2023 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3503-0483-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/EDGE60047.2023.00042

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

Paul Agbaje*, Afia Anjum*, Zahidur Talukder*, Mohammad Islam*, Ebelechukwu Nwafor†, Habeeb Olufowobi*
*Department of Computer Science and Engineering, University of Texas at Arlington
†Department of Computing Sciences, Villanova University
{pauloluwatowoju.agbaje, habeeb.olufowobi}@uta.edu

***Abstract*—Federated learning (FL) enables collaborative training of a global model using localized data from multiple devices. However, in resource-constrained mobile edge computing (MEC) environments, non-independent and identically distributed (non-IID) data generated by these devices poses challenges for traditional FL algorithms like Federated Averaging (FedAvg), leading to decreased accuracy of the global model. In addition, dynamic mobile networks with intermittent connectivity, dropouts, and high migration rates hinder the communication of model updates to the central server. To address these challenges, we present FedCime, a novel tier-based FL approach that selects high-utility mobile clients likely to complete training to replace migrating clients during the round of training. Our evaluation shows that FedCime is scalable and significantly improves training performance in terms of accuracy and computational efficiency compared to state-of-the-art FL algorithms.**

***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 data sources for machine learning (ML) applications. These devices gather data and send them to the cloud for training machine learning algorithms or inference. However, the growing number of IoT devices and the associated data raise challenges in terms of data privacy, integration, security, and latency, affecting the efficiency of ML applications. Federated Learning (FL) addresses these challenges by allowing distributed clients to cooperatively train ML models on decentralized data across multiple edge devices. FL reduces latency by enabling local training with client data, and clients update a central server with their model weights, as shown in Fig. 1. 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].

One major challenge of FL is the variation in data distribution among clients, including differences in class distribution, quality, and quantity. This variation is known as non-independently and identically distributed (non-IID) data and can lead to degraded training accuracy [2]. FL also faces challenges due to dynamic device availability and high traffic

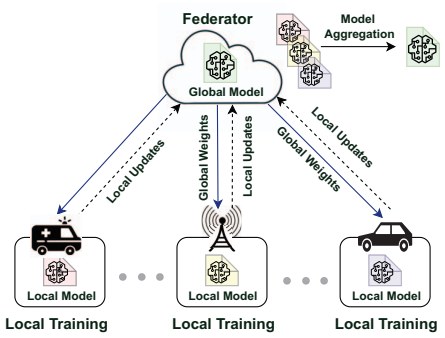


Fig. 1: Overview of federated learning with different devices

from numerous devices in each training round. Therefore, it becomes necessary to select clients strategically to improve training performance and model accuracy in each round. However, communication between the selected clients and a single central server can become a bottleneck, slowing down the training process. To address this challenge, edge servers in mobile edge computing (MEC) bring computational power closer to devices in mobile environments. The edge servers facilitate a hierarchical setting to coordinate clients within their vicinity, reducing traffic load on core cloud networks and minimizing latency in end-to-end communication between clients and servers [3]. 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.

Our contribution. We propose FedCime to address the mobility challenge in FL. 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 additional clients for training. Furthermore, to address the issue of data heterogeneity, we prioritize model

2767-9918/23/$31.00 ©2023 IEEE
DOI 10.1109/EDGE60047.2023.00042

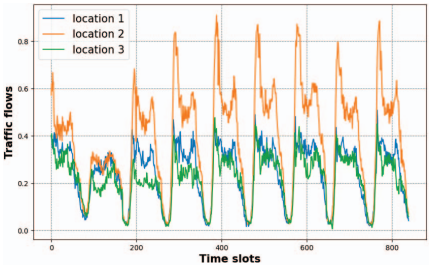
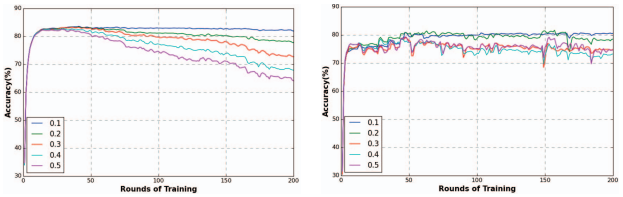


Fig. 2: Traffic flow changes. The traffic volume in 840 continuous 15-minute intervals for 3 locations in Northern Virginia/Washington D.C. capital region [4].



(a) Impact of clients migration without replacement

(b) Impact of clients migration with replacement

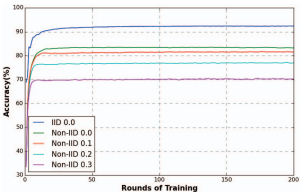
Fig. 3: (a) Accuracy of the global model with different migration rates without replacement using Non-IID data (b) Accuracy of the global with different migration rates with replacement using Non-IID data.

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.

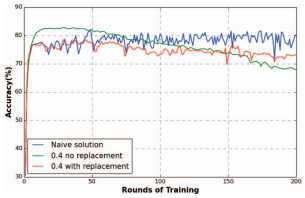
II. Motivation

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 FL, mobile devices in edge networks may not be consistently accessible since the number of nodes 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 [4]. In our experiments, we investigate the impact of client migration on the performance of the global model. 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 using the Federated Averaging (FedAvg) algorithm. The results reveal that migration without replacement leads to accuracy and convergence degradation, while replacing the clients results in better convergence but with some oscillation



(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 global model accuracy (b) Impact of the naive solution with 40% migration rate with and without replacement.

due to non-IID data. Using FedAvg, we observe that as the migration rate increases, the accuracy of the algorithm decreases. These findings highlight the need for a technique that limits the effect of client migration and non-IID data on accuracy and convergence.

2) Impact of data heterogeneity: In networked systems such as the Internet of Vehicles (IoV), 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. We conducted experiments to assess the impact of heterogeneous data on the training performance of 200 clients over 200 rounds using both IID and non-IID MNIST datasets. The results presented in Fig. 4(a) demonstrate that using non-IID data leads to poorer accuracy over the 200 training rounds than 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. Since FedAvg does not address the effect of data heterogeneity, the model's accuracy decreases when clients with non-IID data participate in training. 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 sample additional clients for training and randomly select weight updates from them during aggregation. Our experiment involves allowing a 40% migration rate during training. If N clients drop out, we sample N weight updates from the additional clients randomly preselected 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 the naive approach significantly mitigates the effect of migration. Furthermore, this result highlights that limiting migration's impact can improve the overall performance of FL.

Our work explores the challenges of training an FL model in a heterogeneous mobile network, such as vehicular networks, where sensors are diverse and data quality may be degraded. In contrast to the extreme cases studied in this section, we assume that some clients have IID data and others have non-IID data of varying quality due to sensor age and resource constraints. To simulate this scenario, we evaluate the model's accuracy 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

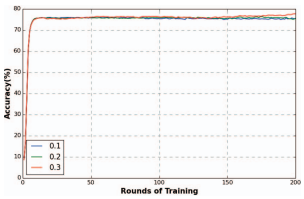


Fig. 5: Accuracy with 30% non-IID data. As the migration rate increases from 0.1 to 0.3, the accuracy of the global model drops.

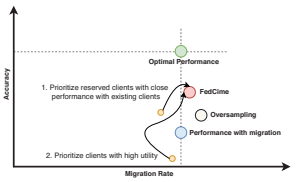


Fig. 6: Performance of FedCime. FedCime prioritizes high utility clients and reserved clients similar to those that did not migrate.

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 enhance the model's accuracy as the migration rate increases.

4) Challenges: Our motivating examples have shown that the naive approach can reduce the impact of migration in mobile environments. However, this method does not address two significant challenges: the limitation of dropout rates during training and the reduction of the effect of data heterogeneity on the global model. FedCime extends 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

In our proposed FL architecture, all clients can communicate with the *federator*, but we consider the scenario where clients are mobile, and their connectivity to the *federator* can vary with mobility. 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 for aggregation. Additionally, if a mobile node migrates, it will no longer be able to communicate with either the *federator* or other nodes.

A. Client Selection

The *federator* selects a subset of clients for training and 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 response time 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 $T_k = T_k^u - T_k^d$, 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:

$$Tier_k = \left\lceil \frac{T_k}{T_{max}} \times N_{tiers} \right\rceil \tag{1}$$

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. 1 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.

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 $0 < \alpha \leq 1$. 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 [5]. The cosine similarity for each client is calculated using the following:

$$S_k = \frac{\langle \Delta\theta_{c^k}, \Delta\theta_{c^a} \rangle}{\|\Delta\theta_{c^k}\| \|\Delta\theta_{c^a}\|} \tag{2}$$

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:

$$\gamma_k = 1 - \frac{\tau}{e^{Loss_k^2}} \tag{3}$$

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 $S_k := \gamma_k S_k$.

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 $S_1' \geq S_2' \geq ... \geq S_n'$ and $n = (1-\alpha)K$. The *federator* then chooses the first few clients

Algorithm 1 Improved Client Selection Algorithm

Require: M: number of local epochs; K: number of clients to be oversampled; α: proportion of clients selected for aggregation; τ: scaling factor for similarity weight.
Ensure: Global model θ.
1: **Initialization:**
2: $D_l \leftarrow 0$ {initialize the number of dropped clients}
3: $C \leftarrow$ random sample of K clients {oversample clients for training}
4: $\mathcal{S} \leftarrow$ random subset of αK 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**

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.

A summary of the approach is given in Algorithm 1. 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—MNIST and FashionMNIST—and compare it with FedAvg, FedProx, and the oversampling technique. 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 $0 < \sigma^2 \leq 1$.

Similar to Talukder and Islam [6], we used a logistic regression classifier for our experiments targeting sensors in MEC 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 efficient optimization and the categorical cross-entropy loss function to measure the dissimilarity in predicted and true class probabilities.

TABLE I: Accuracy of FL algorithms.

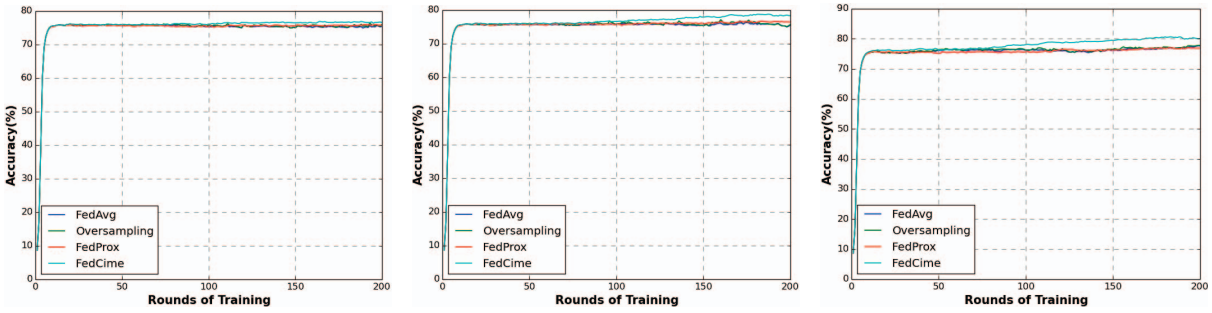
Datasets	Algoirthms	Migration Rates		
		10%	20%	30%
MNIST	FedAvg	75.51	76.59	77.72
	FedProx	75.75	76.51	76.84
	Oversampling	**76.02**	**75.69**	**77.74**
	FedCime	**76.64**	**78.37**	**80.08**
FashionMNIST	FedAvg	64.76	64.84	65.11
	FedProx	64.59	64.79	64.88
	Oversampling	**64.95**	**64.89**	**65.13**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

A. Evaluation Results

1) Comparison with baselines: We evaluate the performance of FedCime against the baselines. The result of our evaluation is presented 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 MNIST, the best performance of the FedAvg algorithm is 77.72%. FedProx handles data heterogeneity 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.

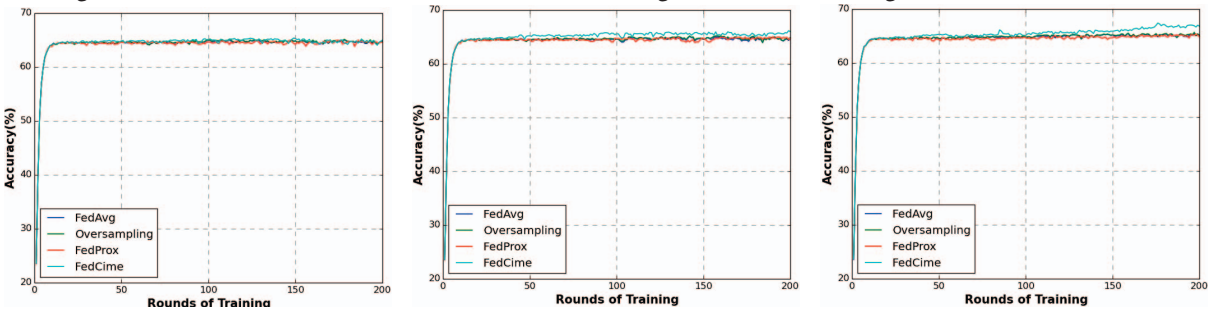
Since the FashionMNIST dataset is more complex than MNIST, the accuracy is lower compared to MNIST for all algorithms. However, FedCime algorithm outperforms the baselines, improving the accuracy by up to 1.95% in the best case. The oversampling approach improves the performance of both FedAvg and FedProx, 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: 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 MNIST, 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 FashionMNIST is noisier due to its higher complexity than the MNIST dataset, however, in most cases, FedCime performs better with higher accuracy than the baselines, improving the model's performance.



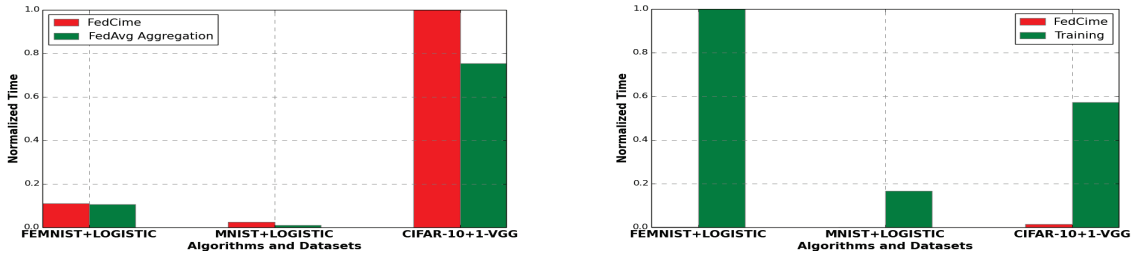
(a) 0.1% migration rate (b) 0.2% migration rate (c) 0.3% migration rate

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



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

Fig. 9: Performance of FedCime's computation time with FedAvg.

3) Computation cost analysis: We perform a computational analysis of FedCime, 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 1. We use three datasets for our evaluation: FEMNIST, MNIST, and CIFAR-10. We used logistic regression for FEMNIST and MNIST, while for CIFAR-10, we used a one-block VGG network [7].

Fig. 9(a) shows that 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 CIFAR-10, 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?: We evaluate FedCime across different scales of clients with the MNIST dataset, terminating the training after 150 rounds. We choose K clients and randomly divide the training set among the K clients, where we test $100 \leq K \leq 1000$. Fig. 10 shows that FedCime outperforms other algorithms with a small number of clients, and continues to perform well as the number of clients increases. Furthermore, our algorithm performs better than FedProx, as FedProx converges slowly. Combining FedCime with FedProx will enhance its performance, as we have accomplished with FedAvg.

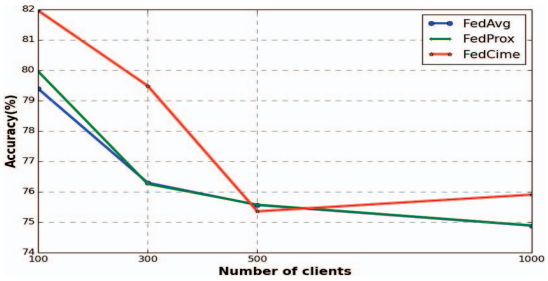


Fig. 10: Accuracy with different numbers of clients

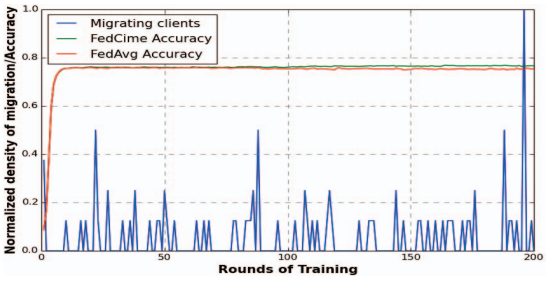


Fig. 11: FedCime's improvement despite migration

5) How good is FedCime at replacing migrating clients?: In this experiment, we track the number of migrating clients and calculate FL accuracy. We compare the accuracy of FedAvg and FedCime's in each training round. As depicted in Fig. 11, FedCime can leverage migration better than FedAvg. In the extreme scenarios where many clients migrate, FedCime is capable of selecting clients that enhance the model's accuracy rather than diminish it. This result implies that FedCime can enhance traditional FL techniques and ensure that mobile clients can participate effectively in FL training.

V. Related Work

FL is a privacy-preserving ML approach that uses approaches such as the FedAvg algorithm to allow global averaging on a server after completing local stochastic gradient descent on a subset of devices [8]. While FedAvg has been shown to converge under realistic settings, data heterogeneity can slow down the convergence rate [9]. Techniques such as sharing common data [10] and FedProx [11] 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 and CSFedAvg [12], [13] 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. [14] 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. [15] 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. [16] 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.

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 scalable and 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.

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] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.