FedHusky: Accelerating Hybrid Federated Learning with Client Hopping

Fangtong Zhou*, Yi Shi†, Wenjing Lou*, Y. Thomas Hou*
*Virginia Tech, Blacksburg, VA, USA
†Commonwealth Cyber Initiative, Virginia Tech, Arlington, VA, USA

***Abstract*—Classic federated learning (FL) trains client models in parallel, and its accuracy hinges on each client's local update. But when those updates come from small, highly heterogeneous datasets, parallel FL's performance collapses. Sequential FL alleviates this problem by letting clients update the model in a serial manner. But this sequential execution leaves all but one client idle and wastes computation resources for training. By combining both parallel and sequential FL, hybrid FL partitions clients into groups to run groups in parallel while keeping sequential updates within each group. Motivated by hybrid FL, we propose a novel solution called FedHusky that can fundamentally advance the hybrid FL solution. Most notably, FedHusky employs a novel calendar mechanism at each client to record its busy intervals that have been booked by different working groups. A novel optimization-based approach is employed to generate new groups, aiming at maximizing the number of busy intervals on each client's calendar while avoiding clients being double-booked by different groups at any time. FedHusky also employs a dynamic birth–death process to maintain the active groups during training, enabling a vibrant ecosystem by allowing the membership (clients) in each group to change over time while striving for the maximum number of simultaneous groups. By incorporating a simple delay mechanism, FedHusky can be highly robust to potential estimation errors in key timing parameters, capable of maintaining smooth operation without interruption. Experiments show that under small datasets and high heterogeneity, FedHusky accelerates convergence by at least $3\times$ and increases the average busy ratio by $6.7\times$ when compared with hybrid FL.**

I. Introduction

Federated Learning (FL) is a distributed machine learning (ML) paradigm that enables multiple clients to collaboratively train a shared model while preserving the privacy of their local data from the central server [1]–[3]. The most well-known algorithm for FL is FedAvg [4], introduced by Google in 2017. In FedAvg, the central server broadcasts the global model to all clients, each of which performs local training on its local data and then uploads its updated model to the central server, which then aggregates these models to update the global model. This process is repeated until the global model converges. Under FedAvg, clients work in a parallel manner (see Fig. 1(a)), i.e., all clients perform their local training in parallel upon receiving a global model. The advantage of parallel FL is evident: it incorporates data from all clients while shortening the duration of each training round. However, under parallel FL, its performance can be severely degraded by *small-scale* and highly *heterogeneous* client datasets [5].

This research was supported in part by ONR grant N00014-24-1-2730 and Virginia Commonwealth Cyber Initiative (CCI).

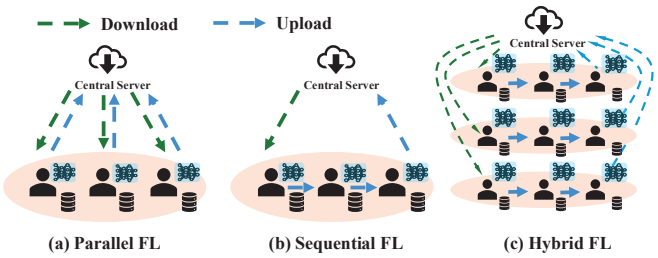


Fig. 1. An illustration that compares parallel, sequential, and hybrid FL.

A small-scale dataset is characterized by a limited number of local samples per client, while a highly heterogeneous dataset means pronounced disparities in label and feature distributions across clients (usually refers to non-i.i.d.). Under parallel FL, clients' updates are aggregated simultaneously. Scarce data makes these updates noisy and biased [6], and heterogeneity steers them in conflicting directions [7], [8]. As a result, naive averaging suppresses useful signals and magnifies noise. Together, small-scale datasets and heterogeneity slow down or even prevent convergence, sharply limiting the accuracy of parallel FL [9].

To address the limitation of parallel FL under small-scale datasets and heterogeneity, the so-called *sequential* FL training was proposed in [10]. Unlike parallel FL, sequential FL requires clients to perform local training in a serial manner. Each client takes the model that is trained by its predecessor, performs its local training, and then passes the newly trained model to the next client (see Fig. 1(b)). In essence, sequential FL turns global learning (i.e., averaging disparate local models) into incrementally updating the global model with each client's contribution in sequence, thereby removing the variance penalty of tiny local datasets and the bias penalty of extreme non-i.i.d. splits [11]. This ensures each client's information is captured without destructive cancellation.

However, while sequential FL improves parallel FL (in terms of convergence and accuracy) in the presence of small-scale datasets and heterogeneity, its training process appears to be highly inefficient. Specifically, at any time, there is only one client busy training while all the other clients are idle (waiting for their turns). Hence, its training efficiency is low.

To address the efficiency problem in sequential FL, authors in [12]–[14] proposed a hybrid training solution (see Fig. 1(c)), which combines features from both the parallel and sequential FL. Under hybrid FL, clients are partitioned into separate groups. Within each group, sequential training is

employed, while all the groups are running in parallel. Further, to emulate homogeneous parallel FL, hybrid FL puts clients with markedly different data in the same group for sequential training [14]. Intuitively, hybrid FL can be envisioned as a general form for both parallel and sequential FL, with both as its extreme cases. Specifically, when the number of groups equals the number of clients, hybrid FL degenerates into parallel FL. On the other hand, when the number of groups is one, hybrid FL degenerates into sequential training. By tuning the number of groups and the number of clients per group to match the actual data distribution, the hybrid solution can efficiently boost training performance.

Although hybrid FL is a major advancement of parallel and sequential FL, it still inherits the drawback of sequential FL. Namely, within each group, only one client is busy training while the other clients in the group are idle. This is a huge waste of computational resources in the training process. This paper focuses on this problem by proposing a novel FL algorithm called *FedHusky*.[1] The goal of FedHusky is to maximize the system's training efficiency by maximizing each client's training efficiency, i.e., the percentage of time the client is busy training its local model. The novelties and highlights of FedHusky are summarized below.

- FedHusky employs a novel calendar mechanism at each client to record its busy intervals. This calendar allows each client to be booked (reserved) for training by different groups at different times, effectively enabling each client to hop across different groups over time. Compared to hybrid FL, this novel calendar mechanism is the key technology to reduce the downtime of each client and enable it to work much more efficiently.
- FedHusky also employs a novel optimization-based approach to generate new groups and update the calendars at the clients as time progresses. The objective of each optimization problem is to create a new working group, which subsequently increases the number of busy intervals on those clients in the new group, while meeting various scheduling constraints, including no "double-booking" constraint for each client. The finding of a feasible solution to the optimization problem will lead to the generation of a new working group as well as the updates of the calendars of those clients selected by the new group.
- FedHusky employs a dynamic birth-death process for each group so as to maintain a vibrant learning ecosystem among the clients. Once a client completes its training, it is released by the group and immediately triggers a new optimization problem with the attempt to generate a new group. Once the last client in the group completes its training, it uploads its trained model to the server and the group dies. By allowing such dynamic formation and dissolution of working groups, FedHusky strives to generate the maximum possible number of groups while keeping the membership of each new group as fluid as possible.
- Although FedHusky requires knowledge of processing time for busy intervals at each client when solving each optimization problem, it is robust to potential estimation errors in this key parameter. Specifically, whenever there is an underestimation of processing time and thus a partial overlap of actual busy intervals on a client's calendar (i.e., partial double-booking), all that the affected client needs to do is to complete the current training task at hand in one group and then start the next job in another group. Although this will cause a slight delay in the latter group, it does not pose any serious issue in the continuing operation of FedHusky, making FedHusky highly robust and flexible.
- Experimental results show that under small-scale and highly heterogeneous dataset settings, FedHusky significantly outperforms hybrid FL. In particular, it attains a convergence speedup of $5.9\times$ for the MNIST dataset and $3\times$ for the Fashion-MNIST dataset compared to hybrid FL. Moreover, the average client busy ratio of FedHusky is $6.7\times$ higher than that of hybrid FL.

II. FedHusky: An Overview

This paper considers a general FL scenario with a central server and a number of mobile clients. Each mobile client is assumed to be capable of data collection, local training, and wireless communications. The central server performs aggregation and updates the global model.

To address the limitations of hybrid FL, we propose FedHusky, a novel FL solution aiming to maximize the busy intervals of each client, thereby improving training efficiency. As shown in Fig. 2, FedHusky has the following major steps.

- **Step 1: Initial Training and Clustering**: In the initialization phase, the FL server transmits an initial global model to all clients. Each client then trains its local model for a few rounds and then sends it to the server. In particular, each client estimates the processing time to train its model for each round.[2] Following this training of local data, the server groups the clients into clusters based on the similarity of the locally trained models—clients in the same cluster share similar data labels (see Section III-A for more details).
- **Step 2: Generating Initial Groups**: Based on the clusters in Step 1, the server creates groups, with each group containing one client from every cluster. In this step, FedHusky employs a novel distributed *calendar* mechanism at each client and aims to form as many groups as possible—each client can be in as many groups as possible, as long as it is not double-booked by two groups at the same time (i.e., no overlap of two busy intervals on its calendar). Note that this is in fundamental contrast

[1]The codename *FedHusky* is inspired by Alaskan Husky—a hard-working dog primarily for sledding. Husky symbolizes our goal of striving to keep the clients as busy as possible in training their local models.

[2]A client's local processing time, upload and download communication time will be estimated throughout the training process. In Section III-E, we will show that FedHusky has great tolerance of errors incurred in estimation.

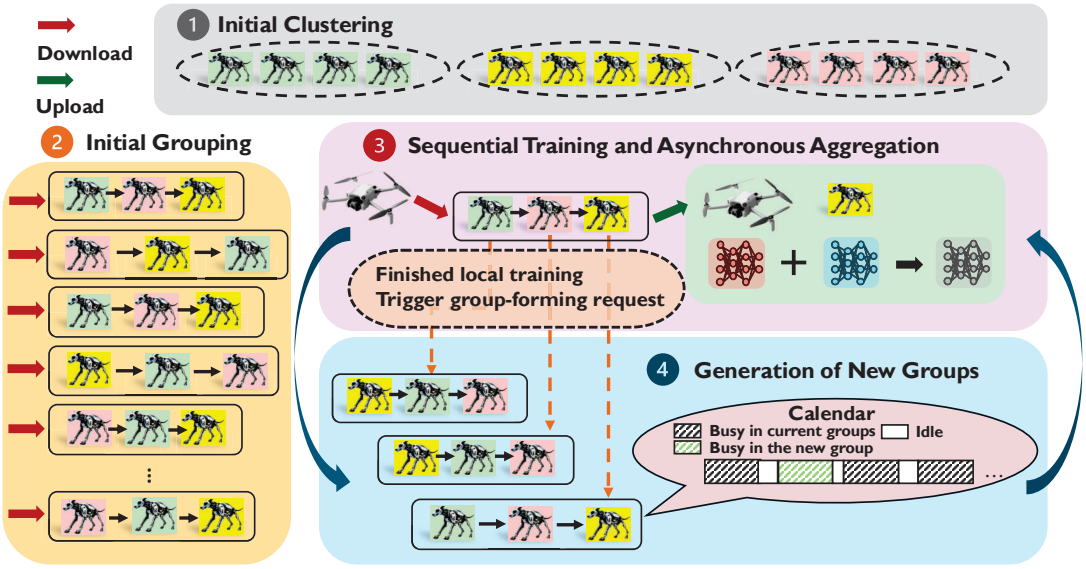


Fig. 2. Workflow of FedHusky.

to hybrid FL, where each client can only participate in training in one group. The calendar-based scheduling mechanism enables a client to *hop* freely across different groups at different times, instead of being bound to only one group as in hybrid FL. The main challenge here is how to set up these initial groups so that as many initial groups as possible can be generated, while none of the clients is double-booked by two groups at the same time on its local calendar. We propose to address this problem by developing and solving a series of optimization problems. The details are given in Section III-B.

- **Step 3: Sequential Training in Each Group and Aggregation**: Once the initial groups are formed, clients in each group start their training, following their local calendars that are created in Step 2. Local training in each group follows a sequential manner. This process stops until the last client in the group completes its training and uploads its trained model back to the server. The details are given in Section III-C. Since each client's processing time is estimated in Step 1 (and subsequently used in Step 2), its actual processing time is likely to differ. In Section III-E, we show that FedHusky is highly robust to such estimation errors and can easily adapt to them as the training process progresses over time.
- **Step 4: Generation of New Groups**: For each group in Step 3, once a client completes its training in the group, the client will be released from the group. Once the last client in the group completes its training, the group ceases to exist (dies). To ensure a vibrant ecosystem (without encountering extinction), FedHusky attempts to generate new groups whenever such an opportunity arises. Specifically, whenever a client completes its task and is released from its current group, it triggers a new group-forming request at the server. The server will attempt to form a new group by solving a new optimization problem (similar to those in Step 2). The goal is to have as many simultaneous groups as possible and to have each client hop around in as many groups as possible. The details on this step are given in Section III-D. For each newly generated group, it goes to **Step 3** to perform training and aggregation.
- **Step 5: Termination**: The algorithm terminates once a pre-defined training time limit is reached.

In the following section, we provide details on these steps.

III. FedHusky: Design Details

In this section, we provide details on the above key steps in FedHusky.

A. Step 1: Initial Training and Clustering

Ideally, clustering should be directly based on clients' data distributions. However, due to privacy concerns, sharing clients' data distributions with the server is prohibited. Instead, the clients initiate a short period of local training in the initial stage and upload their local models $\boldsymbol{w}_i$, $i = 1, 2, \ldots, N$ (N denotes the number of clients) to the server, which will perform clustering based on these local models.

At the server, the first step is to determine how many clusters will be formed, which we denote as C. Too few clusters may put dissimilar clients in the same group, while too many clusters may excessively fragment the data types, both hindering training performance. We adopt the *gap statistic* algorithm [15] to carry out this process.

Once the number of clusters C is determined, the clustering results can also be obtained concurrently via the *K-means* algorithm [16]. Denote the C clusters as $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_C$, respectively, where clients in the same cluster share similar local models and are likely to possess similar data distributions.

A potential problem with clustering is that the number of clients in each cluster may vary widely, which hinders our

subsequent grouping operation. To address this problem, we perform a re-balancing step. Ideally, we should have a cluster size of either $\lceil \frac{N}{C} \rceil$ or $\lfloor \frac{N}{C} \rfloor$. For a cluster with the number of clients greater than $\lceil \frac{N}{C} \rceil$, the server discards the "farthest" clients and reassigns them to the cluster that is "nearest" that currently has fewer clients than $\lfloor \frac{N}{C} \rfloor$. Here, "farthest" and "nearest" refer to the Euclidean distance between each client's local model and the centroid of a cluster as determined by the K-means algorithm.

When a client uploads its model $\boldsymbol{w}_i$, it also submits its local training time δ_i. Upon receiving δ_i from client i, the server computes the total processing time at client i, denoted as Δ_i, as follows: $\Delta_i = \delta_i + \psi_i$, where ψ_i is the communication time of transmitting the model. For each client i, the server maintains a tuple $(\Delta_i, m(i))$, where $m(i) \in \{1, 2, \ldots, C\}$ is a mapping, denoting the specific cluster that client i belongs to.

B. Step 2: Generating Initial Groups

Recall that in hybrid FL, the number of groups (denoted as G) is $\lfloor \frac{N}{C} \rfloor$. Under FedHusky, where a client can hop into other groups as long as it is not double-booked at the same time, we will have many more groups than $\lfloor \frac{N}{C} \rfloor$. An upper bound for G is N, where each client will be the first client in each of the N groups. But in reality, we will have a group number smaller than N, due to heterogeneity in training time at each client and estimation error of various time durations (upload, download, actual training time). So the goal of Step 2 is to form as many groups as possible by allowing a client to participate in as many groups as possible, as long as it is not double-booked at the same time.

To achieve this, the server iteratively considers each client from 1 to N as the first member of a potential group (i.e., a total of N potential groups). That is, for each potential group j, $j = 1, 2, \ldots, N$, the first (head) member of the group is client j. FedHusky selects the remaining clients from each remaining cluster by solving an optimization problem.

We start our discussion with the first group.

1) Generating the first group: For the first group, the first client in the group is client #1, which comes from cluster $m(1)$. For the rest $C-1$ clusters, the server selects one client from each of the other $C-1$ clusters, respectively, and randomly places them in the rest $C-1$ positions in the group. The position of each client in the group corresponds to its training order. Given its position in the group, each selected client i can record its busy interval.

To keep track of the busy intervals at each client i, it is necessary to establish a *calendar* for each client. Define $\mathcal{B}_i$ as the calendar (represented as a set) for client i as follows:

$$\mathcal{B}_i = \{[b_i^{(k)}, e_i^{(k)}], \;\; k = 1, 2, \ldots\}, \tag{1}$$

where each element in set $\mathcal{B}_i$ is an interval $[b_i^{(k)}, e_i^{(k)}]$ representing the begin and ending times of client i's k-th busy interval. The busy intervals are ordered chronologically, i.e., $b_i^{(k)} < e_i^{(k)} \leq b_i^{(k+1)}$ $(k = 1, 2 \ldots)$ and the number of elements (busy intervals) in set $\mathcal{B}_i$ can go to infinite if the training process goes on indefinitely.

After generating group 1, each client i in this group adds its busy interval $[b_i^{(1)}, e_i^{(1)}]$ in its calendar $\mathcal{B}_i$.

2) Generating the second group: After the first group is formed, the server moves on to generate the second group. Recall that the server places the client 2, which is in cluster $m(2)$, as the first client in the second group. For the remaining $C-1$ positions, the server will select one client from each of the other $C-1$ clusters.

It is important to note that the clients in each of the clusters is not changed. Given that the clients in the other $C-1$ clusters remain the same as the previous iteration, the server may select clients that are already selected in the first group. To ensure there is no overlap between the busy intervals for any client, we model the client selection problem mathematically as a feasibility problem.

Denote x_i as a binary variable indicating whether or not client i will be selected for group 2, i.e.,

$$x_i = \begin{cases} 1, & \text{if client } i \text{ is selected for group 2,} \\ 0 & \text{otherwise.} \end{cases}$$

Since exactly only one client is selected from each of the C clusters, we have:

$$x_2 = 1, \tag{2}$$

$$\sum_{i \in \mathcal{K}_c} x_i = 1 \quad (c = 1, 2, \ldots, C). \tag{3}$$

The generation of the second group depends on the ordering of the clients in the first group, as well as the clients to be chosen for the second group so as to avoid any double-booking. Denote $y_{i \mapsto p}$ as a binary variable indicating whether or not client i is placed in position p ($p = 1, 2, \ldots, C$) in the second group:

$$y_{i \mapsto p} = \begin{cases} 1, & \text{if client } i \text{ is placed in position } p \text{ in group 2,} \\ 0 & \text{otherwise.} \end{cases}$$

Since we place client 2 in the first position of group 2, we have:

$$y_{2 \mapsto 1} = 1. \tag{4}$$

Since for each position p, we can have exactly one client, we have:

$$\sum_{i=1}^{N} y_{i \mapsto p} = 1 \quad (p = 1, 2, \ldots, C). \tag{5}$$

Also, for any client i, it can appear no more than once (or none) in group 2. We have:

$$\sum_{p=1}^{C} y_{i \mapsto p} = x_i \quad (i = 1, 2, \ldots, N). \tag{6}$$

Denote $h[p]$ as the start time of the client at the p-th position in group 2, $p = 1, 2, \ldots, C$. Denote t_0 as the start time for group 2, i.e.,

$$h[1] = t_0. \tag{7}$$

Since $h[p]$ is the sum of the start time of the client at position $(p-1)$ and its busy interval, we have:

$$h[p] = h[p-1] + \sum_{i=1}^{N} y_{i \mapsto p-1} \Delta_i \quad (p = 2, 3, \dots C). \tag{8}$$

Denote client i's start time of its busy interval in group 2 as s_i. Then we have the following relationship among s_i, $h[p]$, and $y_{i \mapsto p}$ for group 2:

$$s_i = \begin{cases} h[p], & \text{if } y_{i \mapsto p} = 1 \text{ for group 2,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

To reformulate the above raw formulation into a mathematical program, we have:

$$s_i \geq h[p] - M(1 - y_{i \mapsto p}) \quad (p = 1, 2, \dots, C), \tag{9}$$

$$s_i \leq h[p] + M(1 - y_{i \mapsto p}) \quad (p = 1, 2, \dots, C). \tag{10}$$

where M is a sufficiently large constant.

Based on the calendar concept that we introduced earlier, we must ensure that the busy interval on the calendar of each client in group 2 does not conflict with its busy interval marked in group 1 if the client also belongs to group 1. For a client i that already belongs to group 1, it has its calendar as $\mathcal{B}_i = \{[b_i^{(1)}, e_i^{(1)}]\}$. If this same client i is to be put in position p in group 2, then its new busy interval $[s_i, s_i + \Delta_i]$ can only be before or after the existing busy interval $[b_i^{(1)}, e_i^{(1)}]$ so as not to be double-booked.

Specifically, if client i is in group 1, then there are two gaps in $\mathcal{B}_i$ to place its new busy interval if it is also selected for group 2, i.e., $[0, b_i^{(1)}]$ (which we call "gap 1") and $[e_i^{(1)}, T]$ (or "gap 2"), where T is the termination time of FedHusky. To determine which gap to insert the new busy interval, we introduce a binary variable $z_i^{(k)}$ $(k = 1, 2)$ as follows:

$$z_i^{(k)} = \begin{cases} 1, & \text{if client } i \text{ inserts the new busy interval} \\ & \text{into gap } k, \\ 0 & \text{otherwise.} \end{cases}$$

Since this new busy interval can only be inserted into one gap on client i's calendar (when $x_i = 1$ for group 2), we have:

$$\sum_{k=1}^{2} z_i^{(k)} = x_i \ . \tag{11}$$

Moreover, if $z_i^{(k)} = 1$ (i.e., client i's new busy interval can be successfully inserted into its k-th gap), we have $s_i \geq e_i^{(k-1)}$, $s_i + \Delta_i \leq b_i^{(k)}$, where we define $e_i^{(0)} = 0$ and $b_i^{(2)} = T$. Otherwise, i.e., $z_i^{(k)} = 0$ (client i's new busy interval is not inserted into the k-th gap), there is no relationship between s_i and the k-th gap $[e_i^{(k-1)}, b_i^{(k)}]$. The above two statements can be formulated as:

$$s_i \geq e_i^{(k-1)} - M(1 - z_i^{(k)}) \qquad (k = 1, 2), \tag{12}$$

$$s_i + \Delta_i \leq b_i^{(k)} + M(1 - z_i^{(k)}) \qquad (k = 1, 2), \tag{13}$$

where M is a sufficiently large positive constant so that constraints always hold when $z_i^{(k)} = 0$.

Since the goal of FedHusky is to keep clients as busy as possible, we define a busy ratio, θ_i, for client i. If client i is not selected in group 1, then $\theta_i = 0$ at this point. If client i is selected in group 1 (i.e., it has one busy interval $[b_i^{(1)}, e_i^{(1)}]$ in its calendar), $\theta_i = \left(e_i^{(1)} - b_i^{(1)}\right)/T$. When generating the second group (as well as future groups), the server aims to maximize the minimum current busy ratio, i.e., the busy ratio after the group 2's possible selecting action $\theta_i + \Delta_i x_i / T$ over all clients $i = 1, 2, \dots, N$ (we denote the set of clients as $\mathcal{N}$). We now have the following problem formulation to generate group 2:

$$\textbf{OPT-2}: \max \quad \min_{i \in \mathcal{N}} \theta_i + \frac{\Delta_i x_i}{T} \tag{14a}$$

$$\text{s.t.} \quad \text{Client selection: (3),} \tag{14b}$$

$$\text{Position selection: (5) to (10),} \tag{14c}$$

$$\text{Gap selection: (11), (12), (13),} \tag{14d}$$

$$\text{Pre-assigned parameters: (2), (4),} \tag{14e}$$

$$x_i, y_{i \mapsto p}, z_i^{(k)} \in \{0, 1\}. \tag{14f}$$

Problem OPT-2 is an integer program (IP), which can be solved efficiently by the Gurobi solver [17]. After solving OPT-2, the selected clients for group 2 update each of their calendars with their new busy intervals.

3) Generating the i-th group ($i = 3, 4, \dots N$): Based on our experience in generating group 2, it is easy to move on to groups $i = 3, 4, \dots, N$. In essence, for each group i, we need to solve an optimization problem OPT-i, based on the calendars for all clients that are determined from the previous optimization problems.

Specifically, for group i, we have client i selected (i.e., $x_i = 1$) and placed in the first position of the group (i.e., $y_{i \mapsto 1} = 1$). Accordingly, constraint (2) and (4) should be updated for client i.

For each client $j = 1, 2, \dots, N$, the number of busy intervals K_j in its current calendar $\mathcal{B}_j$ is at most $i - 1$, representing that it has been selected in all previous groups. Therefore, there are $K_j + 1$ gaps for potential insertion of a new busy interval. So for OPT-i, constraints (11), (12), and (13) must all update the range of k to span from 1 to $K_j + 1$ (for each j). Also, the busy ratio θ_j in the objective function is updated to $\theta_j = \sum_{k=1}^{K_j} \left(e_j^{(k)} - b_j^{(k)}\right)/T$. Note that if $K_j = 0$, θ_j is 0. For the other constraints in OPT-i, they have the same mathematical formulation as (3), (5) to (10) in OPT-2.

For the updated optimization problem OPT-i, it will either have an optimal solution or be infeasible. If an optimal solution exists, then we have a new group i and will need to update the calendars of those clients in this new group. Otherwise (OPT-i is infeasible), we move on to develop OPT-$(i+1)$. After FedHusky solves the last (i.e., OPT-N) optimization problem, we have completed Step 2.

In Fig. 3, we present the process of inserting client i's new busy interval $[s_i, s_i + \Delta_i]$ into its calendar $\mathcal{B}_i$ once we have a new group.

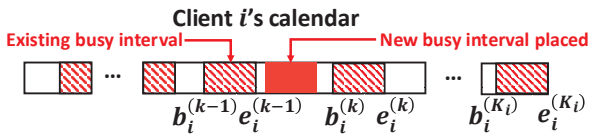


Fig. 3. Insert the new busy interval $[s_i, s_i + \Delta_i]$ into client i's calendar $\mathcal{B}_i$.

C. Step 3: Sequential Training in Each Group and Aggregation

1) Sequential training: Upon forming the initial groups, the server broadcasts its current global model to all clients. The first client in each group starts to train its model and then passes it to the next client in the group and so forth. This process continues sequentially until the last client in each group completes its training and uploads the final model back to the server.

Denote $\boldsymbol{w}_{(j,p)}^{\tau}$ and $F_{(j,p)}(\cdot)$ as the local model and loss function of the client in the p-th position from the j-th group. τ denotes the version of the global model received by the group, and η denotes the learning rate. Each client performs local training through stochastic gradient descent (SGD) as follows:

$$\boldsymbol{w}_{(j,p)}^{\tau} = \boldsymbol{w}_{(j,p-1)}^{\tau} - \eta \nabla F_{(j,p)}(\boldsymbol{w}_{(j,p-1)}^{\tau}) \quad (p = 1, 2, \dots C), \tag{15}$$

where for $p = 1$, $\boldsymbol{w}_{(j,0)}^{\tau}$ represents the global model received from the server.

By denoting $\mathcal{D}_{(j,p)}$ as the client's dataset from the p-th position in group j and ξ_ℓ as the data sample from the local dataset $\mathcal{D}_{(j,p)}$. In (15), the local loss function can be further expressed as follows,

$$F_{(j,p)}(\boldsymbol{w}_{(j,p-1)}^{\tau}) = \mathbb{E}_{\xi_\ell \in \mathcal{D}_{(j,p)}} \left[f_{(j,p)}(\boldsymbol{w}_{(j,p-1)}^{\tau}, \xi_\ell) \right], \tag{16}$$

where $f_{(j,p)}(\boldsymbol{w}_{(j,p-1)}^{\tau}, \xi_\ell)$ represents the sample-wise loss function.

The relaying of a locally trained model from one client to the next in the group can be done via ad hoc networking or using the server as a relay, whichever is faster. So the processing time for each client i (including communication time) is upper bounded by Δ_i.

2) Asynchronous aggregation: Whenever the last client in a group completes its training, it sends its model to the server for aggregation. Since $\boldsymbol{w}_{(j,0)}^{\tau}$ is the global model received by the first client in group j, then the model trained by the last client in group j is $\boldsymbol{w}_{(j,C)}^{\tau}$. Denote $\boldsymbol{w}^t$ as the current global model residing at the server, where t represents its version. Denote β as an aggregation coefficient. Then the aggregation process performed by the server is:

$$\boldsymbol{w}^{t+1} = (1 - \beta)\boldsymbol{w}^t + \beta \boldsymbol{w}_{j,C}^{\tau}, \tag{17}$$

where β can be expressed as:

$$\beta = (1 + (t - \tau))^{-\zeta}. \tag{18}$$

In (18), $\zeta > 0$ is a decay parameter and $t - \tau$ is called the model-version gap.

D. Step 4: Generation of New Groups

In FedHusky, each generated group is used only once. That is, whenever a client in a group completes its training and relays its model to the next client, it is released from the group. When the last client in a group completes its training and uploads its model to the server, the group ceases to exist (dies). The reason why FedHusky avoids reusing the same group is that having the same set of clients perform sequential training in a cyclical manner will lead the model to fall into a local optimum, potentially degrading the overall model performance.

To ensure the number of active groups does not go "extinction" and that each client can continue hopping across different groups, new groups must be formed as time progresses. Although there are different ways to form new groups, FedHusky aims to maximize the number of active groups throughout the training process.

Specifically, after a client is released from a group, it immediately notifies the server. Upon receiving this notification, the server will attempt to generate a new group by solving problem OPT-k, where $k > N$. Note that OPT-k will drop (2) and (4) from its constraints, meaning that no specific client is chosen for the first position in this k-th new group. Further, t_0 in (7) (i.e., the start time of the first position $h[1]$) is shifted to the current wall clock time.

If there is a feasible solution to OPT-k, a new k-th group is generated and each client in this group will need to update its calendar. Otherwise (i.e., no feasible solution to OPT-k), there is no new group; the server waits until the next client notification arrives and then starts the same process for OPT-$(k+1)$.

E. Relaxation of Key Assumption

In this subsection, we show that a key assumption in developing FedHusky can be relaxed, making it a highly robust solution in the field.

Recall that generating a new group requires the estimated Δ_i for each client i. Since Δ_i includes both training and communication times, minor estimation errors are inevitable. However, because FedHusky dynamically routes models via ad hoc networks or a central server—whichever is faster (Step 3)—the estimated Δ_i effectively acts as an upper bound. Therefore, Δ_i is highly likely to be overestimated, which is harmless since clients simply finish their tasks early.

Consequently, the potentially disruptive scenario where Δ_i is underestimated is rare. Even when an occasional underestimation occurs, the resulting partial overlap of *actual* busy intervals on client i's calendar is easily managed. The client simply completes its current job before starting the next. This partial "double-booking" causes only a minor delay in the subsequent group and does not disrupt FedHusky's overall operation.

IV. Experimental Results

In this section, we evaluate the performance of FedHusky and compare it to hybrid FL.

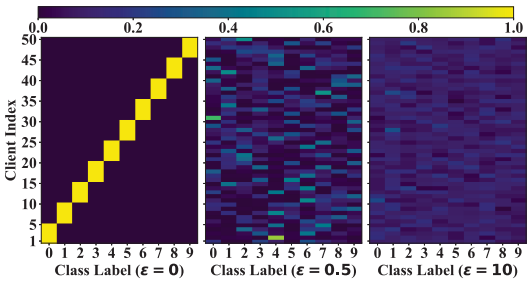
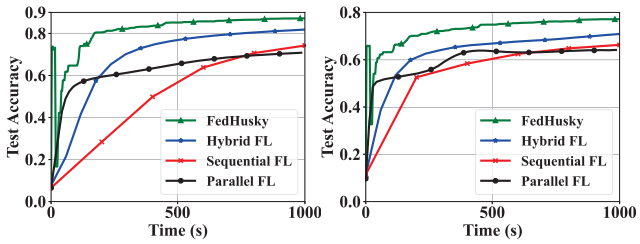


Fig. 4. Different heterogeneity scenarios.



(a) MNIST (b) Fashion-MNIST

Fig. 5. Test accuracy comparison under two different datasets. $N = 50$.

A. Experimental Settings

We assume the server is equipped in a UAV, hovering at a fixed altitude of 100 meters. A total of $N = 50$ clients are randomly deployed on the ground within a circular area of 1000 meters radius. Both uplink and downlink channel bandwidth are 50 MHz, respectively. The wireless channel is modeled by large-scale path loss as: $\mathrm{PL}(d) = \mathrm{PL}_0 + 10 \cdot \alpha \log_{10}(\frac{d}{d_0})$, where d denotes the distance between a client and the UAV server and PL_0 denotes the reference path loss (free space path loss) at $d_0 = 10$ meters and the path-loss exponent is set to $\alpha = 2.8$. Each client transmits with a fixed power of 23 dBm, and the noise power spectral density is -174 dBm/Hz.

We perform the simulation on two common-use datasets: MNIST [18] and Fashion-MNIST [19]. A convolutional neural network (CNN) with two convolutional and three fully connected layers with a model size of 1.7 MB is employed to train both datasets. The learning rate is set to $\eta = 0.01$. The decay parameter in (18) is set to $\zeta = 0.9$. The clients' local training times δ_i are uniformly distributed over $[1, 10]$ seconds.

To create small-scale, heterogeneous datasets, we set each client's dataset size to $|\mathcal{D}_i| = 100$ and use the Dirichlet distribution [20] to simulate heterogeneity across clients. Denote ϵ as the Dirichlet coefficient, which controls the heterogeneity level of data across clients. Clients' data distribution under different ϵ is shown in Fig. 4. While $\epsilon = 0$ represents the extreme non-IID case where each client holds only one label, $\epsilon = 0.5$ illustrates moderate heterogeneity, and $\epsilon = 10$ approaches a completely IID scenario where clients hold a uniform mix of all labels. We set $\epsilon = 0$ in our simulation, representing the most heterogeneous scenario.

The termination time for FedHusky is set to $T = 1000$ seconds.

B. Results

In this section, we provide results for FedHusky through a case study.

The experimental settings are given in the previous section. According to the data distribution of each client shown in Fig. 4 ($\epsilon = 0$), $N = 50$ clients are formed into $C = 10$ clusters, where each cluster has 5 clients.

First, we present the test accuracy performance of FedHusky and compare it with other FL solutions, i.e., hybrid FL (a.k.a PSFL in [14]), parallel FL (FedAvg in [4]) and sequential FL [11]. Fig. 5 shows the results under two datasets. Clearly, under the given termination time ($T = 1000$ seconds), FedHusky outperforms hybrid FL, parallel FL, and sequential FL in accuracy and convergence speed. For example, under the MNIST dataset, to achieve an 80% test accuracy, FedHusky converges $5.9\times$ faster than hybrid FL, $7.9\times$ faster than sequential FL, whereas parallel FL fails to converge to 80%. Under both datasets, we find that parallel FL performs the worst in terms of test accuracy; hybrid FL achieves a noticeable performance improvement over sequential FL, primarily due to its incorporation of multiple groups. FedHusky performs better than hybrid FL due to its higher utilization of clients in training. Note that there are some early fluctuations in FedHusky. This is because in FedHusky, the early global model aggregations include only a subset of clients, which causes temporary instability.

In Fig 6, we show the busy ratio (i.e., θ_i) for all 50 clients over T. For FedHusky, the busy ratio varies from 35.8% to 86.9% over the 50 clients, with the average ratio being 63.4%. In contrast, the busy ratio for hybrid FL varies from 1.8% to 15.5%, with the average ratio being 8.2%. That is, the average client busy ratio for FedHusky is 670% more than that for hybrid FL. Such improvement in each client's work efficiency by FedHusky is the key to its performance improvement in accuracy and convergence speed over hybrid FL.

In Fig. 7, we show the number of active (live) working groups over time throughout the simulation period ($T = 1000$ seconds). The result shows that hybrid FL (orange color) has no more than 5 groups at any time (10 clients per group). In contrast, the average number of active groups in FedHusky is 32 per second. Note that the number of active working groups in FedHusky drops towards the end (right before termination time). This is because, when it gets close to the time limit $T = 1000$ seconds, an attempt to generate a new working group will find that the potential new group's ending time will exceed the time limit. As a result, FedHusky will abandon this attempt without generating a new group.

Finally, we zoom in to examine the calendar of each individual client over time. Due to space limitations, we show, in Fig. 8(a), the calendars of a few randomly selected clients, namely, clients 8, 18, 22, 35, and 45, over the first 100-second time period. In Fig. 8(b), we show the busy intervals for these clients under hybrid FL. Clearly, each client under FedHusky works much harder (more efficiently) than it works under hybrid FL.

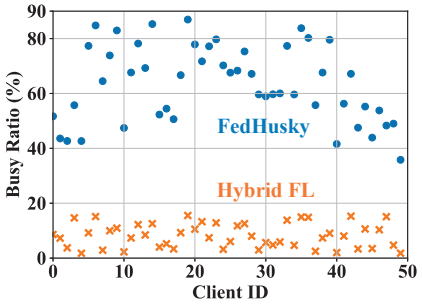


Fig. 6. A comparison of recorded busy ratio for each of the 50 clients in the case study under FedHusky and hybrid FL.

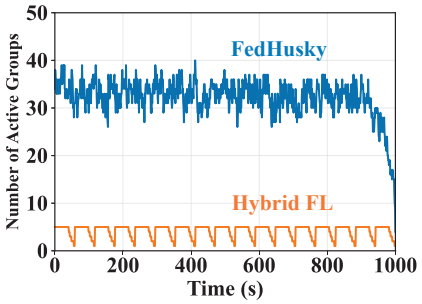
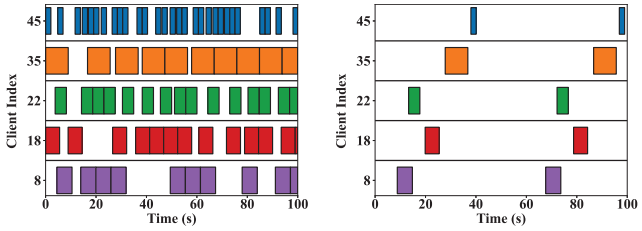


Fig. 7. Number of busy clients at any moment.



(a) FedHusky (b) Hybrid FL

Fig. 8. Comparison of busy intervals between FedHusky and hybrid FL for randomly selected clients over 100 seconds.

V. Conclusions

In this paper, we address the inherent client idleness problem embedded in hybrid FL, which is currently the state-of-the-art solution that addresses small datasets and heterogeneity problems in FL. We propose FedHusky—a novel FL algorithm designed to maximize client utilization. The key enabling mechanism in FedHusky is a distributed calendar for each client, which allows it to keep track of its reservations of busy intervals by different working groups, effectively enabling a client to hop around groups as time progresses. The second novelty in FedHusky is its use of an optimization-based approach to generate new groups, which maximizes work efficiency across clients while preventing double-booking. The third novelty in FedHusky is its employment of a dynamic birth–death process for working groups, ensuring a vibrant client-group ecosystem during the learning process. Finally, FedHusky is highly robust to potential estimation errors in processing time, a key parameter required in the optimization problem to generate new groups and update calendars. Extensive experiment results show that FedHusky significantly outperforms hybrid FL in terms of test accuracy, convergence speed, and per-client utilization. The proposed FedHusky represents a major advancement in addressing small datasets and heterogeneity problems in FL.

References

[1] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A Joint Learning and Communications Framework For Federated Learning Over Wireless Networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[2] F. Zhou, Z. Wang, H. Shan, L. Wu, X. Tian, Y. Shi, and Y. Zhou, "Over-the-Air Hierarchical Personalized Federated Learning," *IEEE Trans. Veh. Technol.*, vol. 74, no. 3, pp. 5006–5021, Mar. 2025.

[3] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, "Federated Learning for the Internet of Things: Applications, Challenges, and Opportunities," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 24–29, Feb. 2022.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2017, pp. 1273–1282.

[5] F. Zhou, Y. Shi, Y. Wu, S. Acharya, L. DaSilva, S. Kompella, W. Lou, and Y. T. Hou, "WOS: An Optimized Scheduling Scheme for Federated Learning in Dynamic Wireless Networks," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Los Angeles, CA, USA, Oct. 2025, 6 pages.

[6] M. Kamp, J. Fischer, and J. Vreeken, "Federated Learning from Small Datasets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Singapore EXPO, Singapore, Apr. 2025.

[7] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning On Non-IID Data With Reinforcement Learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Toronto, ON, Canada, Jul. 2020.

[8] F. Zhou, Z. Wang, Y. Shi, and Y. Zhou, "Decentralized Satellite Federated Learning via Intra-and Inter-Orbit Communications," in *Proc. IEEE Int. Conf. Commun. Wkshps (ICC Wkshps)*, Denver, CO, USA, Jun. 2024, pp. 786–791.

[9] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "A Novel Framework For The Analysis And Design of Heterogeneous Federated Learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 5234–5249, Sep. 2021.

[10] K. Kopparapu and E. Lin, "FedFMC: Sequential Efficient Federated Learning on Non-iid Data," *arXiv preprint*, 2020, accessed: July 31, 2025. [Online]. Available: https://arxiv.org/abs/2006.10937

[11] Y. Li and X. Lyu, "Convergence Analysis of Sequential Federated Learning on Heterogeneous Data," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, New Orleans, LA, USA, Dec. 2023.

[12] R. Zaccone, A. Rizzardi, D. Caldarola, M. Ciccone, and B. Caputo, "Speeding Up Heterogeneous Federated Learning With Sequentially Trained Superclients," in *Proc. 26th Int. Conf. Pattern Recognit. (ICPR)*, Montreal, QC, Canada, Aug. 2022.

[13] S. Zeng, Z. Li, H. Yu, Y. He, Z. Xu, D. Niyato, and H. Yu, "Heterogeneous Federated Learning via Grouped Sequential-to-Parallel Training," in *Proc. Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, Virtual Conference, Apr. 2022.

[14] J. Zhou, Y. Zhao, Y. Xu, M. Xiao, J. Wu, and S. Zhang, "PSFL: Parallel-Sequential Federated Learning with Convergence Guarantees," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, London, United Kingdom, May 2025.

[15] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the Number of Clusters in a Data Set Via the Gap Statistic," *J. R. Stat. Soc. Ser. B (Stat. Methodol.)*, vol. 63, no. 2, pp. 411–423, Apr. 2001.

[16] D. Steinley, "K-Means Clustering: A Half-century Synthesis," *Br. J. Math. Stat. Psychol.*, vol. 59, no. 1, pp. 1–34, May 2006.

[17] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," https://www.gurobi.com, 2024, accessed: July 31, 2025.

[18] L. Deng, "The MNIST Database of Handwritten Digit Images For Machine Learning Research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[19] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint*, 2017, accessed: July 31, 2025. [Online]. Available: https://arxiv.org/abs/1708.07747

[20] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the Effects of Non-identical Data Distribution for Federated Visual Classification," *arXiv preprint*, 2019, accessed: July 31, 2025. [Online]. Available: https://arxiv.org/abs/1909.06335