FedCime: An Efficient Federated Learning Approach For Clients in Mobile Edge Computing

Anonymous Submission

***Abstract*—Federated learning (FL) is a privacy-enhancing distributed machine learning technique that allows multiple devices with localized data to train a global model collaboratively. However, in Mobile Edge Computing (MEC) environments, resource-constrained devices generate non-independent and identically distributed (non-IID) data, which traditional FL algorithms such as Federated Averaging (FedAvg) struggle to handle, resulting in accuracy degradation of the global model. In addition, dynamic mobile networks present challenges such as intermittent or poor network connectivity, dropouts, and high migration rates, making it difficult for mobile clients to communicate model updates to the central server. Moreover, our observations reveal that networks with high migration rates have degraded model performances. To address these challenges, we present FedCime, a novel tier-based FL approach that selects mobile clients with high utility likely to complete training and replaces migrating clients during the round of training. Our evaluation shows that FedCime significantly improves training performance in terms of accuracy and computational efficiency compared to state-of-the-art FL algorithms. Moreover, FedCime leverages client migration to improve the accuracy of the global model by up to 3.24% with a 30% migration rate and is scalable to handle large numbers of clients in dense networks, making it suitable for practical applications.**

***Index Terms*—Federated learning, machine learning, mobile edge computing, data heterogeneity**

I. Introduction

With the proliferation of sensors and their increasing connectivity to the Internet, many Internet of Things (IoT) devices have emerged as important sources of data for machine learning (ML) applications. These sensors gather data from the environment and send them to the cloud to train different machine learning algorithms or use trained algorithms for inferencing. As the number of IoT devices and the data produced continues to increase, data privacy, integration, security, and latency are key challenges affecting the efficiency of ML applications. Federated Learning (FL) is a privacy-preserving ML training technique that addresses these challenges by allowing distributed clients to cooperatively train machine learning models on decentralized data spread across multiple edge devices. Furthermore, since clients can train with their local data, FL reduces latency issues that may arise when clients try to access a central cloud server for data. Fig. 1 illustrates the basic architecture of FL. In FL, clients iteratively download the global model weights from the cloud, locally train their models, and upload the new weights to the server. The cloud server aggregates the clients' updates to improve the model and shares the global model weights with all the clients for the next round of training. This process can continue until training converges [1].

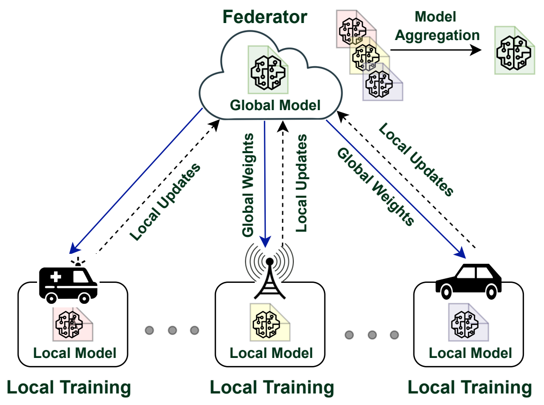


Fig. 1: Overview of federated learning with different devices

One major challenge of FL is that data across clients may differ in class distribution, quality, and quantity. With non independently and identically distributed (non-IID) and unbalanced data in FL settings, training accuracy can become degraded [2]. In addition, FL training may face challenges due to dynamic device availability and traffic from millions of devices for any training round. Thus, sampling clients that will improve training performance and model accuracy for each training round becomes necessary. However, communication between the selected clients with a single central server can still lead to a communication bottleneck that slows the training process. To address this challenge, edge servers in Mobile Edge Computing (MEC) bring computational power closer to devices in mobile environments and assist in a hierarchical setting to coordinate clients in their vicinity, reducing traffic load to core cloud networks and minimizing the latency of end-to-end communication between clients and servers [3]. Furthermore, MEC environments typically have low-latency, high-bandwidth connections to edge devices, which can be important for real-time and interactive applications like self-driving cars. Since vehicles are equipped with various sensors that generate large amounts of data, MEC servers can allow vehicles to participate in FL training, thus expanding the capabilities of FL and making it more suitable for real-world applications. Although MEC servers can assist in FL aggregation, the dynamic nature of the network due to clients' mobility still poses the challenge of maintaining a consistent set of devices for any training round, thereby affecting the overall performance of the FL model. The challenges become even more complicated when the *federator* does not have information about each client's location to estimate the dropout rate due to migration out of its coverage area.

Despite these challenges, the privacy-preserving advantage of FL makes it an essential tool for supporting machine learning applications for mobile nodes. Thus, it is necessary to address the challenges of data heterogeneity and client mobility in mobile networks and ensure that FL can support mobile applications efficiently.

Limitations of existing approaches. Several studies have attempted to address the issue of data heterogeneity in client devices by distributing common data to all clients during the training process to increase the level of IID data during the training process. An approach proposed by Zhao et al. [4] involves sharing a small dataset between all edge devices participating in the training. However, some clients may be unwilling to download public data due to trustworthiness and privacy concerns. Another approach, used by Yoshida et al. [5], allows incentivized clients to upload a portion of their data to the *federator* to improve the global accuracy of this shared data. However, the approach may pose privacy risks for clients participating in the process, especially in the presence of attackers. Other techniques, such FedProx and Yogi [6], [7], focus on optimizing the model aggregation process to improve the global model's accuracy. For example, FedProx penalizes clients that drift from the global model to ensure that accuracy is not degrading. While these techniques offer different degrees of efficiency in addressing data heterogeneity in FL, they do not account for problems that can arise in mobile environments.

These approaches are unsuitable for mobile nodes that frequently migrate from the coverage area of the *federator* aggregating the weights uploaded by each client at every training round. One way to address this challenge is to actively select clients more likely to remain throughout the training process. This approach can help in minimizing the dropout rates during FL training. However, existing methods, such as those proposed by Zhang et al. [8] that consider only data distribution to reduce the impact of dropout and select clients with a lower degree of non-IID data during training do not address the issue of device mobility.

To address the mobility challenge, solutions are required that can mitigate the effect of nodes moving out of the *federator's* coverage area before training is completed.

Our contribution. We propose FedCime to address the mobility challenge in federated learning. FedCime actively selects clients likely to remain within the *federator's* coverage area during the majority of training. The selection strategy is based on the client's delay metrics, where we assess the delays of each client's transmission when sending updates for aggregation. Using this metric, we group clients into different tiers and prioritize clients in tiers with lower delays for the next round of training. Also, to mitigate any dropout that might occur, we choose some N additional clients for training. Furthermore, to address the issue of data heterogeneity, we prioritize model updates with similar characteristics by analyzing updates from reserve clients selected based on their aggregated weights. Specifically, we calculate the cosine similarity of the aggregated weights between the reserved and new clients and select new clients with similar distance values to improve the model's performance.

Overall, FedCime is an effective solution for addressing the challenges of mobility and data heterogeneity in federated learning, particularly in highly dynamic networks like vehicular networks. By actively selecting clients with a lower probability of migration and prioritizing those with similar updates based on a distance-based similarity technique, FedCime can minimize the dropout effect of migration and reduce the impact of data heterogeneity on accuracy. Hence, FedCime is a useful and efficient technique supporting FL for mobile network machine learning applications.

To evaluate the performance of FedCime, we implemented the proposed approach using Pytorch and PySyft and tested it on two real-world FL datasets: MNIST and FashionMNIST. In addition, we compared our approach with FedAvg and FedProx, commonly used baselines in prior work. We also compared our approach with oversampling technique, a naive approach that can improve the global model due to client dropout. Our extensive evaluation demonstrates that FedCime is capable of improving the accuracy of FL models while mitigating the effects of migration and data heterogeneity in mobile networks. Compared to the baselines and oversampling technique, FedCime achieves higher accuracy and lower loss rates, indicating its superiority in addressing the challenges of mobile network machine learning applications. Our results demonstrate the effectiveness of FedCime as a solution for improving FL in mobile networks.

II. Background and Motivation

In this section, we present details of FL and highlight some of its challenges in practical settings using real-world datasets. We also describe the motivation for this work.

A. Federated Learning

In centralized ML, data from different IoT devices are uploaded to a central server, and model training is done using this data repository. However, this approach can lead to data leakages that may compromise users' privacy [9]. FL is a decentralized ML architecture that addresses this challenge by allowing multiple mobile devices to train a shared model without sharing raw data, i.e., FL allows a model's training among a set of clients $C = \{C_1, C_2, C_3, ..., C_k\}$. The model training process occurs on the device, with only the updated model parameters being transmitted back to the central server. This training is completed after T consecutive rounds, and each client trains with its own local dataset, D_k, enabling efficient and privacy-preserving collaborative learning while minimizing the impact on device performance and user privacy. For local training across all clients, the goal is to optimize the loss function $F(w)$ in a distributed way, which measures the difference between the model's predictions and the true values. Thus, each client C_k solves the following distributed optimization problem:

$$\min_{w_k(t)} \quad \frac{1}{|D_k|} \sum_{i=1}^{|D_k|} f_k(w_k(t), x_i, y_i), \forall x_i, y_i \in D_k \qquad (1)$$

where $w_k(t)$ is the model weight from client k, x_i represents the input examples in D_k, y_i denotes the corresponding labels, and f is any suitable local loss function, such as mean-squared error (MSE). In FL, a centralized server known as the *federator* receives local weights from clients after each round for aggregation. The *federator* can choose from available aggregation algorithms to optimize training efficiency. For instance, in FedSGD [10], the *federator* receives gradients of the loss function after each epoch and performs a gradient descent using the average gradient from all clients. Alternatively, in FedAvg [10], each client runs multiple epochs before sending an update to the *federator*, thereby reducing the number of communication rounds needed for training and improving aggregation efficiency. In FedAvg, clients compute new sets of weights and send these to the *federator*, which averages the weights from each client using a defined aggregation rule, such as simple averaging or weighted averaging. The resulting weighted average of the model weights becomes the new global model, which is then used as the starting point for the next round of training. FedAvg averages the weights of local models from multiple clients to improve the accuracy and generalization of the global model while protecting client privacy. When weighted average is used, FedAvg gives more weight to better-performing models, leading to improved global model performance. The weighted average of the local model weights from multiple clients is computed as follows:

$$w(t) := \sum_{k=1}^{K} \frac{D_k}{D} w_k(t) \tag{2}$$

$w(t)$ is the new global model, D is the datasets from all clients participating in training, and K is the total number of clients. Here, FedAvg assigns weights to local models based on the number of samples used for training by each client. This weighting scheme gives more importance to clients with more data.

B. Heterogeneity in FL

Heterogeneity in FL can arise due to differences in hardware specifications, network connectivity, data distribution, or the learning objective. This is a significant challenge that can affect the quality and efficiency of the learning process. Additionally, in networked systems such as the Internet of Vehicles, clients involved in FL can have heterogeneous and non-IID datasets due to variations in the data captured by different vehicles at different times and locations. In non-IID scenarios, where the divergence in clients' datasets is extreme, the size or type of data can also be unequal. This heterogeneity can lead to the degradation of the global model accuracy and slower convergence. Therefore, appropriate techniques must be used to mitigate these challenges and improve the overall performance of the FL training process.

C. Mobility Challenge in MEC

Mobility presents a significant challenge in MEC as mobile devices move from one location to another, changing their

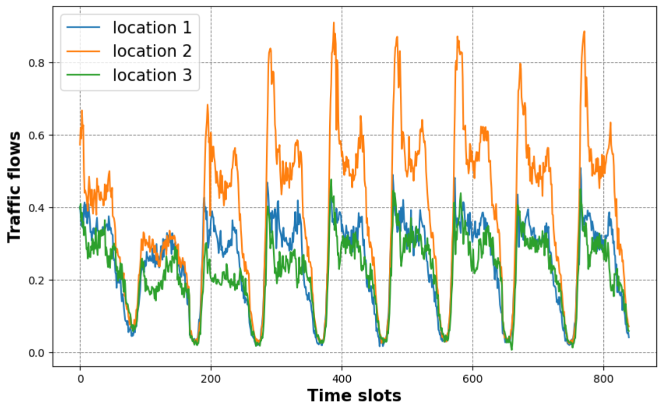


Fig. 2: Traffic flow changes. The traffic volume in 840 continuous 15-minute intervals for 3 locations in Northern Virginia/Washington D.C. capital region [11]

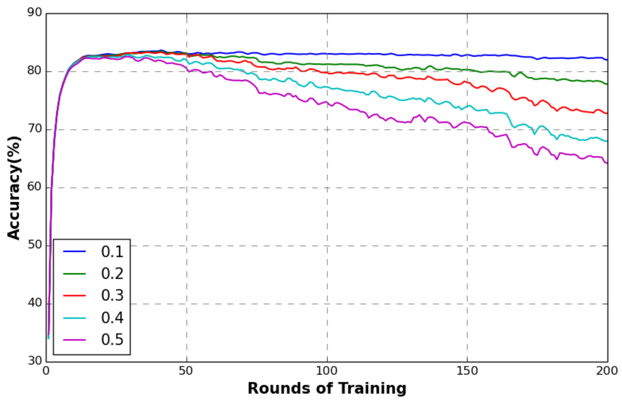
connectivity status, network conditions, and resource availability. In mobile edge networks, participating clients in FL may not be available for the whole duration of training. This is further complicated by the fact that the number of vehicles in a particular region may not stay the same for an extended period of time, as shown in Fig. 2, using a Traffic Flow Prediction Dataset [11]. This inconsistency in the number of vehicles can result in clients moving out of the coverage area of the *federator*, and thus, may upload low-quality models back for aggregation or even completely drop out of the training [12]. Therefore, the mobility challenge in FL requires the *federator* to use techniques that limit the number of clients that drop out due to migration during training while mitigating the effect of dropout on the model's performance.

D. Motivation

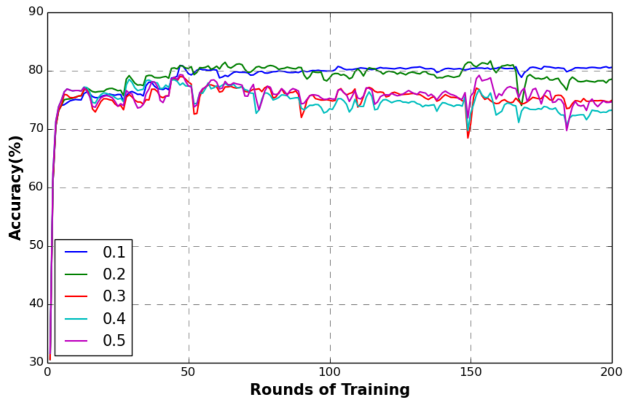
Our approach is motivated by the potential negative impact of data heterogeneity and client mobility on the performance of the global model in FL. To assess this impact, we perform experiments with varying numbers of clients using both IID and non-IID data, as well as in different network scenarios, and analyze their effect on the training accuracy.

1) Impact of mobility: In our experiments, we investigate the impact of client migration on the performance of the global model in FL. We show two scenarios in Fig. 3: one where clients migrate out of the *federator's* coverage area without replacement and another where they are replaced by other clients. The results reveal that client migration without replacement leads to a decrease in accuracy and convergence, while replacing the clients results in better convergence but with some oscillation due to the non-IID nature of the data. As the migration rate increases, the model's accuracy decreases when using the FedAvg algorithm. These findings highlight the need for techniques that limit the effect of client migration and non-IID data on the global model's accuracy and convergence in FL.

2) Impact of data heterogeneity: We conducted experiments to assess the impact of heterogeneous data on the training performance of 200 clients using both IID and non-IID MNIST datasets, varying the number of clients and

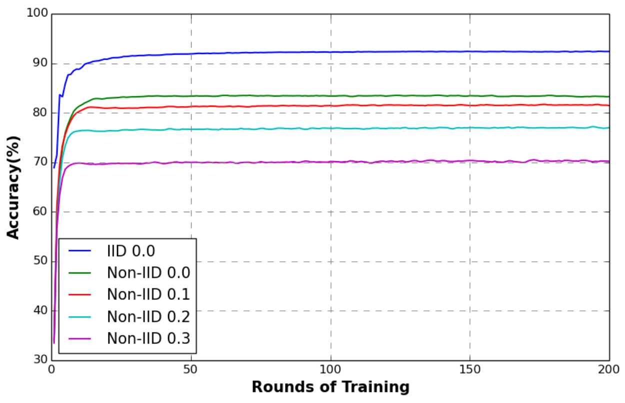


(a) Impact of clients migration without replacement

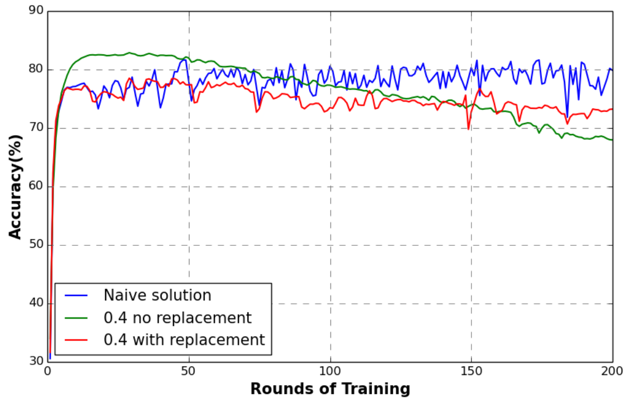


(b) Impact of clients migration with replacement

Fig. 3: (a) Accuracy of the global model with different migration rates without replacement using Non-IID data (b) Accuracy of the global with different migration rates with replacement using Non-IID data



(a) Impact of heterogeneous data and available number of clients



(b) Naive solution using oversampling

Fig. 4: (a) Impact of Non-IID data and the number of clients on the accuracy of the global model (b) Impact of the naive solution with 40% migration rate with and without replacement.

evaluating the performance over 200 training rounds. The results presented in Fig. 4(a) demonstrate that using non-IID data leads to poorer accuracy over the 200 training rounds compared to when clients use uniformly distributed IID data. Moreover, the results show that having fewer clients available for training also degrades the accuracy of the global model. Therefore, high migration rates can significantly impact the model's accuracy. Since FedAvg does not address the effect of data heterogeneity among clients, the model's accuracy decreases when clients with non-IID data participate in the training process. Additionally, the accuracy decreases with FedAvg when the number of participating clients reduces.

3) Naive approach: To minimize the negative impact of dropout on the model's accuracy, we adopt a strategy of sampling additional clients for training and randomly selecting weight updates from them during aggregation. Our experiment involves allowing 40% clients to migrate during training. In the event that N clients drop out, we sample N weight updates from the additional clients randomly pre-selected by the *federator*. We then compare the performance of the naive approach with that of clients that migrate with and without replacement. Fig. 4(b) demonstrates that over 200 rounds, the naive approach significantly mitigates the effect of migration. Furthermore, this result highlights that using techniques that limit migration's impact can improve the overall performance of FL. Therefore, our proposed approach, FedCime, aims to enable training to proceed seamlessly in each round despite node migration.

This paper explores the challenges of training an FL model in a mobile network of diverse clients, such as vehicular networks, where sensors are heterogeneous, and data quality may be degraded. In contrast to the extreme cases studied in Fig. 3 and Fig. 4, we assume that some clients have IID data and others have non-IID data, which varies in quality depending on the age and resource constraints of the sensors. To simulate this scenario, we evaluate the accuracy of the model using different dropout rates and 30% of clients with non-IID data, where we intentionally degrade their data quality by adding Gaussian noise. Our experiment (Fig. 5) shows that as clients drop out, the *federator* can replace them with new clients entering the network with high-quality data, thus improving the model instead of degrading it. This observation motivates our approach to search for clients that can improve the model's accuracy as the migration rate increases.

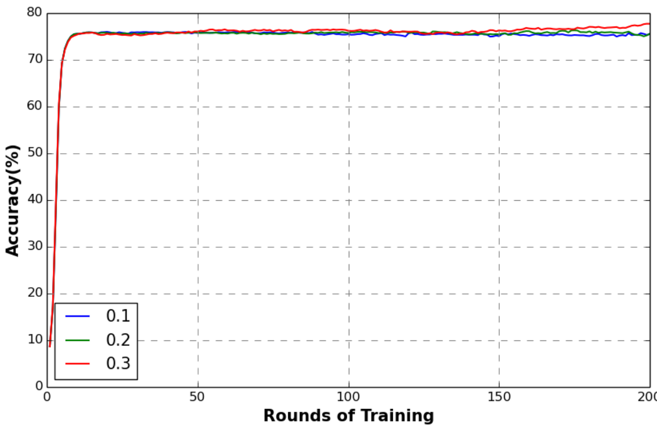


Fig. 5: Accuracy with 30% non-IID data. As the migration rate increases from 0.1 to 0.3, the accuracy of the global model drops.

4) Challenges: Our motivating examples have shown that incorporating additional clients for training and using their updates can reduce the impact of migration in mobile environments. However, this method does not address two significant challenges: the limitation of the number of dropouts during training and the reduction of the effect of data heterogeneity on the global model. The following section introduces a novel technique called FedCime, which guarantees a reduction in the number of dropouts and the mitigation of the non-IID effect. Using the naive oversampling approach helps to alleviate the impact of migration. However, FedCime goes beyond this approach by prioritizing clients with similar performance to the remaining clients and those with high utility, thereby improving the model's performance even further, as shown in Fig. 6.

III. FedCime Design Details

This section discusses the essential aspects of FedCime. Firstly, we explore how FedCime leverages its online profiling technique to limit the selection of likely migrating clients. Subsequently, we discuss the methodology used by the *federator* to compute the similarity between the model updates of various clients and choose the most suitable one for aggregation. Finally, we examine how the offloading process impacts the model aggregation.

In our proposed FL architecture, all clients can communicate with the *federator*, but we consider the scenario where clients are mobile and can be located at different places at any given time, which may affect their connectivity to the *federator*. As a result, a client's poor connectivity and limited computational power can lead to the straggler effect since it cannot efficiently communicate its update to the *federator*. Additionally, if a mobile node migrates out of the network, it will no longer be able to communicate with either the *federator* or other nodes.

A. Client Selection

In our approach, we assume the existence of a *federator* that can be connected to a base station, which selects a subset of clients for training, similar to traditional FL settings. The

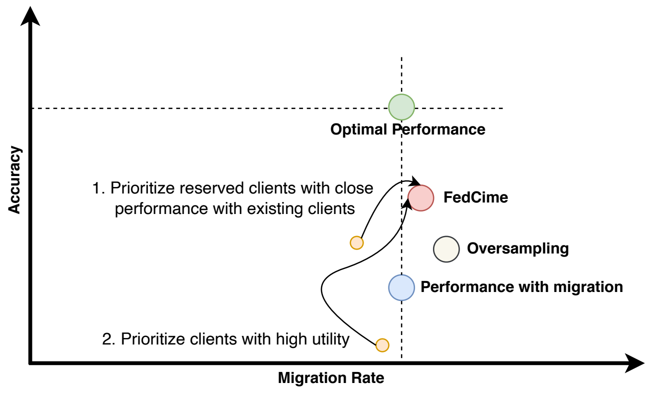

Accuracy
Migration Rate
Optimal Performance
1. Prioritize reserved clients with close performance with existing clients
FedCime
Oversampling
Performance with migration
2. Prioritize clients with high utility

Fig. 6: Performance of FedCime with migration. With increasing migration, accuracy degrades using traditional FL approaches. Using the naive oversampling technique helps to improve training performance when the migration rate increases. With FedCime, the *federator* improves accuracy by prioritizing clients with high utility and selecting reserved clients with performances similar to those that did not migrate.

federator monitors the time it sends the global model to all clients and the time each client returns their local update. To identify stragglers and migrating clients, the *federator* calculates the mean computation duration of results obtained after profiling, then drops clients with high response time. We assume that clients farther away from the *federator* will take longer to send updates. We express the delay metric of each client as follows:

$$T_k = T_k^u - T_k^d \tag{3}$$

where T_k^d is the time when client C_k starts downloading the global model from the *federator*, and T_k^u is the time the *federator* receives an update from client C_k.

After obtaining the delay metrics from each client, the *federator* divides clients into n tiers using the following equation:

$$Tier_k = \left\lceil \frac{T_k}{T_{max}} \times N_{tiers} \right\rceil \tag{4}$$

where T_{max} is the maximum delay among all clients, and N_{tiers} is a predetermined number used by the *federator* to determine the number of allowed tiers. Eq. 4 groups clients based on their delay metrics and assign clients with high delays to the highest tier. The *federator* can then use this information to retain clients in lower tiers and replace clients in tier N in the next round of training. We summarize this procedure in Algorithm 1. In line 1, the *federator* initializes the maximum delay for the current training round. Lines 2-7 show the calculation of the delay metric for each of the clients selected for training. In lines 8-11, the *federator* groups each client into different tiers using the delay metric and adds each client to its appropriate tier. The *federator* selects clients for the next round of training and returns the set of selected clients in lines 12-16.

Algorithm 1 Client Selection using Delay Metrics

```
Require: $\mathcal{S}$: subset of clients selected for training, $N_{tiers}$: number of tiers
 1: $T_{max} \leftarrow 0$ {initialize maximum delay}
 2: for $S_k \in \mathcal{S}$ do
 3:     $T_k \leftarrow T_k^u - T_k^d$ {calculate delay metric for client $S_k$}
 4:     if $T_k > T_{max}$ then
 5:         $T_{max} \leftarrow T_k$ {update maximum delay}
 6:     end if
 7: end for
 8: for $S_k \in \mathcal{S}$ do
 9:     $Tier_k \leftarrow \lceil \frac{T_k}{T_{max}} \times N_{tiers} \rceil$ {assign client $S_k$ to tier}
10:     $\mathcal{T}Tier_k \leftarrow \mathcal{T}Tier_k \cup S_k$ {add client $S_k$ to tier}
11: end for
12: $\mathcal{S} \leftarrow \emptyset$ {initialize selected subset}
13: for $i = 1$ to $N_{tiers} - 1$ do
14:     $\mathcal{S} \leftarrow \mathcal{S} \cup \text{RandomSelect}(\mathcal{T}i)$ {select clients from tier $i$}
15: end for
16: return $\mathcal{S}$ {return selected subset of clients}
```

B. Improved Client Selection and Dropout Mitigation

Although the tier-based method limits the number of migrating clients that will be chosen over the total training rounds, we assume that some clients will still migrate or drop out due to other factors, such as slow computation speed. To mitigate the dropout effect, we allow the *federator* to oversample by choosing K clients that are more than the actual number of clients that will be used for aggregation. From this set of clients, the *federator* will choose αK clients for training and keep the remaining $(1-\alpha)K$ as reserve clients. Here, α is the proportion of clients selected for aggregation and $0 < \alpha \le 1$. If any client drops out, the *federator* will replace such clients from the reserve clients based on the similarity of their updates.

Similarity Scores: After receiving updates from αK selected clients, the *federator* aggregates their weights. If there is any dropout, the *federator* checks the updates of the reserved clients and calculates their similarities to the αK clients that did not migrate. We use cosine similarity, commonly used in literature for calculating similarities in machine learning applications, as the measure of similarity [13], [14]. The cosine similarity for each client is calculated using the following:

$$S_k = \frac{\langle \Delta\theta_{c^k}, \Delta\theta_{c^a} \rangle}{\|\Delta\theta_{c^k}\| \|\Delta\theta_{c^a}\|} \tag{5}$$

where $\langle \Delta\theta_{c^k}, \Delta\theta_{c^a} \rangle$ gives the dot product between the model update $\Delta\theta_{c^k}$from client k and the aggregated weights $\Delta\theta_{c^a}$ from the selected αK clients. $\|\Delta\theta_{c^k}\| \|\Delta\theta_{c^a}\|$ is the product of the norms of the updates.

Similarity Weights: Although the reserve clients are now weighted such that the clients with updates that are similar to the originally sampled αK clients can be selected, it is also important to use updates that will be significant to the global model. Since a client's loss will be high if the model is not adequately generalized to its dataset, we use each client's loss as a metric to determine the weights given to its similarity scores. The similarity weight for each client can be calculated using the following:

$$\gamma_k = 1 - \frac{\tau}{e^{Loss_k^2}} \tag{6}$$

where $Loss_k$ is the loss from client k and τ is a scaling factor.

Using the γ_k calculated for each client, the *federator* updates the similarity score using:

$$S_k := \gamma_k S_k \tag{7}$$

After calculating the similarity, the *federator* sorts the reserve clients in non-increasing order of their similarity scores, i.e., $C' = \{S_1', S_2', ...S_n'\}$, where $S_1' \ge S_2' \ge ... \ge S_n'$ and $n = (1-\alpha)K$. The *federator* then chooses the first few clients from C' needed to replace the dropped clients from the initial set of selected αK clients.

Model Aggregation: Using the model updates received from all selected clients, the *federator* aggregates the updates to compute the new weight for the global model using Eq. 2.

A summary of the approach is given in Algorithm 2. In lines 1-6, the *federator* initializes the number of dropped clients to zero, selects clients for training, and calculates the number of expected updates. After all selected clients send their updates, the *federator* checks if there is any dropout, aggregates the weights, and calculates the similarity scores for reserved clients in lines 7-17. In line 18, the *federator* sorts the reserve clients based on their similarity scores and replaces the dropped clients in lines 19-20. Line 22 shows the global weight update after replacement.

IV. Evaluation

We perform simulation experiments with one *federator* and K clients to evaluate our proposed approach. We test the performance of FedCime using two commonly used datasets and compare it with FedAvg, FedProx, and the oversampling technique. Our implementation follows the classic FL architecture with one server and 200 clients. To simulate degraded data in wireless sensors, we add Gaussian noise $x = x + \epsilon$, where $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$, $\mu = 0$, and $0 < \sigma^2 \le 1$. We assume that the *federator* can communicate with all clients, and the delay incurred in downloading and uploading models increases with the distance of the client from the *federator*.

A. Evaluation Settings

1) Datasets: We used two publicly available datasets, MNIST and FashionMNIST, widely used in literature [8], [14].

MNIST: The MNIST dataset consists of 10 classes of handwriting images ranging from 0 to 9. This dataset comprises 60,000 training and 10,000 testing images, all grayscale, and with dimensions of 28 x 28 pixels. We divide the image dataset among 300 clients and use the original test set for evaluating the model.

FashionMNIST: The FashionMNIST dataset comprises 70,000 grayscale images divided into 60,000 training and 10,000 testing sets. We distribute the training set among 300 clients and evaluate the global model using the testing set.

Algorithm 2 Improved Client Selection Algorithm

```
Require: $M$: number of local epochs; $K$: number of clients to be
    oversampled; $\alpha$: proportion of clients selected for aggregation;
    $\tau$: scaling factor for similarity weight.
Ensure: Global model $\theta$.
 1: Initialization:
 2: $D_l \leftarrow 0$ {initialize the number of dropped clients}
 3: $C \leftarrow$ random sample of $K$ clients {oversample clients for train-
    ing}
 4: $\mathcal{S} \leftarrow$ random subset of $\alpha K$ clients from $C$ {select subset of
    clients for training}
 5: $\mathcal{R} \leftarrow$ remaining clients in $C$ {reserve clients for replacement}
 6: $A_l \leftarrow |\mathcal{S}|$ {number of expected updates}
 7: if clients dropped out then
 8:     $D_l \leftarrow |A_l| - |\mathcal{S}|$ {update number of dropped clients}
 9:     for $k \in \mathcal{S}$ do
10:         $\Delta\theta c^a \leftarrow$ weighted aggregation of updates from $\mathcal{S}$
11:     end for
12:     for $k \in \mathcal{R}$ do
13:         $\Delta\theta_{c^k} \leftarrow$ local update from $k$ with $M$ epochs
14:         $S_k \leftarrow \frac{\langle \Delta\theta_{c^k}, \Delta\theta_{c^a} \rangle}{\|\Delta\theta_{c^k}\| \|\Delta\theta_{c^a}\|}$ {calculate similarity score}
15:         $\gamma_k \leftarrow 1 - \frac{\tau}{e^{Loss_k^2}}$ {calculate similarity weight}
16:         $S_k \leftarrow \gamma_k S_k$ {update similarity score}
17:     end for
18:     $C' \leftarrow$ sort $\mathcal{R}$ in non-increasing order of similarity scores
        {sort reserve clients by similarity score}
19:     for $i \in [1, D_l]$ do
20:         $\mathcal{S} \leftarrow \mathcal{S} \cup C'i$ {replace dropped clients with reserve clients}
21:     end for
22:     $\theta \leftarrow$ update global model with aggregate of update in $C$
23: end if
```

2) Model Parameters: Similar to Talukder and Islam [15], we used a logistic regression classifier for our experiments targeting sensors in mobile networks using Pytorch and Pysyft rather than TensorFlow. To preprocess the data, we flattened the input features and encoded the labels using one-hot encoding. We utilized the softmax activation function for the MNIST and FashionMNIST datasets and set the L1 and L2 regularization values to 0.01. We chose the Adam optimizer for an efficient optimization process and the categorical cross-entropy loss function to measure the dissimilarity between the predicted probability distribution and the true probability distribution of the classes.

3) Baselines: We evaluate the performance of FedCime using the following baselines:

FedAvg [10]: FedAvg is a commonly used FL algorithm that aggregates updates from all clients that did not drop out but participated in the round of training. The weight of each client's update in the final global model is determined by their respective local dataset size.

FedProx [6]: FedProx is an improved version of the FedAvg algorithm that addresses the issue of data heterogeneity among clients. FedProx incorporates a proximal term in the optimization objective of FedAvg to minimize the divergence of local models from the global model. The proximal term acts as a penalty for updates that deviate excessively from the global model and reduces the influence of such updates in the aggregation process. As a result, the FedProx algorithm generates a more stable and accurate global model.

TABLE I: Accuracy of FL algorithms under different migration rates. In each round, 50% of clients have non-IID data.

Datasets	**Algoirthms**	**Migration Rates**		
		10%	20%	30%
MNIST	FedAvg	75.51	76.59	77.72
	FedProx	75.75	76.51	76.84
	Oversampling	**76.02**	**75.69**	**77.74**
	FedCime	**76.64**	**78.37**	**80.08**
FashionMNIST	FedAvg	64.76	64.84	65.11
	FedProx	64.59	64.79	64.88
	Oversampling	**64.95**	**64.89**	**65.13**
	FedCime	**64.99**	**66.01**	**66.83**

Oversampling: This approach involves oversampling the clients using our naive approach. The oversampling approach allows the *federator* to replace clients that dropout with another client from the oversampled set. This technique aims to address the impact of dropout resulting from node migration.

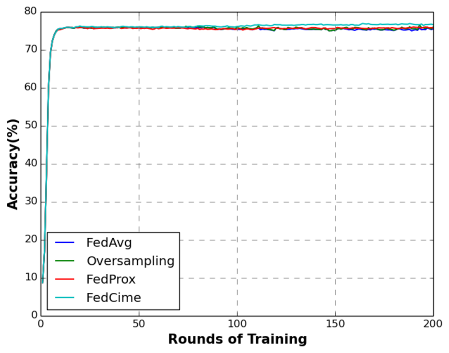
B. Evaluation Results

1) Comparison with baselines: We evaluate the performance of FedCime against the baselines. The result of our evaluation is given in Table I. We varied the migration rate from 10% to 30% in each round, with 50% of clients having IID and non-IID data each.

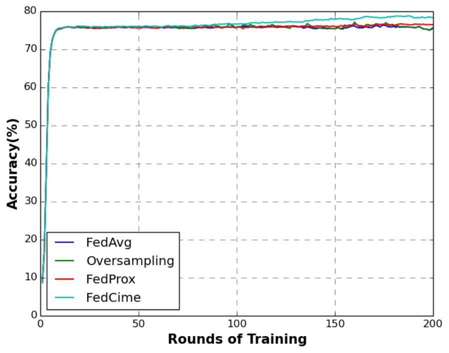
For the MNIST dataset, the best performance of the FedAvg algorithm is 77.72%. FedProx handles the heterogeneity in the dataset better than FedAvg when the migration rate is 10%, achieving an accuracy of 75.75%. Using the oversampling approach, the accuracy is improved to 77.74% when the migration rate is 30%. The results show that FedCime achieves higher accuracy of 76.64%, 78.37%, and 80.08% for 10%, 20%, and 30% migration rates, respectively, outperforming all other baselines. The accuracy increases with increasing migration when available IID clients join the network, but FedCime leverages this advantage more effectively than other algorithms.

Since the FashionMNIST dataset is more complex than MNIST, the accuracy is lower compared to MNIST for all algorithms. However, FedCime algorithm outperforms the baselines, improving the accuracy by up to 1.95% in the best case. The oversampling approach improves the performance of both FedAvg and FedProx, reaching an accuracy of 65.13% compared to FedAvg's 65.11% and FedProx's 64.88% accuracy when the migration rate is 30%. However, the ability of FedCime to select clients that significantly improve the model's performance helps it to perform better than other algorithms. By using FedCime, we improve the accuracy of the models to 64.99%, 66.01%, and 66.83% for 10%, 20%, and 30% migration rates, respectively.

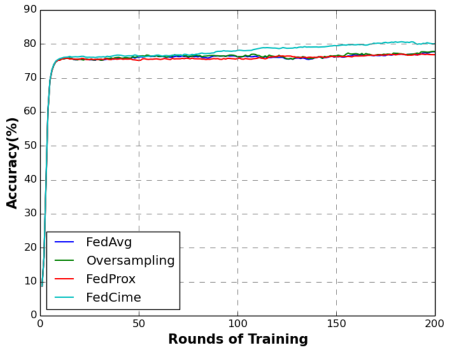
2) Convergence analysis: Convergence analysis is an important aspect of evaluating the performance of machine



(a) 0.1% migration rate

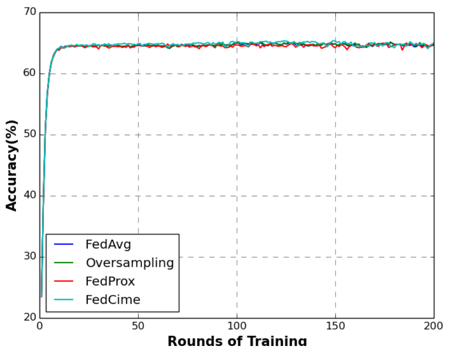


(b) 0.2% migration rate

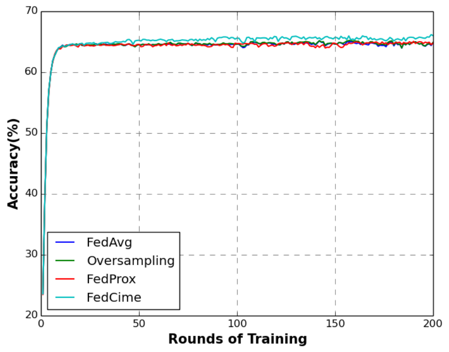


(c) 0.3% migration rate

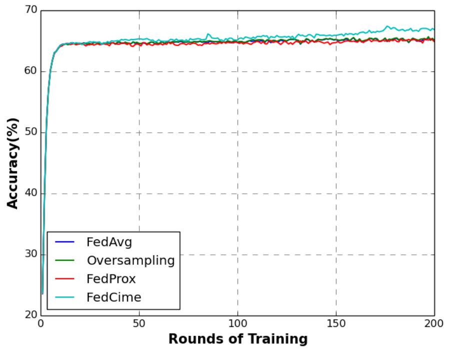
Fig. 7: Performance of FedCime with MNIST dataset using different rates of migration and 0.5% Non-IID



(a) 0.1% migration rate



(b) 0.2% migration rate



(c) 0.3% migration rate

Fig. 8: Performance of FedCime with FashionMNIST dataset using different rates of migration and 0.5% Non-IID

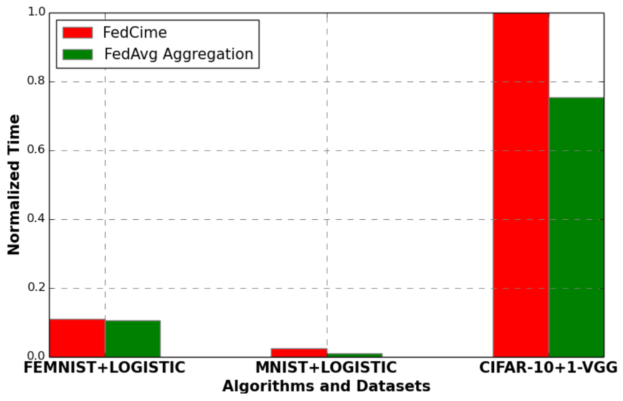
learning algorithms, including FL. It measures the rate and stability of algorithm convergence as it iteratively updates the model weights based on the training data. We evaluate the convergence of FedCime with existing approaches as shown in Fig. 7 and Fig. 8. The figures show the accuracy of each technique for 10%, 20%, and 30% migration rates. Using the MNIST dataset, Fig. 7 illustrates that all algorithms converge smoothly for a migration rate of 10%. However, FedCime converges with higher accuracy compared with other baselines. As the degree of migration increases, FedCime performs better compared to other baseline algorithms, as evident in Fig. 7(b) and Fig. 7(c), where FedCime selects better clients and converge with higher accuracy.

The convergence for the FashionMNIST dataset is a bit noisier due to its higher complexity than the MNIST dataset, but all algorithms were able to converge as depicted in Fig. 8. However, in most cases, FedCime converges with higher accuracy than the baselines, improving the model's performance.

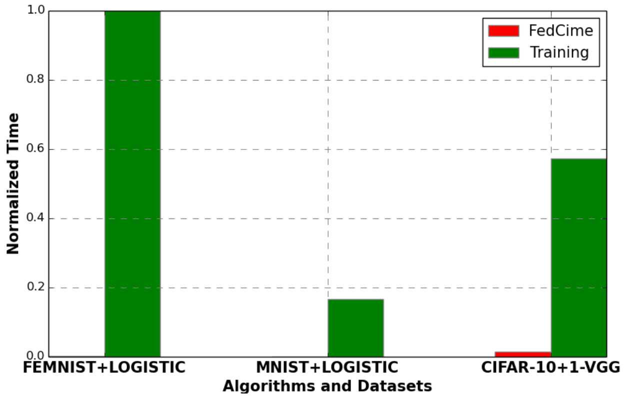
3) Computation cost analysis: In addition to evaluating the accuracy of FedCime, we perform a computational analysis, i.e., we evaluate the time and resource required and compare the overhead with FedAvg's weights aggregation time. The weights aggregation time is the duration it takes for the server to find the weighted average of all weights uploaded by the clients after completing local training. We compare this time to the time required to complete the procedures in Algorithm 2. We use three datasets for our evaluation: FEMNIST, MNIST, and CIFAR-10. FEMNIST is a non-IID and heterogeneous dataset comprising 341,873 training sets divided among 3,383 users. It contains 40,832 grayscale test images of sizes 28 x 28 pixels. CIFAR-10 is a dataset of 60,000 colored images of size 32 x 32 pixels, consisting of 10 classes, including 50,000 training images and 10,000 testing images. We used logistic regression for the FEMNIST and MNIST datasets, while for the CIFAR-10 dataset, we used a one-block VGG network [16].

Fig. 9(a) shows the result of our evaluation. From the results, FedCime's overhead is comparable to the aggregation time when using logistic regression. Due to the number of parameters in the VGG network when using the CIFAR-10 dataset, the time for FedCime's computation is higher than the aggregation time. Comparing the overhead of FedCime's computation with the overall training and weight aggregation time for all clients in the network, Fig. 9(b) shows that the overhead of FedCime's computation is negligible for all datasets and networks. Therefore, the demonstrated overhead of FedCime's computation in Fig. 9(a) becomes insignificant in the context of the overall aggregation time.

4) Is FedCime scalable?: In this experiment, we evaluate FedCime across different scales of clients with the MNIST datasets, terminating the training after 150 rounds. We choose K clients and randomly divide the training set among the K



(a) FedCime's computation and FedAvg's aggregation time



(b) FedCime's computation and FedAvg's overall training time

Fig. 9: (a) Comparison of FedCime's computation time and FedAvg's time for aggregation on the server side (b) Comparison of FedCime's computation time and FedAvg's overall time for training and aggregation on the client and server side.

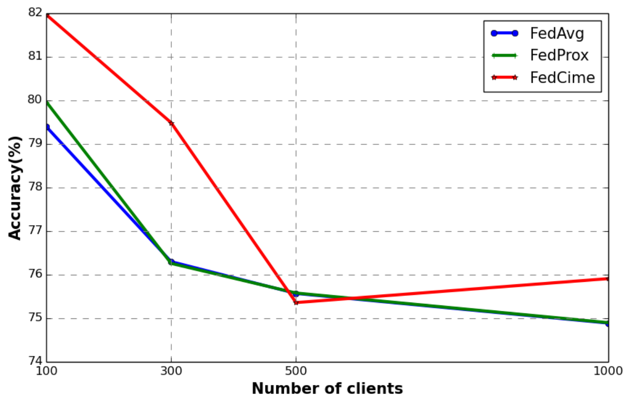


Fig. 10: Accuracy with different numbers of clients in the network

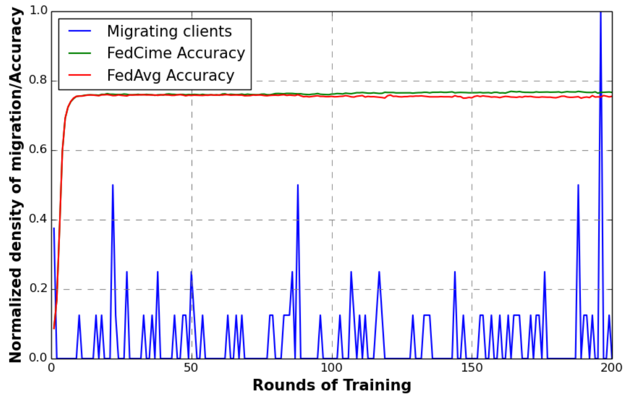


Fig. 11: FedCime's improvement despite clients' migration

clients, where we test $100 \leq K \leq 1000$. Fig. 10 shows that FedCime outperforms other algorithms with a small number of clients, and it continues to perform well as the number of clients increases. Furthermore, our algorithm improves the accuracy of the global model more than FedProx does, as FedProx converges slowly. We can also use FedCime in combination with FedProx to enhance its performance, as we have accomplished with FedAvg.

5) How good is FedCime at replacing migrating clients?: To demonstrate the effect of client migration on FedCime, we track the number of migrating clients and calculate its accuracy. We compare the accuracy of FedAvg and FedCime's in each training round. Fig. 11 shows the results of our evaluation. As depicted, FedCime can leverage migration better than the traditional FedAvg algorithm and improve the performance of the global model. In the extreme scenarios where many clients migrate, FedCime is capable of selecting clients that will enhance the model's accuracy rather than diminish it. This result implies that our algorithm can enhance traditional FL techniques and ensure that clients in mobile environments can participate effectively in FL training.

V. Related Work

This section summarizes some of the state-of-the-art approaches related to the proposed method.

FL is a privacy-preserving ML approach that uses the FedAvg algorithm to allow global averaging on a server after completing local stochastic gradient descent on a subset of devices [10]. While FedAvg has been shown to converge under realistic settings, data heterogeneity can slow down the convergence rate [17]. Techniques such as sharing common data [4] and FedProx [6] have also been proposed to address this challenge. However, sharing common data may violate clients' security policies, while the FedProx requires parameter tuning that may lead to slow convergence. Other optimization techniques, such as Yogi, FedNova, Scaffold, CSFedAvg [7], [8], [18], [19] have also been proposed, but they do not fully account for migration in MEC environments.

Split learning (SL) enables clients in FL training to offload some layers of their ML models to maximize efficiency. Tharpa et al. [20] introduced SplitFed, a technique combining SL and FL to reduce computation and improve model robustness. However, it does not address the challenge of device mobility, which can impact training accuracy when devices move out of the network before training is completed. Wu et al. [21] proposed FedAdapt, an adaptive framework that accounts for dynamic network bandwidth and heterogeneous devices during training. The approach uses a reinforcement learning agent to make offloading decisions. Cox et al. [22] proposed Aergia, a technique that boosts training speed in FL by allowing slow clients to freeze part of their models and transfer the frozen layers to a faster client for training. However, these techniques are unsuitable for mobile devices due to the challenge of device mobility.

Edge networks involving mobile devices have unstable connectivity and network conditions, and the clients presented in the server's coverage area may vary during training. Therefore, creating a framework that ensures the training process is completed successfully to maintain a stable model, even when nodes move out of the network, is essential. However, research in this area is limited. To address this issue, Ullah et al. [23] proposed FedFly, which reduces the training costs when devices migrate between edge servers during the training phase by transferring the model to the source edge server. We aim to enhance the accuracy and stability of the model trained by the initiating edge server in this context.

VI. Conclusion

In this paper, we present FedCime, an efficient and effective tier-based approach for FL in MEC environments to minimize the effect of non-IID and heterogeneous datasets while ensuring better convergence and accuracy. FedCime leverages client migration in mobile networks to select clients that can improve the accuracy of the global model. We have shown that FedCime is resilient to client migrations and can efficiently select clients likely to remain in the training process for extended periods and will improve the model's accuracy, thereby ensuring that clients in mobile environments can participate effectively in the training process. Our extensive evaluation of FedCime with different datasets shows that FedCime outperforms state-of-the-art algorithms such as FedAvg and FedProx in terms of accuracy, communication overhead, and computational efficiency. We also showed through experiments that FedCime is scalable and can handle a large number of clients in dense networks, making it suitable for practical applications.

We plan to test FedCime with more heterogeneous datasets using extreme non-IID cases in future work. We also envision expanding FedCime to choose trustworthy clients and detect and defend against malicious clients in the network, thereby contributing significantly to the development of FL in MEC environments.

References

[1] H. Ludwig and N. Baracaldo, "Introduction to federated learning," in *Federated Learning*. Springer, 2022, pp. 1–23.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[3] R. Yu and P. Li, "Toward resource-efficient federated learning in mobile edge computing," *IEEE Network*, vol. 35, no. 1, pp. 148–155, 2021.

[4] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[5] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data," in *ICC 2020-2020 IEEE International Conference On Communications (ICC)*. IEEE, 2020, pp. 1–7.

[6] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[7] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[8] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang, "Client selection for federated learning with non-iid data in mobile edge computing," *IEEE Access*, vol. 9, pp. 24 462–24 474, 2021.

[9] F. Tramèr, R. Shokri, A. S. Joaquin, H. Le, M. Jagielski, S. Hong, and N. Carlini, "Truth serum: Poisoning machine learning models to reveal their secrets," *arXiv preprint arXiv:2204.00032*, 2022.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017.

[11] L. Zhao, O. Gkountouna, and D. Pfoser, "Spatial auto-regressive dependency interpretable learning based on spatial topological constraints," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 5, no. 3, pp. 1–28, 2019.

[12] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020.

[13] S. Sohangir and D. Wang, "Improved sqrt-cosine similarity measurement," *Journal of Big Data*, vol. 4, no. 1, pp. 1–13, 2017.

[14] P. Tian, W. Liao, W. Yu, and E. Blasch, "Wscc: A weight-similarity-based client clustering approach for non-iid federated learning," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20 243–20 256, 2022.

[15] Z. Talukder and M. A. Islam, "Computationally efficient auto-weighted aggregation for heterogeneous federated learning," in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 2022, pp. 12–22.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[18] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, 2020.

[19] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020.

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

[21] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "Fedadapt: Adaptive offloading for iot devices in federated learning," *IEEE Internet of Things Journal*, 2022.

[22] B. Cox, L. Y. Chen, and J. Decouchant, "Aergia: leveraging heterogeneity in federated learning systems," in *Proceedings of the 23rd conference on 23rd ACM/IFIP International Middleware Conference*, 2022.

[23] R. Ullah, D. Wu, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "Fedfly: Toward migration in edge-based distributed federated learning," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 42–48, 2022.