Communication-Pipelined Split Federated Learning for Foundation Model Fine-Tuning in UAV Networks

Zizhen Zhou, Ying-Chang Liang, Yanyu Cheng, and Wei Yang Bryan Lim

***Abstract*—Deploying foundation models (FMs) on uncrewed aerial vehicles (UAVs) promises broad "low-altitude economy" applications. Split federated learning (SFL)-based fine-tuning leverages distributed data while keeping raw data local and reduces client-side burden by partitioning the model between client and server. However, the per-round training latency is dominated by stragglers. Training paradigms featuring parallel gradient transmission (GT) allocate dedicated portions of downlink communication resources to each client. They may leave resources idle and suffer from prolonged GT latency, especially in UAV networks, where the communication latency typically far exceeds the computation latency. To address this, we propose a sequential GT paradigm, where the server dedicates all downlink resources for the current GT. We further propose communication-pipelined SFL (CPSFL), characterized by downlink GT priority scheduling and intra-round asynchronous training. We investigate CPSFL-based LoRA fine-tuning of FMs in UAV networks and formulate an optimization problem to minimize a weighted sum of per-round training latency and worst-case client energy consumption by optimizing the split point selection (SPS) and the computing and communication resource allocation (CCRA) (the uplink bandwidth allocation and the server computing frequency allocation). To solve this problem, we develop an attention-based deep reinforcement learning (DRL) framework, where the base station agent decides the split point and the CCRA in each round by leveraging previous round information, including UAV trajectories. Simulation results show that the proposed DRL-based CPSFL scheme outperforms the parallel GT benchmarks, the ablation variants, the fixed CCRA scheme, while approaching the best fixed-SPS scheme.**

***Index Terms*—Split federated learning, UAV network, resource allocation, reinforcement learning**

I. Introduction

Uncrewed aerial vehicles (UAVs) are widely applied in low-altitude economic activities such as natural resource management, power facility inspection, and public security patrol, owing to their flexible deployment and controllable mobility [1]. During missions, UAVs collect large volumes of sensory data (e.g., images and videos), which can be leveraged to adapt foundation models (FMs) to diverse downstream tasks [1]. Fine-tuning FMs, rather than training from scratch, substantially reduces the required data and computation [2].

Centralized fine-tuning in a data center after the UAVs land is a straightforward approach, but it suffers from some limitations: (i) bandwidth-constrained backhaul links from the landing site to the data center; (ii) privacy or regulatory concerns with sensitive data; and (iii) the need for near-real-time adaptation to dynamic environments (e.g., weather, lighting). These challenges motivate edge-side, ongoing FM adaptation during missions. Federated learning (FL) offers a promising solution, enabling collaborative model improvement while avoiding raw data transmission: clients train locally and upload only model parameters to a server for aggregation [3], [4]. Nevertheless, full-parameter fine-tuning (FPFT) of an entire FM is often impractical on UAVs with limited memory, computing capability, and energy resources.

Low-rank adaptation (LoRA) [5], a prominent parameter-efficient fine-tuning (PEFT) method, greatly reduces the number of trainable parameters (TPs) while achieving accuracy comparable to FPFT. LoRA is thus attractive for FL in UAV networks because it (i) reduces memory usage by eliminating the need to store optimizer states for frozen parameters and (ii) reduces communication overhead, as UAV clients only need to transmit a small number of TPs for federated aggregation. However, even with LoRA, hosting and updating an entire FM on resource-constrained UAVs can still be challenging.

Split federated learning (SFL) further alleviates the client-side burden by splitting the FM into a client-side model and a server-side model, while still enabling parallel client training [6]. However, due to synchronized federated aggregation, the training latency in the SFL is determined by the slowest client, commonly referred to as the "straggler" [7]. In UAV networks, the straggler issue is aggravated by the heterogeneous computing and communication capabilities, mobility, and limited energy of UAVs [8]. Split point selection (SPS) significantly influences both training latency and client-side energy consumption, as it determines the computational load on both the client and server, as well as the communication overhead [4], [7]–[10]. Efficient utilization of server-side computing and communication resources is also critical in mitigating the straggler problem [7]–[10].

A. Related Works and Challenges

Recently, SPS and communication-computing resource allocation (CCRA) for SFL in wireless networks have received

Z. Zhou is with the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China (UESTC), Chengdu 611731, China (e-mail: zhouzizhen@std.uestc.edu.cn).

Y.-C. Liang is with the Center for Intelligent Networking and Communications (CINC), University of Electronic Science and Technology of China (UESTC), Chengdu 611731, China (e-mail: liangyc@ieee.org).

Y. Cheng is with the School of Cyberspace, Hangzhou Dianzi University, Hangzhou 310018, China (email: yycheng@hdu.edu.cn).

W. Y. B. Lim is with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: bryan.limwy@ntu.edu.sg).

TABLE I: Comparison of server-side computing and communication resource utilization paradigms

Paradigms	Server-side computing	Downlink gradient transmission	Related studies
SFL-PP	Parallel	Parallel	[11]–[25] [26]–[33] and PipeSFL [37] CPSFL (ours)
SFL-SP	Sequential	Parallel	
SFL-PS	Parallel	Sequential	

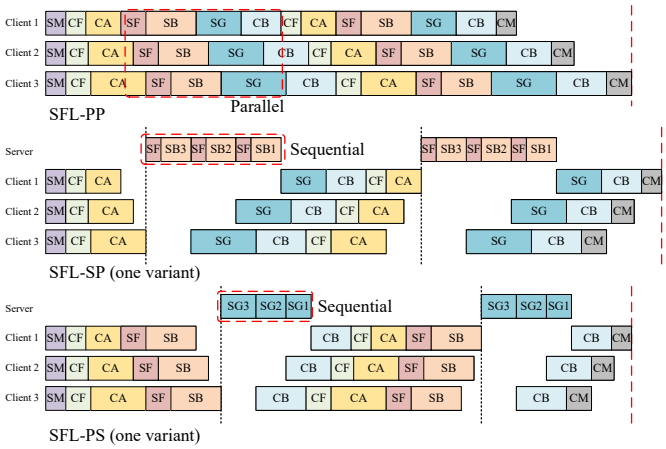


Fig. 1: The timeline of one training round (consisting of two local iterations) for three clients when SFL-PP (top), SFL-SP (middle), or SFL-PS (bottom) is applied. The notations SM, CF, CA, SF, SB, SG, CB, and CM correspond to the eight steps in Fig. 2, respectively.

significant attention [11]–[36]. To mitigate stragglers, existing works explore various strategies, including: (i) individual SPS for each client [11]–[18], [21]–[25], [29]–[32]; (ii) client grouping [19], [26]; (iii) parallel split learning without client-side model aggregation [27], [28]; and (iv) inter-round asynchronous training [12], [16]. Moreover, SFL for LoRA-based fine-tuning in wireless networks has been studied in [20]–[22], [32]–[34], considering aspects such as server energy consumption [21], client storage constraints [20], [22], privacy preservation [22], LoRA rank optimization [33], and smashed data compression [20]. However, they may incur long per-round latency due to inefficient utilization of downlink communication resources. Specifically, we analyze as follows.

We classify SFL into three types based on whether the server can compute tasks for different clients in parallel and whether it can transmit gradients to different clients simultaneously over the downlink. Table I summarizes these paradigms, referred to as SFL-PP, SFL-SP, and SFL-PS, and their time diagrams are illustrated in Fig. 1.

SFL-PP with full parallelism: In SFL-PP, all steps of different clients are executed in parallel. Specifically, the server allocates a fixed portion of the shared resources, including its computing frequency, the bandwidth for downlink gradient transmission (GT), and corresponding transmit power, to each client and keeps this allocation unchanged in one training round. SFL-PP is widely adopted in existing works [11]–[25] due to its analytical tractability: the per-round training latency equals the maximum per-round latency among all clients. However, idle resources allocated for one client cannot be utilized by the ongoing computations or downlink GT of other clients, which may increase the per-round latency. For example, in Fig. 1, when the server begins GT for client 1, the downlink resources allocated to client 3 remain idle.

SFL-SP with sequential server-side computing: SFL-SP uses all computing resources for the current computing task and decides the scheduling order of the client tasks. We refer to the most widely adopted form as vanilla SFL-SP [26]–[31], [33], which exhibits two synchronizations per local iteration: (i) the server starts computing only after receiving the smashed data from all clients, and (ii) it starts GT only after completing the computing tasks of all clients. Thus, the vanilla SFL-SP is vulnerable to the straggler issue. To mitigate this, the pipelining is introduced to overlap the communication latency and the client computing latency with the server computing latency (the SF and SB steps in Fig. 1) as much as possible [32], [37]. In [32], the server prioritizes tasks from clients with longer backward propagation (BP) times. In [37], PipeSFL is proposed to enable finer-grained pipelined computing scheduling via server-side computing priority scheduling and intra-round asynchronous training. Nevertheless, in wireless networks such as UAV networks, where the computing latency is much smaller than the communication latency, the performance gain offered by PipeSFL may be limited.

SFL-PS with sequential downlink GT: Different from SFL-PP and SFL-SP, we propose SFL-PS, which uses all downlink resources for the current GT and decides the scheduling order across clients. By reducing the GT latency and allowing clients to immediately proceed to subsequent steps upon receiving gradients, SFL-PS has the potential to reduce the per-round latency of SFL in wireless networks. Notably, SFL-PS has not been explored in existing studies.

In addition to the straggler issue caused by the heterogeneity of UAV clients, their mobility introduces another major challenge. The SFL with mobile clients, such as vehicles, UAVs, and satellites, has been studied in [23]–[25], [31], [35]. Nevertheless, these works oversimplify the channel variations caused by the moving clients: they either assume the channel remains unchanged within a training round [25], [31], [35] or approximate communication rates using time-averaged values over the coverage period [23], [24]. Given that a single training round often exceeds ten seconds [24], [31], a finer-grained latency analysis is essential. Building on this, the historical trajectory data can be leveraged to infer future mobility patterns and enable proactive resource management. To handle the environment with uncertain UAV trajectory and the complexity of the optimization problem, deep reinforcement learning (DRL) techniques are promising for decision-making in SFL scenarios [13], [23], [24], [36].

B. Main Contributions

In summary, existing SFL training paradigms underutilize downlink communication resources, and existing SFL resource allocation studies inadequately address mobile clients, particularly lacking slot-level channel modeling and trajectory-aware decision-making. To tackle these challenges, we propose communication-pipelined SFL (CPSFL) and an attention-

based DRL framework for joint SPS and CCRA in CPSFL-enabled UAV networks, supported by the fine-grained latency analysis and the UAV trajectory features extraction. The main contributions are summarized as follows:

- To improve downlink resource utilization, we propose SFL-PS, where the server transmits gradients sequentially. To mitigate stragglers in vanilla SFL-PS, we further propose CPSFL, incorporating two PipeSFL-inspired enhancements: (i) downlink GT priority scheduling that increases latency overlap by prioritizing clients with larger lags, and (ii) intra-round asynchronous training that reduces downlink idling by enabling immediate GT upon server computation completion. We also provide the optimality proofs for the scheduling policy under certain simplifying assumptions, along with the latency analysis and comparisons.
- To finely capture the UAV mobility, we propose a fine-grained latency analysis under SFL-PS where the channel varies per time slot instead of per training round. Based on this, we formulate an optimization problem to minimize both the per-round training latency and the maximum per-client energy consumption by jointly optimizing SPS and CCRA (i.e., the uplink bandwidth allocation and the server computing frequency allocation).
- Decision-making at the round start is intractable due to the problem complexity and the unavailability of current-round channel knowledge. To address this, we design an attention-based DRL framework, where the base station (BS) agent leverages previous round information to determine the split point and the CCRA. An attention mechanism enables effective feature extraction from variable-length UAV trajectories.
- Simulation results show that CPSFL achieves lower per-round latency than SFL-PP, PipeSFL, and ablation variants in UAV networks where the communication latency is dominant. Moreover, the DRL-based CPSFL scheme outperforms its variant that does not leverage UAV trajectory and the fixed CCRA scheme, and approaches the best fixed-SPS scheme.

C. Organizations

The rest of this paper is organized as follows: Section II presents the system model, the slot-level latency analysis under the SFL-PS paradigm, and the optimization problem. Section III presents the proposed CPSFL. In Section IV, we propose the attention-based DRL framework to decide the SPS and the CCRA. Section V shows the simulation results. Finally, Section VI concludes this paper.

Notations: The lowercase, bold lowercase, and bold uppercase, i.e., a, $\mathbf{a}$, and $\mathbf{A}$ are scalar, vector, and matrix, respectively. $\mathbb{R}^{a\times b}$ denotes the space of $a \times b$ real-valued matrices. $|\mathcal{A}|$ denotes the cardinality of set $\mathcal{A}$. $(\cdot)^\top$ denotes transpose. $\mathbb{E}\{\cdot\}$ denotes the average operation.

II. System Model

As shown in Fig. 2, we consider a CPSFL-enabled UAV network, where a BS collaborates with K UAVs to fine-tune a complete FM $\mathbf{w}$ through the CPSFL. Specifically, the BS acts as the server, while the UAVs serve as the clients. The FM is split at the split point u into a server-side FM $\mathbf{w}_s$ and a client-side FM $\mathbf{w}_c$, which are fine-tuned at the BS and the UAVs, respectively. In each local iteration, UAV k processes a data batch $\mathcal{D}_k = \{\mathbf{X}_k, \mathbf{y}_k\}$ of size B, where $\mathbf{X}_k = \{\mathbf{x}_k^b\}_{b=1}^B$ is the set of raw data and $\mathbf{y}_k = \{y_k^b\}_{b=1}^B$ is their corresponding labels. We denote the set of UAV indices as $\mathcal{K}$.

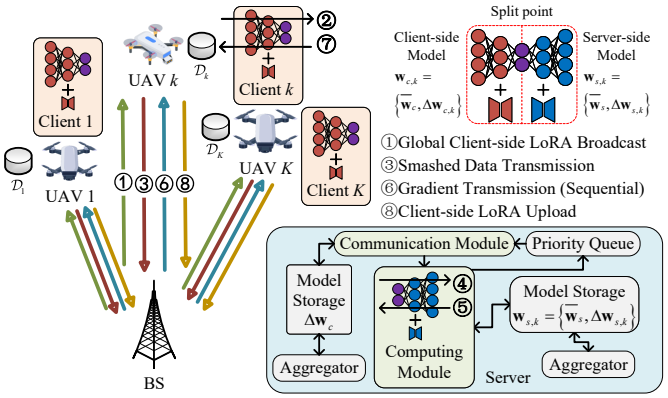


Fig. 2: Communication-pipelined split federated learning for foundation models LoRA fine-tuning in UAV networks.

A. Preliminaries of LoRA and SFL

LoRA adapts pre-trained models by injecting trainable low-rank matrices while freezing original weights [5]. For a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times h}$, LoRA represents updates as $\mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{BA}$, where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times h}$, and the rank $r \ll \min(d, h)$. This reduces the number of TPs from $d \times h$ to $(d + h) \times r$ while preserving performance.

We denote the client-side FM as $\mathbf{w}_c = \{\overline{\mathbf{w}}_c, \Delta\mathbf{w}_c\}$, where $\overline{\mathbf{w}}_c$ is the frozen pre-trained FM parameters and $\Delta\mathbf{w}_c$ is the client-side TPs, i.e., the LoRA module weights. Similarly, we denote the server-side FM as $\mathbf{w}_s = \{\overline{\mathbf{w}}_s, \Delta\mathbf{w}_s\}$, where $\Delta\mathbf{w}_s$ denotes the TPs including the LoRA module weights and the task module, e.g., the classification head in the classification task.

The SFL process consists of N training rounds, each comprising eight steps as shown in Fig. 2. At the start of round n, the server broadcasts the latest global client-side TPs $\Delta\mathbf{w}_c(n-1)$ (i.e., $\Delta\mathbf{w}_{c,k}(n, 0)$, $\forall k$) to all clients (Step 1). The server set $\Delta\mathbf{w}_{s,k}(n, 0) = \Delta\mathbf{w}_s(n-1)$, $\forall k$. Subsequently, the K clients perform I local iterations, with each local iteration involving Steps 2 to 7, detailed as follows: Step 2: Client k performs the forward propagation (FP) of the client-side model $\mathbf{w}_{c,k}(n, i-1)$ and obtains the output (called smashed data) $\mathbf{A}_k(n, i) = f(\mathbf{X}_k(n, i); \mathbf{w}_{c,k}(n, i-1))$. Step 3: Client k transmits $\mathbf{A}_k(n, i)$ and label $\mathbf{y}_k(n, i)$ to the server. Steps 4 and 5: The server performs the FP and BP of the server-side model $\mathbf{w}_{s,k}(n, i-1)$ and obtains $\Delta\mathbf{w}_{s,k}(n, i)$. Step 6: The server transmits the gradients of the smashed data $\mathbf{G}_k(n, i) = \nabla\ell(\mathbf{A}_k(n, i), \mathbf{y}_k(n, i); \mathbf{w}_{s,k}(n, i-1))$ to client k. Step 7: Client k performs the BP of the client-side model to obtain $\Delta\mathbf{w}_{c,k}(n, i)$. After completing I local iterations, each client transmits $\Delta\mathbf{w}_{c,k}(n, I)$ to the server (Step 8). Finally, the

server aggregates the $\Delta \mathbf{w}_{c,k}(n, I)$ from K clients to obtain the updated global client-side TPs $\Delta \mathbf{w}_c(n)$. Concurrently, the server aggregates the $\Delta \mathbf{w}_{s,k}(n, I)$ to obtain the updated global server-side TPs $\Delta \mathbf{w}_s(n)$, completing one training round.

B. SFL-PS Paradigm

We propose a training paradigm, SFL-PS, characterized by parallel server-side computing and sequential downlink GT, as illustrated in Table I and Fig. 1. Specifically, in steps 4 and 5, the server partitions its computing resources into dedicated portions for individual clients, with the allocation fixed in one training round[1]. In step 6, all downlink resources are used for the current GT, and the GTs are scheduled in a specific order. We assume that the uplink bandwidth W_U and the downlink bandwidth W_D do not overlap. At the start of each round, the server decides the following variables

- $u(n)$ is the split point for all clients.
- $\alpha_k(n)$ is the fraction of server computing frequency allocated to client k.
- $\beta_k(n)$ is the fraction of uplink bandwidth allocated to client k.

Vanilla SFL-PS employs intra-round synchronous training and undesignated GT scheduling. Specifically, during a local iteration, the server starts GT only after completing the server-side model FP and BP for all clients, and the scheduling order for downlink transmission is undesignated, for example, random or first-come-first-served (FCFS). These two characteristics may lead to long per-round training latency. To address these limitations, we propose CPSFL in Section III.

C. Fine-Grained Latency and Energy Consumption Analysis

In this section, we first analyze the achievable communication rate for each client, and then analyze the time and energy consumption of the eight steps in each training round.

To finely capture the UAV client mobility, we propose a fine-grained latency analysis where the channel varies per time slot instead of per training round. We denote the channel gain between the BS and UAV k in the s-th time slot as $h_k(s)$, which is a function of the distance between the BS and UAV k and assumed to be constant over a time slot with length τ_0. At the start of round n, the server allocates $W_k(n) = \beta_k(n) W_U$ bandwidth to client k for the uplink transmission. The uplink rate from client k to the server in round n is given by

$$R_{U,k}(n, h_k(s)) = W_k(n) \log_2(1 + p_k h_k(s)/(W_k(n) N_0)), \quad (1)$$

where p_k is the transmit power of client k and N_0 is the noise power spectral density (PSD). The downlink rate from the server to client k in round n is given by

$$R_{D,k}(s) = W_D \log_2(1 + P_S h_k(s) / (W_D N_0)), \quad (2)$$

where P_S is the total transmit power of the server.

[1]To enable parallel server-side computing, the server model storage should include K server-side FMs $\mathbf{w}_s$ for K clients, respectively.

Then, in round n, the average communication rate of a uplink transmission step for client k, starting from t_B and ending at t_E, can be expressed as

$$\begin{aligned}\overline{R}_{U,k}(n, t_B, t_E) = \Big(& R_{U,k}(n, h_k(s_B))((s_B + 1)\tau_0 - t_B) \\ & + \sum_{j=s_B+1}^{s_E-1} R_{U,k}(n, h_k(j))\tau_0 \\ & + R_{U,k}(n, h_k(s_E))(t_E - s_E\tau_0) \Big) / (t_E - t_B), \end{aligned} \quad (3)$$

where $s_B = \lfloor t_B/\tau_0 \rfloor$ and $s_E = \lfloor t_E/\tau_0 \rfloor$ are the time slots corresponding to t_B and t_E. If $s_B = s_E$, then $\overline{R}_{U,k}(n, t_B, t_E) = R_{U,k}(n, h_k(s_B))$. Similarly, by replacing $R_{U,k}(n, h_k(s))$ with $R_{D,k}(s)$ in (3), the average rate for the downlink transmission $\overline{R}_{D,k}(t_B, t_E)$ can be obtained.

1) Step 1: The client-side TPs broadcasting latency is $\tau_{SM}(n) = \max\{\tau_{SM,k}(n)\}$ and $\tau_{SM,k}(n)$ is given by

$$\tau_{SM,k}(n) = \Gamma_M(u(n)) / \overline{R}_{D,k}(t_B, t_B + \tau_{SM,k}(n)), \quad (4)$$

where $\Gamma_M(u)$ is the data size (in bits) of the client-side TPs when the split point is u and $t_B = t_{SM,k,B}(n)$ is the start time of this step. Note that $\tau_{SM,k}(n)$ is obtained by solving equation (4). The latencies of each communication step below are also obtained by solving the corresponding equations.

2) Step 2: The client-side model FP latency of client k is

$$\tau_{CF,k}(n, i) = B\Psi_{CF}(u(n)) / (\kappa_k f_k), \quad (5)$$

where $\Psi_{CF}(u)$ is the computation workload (in FLOPs) with one data sample when the split point is u, κ_k is the computing intensity (in FLOPs/cycle) of client k, and f_k is the computing frequency of client k. The corresponding energy consumption is $e_{F,k}(n, i) = \omega_k f_k^3 \tau_{CF,k}(n, i)$, where ω_k is the coefficient (in Watt/(cycle/s)^3) according to the chip architecture [23].

3) Step 3: The smashed data transmission latency of client k is

$$\tau_{CA,k}(n, i) = B\Gamma_A(u(n)) / \overline{R}_{U,k}(n, t_B, t_B + \tau_{CA,k}(n, i)), \quad (6)$$

where $\Gamma_A(u)$ is the data size (in bits) of the smashed data when the split point is u and $t_B = t_{CA,k,B}(n, i)$. The corresponding energy consumption is $e_{A,k}(n, i) = p_k \tau_{CA,k}(n, i)$.

4) Steps 4 and 5: The server-side FP and BP latency for client k is

$$\tau_{S,k}(n, i) = B(\Psi_{SF}(u(n)) + \Psi_{SB}(u(n))) / (\kappa_S \alpha_k(n) f_S), \quad (7)$$

where $\Psi_{SF}(u)$ and $\Psi_{SB}(u)$ denote the computational workloads (in FLOPs) per data sample for FP and BP, respectively, when the split point is u; κ_S is the server's computing intensity (in FLOPs/cycle); and f_S is the server's computing frequency.

5) Step 6: The gradient transmission latency for client k is

$$\tau_{SG,k}(n, i) = B\Gamma_G(u(n)) / \overline{R}_{D,k}(t_B, t_B + \tau_{SG,k}(n, i)), \quad (8)$$

where $\Gamma_G(u)$ is the data size (in bits) of the gradients of the smashed data when the split point is u and $t_B = t_{SG,k,B}(n, i)$.

6) Step 7: The client-side model BP latency of client k is

$$\tau_{CB,k}(n, i) = B\Psi_{CB}(u(n)) / (\kappa_k f_k), \quad (9)$$

where $\Psi_{CB}(u)$ is the computation workload (in FLOPs) with one data sample when the split point is u. The corresponding energy consumption is $e_{B,k}(n, i) = \omega_k f_k^3 \tau_{CB,k}(n, i)$.

7) Step 8: The client-side TPs uplink transmission latency of client k is

$$\tau_{CM,k}(n) = \Gamma_M(u(n)) / \overline{R}_{U,k}(n, t_B, t_B + \tau_{CM,k}(n)), \quad (10)$$

where $t_B = t_{CM,k,B}(n)$. The corresponding energy consumption is $e_{M,k}(n) = p_k \tau_{CM,k}(n)$.

8) Total energy consumption for one round: In round n, the total energy consumption of client k is

$$e_k(n) = \sum_{i=1}^{I} (e_{F,k}(n,i) + e_{A,k}(n,i) + e_{B,k}(n,i)) + e_{M,k}(n). \quad (11)$$

9) Lag of clients: We define the lag of client k as the time interval from the start of its BP in the previous local iteration to the completion of its server-side computation in the current iteration, as illustrated in Fig. 3. Specifically, the lag of client k in local iteration i of round n is defined as

$$\begin{aligned} l_k(n,i) = &\tau_{CB,k}(n, i-1) + \tau_{CF,k}(n,i) + \\ &\tau_{CA,k}(n,i) + \tau_{S,k}(n,i). \end{aligned} \quad (12)$$

10) The upper bound of the total latency for one round: The total training latency of round n is denoted by $\tau(n)$. Under the SFL-PS paradigm, the upper bound of $\tau(n)$, denoted as $\tau_{\max}(n)$, is achieved when the GT of the client with the highest lag in each iteration is performed at the end, i.e.,

$$\begin{aligned} \tau(n) \le &\tau_{\max}(n) = \max_k \{\tau_{SM,k}(n)\} + \\ &\max_k \{\tau_{CF,k}(n,1) + \tau_{CA,k}(n,1) + \tau_{S,k}(n,1)\} + \\ &\sum_{i=1}^{I-1} \left(\sum_{k=1}^{K} \tau_{SG,k}(n,i) + \max_k \{l_k(n,i+1)\} \right) + \\ &\sum_{k=1}^{K} \tau_{SG,k}(n,I) + \max_k \{\tau_{CB,k}(n,I) + \tau_{CM,k}(n)\}. \end{aligned} \quad (13)$$

D. Problem Formulation

In the SFL-PS-enabled UAV network, the client mobility induces time-varying wireless channels and achievable communication rates, which affect the latency and the energy consumption. Therefore, to minimize the training latency and energy consumption per round, adjusting the SPS and the CCRA when each round begins is necessary. To prevent UAVs from depleting their energy too quickly, we focus on the maximum energy consumption of UAVs. The optimization problem can be formulated as

$$\min_{\{\alpha_k(n), \beta_k(n), u(n)\}} \quad \tau(n) + \lambda \max_k \{e_k(n)\} \quad (14a)$$

$$\text{s.t.} \quad \alpha_{\min} \le \alpha_k(n) \le 1, \sum_{k=1}^{K} \alpha_k(n) = 1, \quad (14b)$$

$$\beta_{\min} \le \beta_k(n) \le 1, \sum_{k=1}^{K} \beta_k(n) = 1, \quad (14c)$$

$$u(n) \in \mathcal{U}, \quad (14d)$$

where $\lambda > 0$ is the weight of the energy term and $\mathcal{U}$ is the set of possible values of the split point. Besides, $\alpha_{\min} \in [0, 1/K]$ and $\beta_{\min} \in [0, 1/K]$ are the limits of the transmitting power fraction and the bandwidth fraction, respectively. To keep $\tau(n)$ away from its upper bound $\tau_{\max}(n)$ in (13), we propose CPSFL in the next section to enhance vanilla SFL-PS.

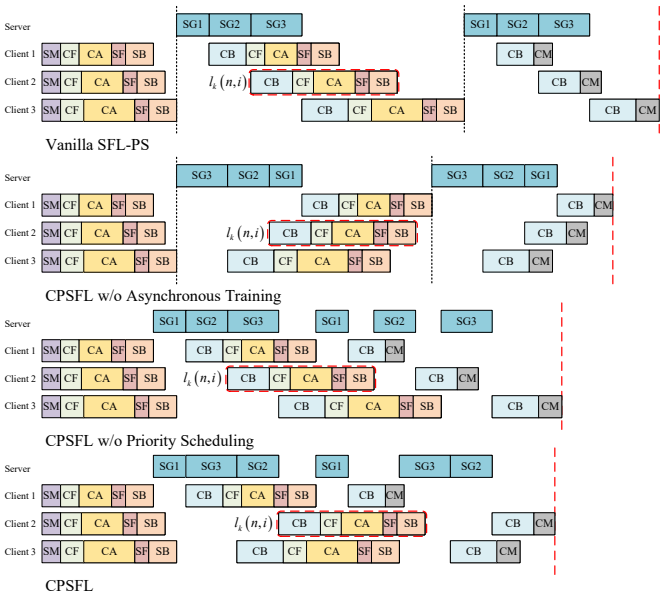


Fig. 3: The timeline of one training round (consisting of two local iterations) for three clients when the proposed CPSFL and its ablation variants are applied. The notations SM, CF, CA, SF, SB, SGk, CB, and CM correspond to the eight steps in Fig. 2, respectively.

III. Communication-Pipelined SFL

In this section, we propose the CPSFL, which enhances the vanilla SFL-PS through the downlink GT priority scheduling and intra-round asynchronous training. Specifically, the priority scheduling allows clients who are likely to become stragglers to receive the gradients required for BP earlier. The asynchronous training reduces the idle downlink waiting time by enabling immediate GT upon server computation completion. Fig. 3 illustrates the timelines of four paradigms: vanilla SFL-PS, CPSFL without the intra-round asynchronous training (CPSFL w/o AT), CPSFL without the downlink GT priority scheduling (CPSFL w/o PS), and CPSFL, showing that CPSFL achieves the shortest training latency for one round.

In the following, we first present the priority scheduling mechanism in CPSFL w/o AT and establish its optimality via theoretical analysis. Next, we introduce CPSFL and prove the scheduling optimality under additional simplifying assumptions. We then analyze the per-round latency of CPSFL and its ablation variants, and finally compare them against the other two paradigms: SFL-PP and PipeSFL.

A. Downlink Gradient Transmission Priority Scheduling

Due to client heterogeneity in channel conditions and computing capabilities, the order of downlink GT significantly impacts the per-iteration training latency. Under the synchronous training setting, there are theoretically $K!$ possible scheduling

Algorithm 1 CPSFL (Two Server-Side Improvements)

Procedure: Downlink Gradient Transmission Priority Scheduling
if the server completes the server-side model FP and BP for client k and obtains the gradient $\mathbf{G}_k(n,i)$ **then**
  The server adds $\mathbf{G}_k(n,i)$ to the priority queue with the lag $l_k(n,i)$ in (12) as its priority.
end if
Procedure: Intra-Round Asynchronous Training
if the server is not transmitting gradients and the priority queue is not empty **then**
  The server retrieves the gradient $\mathbf{G}_k(n,i)$ of the highest-priority client k from the priority queue.
  The server transmits $\mathbf{G}_k(n,i)$ to client k.
  The server removes $\mathbf{G}_k(n,i)$ from the priority queue.
end if

orders with K clients. Thus, finding the optimal orders is NP-hard, and exhaustive search is prohibitively time-consuming. To address this, we design a priority scheduling mechanism for downlink GT and prove its optimality. This paradigm is called CPSFL w/o AT.

As shown in Algorithm 1, upon completing the computing task for client k, the server obtain the gradient $\mathbf{G}_k(n,i)$ for client k and inserts it into the priority queue along with its lag $l_k(n,i)$ from (12), which serves as the transmission priority. Consequently, clients with larger lags are assigned higher transmission priority and can start BP earlier. When $I=0$, the term $\tau_{CB,k}(n,0)$ in (12) can be replaced by $\varsigma\tau_{CF,k}(n,1)$ with a constant $\varsigma>0$.

1) Optimality analysis: We first analyze the timeline when the server employs the priority scheduling, based on the following simplifying assumption.

Assumption 1. *The wireless channel remains constant within each training round. Thus, we can omit the round index n and local iteration index i for brevity. Without loss of generality, we reindex the K clients in non-decreasing order of their lags, i.e., $l_1 \le l_2 \le \cdots \le l_K$, so that their transmission priorities are client 1 $<$ client 2 $<\cdots<$ client K.*

For CPSFL w/o AT, we define the per-iteration training latency as the time interval between two consecutive instants when the server finishes the computing tasks of all clients, which is illustrated in Fig. 3 as the gap between two vertical black dashed lines. This latency is given by

$$T_{\text{iter}} = \max_k \left\{ \sum_{j=k}^{K} \tau_{SG,j} + l_k \right\}. \tag{15}$$

Then, the following theorem shows the optimality of the proposed GT priority scheduling.

Theorem 1. *The optimal strategy to minimize the per-iteration training latency T_{iter} is to prioritize the GT task of clients with larger lag, i.e., schedule the GT task of client m before client k if and only if $l_m \ge l_k$, $\forall m,k \in \mathcal{K}$.*

Proof. See Appendix A. □

B. Intra-Round Asynchronous Training

In CPSFL w/o AT, the server's downlink communication resources remain idle between the completion of the computing task of one client and the completion of the computing tasks of all clients. To improve resource utilization, the proposed CPSFL adopts the intra-round asynchronous training. As shown in Algorithm 1, whenever the server is not transmitting gradients, it immediately retrieves the highest-priority gradient from the priority queue, starts transmission, and then removes this gradient and its priority from the queue. This allows the server to start downlink GT without waiting for the computing tasks of all clients to complete.

Notably, the gradients in the priority queue may originate from different local iteration counts across clients. For example, client j may expect to receive a gradient of iteration 2, while client k expects one of iteration 3. In this study, the transmission priority of each gradient is independent of the number of local iterations completed by the corresponding client.

1) Optimality analysis: To simplify the per-round latency analysis, we adopt Assumption 1 and two other assumptions.

Assumption 2. *The uplink transmission latency of the client-side TPs, $\tau_{CM,k}$, is negligible.*

Assumption 2 approximately holds in practice for two reasons. First, LoRA and model splitting significantly reduce the number of client-side TPs, making the data size $\Gamma_M(u)$ in (10) much smaller than $B\Gamma_A(u)$ in (6), so that $\tau_{CM,k} \ll \tau_{CA,k}$. Second, when the number of local iterations I is large, the contribution of $\tau_{CM,k}$ to the total per-round latency becomes negligible.

Assumption 3. *The GT latencies for all clients are equal, i.e., $\tau_{SG,k} = \tau_{SG}$, $\forall k$ (based on Assumption 1).*

Assumption 3 holds when the downlink communication rates between the clients and the server are identical. Since the server uses all downlink resources (bandwidth and transmit power) for each GT, as shown in (2) and (8), this assumption is equivalent to assuming identical channel gains: $h_k = h$, $\forall k$. Moreover, Assumption 3 is approximately valid in practice: due to limited client uplink bandwidth and transmit power, $\tau_{SG,k}$ is much smaller than $\tau_{CA,k}$, and the differences among $\tau_{SG,k}$ across clients is relatively small.

Under Assumptions 1–3, we establish the following optimality theorem for the proposed scheduling policy.

Theorem 2. *The optimal strategy to minimize the per-round training latency of CPSFL is to prioritize the GT task of clients with larger lag, i.e., schedule the GT task of client m before client k if and only if $l_m \ge l_k$, $\forall m,k \in \mathcal{K}$.*

Proof. See Appendix B. □

C. Per-Round Training Latency Analysis

In the analysis, we omit the broadcast latency of the global client-side TPs $\tau_{SM}(n)$, as it is identical for all clients and paradigms. Besides, we adopt Assumptions 1 and 2 for brevity.

1) Per-round latency of the vanilla SFL-PS: We denote the per-round latency of the vanilla SFL-PS by τ_1. From (13), we have $\tau_1 \leq \tau_{\max}$. The minimum value of τ_1 is achieved when CPSFL w/o AT is applied, i.e., the GT follows the priority scheduling in Theorem 1. We denote the per-round latency of CPSFL w/o AT by τ_2, so $\tau_1 \geq \tau_2$.

2) Per-round latency of CPSFL w/o AT: The per-round latency of CPSFL w/o AT τ_2 is given by

$$\tau_2 = \max_k \left\{\tau_{CF,k} + \tau_{CA,k} + \tau_{S,k}\right\} + (I-1)\cdot \max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + l_k\right\} + \max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + \tau_{CB,k}\right\}. \tag{16}$$

Thus, $\tau_2 \geq I \max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + l_k\right\} = \widehat{\tau}_2$, where the equality holds when $\arg\max_k \left\{\tau_{CF,k} + \tau_{CA,k} + \tau_{S,k}\right\} = \arg\max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + \tau_{CB,k}\right\}$ or when I is large.

3) Per-round latency of CPSFL: Due to the intra-round asynchronous training, the per-round latency of CPSFL, denoted by τ_{CPSFL}, lacks a closed-form expression. Instead, we characterize its bounds. As described in Section III-B, under asynchronous training, a lower-priority client's GT may start earlier than a higher-priority one's, provided no higher-priority GT task is yet enqueued. When each client must wait for all higher-priority GTs to complete before its GT begins in every local iteration, τ_{CPSFL} achieves its upper bound, i.e.,

$$\tau_{\mathrm{CPSFL}} \leq \tau_2. \tag{17}$$

In summary, we have $\tau_{\mathrm{CPSFL}} \leq \tau_2 \leq \tau_1$.

The lower bound is achieved when the downlink GT fully overlaps all other latencies, except for the latency of client 1, the client with the smallest lag, at the start and end of each round.

$$\tau_{\mathrm{CPSFL}} \geq \tau_{CF,1} + \tau_{CA,1} + \tau_{S,1} + \sum_{i=1}^{I}\sum_{j=1}^{K} \tau_{SG,j} + \tau_{CB,1}. \tag{18}$$

D. Comparison of the Per-Round Training Latency of CPSFL, PipeSFL and SFL-PP

In this section, we compare the approximate upper bound of the per-round latency of CPSFL, $\widehat{\tau}_2$, with that of PipeSFL, $\widehat{\tau}_3$, and the per-round latency of SFL-PP, τ_{PP}, to characterize the conditions under which either CPSFL or PipeSFL outperforms the others.

Firstly, we redefine the latency notation for the relevant steps of SFL-PP and PipeSFL since they utilize the server's computing and communication resources differently from the proposed CPSFL, as shown in Table I. The downlink rate from the server to client k in round n is given by

$$R'_{D,k}(n, h_k(s)) = \beta_k(n) W_D \cdot \log_2\left(1 + \rho_k(n) P_S h_k(s) / \left(\beta_k(n) W_D N_0\right)\right), \tag{19}$$

where $\rho_k(n)$ is the fraction of the server transmit power allocated to the client k. Its constraint is given by

$$\rho_{\min} \leq \rho_k(n) \leq 1, \sum_{k=1}^{K} \rho_k(n) = 1. \tag{20}$$

Then, the average rate for the downlink GT $\overline{R}'_{D,k}(n, t_B, t_E)$ can be obtained by replacing $R_{U,k}(n, h_k(s))$ with $R'_{D,k}(n, h_k(s))$ in (3). The GT latency for client k $\tau'_{SG,k}(n, i)$ can be obtained by replacing $\overline{R}_{D,k}(n, t_B, t_E)$ with $\overline{R}'_{D,k}(n, t_B, t_E)$ in (8). To avoid cumbersome and difficult comparisons, we adopt Assumptions 1 and 2 in the following. Obviously, we have $\tau_{SG,k} < \tau'_{SG,k}$.

The per-round latency of SFL-PP can be expressed as

$$\tau_{\mathrm{PP}} = I \max_k \left\{\tau_{CF,k} + \tau_{CA,k} + \tau_{S,k} + \tau'_{SG,k} + \tau_{CB,k}\right\}. \tag{21}$$

In PipeSFL, the lag of client k is given by

$$l'_k = \tau_{CF,k} + \tau_{CA,k} + \tau'_{SG,k} + \tau_{CB,k}. \tag{22}$$

Then, we reindex the K clients by their lags, i.e., $l'_1 \leq l'_2 \leq \cdots \leq l'_K$[2]. Moreover, the server-side computing latency of client k is denoted as $\tau'_{S,k}$, which is identical for all clients since the server allocates all its computing resources to the current task and all clients share the same split point. For brevity, we denote it as τ'_S. Obviously, we have $\tau'_S < \tau_{S,k}$. Similarly to τ_2 in (16) and (17), the upper bound of per-round latency of PipeSFL can be expressed as

$$\tau_3 = \max_k \left\{\tau_{CF,k} + \tau_{CA,k}\right\} + (I-1) \max_k \left\{(K-k+1)\tau'_S + l'_k\right\} + \max_k \left\{(K-k+1)\tau'_S + \tau'_{SG,k} + \tau_{CB,k}\right\}. \tag{23}$$

Thus, we have $\tau_3 \geq I \max_k \left\{\sum_{j=k}^{K} \tau'_S + l'_k\right\} = \widehat{\tau}_3$, where the equality holds when $\arg\max_k \left\{\tau_{CF,k} + \tau_{CA,k}\right\} = \arg\max_k \left\{\sum_{j=k}^{K} \tau'_S + \tau'_{SG,k} + \tau_{CB,k}\right\}$ or when I is large.

Next, we compare the latencies under two extreme cases to derive intuitive insights.

1) Case 1: negligible computing latency: When computing latency is negligible, i.e., $\tau_{CF,k}, \tau_{S,k}, \tau_{CB,k}, \tau'_S \to 0$, we compare the latency of paradigms with parallel GT, τ_{PP} and $\widehat{\tau}_3$, against that of the paradigm with sequential GT, $\widehat{\tau}_2$. In this case, $\widehat{\tau}_2 = I \max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + \tau_{CA,k}\right\}$ and $\tau_{\mathrm{PP}} = \widehat{\tau}_3 = I \max_k \left\{\tau_{CA,k} + \tau'_{SG,k}\right\}$. Thus, if

$$\max_k \left\{\sum_{j=k}^{K} \tau_{SG,j} + \tau_{CA,k}\right\} \leq \max_k \left\{\tau'_{SG,k} + \tau_{CA,k}\right\}, \tag{24}$$

then $\widehat{\tau}_2 \leq \widehat{\tau}_3 = \tau_{\mathrm{PP}}$ and the proposed CPSFL is likely to achieve a lower per-round latency.

Next, we present a scenario under which (24) holds. Firstly, (24) is equivalent to

$$\sum_{j=k}^{K} \tau_{SG,j} + \tau_{CA,k} \leq \max_k \left\{\tau'_{SG,k} + \tau_{CA,k}\right\}, \forall k. \tag{25}$$

If the downlink communication resources are equally allocated among clients, i.e., $\beta_k = \rho_k = 1/K$, $\forall k$ in (19), then $K\tau_{SG,k} = \tau'_{SG,k}$, $\forall k$. Thus, $K\tau_{SG,k} + \tau_{CA,k} \leq \max_k \left\{K\tau_{SG,k} + \tau_{CA,k}\right\} = \max_k \left\{\tau'_{SG,k} + \tau_{CA,k}\right\}$, $\forall k$.

[2]Note that the index k in τ_2 and that in τ_3 may refer to different clients, since k in τ_2 is indexed according to l_k in (12), whereas k in τ_3 is indexed according to l'_k.

Moreover, if the GT latency ordering is opposite to the lag ordering, i.e., $\tau_{SG,k} \geq \tau_{SG,k+1}, \forall k \in [1, K-1]$ (a relaxed version of Assumption 3), we have $\sum_{j=k}^{K} \tau_{SG,j} \leq (K-k+1)\,\tau_{SG,k} \leq K\tau_{SG,k}$, $\forall k$. Under these two conditions, (25) and (24) hold.

2) Case 2: negligible communication latency: When communication latency is negligible, i.e., $\tau_{CA,k}, \tau_{SG,k}, \tau'_{SG,k} \to 0$, we compare the latency of paradigms with parallel server computing, τ_{PP} and $\widehat{\tau}_2$, against that of the paradigm with sequential server computing, $\widehat{\tau}_3$. In this case, $\widehat{\tau}_3 = I \max_k \left\{ \sum_{j=k}^{K} \tau'_S + \tau_{CF,k} + \tau_{CB,k} \right\}$ and $\tau_{\mathrm{PP}} = \widehat{\tau}_2 = I \max_k \left\{ \tau_{CF,k} + \tau_{S,k} + \tau_{CB,k} \right\}$. Thus, if

$$\max_k \left\{ (K-k+1)\,\tau'_S + \tau_{CF,k} + \tau_{CB,k} \right\} \leq \max_k \left\{ \tau_{S,k} + \tau_{CF,k} + \tau_{CB,k} \right\}, \tag{26}$$

then $\widehat{\tau}_3 \leq \widehat{\tau}_2 = \tau_{\mathrm{PP}}$ and the PipeSFL is likely to achieve a lower per-round latency.

Next, we present a scenario under which (26) holds. Firstly, (26) is equivalent to $(K-k+1)\,\tau'_S + \tau_{CF,k} + \tau_{CB,k} \leq \max_k \left\{ \tau_{S,k} + \tau_{CF,k} + \tau_{CB,k} \right\}$, $\forall k$. If the server computing frequency is equally allocated, i.e., $\alpha_k = 1/K$, $\forall k$ in $\tau_{S,k}$ in (7), then $K\tau'_S = \tau_{S,k}$, $\forall k$. Thus, $K\tau'_S + \tau_{CF,k} + \tau_{CB,k} \leq \max_k \left\{ K\tau'_S + \tau_{CF,k} + \tau_{CB,k} \right\} = \max_k \left\{ \tau_{S,k} + \tau_{CF,k} + \tau_{CB,k} \right\}$, $\forall k$. Then, since $(K-k+1)\,\tau'_S \leq K\tau'_S$, $\forall k$, (26) holds.

IV. DRL-based SPS and CCRA for the CPSFL-enabled UAV networks

In this section, we first motivate the use of DRL for decision-making. We then formulate the problem as a partially observable Markov decision process (POMDP). Next, we describe the design of the BS agent, which incorporates an attention-based mechanism to extract features from historical UAV trajectories. Finally, we present the DRL-based SPS and CCRA scheme for the CPSFL-enabled UAV networks.

A. Motivation of DRL-based Solution

Problem (14) presents three major challenges that render conventional optimization methods impractical:

1) Complexity: It is a mixed-integer non-linear program (MINLP) with both discrete and continuous variables, making it non-convex and NP-hard [24].
2) Analytical intractability: As discussed in Section III-C3, due to intra-round asynchrony, the per-round latency $\tau_{\mathrm{CPSFL}}(n)$ lacks a closed-form expression in terms of the decision variables and channel strengths.
3) Uncertainty: At the start of each round, the BS has no access to future UAV trajectories or channel realizations.

Furthermore, the problem exhibits temporal correlation: the UAVs' initial locations in the next round depend on the current round's decisions, creating a sequential decision-making process. These motivate the use of DRL, a technique naturally suited to optimizing policies in complex, uncertain, and temporally correlated environments. Due to the environmental uncertainty, the agent only has partial observability. Thus, we formulate the problem as a POMDP.

B. POMDP Formulation and BS Agent Design

A POMDP model can be described with a tuple $\langle \mathcal{S}, \Omega, \mathcal{A}, \mathcal{P}, r, \mathcal{B}, \gamma \rangle$. Specifically, $\mathcal{S}$ is the state space, $\mathbf{o} \in \Omega$ is the observation, $\mathbf{a} \in \mathcal{A}$ is the action, $P(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \in \mathcal{P}$ is the probabilistic transition function, r is the reward function, and $\gamma \in (0,1)$ is the discount factor. Denote the policy for agent as $\pi : \Omega \times \mathcal{A} \to [0,1]$. The expected discounted cumulative reward for agent is $J(\pi) = \mathbb{E}_{\mathbf{s}(0),\mathbf{a}(0),\ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(t)\right]$, where $\mathbf{a}(t) \sim \pi(\cdot|\mathbf{o}(t))$ and $\mathbf{s}(t+1) \sim P(\cdot|\mathbf{s}(t), \mathbf{a}(t))$. The POMDP aims to find an optimal policy π^* that maximizes $J(\pi)$, i.e.,

$$\pi^* = \mathrm{argmax}_{\pi} J(\pi). \tag{27}$$

To find π^*, the DRL algorithms can be applied.

We designate the BS as the agent for deciding the variables listed in Section II-B. Its observation space, action space, and reward function are detailed as follows.

1) Attention-based UAV trajectory feature extraction: To make informed decisions, the BS agent needs to infer future UAV trajectory features and analyze how past trajectories affect the latency and energy consumption. Thus, the agent leverages historical trajectory data from the previous training round. However, the number of time slots per round varies, posing a challenge for fixed-dimension neural networks [38]. To address this, we employ an attention mechanism with positional encoding to extract fixed-dimensional representations.

Given a query $\mathbf{q} \in \mathbb{R}^{D_Q \times 1}$ and M key-value pairs $\mathbf{K} = [\mathbf{k}_1, \ldots, \mathbf{k}_M] \in \mathbb{R}^{D_K \times M}$, $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_M] \in \mathbb{R}^{D_V \times M}$, the attention mechanism produces a feature vector $\mathbf{f}_k \in \mathbb{R}^{H \times 1}$. Specifically, $\mathbf{q}$, $\mathbf{K}$, and $\mathbf{V}$ are first projected via the learnable weights $\mathbf{W}_Q \in \mathbb{R}^{D_S \times D_Q}$, $\mathbf{W}_K \in \mathbb{R}^{D_S \times D_K}$, and $\mathbf{W}_V \in \mathbb{R}^{H \times D_V}$, respectively. To preserve the temporal order, sinusoidal positional encodings are added to the projected keys and values, as well as to the query at its corresponding time step. The output feature is then obtained via the scaled dot-product attention [39]

$$\mathbf{f} = (\mathbf{W}_V \mathbf{V} + \mathbf{P}_V) \cdot \mathrm{softmax}\left(\frac{1}{\sqrt{D_S}} (\mathbf{W}_K \mathbf{K} + \mathbf{P}_K)^{\top} (\mathbf{W}_Q \mathbf{q} + \mathbf{p}_Q) \right), \tag{28}$$

where $\mathbf{P}_V \in \mathbb{R}^{H \times M}$, $\mathbf{P}_K \in \mathbb{R}^{D_S \times M}$, and $\mathbf{p}_Q \in \mathbb{R}^{D_S \times 1}$ are the positional encodings.

For UAV k at time slot s, we form a vector $\boldsymbol{\chi}_k(s) \in \mathbb{R}^{4 \times 1}$ by combining its 3D coordinates and distance to the BS, which is available to the BS at the start of each slot. Let $\mathcal{S}(n)$ denote the set of time slots in round n, with $M(n) = |\mathcal{S}(n)|$ varying across rounds. The query is set to $\mathbf{q} = \boldsymbol{\chi}_k(s')$, where s' is the last time slot in $\mathcal{S}(n-1)$. The key-value pairs are defined as $\mathbf{k}_m = \mathbf{v}_m = \boldsymbol{\chi}_k(s)$, $\forall s \in \mathcal{S}(n-1)$. Applying (28) yields a fixed-dimensional trajectory feature $\mathbf{f}_k(n-1) \in \mathbb{R}^{H \times 1}$ for UAV k over round $n-1$.

2) Observation space: The observation of the BS is designed as

$$\mathbf{o}(n) = \big[u(n-1), [\alpha_k(n-1), \beta_k(n-1), e_k(n-1)]_{k \in \mathcal{K}}, \tau(n-1), [\boldsymbol{\chi}_k(s)]_{k \in \mathcal{K}, s \in \mathcal{S}(n-1)} \big]. \tag{29}$$

To obtain $\mathbf{o}(n)$, the communication overhead is K positive real numbers (PRNs) (i.e., $e_k(n-1)$, $\forall k$), which is negligible

Algorithm 2 DRL-based SPS and CCRA scheme for the CPSFL-enabled UAV networks

Initialize the whole model $\mathbf{w}$ with the LoRA modules and the batch size B.
The BS initializes the policy network $\pi_{\boldsymbol{\vartheta}}$, the value network $V_{\boldsymbol{\varphi}}$, and the trajectory collector ξ of size B_m.
for round $n = 0, \ldots, N$ **do**
  If $n \geq 1$, the BS obtains observation $\mathbf{o}(n)$.
  If $n \geq 2$, the BS stores the experience $\langle \mathbf{o}(n-1), \mathbf{a}(n-1), r(n-1), \mathbf{o}(n) \rangle$ into ξ.
  if ξ is full **then**
    The BS obtains all experiences $\langle \mathbf{o}_j, \mathbf{a}_j, r_j, \mathbf{o}_{j+1} \rangle$, $j = 1, \ldots, B_m$ in ξ and updates $\pi_{\boldsymbol{\vartheta}}$ and $V_{\boldsymbol{\varphi}}$ by the PPO algorithm, and then clear ξ.
  end if
  The BS obtains the action $\mathbf{a}(n)$, decides the split point $u(n)$, the server computing frequency allocation $\alpha_k(n)$, $\forall k$, and the uplink bandwidth allocation $\beta_k(n)$, $\forall k$.
  The BS splits $\mathbf{w}(n-1)$ into $\mathbf{w}_c(n-1)$ and $\mathbf{w}_s(n-1)$, and sends $\Delta\mathbf{w}_c(n-1)$, $u(n)$, and $\beta_k(n)$ to UAV k, $\forall k$.
  The UAVs and BS perform I local iterations by the CPSFL paradigm in Algorithm 1.
  After receiving $\Delta\mathbf{w}_{c,k}(n, I)$, $\forall k$, the BS calculates their weighted average to obtain $\Delta\mathbf{w}_c(n)$. Concurrently, the BS aggregates the $\Delta\mathbf{w}_{s,k}(n, I)$, $\forall k$ to obtain $\Delta\mathbf{w}_s(n)$.
  UAV k send $e_k(n)$ to the BS, $\forall k$.
  The BS receives reward $r(n)$.
end for
return Trained whole TPs consisting of $\Delta\mathbf{w}_c$ and $\Delta\mathbf{w}_s$.

compared with Γ_M and Γ_A. After obtaining $\mathbf{f}_k(n-1)$, $\forall k$, we concatenate it with the other components of $\mathbf{o}(n)$ as input to the subsequent multi-layer perception (MLP).

3) Action space: The action space of the BS is designed as

$$\mathbf{a}(n) = \left[u(n), \widetilde{\boldsymbol{\alpha}}(n), \widetilde{\boldsymbol{\beta}}(n) \right], \tag{30}$$

where $\widetilde{\boldsymbol{\alpha}} = [\widetilde{\alpha}_1, \cdots, \widetilde{\alpha}_K]$, $\sum_{k=1}^{K} \tilde{\alpha}_k(n) = 1$, and $\tilde{\alpha}_k(n) > 0$. $\widetilde{\boldsymbol{\beta}}$ is defined analogously. To satisfy (14b), we obtain $\alpha_k(n)$ by the following linear scaling

$$\boldsymbol{\alpha}(n) = \begin{cases} \tilde{\boldsymbol{\alpha}}(n), \min\left(\tilde{\boldsymbol{\alpha}}(n)\right) \geq \alpha_{\min}, \\ A\left(\tilde{\boldsymbol{\alpha}}(n) - \min\left(\tilde{\boldsymbol{\alpha}}(n)\right)\right) + \alpha_{\min}, \text{else}, \end{cases} \tag{31}$$

where $A = (1 - K\alpha_{\min}) / \left(1 - K \min\left(\widetilde{\boldsymbol{\alpha}}(n)\right)\right) \geq 0$. Besides, $\beta_k(n)$ can be obtained in a similar way to satisfy (14c).

4) Reward function: The reward function of the BS is designed as

$$r(n) = -\tau(n) - \lambda \max_k \left\{ e_k(n) \right\}. \tag{32}$$

C. The Overall DRL Scheme

The proposed DRL-based SPS and CCRA scheme for the CPSFL-enabled UAV networks is summarized in Algorithm 2. The BS agent is equipped with the proximal policy optimization (PPO) algorithm [40].

V. Simulation Results

In this section, we present the simulation results to demonstrate the performance improvements brought by the proposed CPSFL in Algorithm 1 for the UAV network and the proposed DRL-based SPS and CCRA scheme in Algorithm 2.

TABLE II: Amount of the FP computation, the client-side TPs, and the smashed data of Swin-L with LoRA

u	Ψ_{CF} (GFLOPs)[1]	Ψ_{SF} (GFLOPs)[1]	Γ_M (KB)[2]	Γ_A (KB)[2]	Smashed Data Shape
1	6.18	64.10	192	2352	[56, 56, 192]
2	12.50	57.78	612	1176	[28, 28, 384]
3	64.19	6.09	7596	588	[14, 14, 768]
4	70.28	0.001	9276	294	[7, 7, 1536]

[1] It is calculated by fvcore with 1Mult-Adds $\approx$ 2FLOPs.
[2] It is measured in float32 precision.

A. Benchmarks

We compare CPSFL against the following SFL paradigms and ablation variants:

- CPSFL w/o AT: CPSFL with intra-round synchronous training (see Fig. 3): the server starts GT only after completing server-side computing for all clients.
- CPSFL w/o PS: CPSFL with FCFS scheduling instead of priority scheduling.
- PipeSFL [37]: A state-of-the-art SFL-SP variant featuring server-side computing priority scheduling and intra-round asynchronous training[3].
- PipeSFL w/o AT: PipeSFL with intra-round synchronous training: the server starts computing only after receiving smashed data from all clients.
- PipeSFL w/o PS: PipeSFL with FCFS scheduling instead of priority scheduling.
- SFL-PP: All SFL steps of different clients is executed in parallel (see Fig. 1).

The latency of SFL-PP can be readily extended from (21). In PipeSFL, upon receiving the smashed data, the server calculates the client lag using an extension of (22), where we set $\tau_{CB,k}(n, 0) = \varsigma \tau_{CF,k}(n, 1)$ and estimate the GT latency by utilizing the current-slot channel information, i.e., $\tau'_{SG,k}(n, 0) = B\Gamma_G(u(n)) / R'_{D,k}(n, h_k(s_{CA,k,E}(n, i)))$, with $s_{CA,k,E}(n, i) = \lfloor (t_{CA,k,B}(n, i) + \tau_{CA,k}(n, i)) / \tau_0 \rfloor$[4].

B. Simulation Parameters Setting

We consider a UAV network with a BS and $K = 10$ UAVs. The BS is located at [0,0,30]m. The horizontal locations of the UAVs are within three concentric annular regions centered on the BS. UAVs 1-3, 4-6, and 7-10 are located in the inner, middle, and outer rings, respectively. The boundaries of these regions are 100m, 550m, 820m, and 1000m away from the BS. The UAVs' height is 20m, and their speed is maintained between 0.1m/s and 4m/s. The time slot length is set to $\tau_0 = 0.1$s. The center frequency is $f_c = 2$GHz. The uplink bandwidth and the downlink bandwidth are $W_U = W_D = 20$MHz. The noise PSD is set to $N_0 = -114$dBm/MHz. The path loss $L_{\mathrm{RMa},k}$ in dB follows the RMa-AV LOS model in 3GPP TR 36.777 [41]. The channel gain is $h_k(s) = 10^{-L_{\mathrm{RMa},k}(s)/10}$.

The BS and UAVs collaborate to fine-tune a Swin-L model ($\approx$200M parameters)[5] [42] on the CIFAR-100 dataset, where

[3] For fair comparison, we adopt PipeSFL-V2, where each client's server-side FP is immediately followed by its BP.

[4] In [37], these terms are set to zero, which may result in priority assignments that deviate from the optimal scheduling in Theorem 2 of [37] for minimizing the per-round training latency.

[5] CPSFL is also applicable to billion-scale models, e.g., SwinV2-G (3.0B).

the input size is 3*224*224 and $B = 8$. The LoRA modules of rank 8 are injected into all linear layers except the classification head. All parameters except the LoRA modules and the classification head are frozen. Split point u allocates u stages to the client and the remainder to the server. Thus, $\mathcal{U} = \{1, \ldots, 4\}$ in (14d). The values of $\Psi_{CF}(u)$, $\Psi_{SF}(u)$, $\Gamma_M(u)$, and $\Gamma_A(u)$ are shown in Table II. Besides, we set $\Gamma_G(u) = \Gamma_A(u)$, $\Psi_{SB}(u) = \varsigma\Psi_{SF}(u)$, $\Psi_{CB}(u) = \varsigma\Psi_{CF}(u)$, and $\varsigma = 2$ since the computations required for BP are about twice that of FP [31]. The BS is equipped with a GeForce RTX 4080 for server-side computing, which provides a computing capacity of 780 AI TOPS (≈195TFLOPS) and operates at a frequency of $f_S = 2.51$GHz [43]. Accordingly, we set its computational intensity to $\kappa_S = 195 \times 10^{12}/f_S$ FLOPs/cycle. In addition, we set $P_S = 40$W and $\alpha_{\min} = \beta_{\min} = 1/K/5$ in (14b) and (14c). The UAVs have heterogeneous transmit powers, with p_k for UAV 1-10 set to 1, 0.4, 0.1, 1, 0.4, 0.1, 1, 0.4, 0.2, and 0.1 W, respectively. Each UAV is equipped with a device whose performance is comparable to the Jetson Xavier NX for client-side computation, offering a computing capacity of 10TOPS (≈2.5TFLOPS) and operating at a frequency of $f_k = 1$GHz [44]. Accordingly, we set its computational density to $\kappa_k = 2.5 \times 10^{12}/f_k$ FLOPs/cycle and set the energy consumption coefficient to $\omega_k = 16\text{W/(GHz)}^3$. The weight of the energy term is set as $\lambda = 4$.

For the BS agent, we set $D_S = 8$ and $H = 16$ in (28). Both π_{ϑ} and V_{φ} have three hidden layers with 128, 64, and 32 neurons, respectively. Besides, the third hidden layer and the output layer of π_{ϑ} include three branches for deciding $\alpha_k(n)$, $\beta_k(n)$, and $u(n)$, respectively. The softmax activation function is used in the outputs of the branches for $u(n)$ to normalize the probabilities. The parameters of Algorithm 2 are set as follows: discount factor is $\gamma = 0.5$, learning rates are $\alpha_V = \alpha_\pi = 3 \times 10^{-4}$, and we set $B_m = 12$.

C. Performance Evaluation with Fixed Variables

In this section, we perform comparisons under the following fixed resource allocation settings. For CPSFL, we set $\alpha_k = \beta_k = 1/K$, $\forall k$. For PipeSFL, we set $\rho_k = \beta_k = 1/K$, $\forall k$. For CPSFL, we set $\alpha_k = \beta_k = \rho_k = 1/K$, $\forall k$. The split point is set as $u = 2$ by default. The number of local iteration is set as $I = 3$ by default.

Fig. 4 compares the timelines of SFL-PP, PipeSFL, and CPSFL, intuitively illustrating the source of CPSFL's performance gain. In our simulation setting, the per-round training latency is dominated by the wireless communication, while the client and server computing latencies are relatively small. Consequently, although PipeSFL dedicates all server computing resources on the current task and incorporates priority scheduling and asynchronous training, its performance improvement is limited. In contrast, CPSFL, an advanced SFL-PS variant, focuses all downlink communication resources on the current GT, significantly reducing the GT latency for each client. Moreover, CPSFL integrates priority scheduling and asynchronous training, further lowering overall latency.

Fig. 5 shows the average per-round latency $\overline{\tau}(n)$ over 500 training rounds for different split points u. First, increasing u leads to a decrease in $\overline{\tau}(n)$ across all schemes since it reduces $\Gamma_A(u)$ (see Table II), thereby decreasing $\tau_{CA,k}$, $\tau_{SG,k}$, and $\tau'_{SG,k}$. Moreover, CPSFL achieves the lowest latency. When $u = 2$, it reduces $\overline{\tau}(n)$ by over 30% compared to PipeSFL. This gain is more pronounced for smaller u since in this case, the communication latency constitutes a larger fraction of $\tau(n)$, and the benefit of time-division sequential GT becomes significant.

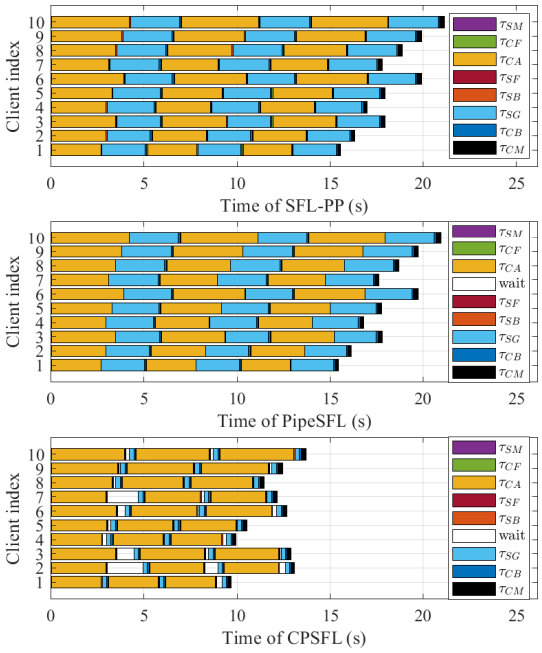


Fig. 4: Timeline of one training round (consisting of two local iterations) for ten clients when SFL-PP (top), PipeSFL (middle), or CPSFL (bottom) is applied.

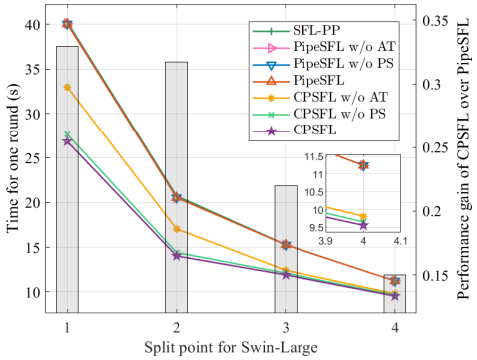


Fig. 5: Average per-round latency with different split points u obtained by CPSFL and the benchmarks.

In Table III, we define five clusters of UAVs with approximately increasing heterogeneity in their uplink communication latencies $\tau_{CA,k}$, achieved by changing their transmit powers p_k while keeping their average uplink communication rates $R_{U,k}$ nearly the same. In cluster 1, $\tau_{CA,k}$ is nearly identical across clients, whereas in cluster 5, the last two UAVs exhibit $\tau_{CA,k}$ values nearly 1.5 times larger than those of the others. Fig. 6 shows the average per-round latency over 500 training rounds for these clusters. CPSFL achieves the lowest latency, and its performance gain over CPSFL w/o PS and PipeSFL basically grows with increasing latency heterogeneity. This is

TABLE III: Transmit power of five UAV clusters (K=12)

Clusters	p_k (W) (inner ring)	p_k (W) (middle ring)	p_k (W) (outer ring)
1	0.2, 0.2, 0.2, 0.2	0.4, 0.4, 0.4, 0.4	0.8, 0.8, 0.8, 0.8
2	0.4, 0.4, 0.4, 0.4	0.8, 0.8, 0.2, 0.2	0.4, 0.4, 0.4, 0.4
3	0.4, 0.4, 0.4, 0.4	0.8, 0.8, 0.8, 0.8	1.6, 0.2, 0.2, 0.2
4	0.8, 0.8, 0.8, 0.4	0.8, 0.8, 0.2, 0.2	0.4, 0.2, 0.2, 0.2
5	0.8, 0.8, 0.8, 0.4	0.8, 0.8, 0.4, 0.4	0.8, 0.1, 0.1, 0.1

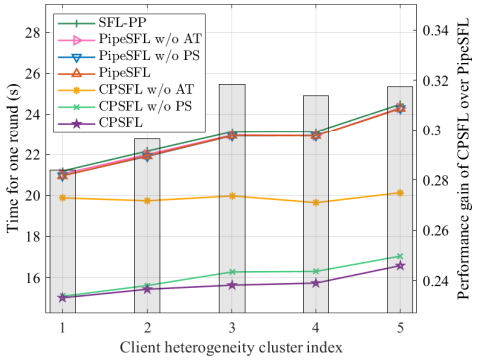


Fig. 6: Average per-round latency with different degrees of client heterogeneity obtained by CPSFL and the benchmarks.

because CPSFL effectively hides the impact of clients with large lag l_k in the GT latencies $\tau_{SG,k}$ via priority scheduling and intra-round asynchronous training. In contrast, PipeSFL struggles to hide the latency of clients with large lag l'_k in the server computing latencies τ'_S since τ'_S is small in our wireless UAV setting.

Fig. 7 and Fig. 8 show the average per-round latency $\overline{\tau}(n)$ over 500 training rounds with different numbers of local iterations I and clients K, respectively. For Fig. 8, there are $K/3$ clients in each of the three rings, with transmit powers p_k set to 0.9W (inner), 0.3W (middle), and 0.1W (outer), respectively. In both figures, CPSFL achieves the lowest latency across all settings, with the performance gain over PipeSFL slightly increases with larger I or K. Specifically, in Fig. 7, $\overline{\tau}(n)$ grows approximately linearly with I. In Fig. 8, increasing K raises $\overline{\tau}(n)$ since the total resource, including the bandwidth and server computing frequency, is fixed.

D. Performance Evaluation with Optimized Variables

In this section, we compare the DRL-based scheme in Algorithm 2 and the fixed variants. The schemes are appended with the suffixes to indicate how the variables are determined.

- DRL (proposed): The SPS and CCRA are decided by Algorithm 2. Besides, DRL(-) denotes a variant that replaces $[\boldsymbol{\chi}_k(s)]_{s\in\mathcal{S}(n-1)}$, $\forall k$ in (29) with the distances from the BS to all UAVs at the last time slot in $\mathcal{S}(n-1)$. Thus, DRL(-) does not involve the attention mechanism.
- fixed RA: The SPS is decided by Algorithm 2, while the CCRA is fixed at $\alpha_k = \beta_k = 1/K$, $\forall k$.
- $u = \widetilde{u}$: The CCRA is decided by Algorithm 2, while u is fixed at $\widetilde{u}$.

Fig. 9 shows the per-round latency $\tau(n)$ and the maximum energy consumption of all UAVs for one round $\max_k\{e_k(n)\}$ obtained by the DRL-based CPSFL scheme and the benchmarks when $I = 5$. Each value is a moving average with a span of 21 rounds. The proposed scheme converges faster

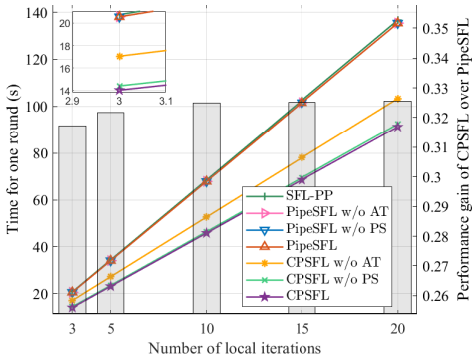


Fig. 7: Average per-round latency with different numbers of local iterations I obtained by CPSFL and the benchmarks.

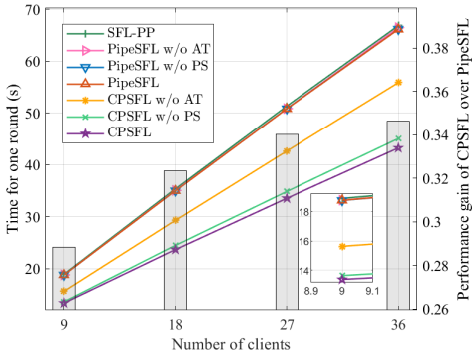
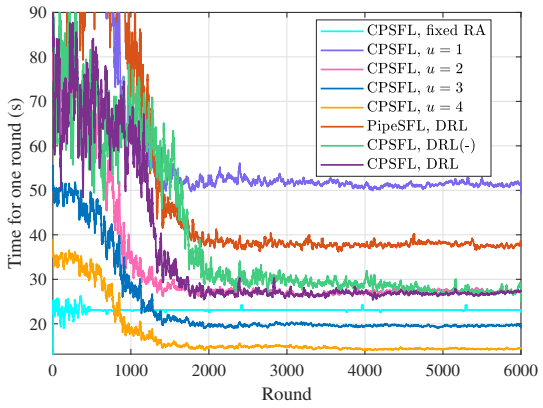


Fig. 8: Average per-round latency with different numbers of clients K obtained by CPSFL and the benchmarks.

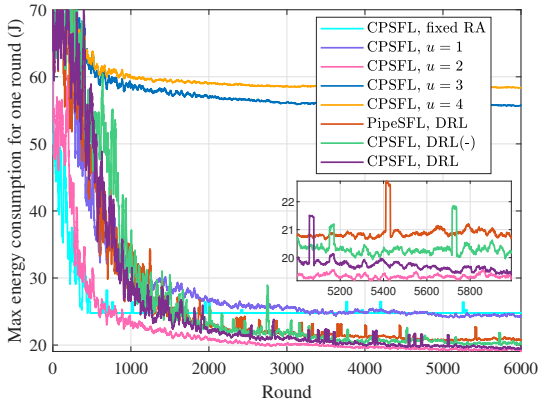
than the DRL(-) scheme, attaining an average objective value (see (14a)) of 105.8 over the last 1000 rounds, compared to 108.6 for the DRL(-) scheme. This improvement stems from its ability to reduce environmental uncertainty by learning the relationship among UAV trajectories, actions, and rewards. Moreover, the proposed DRL scheme can effectively handles the challenging joint optimization of SPS and CCRA within CPSFL. For comparison, the fixed CCRA scheme yields an average objective value of 122.3, confirming that the proposed scheme achieves a superior latency-energy trade-off by adapting to heterogeneous UAV transmit powers and time-varying channel conditions induced by mobility. Furthermore, the performance of the proposed scheme approaches that of the CPSFL scheme with $u = 2$, which is the best fixed split point scheme evaluated by the objective values in (14a). This shows that the BS agent in the proposed scheme can automatically select a high-performance split point. Notably, when both CPSFL and PipeSFL employ the DRL for decision-making, CPSFL consistently achieves lower latency and energy consumption, further validating the benefits of sequential GT and CPSFL's two key enhancements.

VI. Conclusions

In this paper, we have proposed a sequential GT paradigm, where the server dedicates all downlink resources for the current GT. We have further proposed CPSFL, characterized by downlink GT priority scheduling and intra-round asynchronous training. We have investigated CPSFL-based LoRA fine-tuning of FMs in UAV networks and have formulated



(a) Latency for one round.



(b) Maximum energy consumption of all UAVs.

Fig. 9: Latency and the maximum energy consumption of all UAVs for one round obtained by the DRL-based CPSFL scheme and the benchmarks: $I = 5$.

an optimization problem to minimize the per-round training latency and the worst-case client energy consumption by optimizing the SPS, the uplink bandwidth allocation, and the server computing frequency allocation. To solve this problem, we have developed an attention-based DRL framework, where the BS agent decides the split point and the CCRA in each round by leveraging previous round information including UAV trajectories. Simulation results have shown that the proposed DRL-based CPSFL scheme outperforms the benchmarks, the ablation variants, the fixed CCRA scheme, and approaches the best fixed-SPS scheme. This study lays the foundation for finer-grained pipelining paradigms in SFL.

Appendix A
Proof of Theorem 1

With Assumption 1, the lags satisfy $l_1 \leq l_2 \leq \cdots \leq l_K$. The proposed priority scheduling strictly follows this lag ordering. We suppose that there exist indices $m, k \in \mathcal{K}$ with $m > k$ (so $l_m \geq l_k$) such that swapping their priorities, i.e., assigning client m a lower priority than client k, reduces the per-iteration training latency in (15). This latency is rewritten as

$$T_{\text{iter}} = \max\Bigg\{ \max_{j\in[1,k-1]\cup[m+1,K]} \{t_j + l_j\}, t_m + l_m, \max_{q\in[k+1,m-1]} \{t_q + l_q\}, t_k + l_k \Bigg\}. \tag{33}$$

For any $q \in [k+1, m-1]$, we have $l_m \geq l_q \geq l_k$. We denote $t_k = \sum_{j=k}^{K} \tau_{SG,j}$, thus we have $t_k > t_q > t_m$. We use the superscript * to denote all intervals and timestamps after the swap. After swapping, we have $t^*_m > t^*_q > t^*_k$, $t^*_m = t_k$, and $t^*_j = t_j$, $\forall j \in [1, k-1] \cup [m+1, K]$. Besides, the per-iteration training latency becomes

$$T^*_{\text{iter}} = \max\Bigg\{ \max_{j\in[1,k-1]\cup[m+1,K]} \{t^*_j + l_j\}, t^*_m + l_m, \max_{q\in[k+1,m-1]} \{t^*_q + l_q\}, t^*_k + l_k \Bigg\}. \tag{34}$$

Since $t^*_m = t_k$ and $l_m \geq l_k$, we have $t^*_m + l_m \geq t_k + l_k$. Since $t^*_m > t_q$ and $l_m \geq l_q$, we have $t^*_m + l_m > t_q + l_q$. Since $t^*_m > t_m$, we have $t^*_m + l_m > t_m + l_m$. Therefore, $T^*_{\text{iter}} \geq T_{\text{iter}}$, meaning the swap does not reduce latency, contradicting the initial assumption. Moreover, any priority ordering can be obtained by starting from the descending-lag order and performing a sequence of such swaps. Since none of these swaps can reduce the latency, the ordering that prioritizes clients with larger lags is optimal.

Appendix B
Proof of Theorem 2

With Assumption 1, the lags satisfy $l_1 \leq l_2 \leq \cdots \leq l_K$. The proposed priority scheduling strictly follows this lag ordering. We suppose that there exist indices $m, k \in \mathcal{K}$ with $m > k$ (so $l_m \geq l_k$) such that swapping their priorities, i.e., assigning client m a lower priority than client k, reduces the per-round training latency.

We use the superscript * to denote all intervals and timestamps after the swap. Let $T_{CPSFL,k}$ be the latency for client k to complete all iterations in a round. We will show $T^*_{CPSFL} \geq T_{CPSFL}$, thereby proving Theorem 2.

Firstly, lowering client m's priority may cause its GT to wait for those of clients k through $m-1$, so $T^*_{CPSFL,m} \geq T_{CPSFL,m}$. Secondly, raising client k's priority eliminates its need to wait for transmissions of clients $k+1$ through m, so $T_{CPSFL,k} \geq T^*_{CPSFL,k}$.

Thirdly, client m's priority after the swap equals client k's priority before the swap. Thus, the number of higher-priority clients remains unchanged. Let ϱ_k and ϱ^*_k denote the number of GTs client k must wait for before and after the swap, respectively, with $\varrho_k, \varrho^*_k \in \left[I, \sum_{j=k}^{K} I\right]$. Let $\overline{\tau}_{SG,k}$ and $\overline{\tau}^*_{SG,k}$ be the corresponding average per-transmission latencies. Then, we approximate $T_{CPSFL,k} \approx \varrho_k \overline{\tau}_{SG,k} + Il_k$ and $T^*_{CPSFL,m} \approx \varrho^*_m \overline{\tau}^*_{SG,m} + Il_m$. Under Assumption 3, we have $\tau_{SG,k} = \tau_{SG}$, $\forall k$ and $\overline{\tau}^*_{SG,m} = \overline{\tau}_{SG,k}$. Moreover, under Assumption 3 and $l_m \geq l_k$, client k is enqueued more frequently than client m, implying $\varrho^*_m \geq \varrho_k$. Hence, $T^*_{CPSFL,m} \geq T_{CPSFL,k}$.

Similarly, the priority of client j remains unchanged by the swap, $\forall j \in [k+1, m-1]$. We approximate $T_{CPSFL,j} \approx \varrho_j \overline{\tau}_{SG,j} + Il_j$ and $T^*_{CPSFL,j} \approx \varrho^*_j \overline{\tau}^*_{SG,j} + Il_j$. Under Assumption 3, we have $\overline{\tau}_{SG,j} = \overline{\tau}^*_{SG,j}$. Moreover, under Assumption 3 and $l_m \geq l_k$, client k is enqueued more frequently than client m, implying $\varrho^*_j \geq \varrho_j$. Hence, $T^*_{CPSFL,j} \geq T_{CPSFL,j}$.

Finally, since client j's priority are unaffected by the swap, $\forall j \in [1, k-1] \cup [m+1, K]$, their per-round training latency is basically unchanged, i.e., $T^*_{CPSFL,j} = T_{CPSFL,j}$.

These results imply $T^*_{CPSFL} \geq T_{CPSFL}$, contradicting the assumption that the swap reduces latency. Since any priority ordering can be reached via successive swaps from the descending-lag order, without reducing latency, the proposed lag-based priority scheduling is optimal.

REFERENCES

[1] H. Kheddar, Y. Habchi, M. C. Ghanem, M. Hemis, and D. Niyato, "Recent advances in transformer and large language models for UAV applications," *arXiv preprint arXiv:2508.11834*, 2025.

[2] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," *IEEE Commun. Surv. Tut.*, 2025, early access.

[3] Z. Chen, H. H. Yang, Y. Tay, K. F. E. Chong, and T. Q. Quek, "The role of federated learning in a wireless world with foundation models," *IEEE Wireless Commun.*, vol. 31, no. 3, pp. 42–49, 2024.

[4] N. Yan, Y. Su, Y. Deng, and R. Schober, "Federated fine-tuning of LLMs: Framework comparison and research directions," *IEEE Commun. Mag.*, vol. 63, no. 10, pp. 52–58, 2025.

[5] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[6] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 8, 2022, pp. 8485–8493.

[7] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6G edge networks," *IEEE Wireless Commun.*, vol. 31, no. 4, pp. 170–176, 2024.

[8] X. Qiang, Z. Chang, C. Ye, T. Hamalainen, and G. Min, "Split federated learning empowered vehicular edge intelligence: Concept, adaptive design, and future directions," *IEEE Wireless Commun.*, 2025, early access.

[9] B. Xu, H. Zhao, C. Bao, and H. Zhu, "Optimizing efficient federated split learning over wireless networks: Challenges and opportunities," *IEEE Netw.*, 2025, early access.

[10] W. Ni, H. Tian, S. Wang, C. Li, L. Sun, and Z. Yang, "Federated split learning for resource-constrained robots in industrial IoT: Framework comparison, optimization strategies, and future directions," *arXiv preprint arXiv:2510.05713*, 2025.

[11] C. Xu, J. Li, Y. Liu, Y. Ling, and M. Wen, "Accelerating split federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 23, no. 6, pp. 5587–5599, 2024.

[12] S. Chu, Y. Ni, J. Li, K. Wei, and J. Wang, "Online device scheduling and model partition in hybrid asynchronous split federated learning," *IEEE Commun. Lett.*, 2025, early access.

[13] G. Qiang, F. Fang, H. Chen, and X. Wang, "Joint computational resource allocation and layer partitioning for federated learning," *IEEE Internet Things J.*, no. 99, pp. 1–1, 2025.

[14] G. Zhu, Y. Deng, X. Chen, H. Zhang, Y. Fang, and T. F. Wong, "ESFL: Efficient split federated learning over resource-constrained heterogeneous wireless devices," *IEEE Internet Things J.*, vol. 11, no. 16, pp. 27 153–27 166, 2024.

[15] Y. Wen, G. Zhang, K. Wang, and K. Yang, "Training latency minimization for model-splitting allowed federated edge learning," *IEEE Trans. Netw. Sci. Eng.*, 2025, early access.

[16] H. Ao, H. Tian, W. Ni, G. Nie, and D. Niyato, "Semi-asynchronous federated split learning for computing-limited devices in wireless networks," *IEEE Trans. Wireless Commun.*, 2025, early access.

[17] X. Wang, S. Song, Z. Zhang, X. Hou, Z. Li, T. Xing, and X.-P. Zhang, "Split federated learning for resource-constrained edge computing networks," *IEEE Trans. Consum. Electron.*, 2025, early access.

[18] J. Hu, Y. Liang, Y. Chen, G. Liu, W. Chen, and L. Duan, "Performance optimization of split federated learning in heterogeneous edge computing environments," *IEEE Trans. Ind. Inform.*, 2025, early access.

[19] S. Zhang, W. Wu, L. Song, and X. Shen, "Efficient model training in edge networks with hierarchical split learning," *IEEE Trans. Mobile Comput.*, 2025, early access.

[20] S. Zhang, G. Cheng, W. Wu, X. Huang, L. Song, and X. Shen, "Split fine-tuning for large language models in wireless networks," *IEEE J. Sel. Topics Signal Process.*, 2025, early access.

[21] Z. Li, S. Wu, L. Li, and S. Zhang, "Energy-efficient split learning for fine-tuning large language models in edge networks," *IEEE Netw. Lett.*, 2025, early access.

[22] X. Chen, W. Wu, F. Ji, Y. Lu, and L. Li, "Privacy-aware split federated learning for LLM fine-tuning over internet of things," *IEEE Internet Things J.*, 2025, early access.

[23] M. Wu, R. Yang, X. Huang, Y. Wu, J. Kang, and S. Xie, "Joint optimization of model partition and resource allocation for split federated learning over vehicular edge networks," *IEEE Trans. Veh. Technol.*, vol. 73, no. 10, pp. 15 860–15 865, 2024.

[24] L. Yu, Z. Chang, Y. Jia, and G. Min, "Model partition and resource allocation for split learning in vehicular edge networks," *IEEE Trans. Intell. Transp. Syst.*, 2025, early access.

[25] W. Wu and X. Huang, "Split-LEO: efficient AI model training over LEO satellite networks," *Sci. China Inf. Sci.*, vol. 68, no. 9, pp. 1–15, 2025.

[26] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, "Split learning over wireless networks: Parallel design and resource management," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, 2023.

[27] Z. Lin, G. Zhu, Y. Deng, X. Chen, Y. Gao, K. Huang, and Y. Fang, "Efficient parallel split learning over resource-constrained wireless edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9224–9239, 2024.

[28] H. Ao, H. Tian, and W. Ni, "Federated split learning for edge intelligence in resource-constrained wireless networks," *IEEE Trans. Consum. Electron.*, 2024, early access.

[29] Z. Lin, G. Qu, W. Wei, X. Chen, and K. K. Leung, "AdaptSFL: Adaptive split federated learning in resource-constrained edge networks," *IEEE Trans. Netw.*, 2025, early access.

[30] Z. Lin, Z. Chen, X. Chen, W. Ni, and Y. Gao, "HASFL: Heterogeneity-aware split federated learning over edge computing systems," *arXiv preprint arXiv:2506.08426*, 2025.

[31] X. Qiang, Z. Chang, Y. Hu, L. Liu, and T. Hämäläinen, "Adaptive and parallel split federated learning in vehicular edge computing," *IEEE Internet Things J.*, vol. 12, no. 5, pp. 4591–4604, 2025.

[32] X. Chen, L. Li, F. Ji, and W. Wu, "Memory-efficient split federated learning for LLM fine-tuning on heterogeneous mobile devices," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2025, pp. 1–6.

[33] K. Zhao and Z. Yang, "Efficient federated split learning for large language models over communication networks," *arXiv preprint arXiv:2504.14667*, 2025.

[34] Z. Wang, Y. Zhou, Y. Shi, and K. B. Letaief, "Federated fine-tuning for pre-trained foundation models over wireless networks," *IEEE Trans. Wireless Commun.*, 2025, early access.

[35] F. Solat, J. Lee, and D. Niyato, "Split federated learning-empowered energy-efficient mobile traffic prediction over UAVs," *IEEE Wireless Commun. Lett.*, 2024, early access.

[36] L. U. Khan, M. Guizani, S. Muhaidat, and M. Ayyash, "QoS-enabled wireless split federated learning: A reinforcement learning and optimization approach," *IEEE Trans. Consum. Electron.*, 2025, early access.

[37] Y. Gao, B. Hu, M. B. Mashhadi, W. Wang, and M. Bennis, "PipeSFL: A fine-grained parallelization framework for split federated learning on heterogeneous clients," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 1774–1791, 2025.

[38] G. Sun, W. Xie, D. Niyato, F. Mei, J. Kang, H. Du, and S. Mao, "Generative AI for deep reinforcement learning: Framework, analysis, and use cases," *IEEE Wireless Commun.*, 2025, early access.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] 3GPP, "Study on enhanced LTE support for aerial vehicles," Technical Report (TR) 36.777, 2017, version 15.0.0. [Online]. Available: https://www.3gpp.org/DynaReport/36777.htm

[42] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf Comput. Vis.*, 2021, pp. 10 012–10 022.

[43] NVIDIA Corporation, "NVIDIA GeForce graphics cards comparison," 2025. [Online]. Available: https://www.nvidia.cn/geforce/graphics-cards/compare/

[44] ——, "NVIDIA Jetson Xavier," 2025. [Online]. Available: https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-xavier-series/