M2$'s architecture, where we introduced a hidden layer with 128 neurons on the client side. Nevertheless, from $M3$ to $M5$, we behold a consistent increase in the number of trainable parameters and model storage volume due to the shallow depth of newly introduced layers (64, 32, and 32). On the server side, on the contrary, we note a gradual decrease in the model's complexity because of the reduction in the number of layers and their depth in each setup. Consequently, this will necessitate less memory space to store the model.

Regarding memory utilization during training, as well seen from the table, adding more layers in each configuration creates a deeper neural network that actually captures complex relationships in data, but increases the number of neurons leading to an increase in the trainable parameters and therefore a rise in the memory requirements to manipulate these parameters. Optimal memory usage values are achieved with model $M1$ for the client, featuring a memory consumption of $7.4\,\mathrm{MB}$, and model $M5$ for the server, with a memory consumption of $4.5\,\mathrm{MB}$. Similarly, the server and the client experience the highest memory demand with $M1$ and $M5$ model architectures, respectively. It is worth mentioning here that the symmetry in the number of layers between clients and the server (case of model $M3$) does not inherently imply uniformity in terms of computing performance.

For communication overhead, the table shows that, on the client side, $M1$ performs equally to $M2$. As well, $M4$ performs as equally as $M5$. This is due to the fact that during the forwarding phase, the data transmitted from the client to the server consists of the activation (smashed data), originating from the last layer of the client side section. Since both pairs of configurations share identical numbers of neurons in the cut layer, 128 for $(M1, M2)$ and 32 for $(M4, M5)$, the communication overhead exhibits similarity. At the same time, we see that the transmission overhead (gradients) on the server side gradually increases from $M1$ to $M5$, reaching its maximum value with $M5$. This occurs because the client-side model becomes more complex from $M1$ to $M5$, therefore, the magnitude of the gradients becomes more larger, resulting in more significant overhead over the network.

Complexity time is a crucial factor to consider when dealing with resource-constrained UAV devices. It is well known that

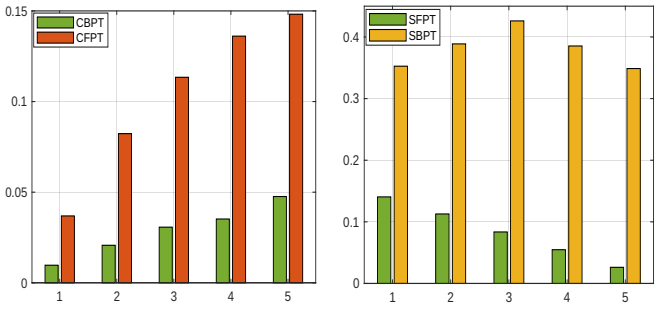
TABLE III: Computing Performance

Computing Performance								
Metrics	Model Total Param		Model Footprint (KB)		Training Memory (MB)		Comm Overhead (MB)	
	Client	Server	Client	Server	Client	Server	Client	Server
M1	71168	226625	278	885.25	7.4	18.5	13.95	12.15
M2	202752	95041	792	371.25	11.7	10.8	13.95	34.65
M3	252160	45633	985	178.25	14.1	6.3	7.2	43.2
M4	264576	33217	1010	129.75	14.3	5.8	3.6	45.45
M5	272896	24897	1040	97.25	21.8	4.5	3.6	46.8

a prolonged training time demands more computational power and energy consumption, leading to depleted UAV batteries.

Fig. 2 illustrates the forward and backward phase times for the client and server sides, respectively, across the five models. From Fig. 2(a), one can observe that clients spend more time in forward propagation compared to backward propagation. This is due to the fact that the SFL clients do not need to compute the model gradients (they are calculated by the server); therefore, the weights update process on the client side is relatively straightforward and computationally less intensive compared to the forward pass that involves matrix multiplications and activation function evaluations. However, the inverse scenario is depicted in Fig. 2(b). The server expends less time on forward propagation and more time on the backward step. The reason is that the server assumes the responsibility for computing the gradients with respect to the neural network parameters. This process is computationally intensive, especially for large networks and datasets, and it typically dominates the overall training time. Among the five models considered, $M1$ demonstrates the lowest forward and backward complexity times on the client side.

Fig. 3 compares the forwarding computation time between the client and the server across the five models. CFPT shows an upward trend attributed to the increase in the number of layers and neurons in each division strategy. Conversely, the server's forwarding processing time (SFPT) demonstrates a downward trend owing to the reduction in the number of layers and learnable parameters that need to be trained from one model to another. From Fig. 4, it is evident that the global training time overlaps across all five models. This observation highlights that splitting the SFL model enhances the training time of resource-constrained clients without impacting the overall training duration. Figures 5 and 6 present the mean and variance values of learning performance (loss and MAE) for the five models over 200 rounds. Fig. 5(a) plots the range of mean loss values across models $M1$ to $M5$. The length of the box, representing the interquartile range, varies from 0.0046 for $M1$ to 0.0048 for $M5$. The median, positioned at 0.0046, indicates the central tendency of the loss. In particular, there are no extreme values present in the results obtained. Furthermore, Fig. 5(b) represents the distribution of loss variance across the different settings. The length of the box varies from approximately 2.67×10^{-5} for $M1$ to $2,8 \times 10^{-5}$ for $M3$. The median value of the loss variance is positioned at approximately 2.66×10^{-5}, which is very low, suggesting that the variance of the loss across all settings is quite minimal. These results indicate that the five trained models are relatively close to each other and exhibit similar stability in terms of loss. Similarly, from Fig. 6, the median values for the mean and variance MAE are $0,0629$ and $1,26 \times 10^{-3}$, respectively, indicating consistent MAE performance across different settings.



(a) Client side (b) Server side

Fig. 2: Forward and Backward time for clients and server

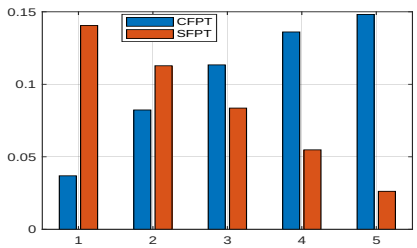


Fig. 3: Client vs Server forwarding time

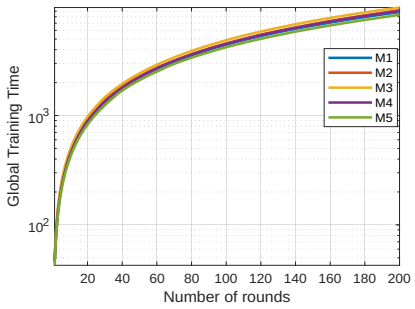
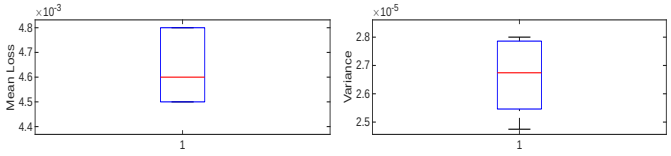
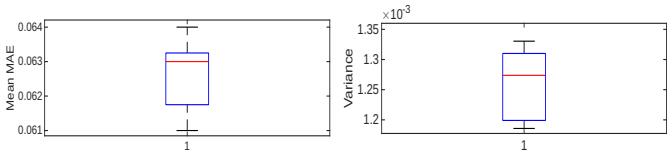


Fig. 4: Global training time



(a) Mean loss (b) Loss variance

Fig. 5: Mean and Variance loss across models



(a) Mean MAE (b) MAE variance

Fig. 6: Mean and Variance MAE across models

V. Conclusion

This study examines the impact of neural network depth on the performance of split federated learning systems and resource utilization in low-capacity devices. The experimental results reveal that split federated learning enhances the computing performance of resource-constrained devices, such as client memory capacity and computation time, while maintaining the overall training duration and learning performance. Through our investigation, we derive the following key observations. First, splitting the model in half does not consistently provide the best solution (case of M3). Moreover, random partitioning of the model is not considered a best practice, because in some cases, the server, as a powerful computational resource, ends up handling relatively simpler computations compared to clients, which are relatively resource-constrained (case of M5). This situation could potentially deplete the client's resources and lead to a premature termination of the training process. Therefore, splitting a model should be done with careful consideration and a clear understanding of the trade-offs at play such as the hardware specifications of the learners involved in the training, along with the available bandwidth, as split federated learning entails significant communication overhead between clients and the server. Another crucial point to consider is that while M1 effectively adapts to UAV requirements in this study, the first split (with the minimum number of layers on the client side) may not always be the optimal choice as it is greatly influenced by the architecture of the neural network.

As a future direction of research, one could extend the current work by investigating the impact of network depth on SFL systems when dealing with heterogeneous devices with varying computational capabilities. In addition, the dynamic adjustment of the depth of the neural network during the training process based on the availability of resources can be analyzed to have a deeper view of the splitting strategies.

Acknowledgment

This work was partially supported by the European Union's Horizon 2020 Research and Innovation Program under the AC3 project (Grant No. 101093129).

References

[1] 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.

[2] 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.

[3] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey, "No peek: A survey of private distributed deep learning," *CoRR*, vol. abs/1812.03288, 2018. [Online]. Available: http://arxiv.org/abs/1812.03288

[4] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[5] N. D. Pham, A. Abuadbba, Y. Gao, T. K. Phan, and N. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," *IEEE Transactions on Information Forensics and Security*, 2023.

[6] 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.

[7] H. Shiri, J. Park, and M. Bennis, "Communication-efficient massive uav online path control: Federated learning meets mean-field game theory," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6840–6857, 2020.

[8] Y. Liu, J. Nie, X. Li, S. H. Ahmed, W. Y. B. Lim, and C. Miao, "Federated learning in the sky: Aerial-ground air quality sensing framework with uav swarms," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9827–9837, 2020.

[9] N. I. Mowla, N. H. Tran, I. Doh, and K. Chae, "Federated learning-based cognitive detection of jamming attack in flying ad-hoc network," *IEEE Access*, vol. 8, pp. 4338–4350, 2019.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] K. Góra, P. Smyczyński, M. Kujawiński, and G. Granosik, "Machine learning in creating energy consumption model for uav," *Energies*, vol. 15, no. 18, 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/18/6810