Journal of Network and Computer Applications 236 (2025) 104105

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Research paper

Efficient privacy-preserving ML for IoT: Cluster-based split federated learning scheme for non-IID data

Mohamad Arafeh [a], Mohamad Wazzeh [a], Hani Sami [a], Hakima Ould-Slimane [b], Chamseddine Talhi [a], Azzam Mourad [c,d,*], Hadi Otrok [e]

[a] Department of Software and IT Engineering, École de Technologie Supérieure (ÉTS), Montreal, QC, Canada
[b] Department of Mathematics and Computer Science, Université du Québec à Trois-Rivières (UQTR), Trois-Rivières, QC, Canada
[c] KU 6G Research Center, Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates
[d] Artificial Intelligence & Cyber Systems Research Center, Lebanese American University, Beirut, Lebanon
[e] Center of Cyber-Physical Systems (C2PS), Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Federated learning
Split learning
Non-IID
Privacy
Machine learning
Participants selection
Clustering

ABSTRACT

In this paper, we propose a solution to address the challenges of varying client resource capabilities in the IoT environment when using the SplitFed architecture for training models without compromising user privacy. Federated Learning (FL) and Split Learning (SL) are technologies designed to maintain privacy in distributed machine learning training. While FL generally offers faster training, it requires clients to train the entire neural network model, which may not be feasible for resource-limited IoT devices. Additionally, FL's performance is heavily impacted by client data distribution and struggles with non-Independent and Identically Distributed (non-IID) data. In parallel, SL offloads part of the training to a server, enabling weak devices to participate by training only portions of the model. However, SL performs slower due to forced synchronization between the server and clients. Combining FL and SL can mitigate each approach's limitations but also introduce new challenges. For instance, integrating FL's parallelism into SL brings issues such as non-IID data and stragglers, where faster devices must wait for slower ones to complete their tasks. To address these challenges, we propose a novel two-stage clustering scheme: the first stage addresses non-IID clients by grouping them based on their weights, while the second stage clusters clients with similar capabilities to ensure that faster clients do not have to wait excessively for slower ones. To further optimize our approach, we develop a multi-objective client selection solution, which is solved using a genetic algorithm to select the most suitable clients for each training round based on their model contribution and resource availability. Our experimental evaluations demonstrate the superiority of our approach, achieving higher accuracy in less time compared to several benchmarks.

1. Introduction

Nowadays, one of the main limitations when considering cybersecurity based on machine learning for IoT devices is their limited computational power and energy capabilities. One method to address this problem is to offload model training to centralized servers. In this case, IoT devices share raw data such as logs, traces, and binaries (Gibert et al., 2020) with third-party servers, which filter and clean the data and proceed with the training process. However, this strategy involves participants sharing their sensitive data, which clearly breaches their privacy. In addition, various organizations in different countries put restrictions on data collection, such as the General Data Protection Regulation implemented by the European Union.

In response to the aforementioned issue, Google introduced Federated Learning (FL) (Chen et al., 2021), which allows ML models to be trained without sharing raw data. In FL, participants train models locally on their devices and share only their weights with a server. The server collects the weights from various clients and merges them to create a global model that contains the knowledge of the merged ones. The study conducted by Rahman et al. (2020a) provides empirical evidence supporting the effectiveness of FL for cybersecurity. The author's findings demonstrate that FL achieves comparable results to centralized approaches, thereby validating FL as a viable and secure solution. Recent studies have proposed integrating FL in IoT environments as a solution for privacy concerns (Abdulrahman et al., 2021) while considering key aspects such as security (Wehbi et al., 2023; Rahman et al., 2020b), quality of service (Hammoud et al., 2022; Chahoud et al., 2023), and data management (Lamaazi et al., 2020). In addition, the

* Corresponding author at: Artificial Intelligence & Cyber Systems Research Center, Lebanese American University, Beirut, Lebanon.
E-mail address: azzam.mourad@lau.edu.lb (A. Mourad).

https://doi.org/10.1016/j.jnca.2025.104105
Received 5 September 2024; Received in revised form 30 November 2024; Accepted 9 January 2025
Available online 17 January 2025

work in Zhang et al. (2024), introduces a novel graph-based algorithm that employs graph sparsification technique to optimize communication efficiency while maintaining the security of resource-constrained IoT devices in the FL environment. However, although FL might be considered a solution to address IoT privacy, it does not yet address the main issue: whether devices can train ML models within a logical amount of time due to the constraints of computational resources.

For such a case, a new scheme was introduced in 2017 called Split Learning (SL) (Vepakomma et al., 2018a), similar to FL in addressing data privacy. However, in SL, devices train only part of the model, called the cut layer, and share the smashed data output with an external server that will complete model training on its resources. Using SL, it is possible to train complex and multilayered models on IoT devices considering that they will commit only a part of the training. However, SL mandates the server's involvement in the model training of each participant, which compels working in a synchronized manner. In the IoT context, the large number of devices renders SL integration impractical due to the substantial increase in completion time caused by the enforced synchronization with a server. Furthermore, using SL can create a bottleneck and a single point of failure issue.

Addressing the aforementioned limitation, the authors in Thapa et al. (2022), Wazzeh et al. (2024), Otoum et al. (2023) propose SplitFed to integrate FL parallelism into SL. Similar to FL, SplitFed allows the clients to train in parallel by assigning each to an external FL Server. When all clients complete the training, the main server collects client and server weights to aggregate the global model. It later sends it back, starting the next training round. Even though such an approach improves the training speed, it inherits all major FL issues, such as non-IID and stragglers. Non-IID is highly emphasized in Zhao et al. (2018a), Wazzeh et al. (2022), affecting the accuracy of the global model and increasing the time it takes to reach convergence. In addition, model aggregation usually occurs when all clients finish and send their weights to the server. Thus, clients who finish faster must wait for the rest of the clients to finish. In the context of IoT, the significant variety of devices' capabilities results in the presence of stragglers, substantially impacting the total time required to complete the task. A cluster-based approach is proposed in Wu et al. (2023) to address the stragglers' presence in the SplitFed IoT environment. The authors aim to solve the problem by clustering devices based on their properties, such as networks, available energy, and computing capabilities. However, such a clustering might result in non-IID clients being in the same cluster, which impacts the global model accuracy and increases the convergence time.

In this regard, SplitFed can play a major role in real-life scenarios. For example, in smart healthcare systems where tasks such as monitoring chronic conditions like diabetes or heart disease are required, a SplitFed approach enables clients to collaboratively train models on their resource-limited wearable devices, enhancing device accuracy while preserving their privacy. However, the models generated across different devices exhibit non-IID distributions due to varying user behaviors. Additionally, consumer-grade wearables may have limited computational power, resulting in straggler clients that delay training, while high-end monitoring equipment offers faster processing but contributes to system heterogeneity. This scenario necessitates an efficient method to address non-IID data and straggler issues, ensuring effective model training across diverse clients.

Several approaches have been proposed to address the challenges of non-IID data and stragglers in split federated learning environments. However, to the best of our knowledge, none of them effectively tackles both issues simultaneously. Furthermore, in this paper, we propose a novel two-stage clustering approach, comprising out-clusters and in-clusters, to address both the non-IID and straggler problems. Out-clusters are designed to tackle the non-IID issue by grouping clients based on their model weights, while in-clusters group clients based on their computational capabilities, ensuring that clients finish together and minimizing waiting times. Clients within each in-cluster train in parallel using the SplitFed strategy, followed by sequential training when transitioning between out-clusters. Additionally, a client selection mechanism is introduced to further enhance our approach by adapting to the dynamic nature of client capabilities in each round, thereby mitigating the impact of stragglers.

Following the out-cluster phase, weight divergence (Zhao et al., 2018a) is used as a criterion for clustering, ensuring that each cluster combines devices with minimal weight divergence. Such devices can be considered IID between themselves; thus, aggregating their models will have minimal impact on the cluster's global model.

In the in-cluster phase, we address the problem of heterogeneous capabilities among IoT devices. We perform additional clustering of clients' devices for each out-cluster obtained from the first phase. Devices within each in-cluster have similar computation capabilities, ensuring simultaneous task completion and reducing overall waiting time. With multiple in-clusters operating in parallel, faster clusters can complete more iterations than slower ones. In this context, we adopt an approach similar to FedAsync (Xie et al., 2019), allowing us to integrate delayed models into the cluster's global model.

Moreover, to further optimize runtime and efficiency, we implement a client selection strategy that adapts to the dynamic nature of client performance, aiming to predict and mitigate stragglers while considering their potential contribution to the global model. Our multi-objective selection criteria are applied through a genetic algorithm integration, identifying the most suitable clients for each training round. This approach focuses on available resources and the impact of the trained model updates on the quality of the global model, all within a manageable search time.

The main contributions are the following:

- Propose a combined Split and Federated approach to enable model training on limited resources on real-world IoT devices. Our approach employs a two-stage clustering mechanism to address the problems of non-IIDness and stragglers.
- Propose the first clustering stage, where the server groups non-IID clients into IID clusters that operate sequentially, thereby avoiding the aggregation of biased weights.
- Present a second-stage clustering approach to address the stragglers issues. In the second stage, clients' capabilities serve as input for clustering, enabling similar clients to complete their execution simultaneously, thereby reducing waiting time. Additionally, our approach incorporates asynchronous capabilities to enable parallelism between in-clusters.
- Propose a client selection algorithm that adapts to the dynamic shifts in clients' capabilities during execution. This algorithm employs clients' available resources to predict their execution time and considers their continuous model updates to assess their impact on the global model.
- Perform empirical experiments using the MNIST and CIFAR-10 datasets, with data simulated using a Dirichlet distribution to imitate real-world data distribution scenarios. This highlights the efficiency of our approach in enhancing accuracy despite the presence of biased client data. Additionally, our approach demonstrates significant improvement in achieving higher accuracy more quickly compared to alternative methods.

2. Literature work

FL was first introduced by Google (McMahan et al., 2017) in 2017 to address the privacy issues of using clients' data to train machine learning models. In their paper, the authors explain the advantages of using their architecture, which are proven by detailed experimentation. Moreover, the authors also described the non-IID problem in the context of local training with part aggregation. The authors stated that the issue can be internally solved at the cost of increasing training time and bandwidth costs. One of the first uses of the FL framework by

Google was in the FLOK system, which uses a kind of FL approach as a recommendation system for clients, mainly emphasizing on respecting their privacy. Later in the following year, split learning (Vepakomma et al., 2018b) was introduced as a solution that allows healthcare providers to collaboratively train ML models without sharing raw data. While both have the same premise of protecting users' privacy, both approaches have very contradicting workflows and target clients. FL is more tailored for environments with a huge number of devices, while SL works well in small environments. This mainly contributes to the enabled parallelism in the FL environment through aggregation countered by the forced synchronization under SL throughout sequential training. As a result, FL was faster but less accurate in non-IID environments due to aggregation, while SL was slower but more accurate due to the non-existing aggregation phase. Many recent approaches have been applied to both FL and SL in the same environment to take advantage of the benefits of both worlds. In the following sections, we detail related work that provides solutions under the context of non-IID, stragglers, and execution speed using one and both FL and SL architectures.

In the following subsections, we elaborate on the recently developed state-of-the-art work regarding FL, SL, and Split-Fed approaches.

2.1. Federated learning

To tackle the non-IID issues in FL, the authors in Zhao et al. (2018b) propose pretraining a booster model using raw data collected from participating clients. Moreover, the collected data is combined and distributed to the rest of the clients. A key issue in this architecture is that raw data is redistributed to the rest of the clients, breaching their privacy. Furthermore, the volume of duplicated data would increase, making model training more time-consuming and less cost-efficient.

An alternative approach to address non-IID data involves grouping clients into clusters. In their work (Briggs et al., 2020), the authors introduced a hierarchical clustering scheme for FL. Their method uses model weights as a parameter for the clustering mechanism, basing their results on the similarity between clients' models. Such an approach aims to reduce non-IID characteristics by grouping similar clients together. However, in this case, the cluster operates independently as a separate FL, heavily changing the structure and reducing the overall efficiency of the results.

In Chen et al. (2023), a proposal addresses the non-IID characteristic in FL with a sequential-based approach. Their approach employs a grouping scheme where clients are grouped randomly or based on specific parameters, such as geographic location. Training proceeds sequentially within each cluster, followed by aggregation across clusters. However, this method is susceptible to sequential learning issues due to the lack of shuffling, which is a significant drawback, especially when combined with the non-IID of clients' data. Additionally, the aggregation step between clusters could lead to biased model aggregation if the data from randomly clustered clients exhibits non-IID characteristics, as noted in Arafeh et al. (2022).

Furthermore, client selection prevails in FL approaches. In Luping et al. (2019), the authors propose an approach to measure the update relevancy to the global model. In such a case, relevance is defined by the number of local models holding the update direction as the estimated global model. Based on the relevancy measure, the authors aim to reduce communication costs by deciding whether model updates should be incorporated into the global model.

In Kang et al. (2019), the authors proposed a game theoretical approach based on contract theory to incentivize clients to participate in an FL task. Clients are rewarded based on the data size used to train the local model and the device's processing capability (CPU). As a result, clients with more data and superior computational powers have a higher selection probability. The main limitation of this approach is the focus on quantity rather than quality. The clients are selected based on the committed data size and processing speed, with little or no mention of the data distribution. That made them intolerant of clients who possess valuable information.

A client selection approach targeting the non-IID in FL is proposed by Duan et al. (2021). A mediator on the server was introduced to solve the global class imbalance of client distribution. Such a mediator adapts to the participants' distribution and requests from clients to update their data by up-scaling or down-sampling based on their distribution. However, access to clients' data distribution is required, causing inference on their private information and breaching their privacy. Similarly, the authors in Yang et al. (2022) address the non-IID challenges by introducing a class distribution estimation approach. This method utilizes client model weights and applies the Kullback–Leibler (KL) divergence to infer data distributions without accessing raw data. Additionally, a combinatorial multi-armed bandit (CMAB) algorithm is proposed to optimize the trade-off between exploration and exploitation, ensuring the selection of clients with balanced datasets. The Hadamard matrix is further employed to efficiently manage class distributions and guide client selection.

Limiting learning parameters plays a significant role in addressing non-IID issues in FL. For instance, the authors in Sahu et al. (2018) proposed the FedProx algorithm introducing a proximal term in the local trainer, acting as a regularization to penalize steep deviations from the global model. Similarly, the authors in Reddi et al. (2021) focused on server-side optimization by adding a regularization step directly after the aggregation to handle noisy gradient updates and dynamically adjust learning rates. While these approaches effectively mitigate non-IID challenges, they also introduce computational overhead. For instance, in Sahu et al. (2018), the proximal term calculation and application during training increases computation time, especially when considering more local epochs, making it less suitable for IoT scenarios where reducing computational load is the main objective.

While existing approaches have successfully addressed the non-IID problem in federated learning, they face challenges when modifying the system's structure. These challenges arise when introducing third parties to manage client distribution, completely isolating clients based on their data distributions, or excluding clients based on their computational weaknesses. In contrast, our approach aims to incorporate all clients without introducing significant complexity to the federated learning environment. This ensures compatibility with other methods while supporting deployment within the split learning workflow.

2.2. Split learning

Due to its ability to address privacy and security concerns, SL has recently gained significant influence and is widely used by industry and academia. Additionally, SL opens opportunities to overcome challenging issues such as limited capabilities, bandwidth, and connectivity, making it especially useful in IoT environments. Numerous works have recently been published that improve SL capabilities to train ML models using IoT devices. For instance, the authors in Ayad et al. (2021) aim to improve the communication efficiency between clients and servers by introducing an auto-encoder that generates a sizable representation of the cut layer output. Furthermore, the authors in Karjee et al. (2021) propose a dynamic model partitioning that finds the optimal cut layer selection to reduce computation latency and computational resources. According to Wang et al. (2022), SL can be used as an efficient scheme with an advantage over the convergence rate while reducing computation time. However, despite the increasing interest, SL still needs to improve its time efficiency. Adding to the privacy of SL, authors in Vepakomma et al. (2018b) propose a security enhancement architecture that makes it possible for the server to complete training the remaining layers without clients sharing their labels. However, SL only works sequentially due to its forced communication with the server. Additionally, since SL relies heavily on its clients' communication and computation capabilities, its applicability in the Internet of Things, which mainly contains heterogeneous devices, is highly affected.

2.3. Split-fed approaches

A combination of FL and SL approaches was first introduced by Thapa et al. (2022). The authors introduced an architecture with two servers, a fed server and a main server. The clients sync their weights in parallel with the main server that handles the computation of the remaining layers and a portion of the backward propagation. Once all clients finish training, the main server collects the clients' cut layers in addition to the trained weights existing in the main server to create a global model of each and share them back with the designated parties. While the proposed approach addressed the relay-based communication drawbacks, it still inherited major issues of using FL, such as non-IID client distribution and the presence of stragglers limiting its applicability in the IoT environment.

Recently, various solutions have been introduced to address these issues. For instance, addressing the existence and heterogeneity of the stragglers, the authors in Wu et al. (2023) propose a clustering mechanism based on the properties of the clients' devices with similar training times into the same group. Additionally, they suggest a dynamic cut layer identification to reduce the training burden on devices with low computational power. Moreover, their main contribution lies in the workflow of their proposal, which makes only the devices within the same cluster run in parallel while the movement from one cluster to another runs sequentially. The authors in Wu et al. (2021) propose an offloading strategy based on the partitioning property provided using SL in an FL context to address the heterogeneity of IoT devices. By employing reinforcement learning, they could dynamically identify the optimal cut layer for each specific client. Similarly, the authors (Samikwa et al., 2022) propose a dynamic split point approach based on network throughput and computation resources. Although the mentioned proposal could handle the straggler issues, they did not focus mainly on targeting the problem of non-IID distribution between clients, which can significantly impact the convergence time when working in an FL environment.

While previous approaches have typically treated the straggler and non-IID problems as separate issues, these problems are interrelated, as both significantly affect accuracy and convergence. Thus, our approach tackles both challenges simultaneously by combining client grouping based on computational capabilities with strategies that address data heterogeneity. This enables balanced and efficient training. Unlike prior works, we prioritize maintaining compatibility with the SplitFed framework while improving convergence rates and reducing training delays.

3. Proposed scheme

3.1. Overview

Our proposed scheme combines the benefits of each of the following technologies: Federated Learning and Split Learning. Fig. 1 depicts our architecture. It comprises three key components: Main Server, Split Servers, and Clients. Our approach comprises performing two-level clustering: Out-Clusters and In-Clusters. The in-clusters train part of the model in parallel at different speeds. Every In-Cluster has its own Split Server, thus eliminating the single point of failure issue. Some clusters can perform more training iterations compared to others. The trained part-of-weights are shared by the split servers asynchronously with the Main Server. In the following, we will elaborate on the logic behind the proposed clustering mechanism while describing each component in the architecture in detail.

The first clustering stage, the out-cluster, is based on clients' weight divergence calculated from their models collected during the initial stage. Client weight divergence has proven effective in accurately identifying client clusters using only shared model weights without requiring access to raw client data (Briggs et al., 2020; Arafeh et al., 2022). This clustering step ensures that clients within the same group have minimal weight divergence, effectively grouping them as IID clients. As a result, these clients can work in parallel, as aggregating their weights will not negatively impact the global model. In contrast, moving from one out-cluster to another occurs sequentially due to the significant weight divergence, which, according to Zhao et al. (2018a), greatly impacts the global model by increasing the convergence time and reducing the overall accuracy.

In the In-Cluster, we cluster the devices based on the amount of computing resources they have. Devices run in parallel in the In-Cluster after performing client selection to increase their availability and reduce training time. Part of the machine learning model is passed to the in-clusters for training. In the typical FL approach, in each round, clients that complete first must wait for the last clients to finish before the server aggregates their model and starts the next round. To overcome this issue, we build in-clusters to partition the clients with similar computation capacity and completion time. Moreover, using a similar approach to FedAsync, the main server can integrate the delayed in-cluster aggregated weight to the global model by diluting them based on the stalled rounds defined by the number of rounds the in-cluster devices missed the aggregation phase.

3.2. Components description

In the following, we describe each of the three components in the proposed scheme: Clients, Split Server, and Main Server.

- **Clients:** are part of the splitfed architecture tasked to train the first part of the model using their local dataset. Later, clients continue the rest of the backward propagation and update the local gradient. Both tasks are carried out in collaboration with a dedicated server assigned to each client.
- **Split Server:** in the splitfed architecture, each client is assigned its own SL server to ensure independent and parallel model training. This differs from traditional SL methods, where all clients rely on a single shared server, leading to synchronized communication and server lockouts. By assigning dedicated servers, clients can train their models asynchronously and in parallel, reducing the load on a single server and decreasing overall completion time through enabled parallelism.
- **Main Server:** is a single entity responsible for coordinating sequential cluster execution and interacting with split servers to perform model aggregation. When the aggregation is due, the main server collects both weights $\{w_c; w_s\}$ from each split server included in the training and aggregates them to generate the new global models. The global models are then transferred to the next cluster assigned to the training.

3.3. Proposed workflow

In this section, we outline the design workflow of our approach, which is summarized in Fig. 2 as a flowchart illustrating the steps in their order of execution. The workflow consists of the following steps:

1. Out-Cluster Selection. The main server starts by shuffling the deferred clusters, selects one randomly, and then distributes the global model to its split servers. The underlying mechanism behind the limited selection and random shuffling of clusters aims to promote unbiased training generalization and mitigate overfitting. Moreover, cluster shuffling acts as a replacement for the missing data shuffling steps traditionally used in centralized mechanisms.
2. In-Cluster Client Selection. Following our approach outlined in the subsequent Section 5, our in-cluster client selection process leverages client-specific data, including available resources and historical model update records. It is important to note that the information utilized in this process does not include sensitive data and will be used to identify a sub-optimal set of clients that minimizes execution time and fastens the convergence rate.

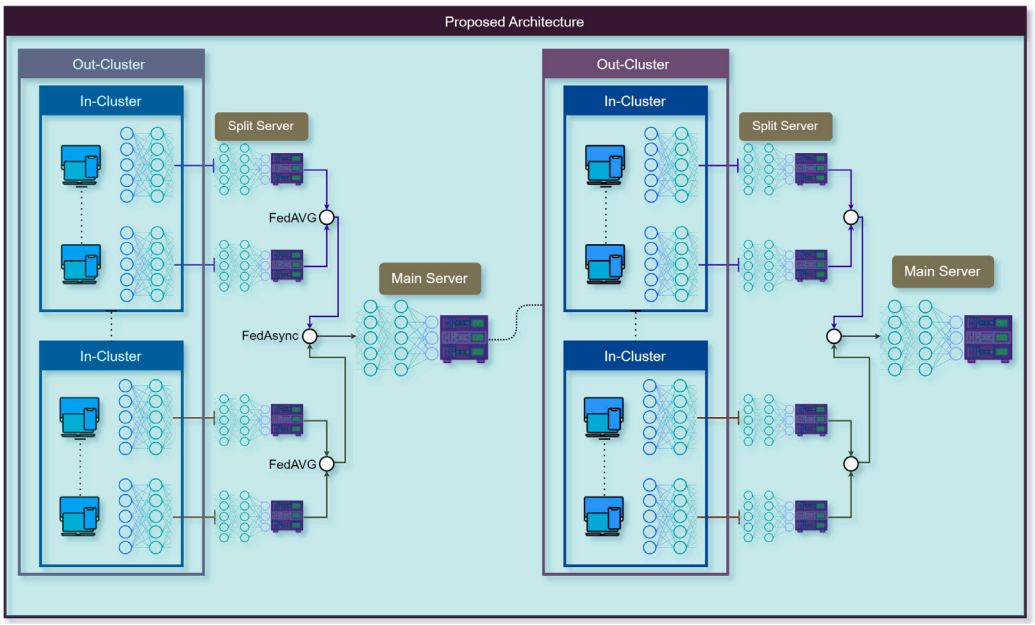


Fig. 1. Framework architecture.

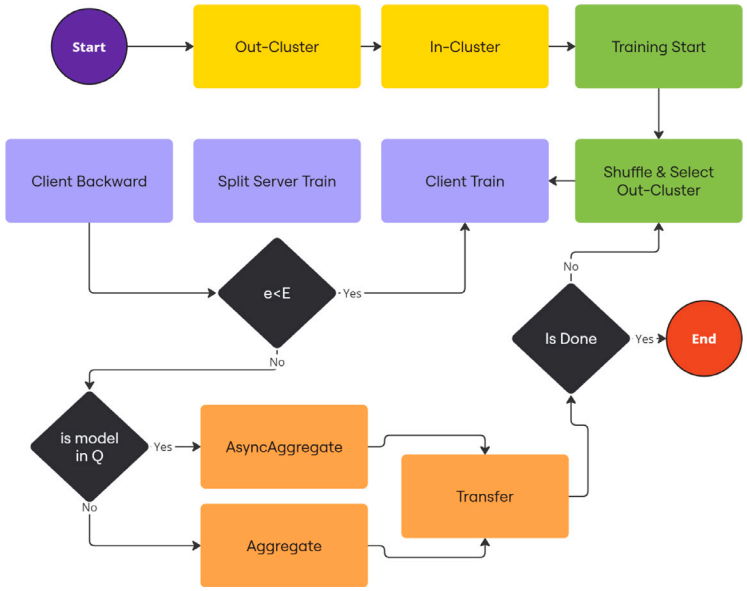


Fig. 2. Workflow execution flowchart. The condition e < E checks whether the trainer has completed all the assigned epochs. The condition is model in Q verifies if there is an older model in the queue to be used for asynchronous aggregation.

3. Training. In this step, the training process is carried out in three phases:
   (a) Client Training. The cluster's selected clients feed-forward their model to the specified cut layer in parallel, generating the smashed data and sending it to their assigned split server.
   (b) Split Train. Split Servers infer the remaining weights using their server model to generate the prediction result drawn from the label samples. Following this step, Split Server computes the loss and performs backward propagation up to the last server layer, which generates the gradients. These gradients are then sent back to the clients so they can continue the process on their layers.
   (c) Client Backward. Each client uses the received gradients to complete the backward propagation and update its optimizer. Steps 3.1–3.3 are repeated in parallel for all in-cluster clients for a specified number of times (split epochs).
4. Aggregation. When at least one cluster's clients finish training, the assigned split servers send the weights to the main server, which will do the aggregation.
5. FedAsync. If the cluster queue contains aggregated weights from another cluster, the main server integrates them following the FedAsync as described in Eq. (4). This step marks the end of the training with one out-cluster.
6. Transfer. After reshuffling, the main server transfers the global model to another randomly selected cluster, repeating Steps 1–6. The main server completes one round once the global model has gone through all out-clusters.

7. Completion. The main server repeats steps 1–7 till it achieves the predefined stopping criteria.

4. Clustering, & SplitFed algorithms

4.1. Clustering

The set of available client devices is denoted by $C = \{c_1, c_2, \ldots, c_N\}$ where N denotes the total number of clients. Each client receives a partial model weight represented by w_c, part of the whole model weights denoted by $w = \{w_c; w_s\}$. Furthermore, the local dataset is denoted as $D_j^{c_i} = \{x_j, y_j\}$, where x_j is a row of raw features input, while y_j represents the row label. We denote the set of Split Servers by $S = \{s_1, s_2, \ldots, s_N\}$ where $|S| = |C| = N$. The server weights is denoted as w_s.

Initialization Step: During the initialization step, the server starts with a randomly generated model's weights and shares them with the clients. Each client trains its model with its assigned split server, which will send the results to the main server. In addition to the weights, the main server collects the clients' capabilities, such as computation speed, available energy, and network resources. Upon receiving all the required information, the server proceeds with the clustering using the KMeans algorithm (Jain and Dubes, 1988). In the following, we mathematically formulate the objectives of the first and second clustering algorithms, denoted as F_1 and F_2, respectively.

Clustering Objective $\mathbf{F_1}$: aims to minimize the impact of non-IID clients, ensuring that clients with similar model weights are grouped together. This enables parallel operation without negatively affecting the accuracy of the global model. The results of client clustering are represented by $V = \{v_1, v_2, \ldots, v_n\}$, where each v_i represents a cluster of clients whose weights are close to each other. By grouping clients with similar weights, we ensure that these clients can train simultaneously without introducing significant bias into the global model. F_1 is mathematically formulated as:

$$F_1 = \underset{V}{arg\,\min} \left(\sum_{k=1}^{n} \sum_{v_a \in V} \left\| w_{v_a} - \mu_k \right\|^2 \right) \tag{1}$$

where w_{v_a} are the merged client and server weights, μ_k is the cluster centroid, and $\|w_i - \mu_k\|^2$ represents the squared Euclidean distance between the client weights and the cluster centroid. This distance reflects how similar or dissimilar a client is to the cluster's center.

Clustering Objective $\mathbf{F_2}$: aims to minimize the idle time of faster clients by clustering clients based on their resource capacities. Clients with similar resources (e.g., CPU, RAM, Disk space, Battery) are grouped together so that they can perform tasks simultaneously, effectively minimizing idle times and improving the overall system's execution time. $v_a \in V, v_a = v_1^a, v_2^a, \ldots, v_p^a$ are internal clusters of clients built based on the clients' resource capacities. In this context, resource capacities are the maximum value of resources a client can access. Objective $\mathbf{F_2}$ is mathematically formulated as follows:

$R'_{v_i^a,\text{CPU}}$ = CPU capacity on client v_i^a.

$R'_{v_i^a,\text{RAM}}$ = Memory capacity on client v_i^a.

$R'_{v_i^a,\text{Disk}}$ = Disk space capacity on client v_i^a.

$R'_{v_i^a,\text{Bat}}$ = Battery capacity on client v_i^a.

$$R'_{v_i^a} = [R'_{v_i^a,\text{CPU}}, R'_{v_i^a,\text{RAM}}, R'_{v_i^a,\text{Disk}}, R'_{v_i^a,\text{Bat}}]$$

$$F_2 = \underset{v_a}{arg\,\min} \left(\sum_{m=1}^{p} \sum_{v_i^a \in v_a} \left\| R'_{i,v_a} - \nu_{cl}^m \right\|^2 \right) \tag{2}$$

where ν_{cl}^m represents the centroid for resource assignments in cluster m, representing the average resource capacities of the clients in that cluster, and $\left\| R'_i - \nu_{cl}^m \right\|^2$ measure the squared Euclidean distance between a client's resource vector and the cluster centroid. This distance quantifies the similarity between a client's resources and the typical resources in the cluster.

4.2. SplitFed algorithm

In Algorithm 1, we show the steps followed in our proposed SplitFed Algorithm as described in Section 3.3. In this algorithm, we start first by performing a one-epoch training, denoted as `one_epoch_training()`, to get an initial insight into clients' weights that will be used further for the iid clustering. In the following step, `iid_clustering()` refers to the first layer of clustering, aligned with objective F_1. Subsequently, we measure clients' resource capacity to be used in resource clustering through `resource_capacities()`. Once these three steps are performed, we start a loop of *nb_rounds*. The second layer of clustering starts by looping over the clients of each outer cluster by calling `resource_clustering()`, aligned with objective F_2.

After obtaining the outer and inner clusters, we start the core of the splitfed algorithm. In each iteration, we perform `suffle_select_next_cluster()` to select the next IID cluster, promoting unbiased training while ensuring randomness and mitigating overfitting. Using the obtained inner clusters, we obtain the available resources of each client by invoking the `update_resources()` function. Afterward, we leverage the selection algorithm through the function described in Section 5 to perform client selection based on clients' available resources. The remaining steps are related to model training and asynchronous aggregation, which we detail next.

Algorithm 1: Double Clustered SplitFed

```
Data: Clients (C)
C.models ← one_epoch_training();
V ← iid_clustering(C.models);
C.R' ← resource_capacities();
foreach C_{v_a} in V do
    v_j^a ← resource_clustering(C_{v_a}.R');
// C_{v_a} are clients of v_a
for r in nb_rounds do
    v_i ← shuffle_select_next_cluster();
    while v_i not None do
        // in sequence
        foreach v_i^a in v_i do
            // in parallel
            v_i^a.R ← update_resources();
            C' ← selection(v_i^a);
            foreach (c_i, s_i) in C' do
                // in parallel
                c_i.feed_forward();
                s_i.feed_forward();
                s_i.backpropagate();
                c_i.backpropagate();
                c_i.optimizer.step();
            aggregate(v_i^a.model, C');
        async(v_i.model, v_i^a.model, α);
        v_i ← shuffle_select_next_cluster();
```

Our approach combines a client $c_i \in C$ and split server $s_i \in S$ to train the model's weights $\mathbf{w} = \{w_c; w_s\}$ asynchronously without sharing raw features with an outside entity (such as a split server or main server). Considering each client's dataset D^{c_i}, the main aim is to identify the optimal model parameters w_c by minimizing the loss function $\mathcal{L}(\mathbf{w}; D^{c_i})$ defined by:

$$\mathcal{L}(\mathbf{w}; D^{c_i}) = \sum_{j=1}^{|D^{c_i}|} f(\mathbf{x}_j, y_j, \mathbf{w}) \tag{3}$$

where f signifies the loss function, which depends on the type of dataset and application we are working on.

After clients complete the training process with their respective split servers, the servers share the weights w_s with the main server. The server then waits for all the clients' of the same clusters to finish and then proceeds with the federated averaging aggregation as in the following:

$$w_{t+1}^{v_a} = \sum_{i=1}^{v_a} \frac{n^i}{n^{v_a}} w_t^i, \tag{4}$$

where $w_{t+1}^{v_a}$ refers to the new global model of cluster v_a while w_{t+1}^i refers to the weight of inner cluster v_a for client i. Furthermore, n^{v_a} corresponds to the total number of rows trained for cluster v_a, while n^i resembles the number of rows trained for client i in the cluster v_a.

The resulting global model is added to a queue when a cluster finishes aggregation. Whenever a faster cluster finishes while there is a global model queued from the slower one, it will proceed with the delayed aggregation according to the following:

$$w_t^{v_a, v'_a} = (1-\alpha) w_t^{v_a} + \alpha w_{t-s}^{v'_a} \tag{5}$$

where $\mathbf{w}_t^{v_a}$ is the fast cluster model at round t, $\mathbf{w}_{t-s}^{v'_a}$ is the slow cluster model at round $t-s$, where s represents the stalled rounds, which correspond to the difference between the round of the faster cluster and the slower cluster. $\alpha = 1/(1+s)$ represents dilution factors on late cluster model (Xie et al., 2019).

5. Client selection problem

5.1. Problem definition

In distributed learning environments involving parallel execution, the completion time of a group of devices working concurrently is determined by the slowest device within the cluster. This issue becomes more noticeable in the context of IoT devices, where the availability of each device goes through frequent and constant changes. While utilizing device resource capacity helps estimate potential execution times, the dynamic resource shifts occurring during the execution significantly impact the overall execution time while increasing the idle time of faster devices. Additionally, the limitation in terms of resources increases the impact, such as minor background tasks running in parallel, which can cause delays in task completion and subsequently affect the entire cluster's execution time.

On the other hand, data availability on each device varies. A device possesses more data, which makes it more important to be selected for more rounds while allowing the global model more opportunities to improve its weight. Meanwhile, other devices might also have uncommon knowledge, such as unique features or classes that are unavailable on most clients' devices. Furthermore, selecting all or a random number of clients in each round is insufficient as it can cause a significant increase in execution time.

In line with the aforementioned problems, we extend our double clustering approach with a selection mechanism that takes advantage of both previous clustering stages for selecting a subset of clients in each round. Our main objective is to reduce the round execution time while also aiming to select a combination of clients with a higher positive impact on the global model.

5.2. Problem formulation

Client selection happens during each round of execution. $\forall i \in v_a : L = \{l_1, l_2, \dots, l_m\}$, where $l_i \in [0,1]$ for $1 \le i \le m$ indicates whether a client is selected to participate in a given round. The client selection process submits to the following objectives:

Objective F_3: seeks to maximize the number of selected clients capable of improving the model convergence.

$$F_3 = \max(\sum_{i=0}^{|L|} l_i) \tag{6}$$

Objective F_4: seeks to mitigate the impact of stragglers in each round. Given the fluctuating availability of clients' resources, our approach involves a per-round client selection process, focusing on their individual resource availability. This selection mechanism excludes clients that may introduce delays or face challenges in executing the training procedures due to insufficient resources.

$R_{v_{i,t}^a,\text{CPU}}$ = CPU available on client v_i^a at round t.

$R_{v_{i,t}^a,\text{RAM}}$ = Memory available on client v_i^a at round t.

$R_{v_{i,t}^a,\text{Disk}}$ = Disk space available on client v_i^a at round t.

$R_{v_{i,t}^a,\text{Bat}}$ = Battery available on client v_i^a at round t.

$R_{v_{i,t}^a,\text{S}}$ = Dataset sample size on client v_i^a at round t.

$$R_{v_{i,t}^a} = [R_{v_{i,t}^a,\text{CPU}}, R_{v_{i,t}^a,\text{RAM}}, R_{v_{i,t}^a,\text{Disk}}, R_{v_{i,t}^a,\text{Bat}}, R_{v_{i,t}^a,\text{S}}]$$

$$\forall i \in v^a : F_4 = \min(\sum_{i=0}^{|L|} l_i \times v_{i,t,\text{exec}}^a) \tag{7}$$

where $v_{i,t,\text{exec}}^a = f_\theta(R_i)$ is the execution time prediction of client v_i^a at time t using a pretrained three-layer neural network model. We build this model by collecting data from our simulated environment.

Objective F_5: is designed to incentivize clients with more substantial updates. The main objective is maximizing the dissimilarity between the newly trained client model and the last server global model. A higher value indicates a more significant update.

$$F_5 = \max\left(\sum_{i=0}^{|L|} l_i \times \|w_{c,t}^{v_i^a} - w_{s,t-1}^{v_a}\|\right) \tag{8}$$

where $\|w_{c,t}^{v_i^a} - w_{s,t-1}^{v_a}\|$ resembles the Euclidean distance between a server and a client.

5.3. Genetic-based client selection

Addressing the complexity of the client selector, we employ a Genetic Algorithm (GA) for its capability to identify a sub-optimal solution within a controlled time frame. Furthermore, GA enables our selector to explore a large portion of the quality solutions involving a combination of clients capable of working in parallel, finishing simultaneously, and substantially improving the global model. Our genetic application seeks a sub-optimal solution that minimizes the aggregation of given objectives. In the following, we provide detailed insights into three key components: initial population, fitness evaluation, and evolution.

Initial Population: We start the process by building the initial population, which consists of chromosomes created through an array of genes. In our scenario, each gene references a client within a chromosome encompassing a subset C' of in-cluster v_a. We aim to validate the effectiveness of their combined models while reducing their impact on the execution time.

Fitness Evaluation: During the evaluation step, we measure each chromosome based on the objectives detailed in Section 5.2 and aggregate the results using the following equation:

$$S = \frac{F_4}{F_3 \times W_{f3} + F_5 \times W_{f5}} \tag{9}$$

where W_{f3} and W_{f5} are the weights assigned to the objectives F_3 and F_5, and they sum to one. In addition, S represents the score calculated for each chromosome. Our goal is to minimize F_4, which considers the combined execution time, while simultaneously maximizing both

F_3, related to the number of selected clients, and F_5, reflecting the chromosome clients' impact on the global model.

Evolution: In the following, we cover the evaluation, selection, crossover, and mutation procedures outlined in Algorithm 2. After building the initial population, we score each chromosome using the proposed fitness function. Subsequently, a filtering procedure is employed, using the Roulette Wheel (Goldberg, 1989) method to retain part of the population. The roulette selection approach assigns a probability of being selected for the next iteration to each gene based on its score.

Algorithm 2: Genetic Algorithm Implementation

```
input : genes = clients, max_iter, r_cross, r_mut
population ← initial_population(genes, population_size,
                                chromosome_size)
solution ← ∅
n_iter ← 0
while n_iter < max_iter do
    scores ← fitness(population)
    solution ← argmin(scores)
    population ← wheel(population, scores)
    children ← ∅
    for i in range(start=0, stop=|population|, step=2) do
        child1 ← population[i]
        child2 ← population[i + 1]
        for c in crossover(child1, child2, r_cross) do
            mutation(c, genes, r_mut)
            children ← children.add(c)
    n_iter ← n_iter + 1
return solution
```

A crossover follows after the selection step, occurring between two chromosomes when a randomly generated float number, ranging from 0 to 1, exceeds a predetermined crossover ratio specified in the GA configuration. During a crossover event, both chromosomes are randomly cut at a specific position, and their respective partitions swap places. Subsequently, the mutation process follows a similar structure, involving a mutation ratio for each gene within a chromosome. Mutation involves swapping an individual gene with another randomly selected from the pool of clients.

These steps are repeated until a satisfactory fitness score is achieved or until we iterate for a specified number of times. Finally, the outcome of genetic selection is a chromosome, a collection of clients whose combined characteristics adhere the most to the given objectives in Section 5.2.

6. Experimental results

In this section, we evaluate the performance of our architectures compared to similar approaches. We used Python as a development environment and PyTorch for model training. Our experiments were conducted on a Linux operating system with an NVIDIA A100 GPU featuring 16 GB of VRAM, a 64-core Intel Xeon CPU, and 128 GB of RAM. For the Federated Learning environment, we utilized the ModularFed (Arafeh et al., 2023) library, which facilitates the integration of distributed training environments for ML models, manages non-IID client data distribution, and simulates clients with varying processing speeds.

We compare our approach to four other approaches: SplitLearning (Split) (Vepakomma et al., 2018b), SplitFed (Thapa et al., 2022), One Layer SplitFed (Wu et al., 2023), and traditional Federated Learning using FedAVG for aggregation. Furthermore, our approach includes two variations: Two Layers Clustering Standard (2LS) and Two Layers Clustering Optimized (2LO), which uses our proposed genetic-based clustering algorithm. The comparison is based on three metrics:

Table 1
Hyperparameters used in the experiments.

Hyperparameter	Value
Learning Rate (client)	0.001
Learning Rate (server)	0.001
Batch Size	100
Epochs	1
Dirichlet Distribution α	0.5
Population Size (Genetic)	20
Max Iterations (Genetic)	50
Mutation Ratio (Genetic)	0.05
Crossover Ratio (Genetic)	0.1
W_{f3}, W_{f5} (Genetic)	0.6, 0.4

- Accuracy per round: the traditional measurement approach to monitor accuracy progress based on the number of rounds. It is important to note that the definition of a round differs for each approach, making these experiments effective for tracking accuracy progress but not reflective of training speed, as each approach has different structural processing. For instance, FL and SFL work completely in parallel, Split works sequentially, 1LS is semi-parallel, and 2LS and 2LO are semi-parallel with asynchronous capabilities.
- Accuracy per time: the primary metric used for our results, monitoring the evolution of accuracy based on the cumulative elapsed time taken between two accuracy measurements.
- Time per round: to show how much time each approach took to complete full iteration during each round.

Regarding the datasets, we used MNIST[1] and CIFAR-10 for conducting the experiments. The MNIST dataset contains images of handwritten digits between 0 and 9, with 60,000 samples for training and 10,000 for testing. CIFAR-10 has a similar size, containing images of 32×32 pixels divided into ten labels describing real-world objects and animals. As for the non-IID data distribution, we opt for Dirichlet Distribution (Kotz et al., 2000) as it emphasizes data distribution in the real world. The non-IID aspect of the data in Dirichlet can be controlled by a concentration parameter α. Dirichlet distributions with small α parameters tend to be sparse. This means that a few categories will dominate the distribution because the probability of observing specific categories is significantly higher than others. In our experiments, we set $\alpha = 0.5$, which creates a highly skewed, non-IID distribution with 120 clients, effectively simulating the imbalanced data distribution typically encountered in real-world scenarios.

In our simulation, we underline the importance of device heterogeneity within the environment. We integrate an additional module into the existing framework, extending traditional training clients with functionalities to measure and account for clients' processing speeds. Our client simulator can create clients with varying capabilities such as RAM, CPU, and Disk size. These parameters are drawn from a multivariate normal distribution, considering expected value ranges based on given means and standard deviations.

To ensure realistic dependencies between these parameters, we introduce correlations using a predefined correlation matrix, ensuring that higher memory correlates with higher processing speed, with some probability of exceptions. The generated parameters are scaled according to the given speed parameter β. We used β values of 0.1, 0.25, and 1 for our experiments to generate the devices, capturing a range of training environments. Furthermore, we introduce a new variable, ψ, to control the probability of having straggler clients in each round. Straggler clients are characterized by using only a fraction of their resource capacity for ML training purposes. We fixed $\psi = 0.05$ for the experiments.

[1] http://yann.lecun.com/exdb/mnist/

Table 2
Split models' configurations.

Model component	Layers
Mnist Client	nn.Linear(784, 1024) nn.Linear(1024, 1024)
Mnist Server	nn.Linear(1024, 1024) nn.Linear(1024, 10)
Cifar Client	nn.Conv2d(3, 64, 3, padding=1) nn.Conv2d(64, 128, 3, padding=1)
Cifar Server	nn.Conv2d(128, 256, 3, padding=1) nn.Linear(256 * 4 * 4, 1024) nn.Linear(1024, 512) nn.Linear(512, 256) nn.Linear(256, 10)

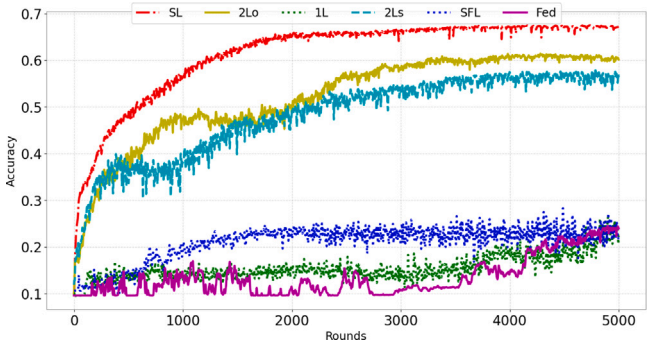


Fig. 3. Cifar accuracy/round.

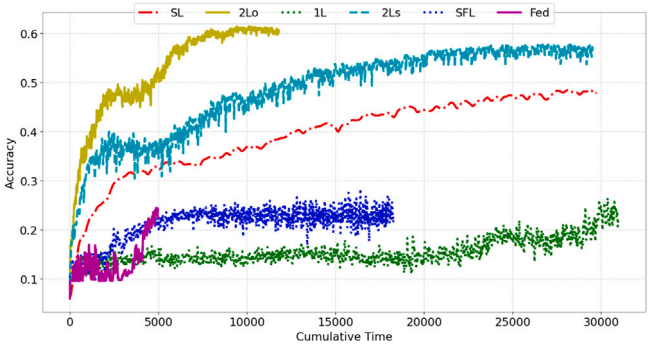


Fig. 4. Cifar accuracy/time.

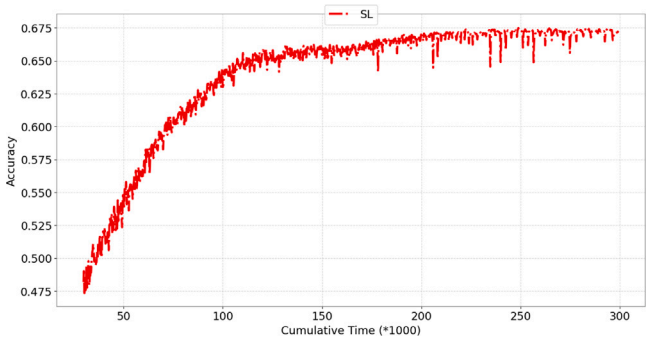


Fig. 5. Cifar accuracy/time (split continuation).

Regarding the GA hyperparameters, as presented in Table 1, we set the number of iterations to 50 to ensure the server completes the selection process before the previously selected clients finish training. The mutation and crossover ratios as fixed to 0.05, and 0.1, respectively. Furthermore, the weights for the genetic objectives (W_{f3}, W_{f5}) are fixed to 0.6, and 0.4, respectively, and remain consistent across all experiments. These weights were chosen to prioritize client diversity while also ensuring that the algorithm focuses on maximizing the number of selected clients for each cluster. Given that the number of available clients in the experiment is relatively small, we had to assign a higher weight to W_{f3} to emphasize the importance of having a larger number of clients in each selection.

Table 2 presents the model configurations used for each dataset. For the MNIST dataset, a model with two hidden layers achieved good results due to the dataset's simplicity and cleanliness. In contrast, the CIFAR10 dataset required a more complex architecture, including CNN layers, to effectively recognize the images. Both models were split to ensure that low-capability devices could handle the training. We assigned the minimum number of hidden layers to the clients for both MNIST and CIFAR while delegating the remaining layers to the split server.

6.1. CIFAR experiments

In Fig. 3, we show the evolution of accuracy in terms of rounds. SL was shown to be the most efficient mainly due to the absence of aggregation and non-IID effects, achieving 0.68 accuracy at the end, followed by our improved approach 2LO and the standard integration without genetic selection at 0.62 and 0.58, respectively. Furthermore, the complexity of the dataset, combined with the non-IID distribution, had a significant impact on the SFL and Fed methods due to the aggregation of biased weights. This effect was mitigated in our approach due to the out-clustering mechanism combined with the sequential workflow.

Figs. 4 and 5 describe the evolution of accuracy in terms of time, drawn from the same experiment as Fig. 3. This representation highlights the impact of the straggler's existence in the environment and shows the efficiency in reaching higher accuracy and convergence when considering time. As shown in Fig. 6, 2LO is the fastest to finish at 11800 Time Units (TU) when using the SL mechanism, followed by SFL at 18200 TU. Both 2LS and 1L had similar completion times at 28000 TU and 31000 TU, respectively.

The most notable aspect of this experiment was the accuracy achieved by each approach over time. For instance, while Fed was the fastest, it couldn't achieve good accuracy given the 5000 rounds it worked on. Furthermore, 2LO was the first to achieve a key point of 0.5 at 2700 TU compared to SL, achieving similar accuracy at 35000 TU and 2LS at 12000 TU. This shows the significant improvement from addressing the straggler's issues using our genetic algorithm enhancement.

Fig. 5 continues the SL, drawn separately for a clearer representation of the experiments.

Fig. 6 illustrates the evolution of round completion time, highlighting the effect of stragglers when comparing our approach to others. As shown in the figure, the forced synchronization made SL take the most time per round, averaging 60000 TU/R, while cycling through the training clients. The fastest method is Fed, followed by our approach, and then SFL, averaging 1200 TU/R, 2300 TU/R, and 3500 TU/R, respectively. By addressing the high amount of time caused by stragglers, our approach is able to achieve similar results in terms of speed when compared to parallel training methods.

6.2. MNIST experiments

In this section, we present the experiments conducted on the MNIST dataset. Fig. 7 shows the accuracy evolution in terms of rounds. As shown in the figure, the SL approach reaches a high accuracy of 0.94 in just 16 rounds; however, it subsequently dropped to 0.189 due to

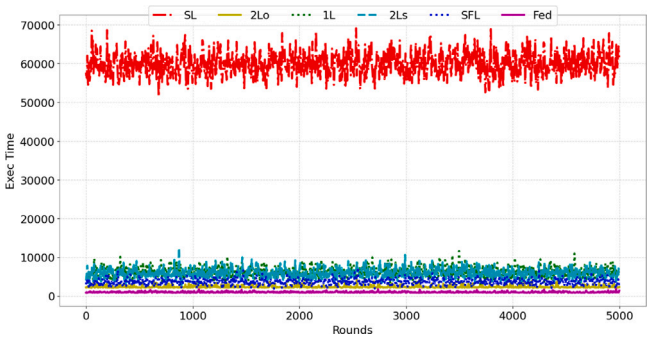


Fig. 6. Cifar round-time.

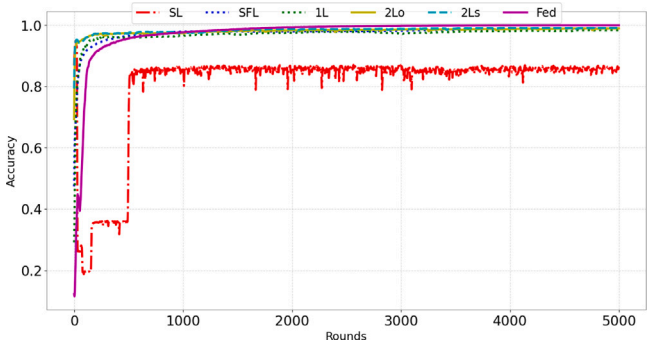


Fig. 7. MNIST accuracy/round.

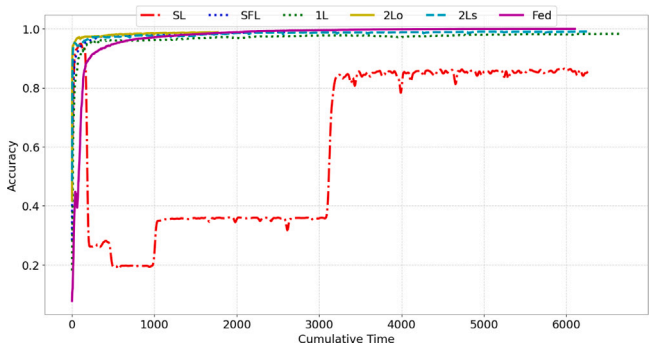


Fig. 8. MNIST accuracy/time.

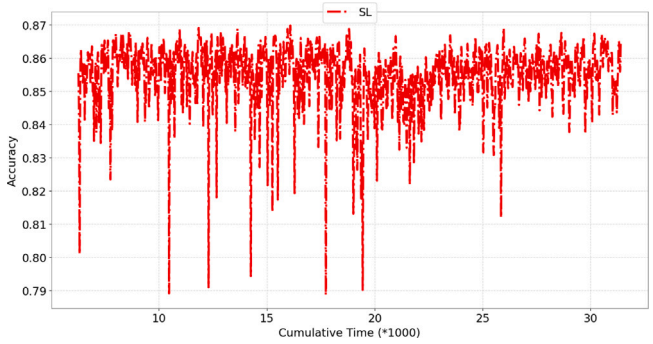


Fig. 9. MNIST accuracy/time (split continuation).

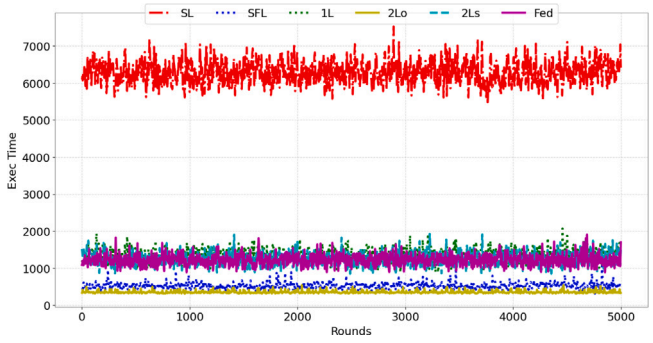


Fig. 10. MNIST time/round.

the lack of shuffling in sequential learning, amplified by the non-IID nature of the client's data. Eventually, the split method converges at an accuracy of 0.85 for the remainder of the experiment. In contrast, the approaches 1L, 2LO, 2LS, and SFL finished with 0.99 accuracy at the end of the experiment.

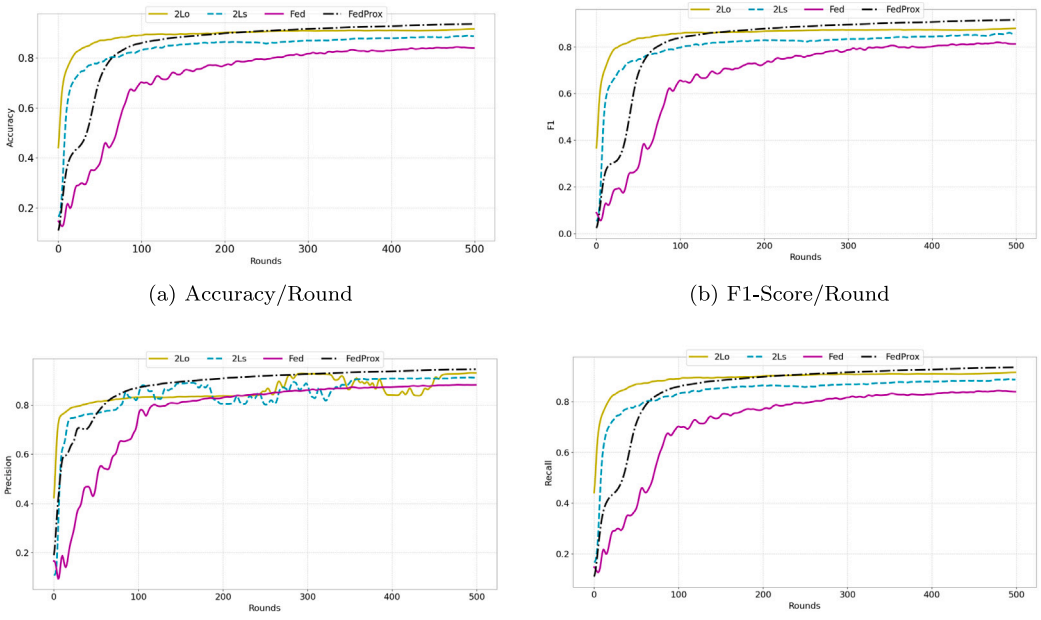
Taking a key accuracy of 0.95 as a benchmark for comparison, we observe that the 2LS method reaches this accuracy in 17 rounds, significantly faster than 64 rounds for 2LO, 253 rounds for 1L, and 403 rounds for SFL. This improvement is primarily attributed to the introduction of the GA mechanism in the 2LO approach, which uses a weight-based objective to diversify the weights of selected clients in each round within the same cluster.

Figs. 8–10 are drawn from the same experiments while presenting different perspectives. Figs. 8 and 9 show the evolution of accuracy in terms of accumulated execution time, which can give more insight into the implication of stragglers in the environment. For instance, SL finishes at 31,420 TU. Furthermore, the Fed approach completes at 434 TU. Besides, the 2LS method reaches the same accuracy in 6,239 TU, while 2LO, 1L, and SFL require 1,776 TU, 6,663 TU, and 2,581 TU, respectively. According to the results, 2LO is the first to achieve 0.96 accuracy at 42 TU, compared to 2LS at time 168 TU, 1L at time 347 TU, and SFL at time 209 TU. These results highlight the efficiency of the 2LO in achieving high accuracy in shorter time frames. Fig. 9 continues Fig. 8 for the SL approach, which is separated for a clearer representation of plots.

Furthermore, 2LO emerges as the fastest method, followed closely by SFL. The 2LO method's speed can be attributed to its ability to predict and exclude stragglers from the selection process. Further noticeable key moments are shown by the asynchronous capabilities of the 2LS approach, which also plays a crucial role in its performance. For instance, 2LS achieves 0.96 accuracy at time 148TU, significantly faster than 1L, which reaches the same accuracy at round 347TU. Similarly, the asynchronous feature contributes to the success of the 2LO approach. This shows the advantage of leveraging asynchronous capabilities to achieve higher accuracies in a shorter time. This is particularly true in an environment where clients are grouped into clusters by speed, eliminating the need to wait for a slow cluster to finish before proceeding with their next training round.

Moreover, we compare the aforementioned approaches with the traditional federated method (FL) in terms of both efficiency and speed. We can observe from Fig. 8 that FL, with no additional features, finishes at a similar time as 2LS. Furthermore, due to the simplicity of the dataset, Fed is not affected much by the non-IID context, reaching 0.99 accuracy at the end of the experiments. However, the non-IID effect is evident in the early stages before reaching convergence, as presented in Fig. 8, where Fed took almost 3000TU to converge compared to our approach taking 42 TU.

Fig. 10 shows the evolution of each round's completion time. As shown in the figure, SL takes the most time per round to complete due to forced synchronization and the existence of stragglers. By addressing both issues in 2LS, we achieve a completion time similar to SFL, which is the baseline for speed comparison. Therefore, our approach can



(a) Accuracy/Round (b) F1-Score/Round

(c) Precision/Round (d) Recall/Round

Fig. 11. Performance Metrics for MNIST Dataset, including benchmarks with optimized federated learning method (FedProx), traditional federated learning (Fed), 2 layers standard (2Ls) and 2 layers with selection (2Lo).

achieve the best performance compared to the existing ones without sacrificing model efficiency, as displayed in Fig. 7.

6.3. Optimized FL experiments

In this section, we evaluate our approach in the context of non-IID environments by comparing it against established benchmarks. These include FedProx (Sahu et al., 2018), a well-known solution aimed at non-IID issues in FL, as well as standard FL without any optimizations to serve as a baseline. To provide a comprehensive comparison, we also assess our double-layered clustering methods, both with and without genetic selection, highlighting how they perform relative to these FL approaches.

It is important to note that enhanced FL approaches like FedProx rely heavily on modifying the full model weights, either on the client side, the server side, or both. Adapting these methods to a split learning environment is highly unfeasible, as such modifications would require extensive back-and-forth communication between clients and the server, introducing significant overhead. For instance, FedProx requires introducing new proximal term variables that modify trainer weights based on the rate of change, multiplied by a regularization term. This process involves iteratively adjusting layer weights after loss computation and before backward propagation. Due to the SplitFed architecture, where the model is divided between server and clients, such computations requiring access to the full model during mid-training are not currently feasible without making significant changes to the approach, which could compromise its original design and purpose. As a result, we evaluate the optimized FL method in its native federated learning environments rather than adapting it to SplitFed.

We include three additional evaluation criteria for metrics: F1-score, precision, and recall, alongside accuracy to provide a more comprehensive analysis of our approaches and their impact on these metrics. Using the same experimental setup as previously described, we evaluate all four methods over 500 rounds. This section emphasizes round-based comparisons, excluding time-based metrics, since our semi-parallel approach inherently differs in execution speed from fully parallel methods.

Figs. 11 present the results of the experiments. Notably, our approach maintained a positive impact on the F1-score, recall, or precision, all showing consistent improvement alongside accuracy. When comparing our approach to optimized FL methods, we observed that both achieved similarly high accuracy, achieving 0.953, 0.924, and 0.89 for FedProx, 2Ls, and 2Ls, respectively, while also reaching convergence within 500 rounds. FedProx, in particular, demonstrated significant improvements over traditional FL methods, about 17%, although at the cost of increased computational demands on the client side to compute and apply the proximal terms. These additions are applied mid-training and grow proportionally with the number of epochs, adding considerable overhead. In contrast, our approach achieves comparable results while being more suitable for IoT devices, as it reduces both computational overhead and communication requirements by only training and communicating part of the model rather than the whole. Moreover, it is essential to note that applying FedProx in a Split Learning setup faces significant limitations, including additional communication costs due to the exchange of proximal terms and increasing computational demands on the client side. These challenges make FedProx largely impractical for Split Learning or SplitFed environments.

7. Conclusion

In this paper, we address the impact of non-IID data distribution and client resource heterogeneity when integrating federated-split learning for IoT devices. We propose a novel double-stage clustering (out,in-clusters) approach and a genetic-based client selection algorithm to overcome these challenges. The out-cluster stage groups clients based on weight divergence to address non-IID data, while the in-cluster stage minimizes stragglers by grouping clients based on execution time. Our workflow uses parallel in-cluster and sequential out-cluster training to ensure efficient aggregation. Additionally, we enhance our approach with a GA selector to optimize client selection based on resource availability and weight compatibility. Experimental results validate the efficiency and improved convergence of our method. However, despite our promising results, there are some limitations and opportunities to consider for future research. Currently, our approach is limited to performing clustering only during the initial stage, which

restricts its ability to adapt dynamically as client participation changes throughout the training process. One potential direction is to enhance our clustering algorithm to dynamically adapt during each round to account for client turnover and resource variations. Another significant area for improvement lies in integrating specialized aggregation algorithms designed for federated split learning environments. The high communication overhead between the server and clients presents scalability and optimization challenges that remain unaddressed in existing approaches. Addressing these limitations, we could benefit from incorporating such techniques tailored to our environment, potentially improving both convergence time and final accuracy.

CRediT authorship contribution statement

Mohamad Arafeh: Writing – review & editing, Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Mohamad Wazzeh:** Writing – review & editing, Writing – original draft, Validation, Methodology. **Hani Sami:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology. **Hakima Ould-Slimane:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology. **Chamseddine Talhi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Conceptualization. **Azzam Mourad:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology. **Hadi Otrok:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by funding from the Innovation for Defence Excellence and Security (IDEaS) program from the Department of National Defence (DND), Canada.

Data availability

The data is publicly available (MNIST and Cifar-10)

References

Abdulrahman, S., Tout, H., Mourad, A., Talhi, C., 2021. FedMCCS: Multicriteria client selection model for optimal IoT federated learning. IEEE Internet Things J. 8 (6), 4723–4735. http://dx.doi.org/10.1109/JIOT.2020.3028742.

Arafeh, M., Hammoud, A., Otrok, H., Mourad, A., Talhi, C., Dziong, Z., 2022. Independent and identically distributed (IID) data assessment in federated learning. In: GLOBECOM 2022 - 2022 IEEE Global Communications Conference. pp. 293–298. http://dx.doi.org/10.1109/GLOBECOM48099.2022.10001718.

Arafeh, M., Ould-Slimane, H., Otrok, H., Mourad, A., Talhi, C., Damiani, E., 2023. Data independent warmup scheme for non-IID federated learning. Inform. Sci. 623, 342–360. http://dx.doi.org/10.1016/j.ins.2022.12.045, URL https://www.sciencedirect.com/science/article/pii/S0020025522015407.

Ayad, A., Renner, M., Schmeink, A., 2021. Improving the communication and computation efficiency of split learning for IoT applications. In: 2021 IEEE Global Communications Conference. GLOBECOM, pp. 01–06. http://dx.doi.org/10.1109/GLOBECOM46510.2021.9685493.

Briggs, C., Fan, Z., Andras, P., 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In: 2020 International Joint Conference on Neural Networks. IJCNN, pp. 1–9. http://dx.doi.org/10.1109/IJCNN48605.2020.9207469.

Chahoud, M., Sami, H., Mourad, A., Otoum, S., Otrok, H., Bentahar, J., Guizani, M., 2023. On-demand-FL: A dynamic and efficient multicriteria federated learning client deployment scheme. IEEE Internet Things J. 10 (18), 15822–15834. http://dx.doi.org/10.1109/JIOT.2023.3265564.

Chen, Z., Li, D., Ni, R., Zhu, J., Zhang, S., 2023. FedSeq: A hybrid federated learning framework based on sequential in-cluster training. IEEE Syst. J. 17 (3), 4038–4049. http://dx.doi.org/10.1109/JSYST.2023.3243694.

Chen, M., Shlezinger, N., Poor, H.V., Eldar, Y.C., Cui, S., 2021. Communication-efficient federated learning. Proc. Natl. Acad. Sci. 118 (17), http://dx.doi.org/10.1073/pnas.2024789118, e2024789118. arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.2024789118.

Duan, M., Liu, D., Chen, X., Liu, R., Tan, Y., Liang, L., 2021. Self-balancing federated learning with global imbalanced data in mobile systems. IEEE Trans. Parallel Distrib. Syst. 32 (1), 59–71. http://dx.doi.org/10.1109/TPDS.2020.3009406.

Gibert, D., Mateu, C., Planes, J., 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J. Netw. Comput. Appl. 153, 102526. http://dx.doi.org/10.1016/j.jnca.2019.102526, URL https://www.sciencedirect.com/science/article/pii/S1084804519303868.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

Hammoud, A., Otrok, H., Mourad, A., Dziong, Z., 2022. On demand fog federations for horizontal federated learning in IoV. IEEE Trans. Netw. Serv. Manag. 19 (3), 3062–3075. http://dx.doi.org/10.1109/TNSM.2022.3172370.

Jain, A.K., Dubes, R.C., 1988. Algorithms for Clustering Data. Prentice-Hall, Inc.

Kang, J., Xiong, Z., Niyato, D., Yu, H., Liang, Y., Kim, D.I., 2019. Incentive design for efficient federated learning in mobile networks: A contract theory approach. CoRR abs/1905.07479, arXiv:1905.07479.

Karjee, J., Anand, K., Bhargav, V.N., Naik, P.S., Dabbiru, R.B.V., Srinidhi, N., 2021. Split computing: Dynamic partitioning and reliable communications in IoT-edge for 6G vision. In: 2021 8th International Conference on Future Internet of Things and Cloud. FiCloud, pp. 233–240. http://dx.doi.org/10.1109/FiCloud49777.2021.00041.

Kotz, S., Balakrishnan, N., Johnson, N.L., 2000. Continuous multivariate distributions: Models and applications. Wiley Ser. Probab. Stat. http://dx.doi.org/10.1002/0471722065.

Lamaazi, H., Mizouni, R., Singh, S., Otrok, H., 2020. A mobile edge-based CrowdSensing framework for heterogeneous IoT. IEEE Access 8, 207524–207536. http://dx.doi.org/10.1109/ACCESS.2020.3038249.

Luping, W., Wei, W., Bo, L., 2019. CMFL: Mitigating communication overhead for federated learning. In: 2019 IEEE 39th International Conference on Distributed Computing Systems. ICDCS, IEEE, pp. 954–964.

McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y., 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, Vol. 54, PMLR, pp. 1273–1282, URL https://proceedings.mlr.press/v54/mcmahan17a.html.

Otoum, S., Guizani, N., Mouftah, H., 2023. On the feasibility of split learning, transfer learning and federated learning for preserving security in ITS systems. IEEE Trans. Intell. Transp. Syst. 24 (7), 7462–7470. http://dx.doi.org/10.1109/TITS.2022.3159092.

Rahman, S.A., Tout, H., Talhi, C., Mourad, A., 2020a. Internet of Things intrusion detection: Centralized, on-device, or federated learning? IEEE Netw. 34 (6), 310–317. http://dx.doi.org/10.1109/MNET.011.2000286.

Rahman, S.A., Tout, H., Talhi, C., Mourad, A., 2020b. Internet of Things intrusion detection: Centralized, on-device, or federated learning? IEEE Netw. 34 (6), 310–317. http://dx.doi.org/10.1109/MNET.011.2000286.

Reddi, S.J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., McMahan, H.B., 2021. Adaptive federated optimization. In: International Conference on Learning Representations. URL https://openreview.net/forum?id=LkFG3lB13U5.

Sahu, A.K., Li, T., Sanjabi, M., Zaheer, M., Talwalkar, A., Smith, V., 2018. On the convergence of federated optimization in heterogeneous networks. CoRR abs/1812.06127, arXiv:1812.06127.

Samikwa, E., Di Maio, A., Braun, T., 2022. ARES: Adaptive resource-aware split learning for Internet of Things. Comput. Netw. 218, 109380. http://dx.doi.org/10.1016/j.comnet.2022.109380, URL https://www.sciencedirect.com/science/article/pii/S1389128622004145.

Thapa, C., Arachchige, P.C.M., Camtepe, S., Sun, L., 2022. Splitfed: When federated learning meets split learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, pp. 8485–8493.

Vepakomma, P., Gupta, O., Swedish, T., Raskar, R., 2018a. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564.

Vepakomma, P., Gupta, O., Swedish, T., Raskar, R., 2018b. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564.

Wang, H., Qu, Z., Zhou, Q., Zhang, H., Luo, B., Xu, W., Guo, S., Li, R., 2022. A comprehensive survey on training acceleration for large machine learning models in IoT. IEEE Internet Things J. 9 (2), 939–963. http://dx.doi.org/10.1109/JIOT.2021.3111624.

Wazzeh, M., Arafeh, M., Sami, H., Ould-Slimane, H., Talhi, C., Mourad, A., Otrok, H., 2024. CRSFL: Cluster-based resource-aware split federated learning for continuous authentication. J. Netw. Comput. Appl. 231, 103987. http://dx.doi.org/10.1016/j.jnca.2024.103987, URL https://www.sciencedirect.com/science/article/pii/S1084804524001644.

Wazzeh, M., Ould-Slimane, H., Talhi, C., Mourad, A., Guizani, M., 2022. Warmup and transfer knowledge-based federated learning approach for IoT continuous authentication. arXiv preprint arXiv:2211.05662.

Wehbi, O., Arisdakessian, S., Wahab, O.A., Otrok, H., Otoum, S., Mourad, A., Guizani, M., 2023. FedMint: Intelligent bilateral client selection in federated learning with newcomer IoT devices. IEEE Internet Things J. 10 (23), 20884–20898. http://dx.doi.org/10.1109/JIOT.2023.3283855.

Wu, W., Li, M., Qu, K., Zhou, C., Shen, X., Zhuang, W., Li, X., Shi, W., 2023. Split learning over wireless networks: Parallel design and resource management. IEEE J. Sel. Areas Commun. 41, 1051–1066.

Wu, D., Ullah, R., Harvey, P., Kilpatrick, P., Spence, I.T.A., Varghese, B., 2021. FedAdapt: Adaptive offloading for IoT devices in federated learning. CoRR abs/2107.04271, arXiv:2107.04271.

Xie, C., Koyejo, S., Gupta, I., 2019. Asynchronous federated optimization. arXiv preprint arXiv:1903.03934.

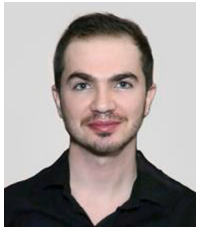
Yang, M., Wang, X., Qian, H., Zhu, Y., Zhu, H., Guizani, M., Chang, V., 2022. An improved federated learning algorithm for privacy preserving in cybertwin-driven 6G system. IEEE Trans. Ind. Inform. 18 (10), 6733–6742. http://dx.doi.org/10.1109/TII.2022.3149516.

Zhang, J., Li, X., Vijayakumar, P., Liang, W., Chang, V., Gupta, B.B., 2024. Graph sparsification-based secure federated learning for consumer-driven internet of things. IEEE Trans. Consum. Electron. 1. http://dx.doi.org/10.1109/TCE.2024.3411551.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V., 2018a. Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.

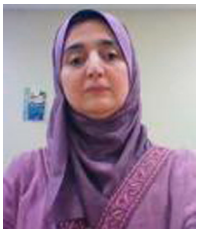
Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V., 2018b. Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.

Mohamad Arafeh is currently pursuing a Ph.D. degree with Ecole de Techńologie Supérieure, Montreal, QC, Canada. He received his BS in Business Computing from the Lebanese University in 2016. His main interests are Federated Learning, Social Network, Recommender Systems, Blockchain, Artificial Intelligence, and Mobile Crowdsensing.



Mohamad Wazzeh is currently pursuing a Ph.D. in the Software and IT Engineering Department at Ecole de Technologie Supérieure in Ḿontreal, QC, Canada. Received the MS degree in business computing from the Lebanese University, Lebanon 2016. He is also a research fellow at the Artificial Intelligence and Cyber Systems Research Center at the Lebanese American University. His research interests include Continuous Behavioral Authentication, Federated Learning, Access Control, IoT, and Smartphone device security.

Hani Sami is currently a postdoctoral student at Ecole de technologie superieure, Montreal, QC, Canada. He completed his Ph.D. studies at Concordia University, Institute for information Systems Engineering (CIISE) in 2023. He received his M.Sc. degree in Computer Science from the American University of Beirut and completed his B.S. and worked as Research Assistant at the Lebanese American University. The topics of his research are Fog Computing, Vehicular Fog Computing, Reinforcement Learning, Reward Shaping, and Blockchain. He is a reviewer of several prestigious conferences and journals.



Hakima Ould-Slimane obtained her Ph.D. degree in Computer Science from Laval University, Quebec, Canada. She is currently a professor at the department of mathematics and computer science at Universite de Quebec a Trois-Rivieres (UQTR, TroisRivieres, Canada). Her research interests include mainly: information security, cyber resilience, homomorphic encryption, federated learning, preserving data privacy in smart environments, machine learning based intrusion detection, access control, optimization of security mechanisms and security of social networks.

Chamseddine Talhi Received the Ph.D. degree in computer science from Laval University, Quebec, QC, Canada, in 2007. He is an Associate Professor with the Department of Software Engineering and IT, ETS, University of Quebec, Montreal, QC, Ćanada. He is leading a research group that investigates smartphone, embedded systems, and IoT security. His research interests include cloud security and secure sharing of embedded systems.

Azzam Mourad received his M.Sc. in CS from Laval University, Canada (2003) and Ph.D. in ECE from Concordia University, Canada (2008). He is currently a Visiting Professor at Khalifa University, a Professor of Computer Science and Founding Director of the Artificial Intelligence and Cyber Systems Research Center at the Lebanese American University, and an Affiliate Professor at the Software Engineering and IT Department, Ecole de Technologie Supérieure (ÉTS), Montreal, Canada. His research interests include Cyber Security, Federated Machine Learning, Network and Service Optimization and Management targeting IoT and IoV, Cloud/Fog/Edge Computing, and Vehicular and Mobile Networks. He has served/serves as an associate editor for IEEE Transactions on Services Computing, IEEE Transactions on Network and Service Management, IEEE Network, IEEE Open Journal of the Communications Society, IET Quantum Communication, and IEEE Communications Letters, the General Chair of IWCMC2020-2022, the General Co-Chair of WiMob2016, and the Track Chair, a TPC member, and a reviewer for several prestigious journals and conferences. He is an IEEE senior member.

Hadi Otrok received his Ph.D. in ECE from Concordia University. He holds a Professor position in the Department of Computer Science at Khalifa University. Also, he is an Affiliate Associate Professor at the Concordia Institute for Information Systems Engineering at Concordia University, Montreal, Canada, and an Affiliate Associate Professor in the Electrical Department at Ecole de Technologie Superieure (ETS), Montreal, Canada. His research interests include the domain of blockchain, reinforcement learning, crowd sensing and sourcing, ad hoc networks, and cloud security. He cochaired several committees at various IEEE conferences. He is also an Associate Editor at IEEE Transactions on Network and Service Management (TNSM), Ad-hoc Networks (Elsevier), and IEEE Network. He also served from 2015 to 2019 as an Associate Editor at IEEE Communications Letters.