scientific reports

Check for updates

OPEN

FedEff: efficient federated learning with optimal local epochs for heterogeneous clients

K. Narmadha✉ & P. Varalakshmi

Federated Learning (FL) enables collaborative model training without centralized data sharing; however, its efficiency often degrades under system and statistical heterogeneity across clients. Increasing the number of local epochs per round can enhance efficiency by enabling the global model to reach target accuracy in fewer communication rounds. Yet, excessive local training may cause client models to diverge from the global model, slowing convergence. To examine this trade-off, we conduct an empirical divergence analysis and show that consistent sufficient local updates across rounds can reduce the mean divergence between local and global models, thereby promoting faster and more stable convergence. Building on this insight, we propose a novel, efficient federated learning algorithm (FedEff) that assigns optimal local epochs to each client in heterogeneous settings. FedEff incorporates a server-side epoch selection mechanism, where the server selects an optimal number of epochs for each client, by considering the computation and communication speeds of all clients. The server uses an Estimated Round Time (ERT) to calculate the optimal number of local epochs for each client. Extensive simulations under heterogeneous computation and communication conditions confirm that the proposed approach achieves notable reductions in client waiting times and overall training duration within the considered simulation framework. Comparative results show that our method achieves better training efficiency than FedAvg and random epoch selection strategies, thereby establishing its effectiveness in improving federated learning performance under heterogeneous settings.

Keywords Federated learning, System heterogeneity, Statistical heterogeneity, Local epochs, Efficiency, FedAvg

Federated learning (FL) facilitates the building of collaborative Machine Learning (ML) models by a centralized server with the help of clients, who participate in the training process[1,2]. In Federated Learning, model training occurs within the clients, and only model parameters are communicated to the server for aggregation. The data privacy of the clients is preserved, as the training data never gets transmitted from the clients. The model-building process in machine learning involves solving an optimization problem to minimize the cost function of the model. In the federated setting, every client builds a local or client model with its own data to minimize its local cost function. A global federated model is maintained by a server, which aggregates local models of all clients. The federated optimization problem focuses on optimizing the global objective function, which is represented as the weighted sum of local objective functions of the respective clients[3].

FL is classified into two settings based on the type of participating clients. In a cross-silo FL, the clients are organizations such as hospitals, banks, educational institutions, etc., whereas in cross-device FL, the clients are smart devices like mobile phones, laptops, tablets, wearable, etc.[1,3]. In cross-device FL, numerous remote, isolated, resource-constrained smart devices participate in the training process. Each device typically shares a low bandwidth and an unstable network connection with the server. In contrast, cross-silo setting involves organizations as clients, which generate more training data and have greater computational, and storage resources. These organizations share a high-speed and stable network connection with the server. The characteristics of clients in both settings differ in terms of data generated by them (statistical heterogeneity) and resources like computational power, storage, memory, etc., available within them (system heterogeneity). Thus, in both settings, each client takes a different computation time to train its local model and a communication time to transfer the model weights to the server. Statistically heterogeneous clients have unbalanced, non-independent and non-identically distributed (non–IID) data[3]. Unbalanced data among clients refers to each client having

Department of Information Science and Technology, CEG Campus, Anna University, Chennai, India. ✉email: narmk27@gmail.com

a varying size of training data, which is very common in any federated environment. The non-IID data across clients indicate that the training data distributions vary from one client to another. Therefore, the above two types of heterogeneity in FL result in heterogeneous local models among clients.

Many federated optimization algorithms have been proposed that focus on various aspects such as the effectiveness, efficiency, and privacy of federated learning models[4]. A popular and widely used algorithm, FedAvg[5], proposed by McMahan et al., is the common starting point for other variants of federated optimization algorithms that have been proposed subsequently. The FedAvg algorithm comprises many iterative communication rounds between clients and the server. In each round, a sample of randomly selected clients trains their local models with a fixed number of epochs from the shared global model parameters sent by the server. After training, each client updates the local model parameters and transmits them to the server. In the initial round, the server randomly initializes the global parameters. The server updates the global parameters in successive rounds, with the weighted sum of the local parameters obtained from all participating clients. Each client is assigned a relative weight proportional to the number of samples in its training dataset. The total number of communication rounds, the number of local epochs chosen by each client in a round, the number of clients participating in a round, and the learning rates used in training processes are the hyperparameters that need to be optimally chosen for the effective and efficient convergence of FL models.

In Federated Learning, minimizing the total number of rounds is always desirable to reduce the transmission overhead between the server and the clients. Some of the methods to improve the communication efficiency in federated optimization are to increase the number of local updates or epochs performed by each client in a round, to compress the parameters that are transmitted between the server and the clients, and to reduce the number of clients participating in each round[4]. Many efficient federated optimization algorithms have been proposed, employing the aforementioned strategies to mitigate communication overhead[4]. Every method involves a hyperparameter, the value of which must be optimally chosen. In our work, the first method of increasing the number of local updates by clients is considered, to improve efficiency and reduce communication overhead during training. However, choosing an optimal value for the desired count of local training steps or local epochs performed by each client in a round is a significant challenge in a heterogeneous federated environment. A varying large number of local updates among clients will result in variable progressions between server and client models and increases the drift between them. However, sufficient local updates per round are needed for quick and stable convergence. In contrast, a smaller number will result in slow convergence of the server model, resulting in a rise in the required count of training rounds.

The number of local epochs done by clients will vary across clients and/or rounds depending upon their computational resource availability, which differs from client to client in a heterogeneous federated setting. This computational or system heterogeneity and data or statistical heterogeneity among the clients leads to heterogeneous local updates across and within training rounds. Eventually, this heterogeneous local progression causes local client models to diverge from the global server model. The client models will deviate towards their own local minima rather than the global minima. Consequently, this leads to an unstable and slow convergence of the global model, resulting in inefficient federated training. Thus, to build an efficient federated model in minimum rounds, there is a need to reduce this local model divergence by assigning an optimal number of local epochs to each client.

Our work aims to propose and validate an efficient federated algorithm for heterogeneous environments, which mitigates the local versus global model divergence by assigning optimal epochs to clients during training. A novel and efficient FL algorithm (FedEff) is proposed that achieves consistent local training by assigning the optimal number of epochs to every client, considering their varying computational heterogeneity. In our proposed algorithm, an epoch selection strategy is incorporated on the server side, where the server computes the optimal number of epochs executed by each client. The server considers both local training time and communication time for transmitting the model parameters of all clients when calculating the Estimated Round Time (ERT). The server uses ERT to find the optimal count of epochs for every client. This ensures that the clients' local updates are consistent throughout the training rounds. Since optimal epochs are selected based on the computation speeds of all clients, the waiting times of all clients are minimized.

Furthermore, our proposed algorithm reduces the impact of stragglers (slow clients), since only an appropriate amount of local training is assigned to clients based on their capabilities. Thus, our proposed algorithm enhances the alignment between local and global models during training by mitigating divergence, therefore producing a more effective and efficient federated model. Our contributions are summarized below.

- An empirical analysis through simulated federated settings is carried out by measuring the progression and divergence of the local versus global model during training rounds, using two different metrics: progression difference (PRD) and parameter deviation (PAD). The progression difference measures the variation between the median of local parameters of all clients and global parameters. The parameter deviation measures the mean Euclidean distance between global and local parameters in each round.
- Computational heterogeneity is simulated by varying the number of local epochs across clients and/or across rounds according to four different classes: Equal Consistent (EC), Unequal Consistent (UEC), Equal Inconsistent (EIC), and Unequal Inconsistent (UEIC). In all classes, 'Equal' means that the number of local epochs performed by every client is the same in any round. 'Consistent' means that any specific client performs the same number of epochs in each round.
- Our analysis indicates that if each client performs an optimal number of local updates and remains consistent across consecutive rounds, the mean divergence between local client models and the global federated model is reduced, resulting in a quick and stable convergence of the global model.
- In our proposed algorithm, FedEff, an epoch selection mechanism is included on the server side, where the server chooses the optimal number of local epochs for each client based on the computation and communica-

tion speeds of all clients. The server estimates the optimal time needed for a training round using the clients' average computation and communication times. This Estimated Round Time (ERT) is used by the server to find the optimal count of local epochs for each client.

The remaining sections of the manuscript are structured as follows. The related work is detailed in Sect. "Related work". Section "Proposed work" elaborates on our proposed work. The experimental results are analyzed in detail in Sect. "Experiments". Section "Results and discussion" presents a case study, while Sect. "Case study" provides the conclusions and discusses future work.

Related work

The most popular and first proposed algorithm for federated optimization, FedAvg, has been analyzed by many existing studies. The experimental results of different models were tabulated by finding the number of rounds needed to achieve the target accuracy. The results are obtained for a varying number of clients in both IID and non-IID scenarios, with the number of local epochs varying between 1, 10, and 20. From the results, it is observed that, in the case of a single local epoch, the number of training rounds required to achieve a target accuracy is very high for any number of participating clients. However, with the local epoch size as 10, the number of rounds is significantly reduced in both IID and non-IID partitions. Thus, as the number of local epochs increases, the number of local updates per round also increases, enabling the model to achieve the target accuracy in fewer rounds. Even though increasing local updates will help in quick convergence of the global federated model, an excessive number of local updates can cause a client drift where the local models move towards their respective local minima instead of the global minima. FedAvg can diverge in the case of a large number of epochs, especially in heterogeneous clients.

FedProx[6] is the first variant of FedAvg, which addresses system and statistical heterogeneity in federated optimization. FedProx is more robust to heterogeneity in a federated setting by allowing clients to do a variable amount of work in each round. However, this will lead to heterogeneous local updates among the clients. As a result, too many local updates in some clients cause their local models to diverge, especially if their data are statistically heterogeneous. To address this, FedProx adds a proximal term to the local objective function to add more stability in convergence. This proximal term acts as a regularization method for clipping the local parameters so that they are closer to the global parameters. However, choosing an optimal value for this regularization parameter is more important for quick and stable convergence.

Juwon Park et al.[7] have discussed a novel approach named AMBLE to eliminate stragglers due to system heterogeneity in federated learning. Their approach aims to adaptively adjust the mini-batch size and the count of local epochs during training to increase the work efficiency of clients with varying system heterogeneity. AMBLE allows clients to select their local epoch sizes based on the local update time of the slowest client in each round. However, finding the slowest client for each round is challenging and adds a lot of computation on the client side, thereby delaying the federated training process. In federated learning, data heterogeneity (non-IID data across clients) leads to client drift, where local model updates deviate from the global optimization direction, slowing convergence, and degrading performance. SCAFFOLD (Stochastic Controlled Averaging)[8] mitigates this by introducing control variates that estimate the update directions of both the server and each client. The difference between these estimates quantifies client drift and is used to correct the local updates. This variance reduction technique ensures stable training, significantly reduces the number of required training rounds, and mitigates the negative effects of heterogeneity. But it needs to maintain the states for the client and server between the rounds. Furthermore, the control variables must be exchanged among the clients and the server, which consumes additional network bandwidth. Wang et al.[9] proposed a novel FL framework to address the heterogeneous local progression of clients. They have provided a theoretical understanding of the heterogeneous local updates of the clients and explained the objective inconsistency problem. To eliminate the inconsistency problem, they proposed FedNova, in which local updates are normalized by the clients before aggregation.

Another approach to reduce local model divergence is to apply variance reduction methods through regularization techniques[10,11]. This paper introduces AIDE[12], an accelerated variant of the DANE algorithm, designed to improve communication efficiency in distributed optimization. Unlike standard distributed methods that require frequent synchronization, AIDE reduces the number of communication rounds needed to reach convergence. This is achieved by allowing each local algorithm to return an approximate solution to a local subproblem. The paper[13] introduces DANE (Distributed Approximate Newton), a communication-efficient distributed optimization method. Instead of frequent gradient exchanges like distributed SGD, DANE lets each client solve a regularized local subproblem that combines its local objective with global gradient information and a quadratic penalty. This design approximates a Newton step, aligning local updates with the global objective and reducing client drift. Yong Shi et al.[14] have added a sparse regularization term to local objectives to mitigate client drift in FL. However, most approaches hinder local convergence while reducing drift. An empirical study on different regularization methods is presented in[15], but all aspects of heterogeneity in FL are not considered. Incorporating acceleration techniques during training, such as including momentum on the server side[16–18] and on the client side[19–21], enhances the performance of FedAvg by speeding up convergence. Liang Gao et al.[22] proposed an algorithm that tracks and minimizes the variations between local and global parameters. Jed Mills et al.[23] have written a survey on client-side optimization strategies used to reduce communication overhead in federated learning. Chengxu Yang et al.[24] have characterized the impact of heterogeneity in FL through large-scale smartphone data. Ahmed et al.[25] have conducted a comprehensive empirical study on heterogeneous FL and have stated that heterogeneity significantly impacts a model's divergence.

To select optimal epochs for clients, appropriate training intensities should be considered based on their training speeds and transmission times. Existing work has addressed this problem by proposing different

strategies, such as selecting suitable clients based on their resource availability, choosing different local iterations for different clients depending on their computation speeds, etc. FedCS[26] adopts a client selection mechanism based on the resources possessed by each client. Clients with more efficient resources are likely to be repeatedly selected for training. However, this leads to a biased global model, as updates from resource-constrained clients are rarely included. FedAda[27] chooses different local iterations for clients based on their training capabilities. It assigns a different local iteration for each client in each round. Zeng et al.[28] have proposed an algorithm to assign the appropriate training intensities to heterogeneous clients. In their proposed algorithm, every pair of clients is made as the slowest and fastest client, to choose the correct pair with minimum training latency. However, this will add additional computational overhead during the training process, especially if the number of clients is large. Xia et al.[29] have introduced a reinforcement strategy to enable partial local adaptive model training in heterogeneous FL. Yujia Wang et al.[30] have discussed asynchronous adaptive FL optimization by incorporating a delay-adaptive learning adjustment approach. Unlike existing mechanisms, our proposed algorithm adopts a feasible approach to select optimal epochs with a minimum overhead and helps reduce the impact of stragglers.

During federated training, protecting the integrity of the model and ensuring that the model remains trustworthy and untampered with are crucial due to threats such as poisoning and malicious updates. Standard defensive methods include robust aggregation to filter abnormal contributions, anomaly detection to spot poisoned updates, and gradient clipping to bound adversarial effects. Cryptographic methods such as secure multiparty computation, homomorphic encryption, and differential privacy further protect updates in transit while reducing manipulation risks. A promising direction to enhance model integrity in Federated Learning is the integration of blockchain. For example, VFChain[31] introduces a blockchain-based framework that ensures verifiability through committee-based aggregation and on-chain proof recording. Auditability is supported by an authenticated data structure that improves search efficiency and allows for secure committee rotation. FedPD[32] enables federated open set recognition by separating client models into closed-set and open-set subnetworks and aligning them between clients. This allows accurate detection of unknown classes while outperforming existing methods.

This FL framework[33] uses a Conjoint Prototype-Aligned loss to address class imbalance by aligning client-side training with a globally balanced objective. It defines a global class-level target based on the overall data distribution. It refines the local training of each client using class prototypes, ensuring that underrepresented classes are properly learned in distributed medical data sets. Federated open set recognition (FedOSR)[34] enables open set recognition in federated medical settings, allowing models to identify known diseases and detect unseen ones without sharing patient data. The Federated Open Space Sampling (FedOSS) framework addresses the lack of unknown samples by synthesizing virtual unknowns near decision boundaries and aggregating them across clients, improving the model's ability to separate known and unknown classes. FedRVR[35] is a federated semi-supervised learning (FSSL) method for clients with limited labeled data. It employs versatile regularization, combining data-guided regularization (pseudo labels from a strong global model) and model-guided regularization (feature perturbations from a weaker global model) to improve prediction consistency. At the server, a relation-guided directional aggregation ranks client models based on pairwise relationships, producing a superior global model for robust training and an inferior model to guide client training further. This approach improves local training and global aggregation, outperforming existing FSSL methods across benchmarks.

Proposed work

Local vs global model progression and divergence

In FedAvg, clients train their local models for more than one epoch by doing a local update for every mini-batch of data in each round. Each client splits its training dataset into many batches and performs an SGD (Stochastic-Gradient Descent) update for each mini-batch. The total number of SGD updates done by any client in a round depends on its mini-batch size and the number of epochs. It is well known that in a federated setting, the clients are statistically heterogeneous, with each client having a different distribution of training data and a dataset of varying size. Each client chooses a different number of epochs (E) and divides the training data (n_i) by varying batch sizes (b). Hence, this will lead to a highly varying number of heterogeneous local updates ($u = E\frac{n_i}{b}$) done among all clients in a round. Moreover, in a federated environment, the system characteristics and network connectivity to the server differ for each client. Clients with more efficient computational resources do more local updates than clients with fewer resources, within a specified time. Clients located closer to the server and having high-speed network connections can significantly reduce the communication time required to exchange model parameters with the server. Hence, the amount of local training performed by each client within a stipulated time varies, resulting in a heterogeneous local progression between clients.

An empirical analysis of the progressions of the client models versus the global federated model is performed in both homogeneous and heterogeneous federated settings. Statistical and system heterogeneity are the main contributing factors that determine the nature of local model progressions during training. They also play an essential role in determining how much the local models diverge from the global model in subsequent training rounds. Thus, both types of heterogeneity are simulated in the experiments for a detailed analysis. The data or statistical heterogeneity is simulated by splitting the dataset into three categories: balanced IID, unbalanced IID, and non-IID. To simulate computational heterogeneity in federated experiments, we choose the number of local epochs as the main hyperparameter, which is varied across clients and/or rounds. This value is assigned according to four different classes – Equal Consistent (EC), Unequal Consistent (UEC), Equal Inconsistent (EIC), and Unequal Inconsistent (UEIC) which collectively reflect the entire nature of system heterogeneity in clients. In all classes, 'Equal' means that the number of local epochs performed by each client is the same in any round and 'Consistent' refers to the same number of local epochs performed by any specific client in every round. A brief description of the four classes is given in Table 1 and more explanations are provided below.

Class	Epochs/Client	Epochs/Round	Description
EC	Same	Same	Epochs do not vary across clients and across rounds
UEC	Varies	Same	Epochs vary across clients but do not vary across rounds
EIC	Same	Varies	Epochs vary across rounds but do not vary across clients
UEIC	Varies	Varies	Epochs vary across clients and across rounds

Table 1. Different classes of local epochs among clients.

a) Class 1: EC (Equal Consistent Epochs) This is the simplest, ideal, and straightforward case, which is the least practical in a federated real-time setting. In this case, all clients execute the same number of local epochs in every round. For example, if the local epoch size is *e*, all participating clients execute *e* epochs in every round.

b) Class 2: UEC (Unequal Consistent Epochs) In this category, each client executes a different number of local epochs, but that number remains the same in all subsequent rounds. For example, a client *i* executes e_i epochs in all training rounds, whereas another client *j* executes e_j epochs in every round. In this case, the total number of epochs contributed by all clients in each round remains the same. Clients consistently execute the same number of epochs in all training rounds.

c) Class 3: EIC (Equal Inconsistent Epochs) In this case, all clients execute the same number of local epochs in any round, and that number varies between rounds. In one round *t*, all clients execute e_t epochs, and in another round *k*, all clients execute e_k epochs. In this case, the total number of epochs trained by all clients varies in each round.

d) Class 4: UEIC (Unequal Inconsistent Epochs) This is the most practical case applicable in any federated setting. Each client chooses the number of local epochs within the baseline value *e* in each round. Hence, the number of local epochs among different clients varies within a round and between subsequent rounds.

All four different classes of local epochs are incorporated in FedAvg and presented as DLE-FedAvg (Different Local Epochs – FedAvg) in Algorithm 1. Consider a centralized federated setting with M clients and one server communicating for *R* rounds. Each round is denoted by *r*. Each client *i* has a training dataset X_i of size $|X_i| = n_i$. The total number of training samples across the client population is given by $|X| = \sum_{i=1}^{M} |X_i| = n$. The global model parameters are denoted by *GW* and *LW* refers to the local model parameters. η is the learning rate used to train local models in each round. B_i is the number of mini-batches for any client *i* and E_b is the baseline value for local epochs for all classes. The local epochs are assigned to clients according to any one of the four classes: EC, UEC, EIC, or UEIC.

In the EC class, the number of local epochs E_i for each client *i* is assigned as the baseline value E_b in every round *r* as shown in Equation (1).

$$E_i = E_b \quad \forall i \quad \forall r \tag{1}$$

In the UEC class, each client *i* chooses a random value y_i between one and E_b as E_i, which remains the same in all training rounds. This is represented in Equation (2).

$$E_i = y_i \quad \forall r \quad where \;\; y_i \in (1, E_b) \tag{2}$$

A uniformly chosen value y_r, between one and E_b is assigned to all clients as their local epoch values in a training round *r* in the EIC class. y_r varies between subsequent rounds, and a new random value is selected in each round. Equation (3) represents this case.

$$E_i = y_r \quad \forall i \quad where \;\; y_r \in (1, E_b) \tag{3}$$

In the UEIC class, the number of local epochs chosen by each client *i* in each round *r*, is a random value y_{ri} between one and E_b as shown in Equation (4). Each client is assigned a new value in each round.

$$E_i = y_{ri} \quad where \;\; y_{ri} \in (1, E_b) \tag{4}$$

In each round, all clients train their local models for their respective assigned number of epochs. Each client splits its dataset into many mini-batches, with a single update done per batch in every epoch. An SGD weight update (W_k) is done per mini-batch *k*, by calculating the gradient of the loss function (h_k) of the mini-batch *k* as shown in Equation (5).

$$W_k := W_{k-1} - \eta \nabla h_k(W_{k-1}) \tag{5}$$

In the first round, all clients train their local models with randomly initialized global model parameters GW_0. The global parameters (*GW*) are updated with the weighted sum of local parameters (*LW*) obtained from all M clients, after each round, as shown in Equation (6). The relative weight is assigned as $\frac{n_i}{n}$ for any *i*th client.

$$GW_r = \sum_{i=1}^{M} \frac{n_i}{n} LW_i \tag{6}$$

Computational heterogeneity among clients is simulated in our experiments by implementing DLE-FedAvg in all four classes of local epochs. Similarly, statistical heterogeneity is simulated in DLE-FedAvg by splitting the training dataset according to the three categories mentioned earlier.

Progression and divergence metrics

In federated learning, it is desirable to see global and local models with uniform progression and similar trajectories as training progresses in subsequent rounds. Any significant divergence will result in a slow and unstable convergence of the global model. Deviation between client and global models occurs in the case of too many local updates and/or heterogeneous local progression within clients. Our study uses two metrics, progression difference (PRD) and parameter deviation (PAD), to quantify this deviation. The progression difference (PRD) measures the variation between a representation of local model parameters of all clients and global model parameters in each round. The median of local model parameters is taken as the representation of the local models of all the clients. The absolute difference between the median of the Euclidean norm of local parameters and the Euclidean norm of global parameters is calculated. A minimum difference will ensure that there is not much variation between the local and global models regarding their progressions after each round. A high PRD value implies that the local models are progressing at different rates than the global model. Ideally, this value should decrease to zero as the training moves closer to convergence. This shows that the local models and the global model are progressing uniformly.

Consider M clients, x^i as the local parameters of any client i and x^* as the global parameters. The PRD is calculated as given in Equation (7).

$$PRD = Median[||x^1||, ||x^2||, ..., ||x^M||] - ||x^*|| \tag{7}$$

The parameter deviation (PAD) measures the average distance between the global model and the local model parameters. The average distance is the mean Euclidean distance between the local and global parameters in each round, specifying the mean deviation between them. This value should be minimum and decrease as training progresses, indicating that the global and local models are aligning together. The PAD value is calculated as given in Equation (8).

$$PAD = Mean[||d^1||, ||d^2||, ..., ||d^M||] \quad where \quad d^i = x^i - x^*. \tag{8}$$

The progression and deviation of the local vs. global models are analyzed with the PRD and PAD metrics. As shown in Fig. 1, the length of the vectors, that is, the parameters or weights of a model $(x^1, x^2, ...x^*)$ and the distance between them $(d^1, d^2, ...)$ are measured using the L2 norm. Larger distances $d^1, d^2..$, will lead to higher PAD values and indicate greater divergence between models. An experimental analysis is carried out by observing how these values change when the local epochs are assigned according to four different classes (EC, UEC, EIC, UEIC) and under three different data distributions – balanced IID, unbalanced IID, and non-IID. Experiments are carried out by measuring PRD and PAD values in two different classification models – a Logistic Regression model and a Multi-Layer Perceptron classifier.

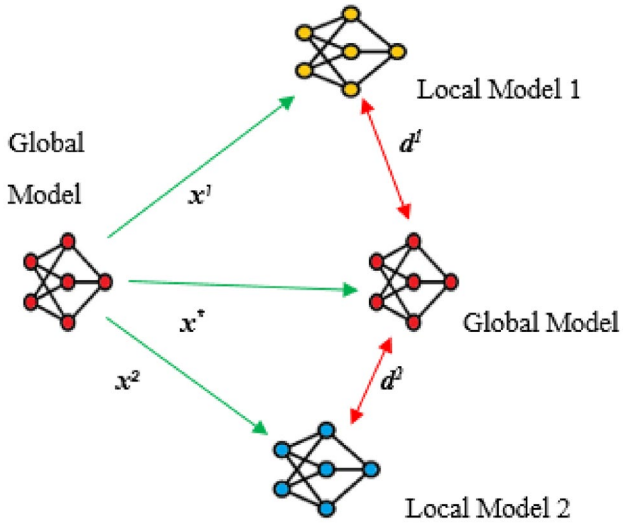

Fig. 1. Local(x^1,x^2,..) vs Global(x^*) parameters progression and divergence(d^1,d^2,..).

Intialize M, E_b, R, GW_0, η
Choose Class $\in ([EC] or [UEC] or [EIC] or [UEIC])$
while each round r from 1 to R **do**
  while each client i from 1 to M in parallel **do**
    if Class EC: **then**
      $E_i = E_b$
    else[Class UEC:]
      $E_i = y_i \in (1, E_b)$
    end if
    if Class EIC: **then**
      $E_i = y_r \in (1, E_b)$
    else[Class UEIC:]
      $E_i = y_{ri} \in (1, E_b)$
    end if
    $LW_i = ClientLocalUpdate(i, GW_{r-1}, E_i)$
  end while
  $GW_r = \sum_{i=1}^{M} \frac{n_i}{n} LW_i$
end while
ClientLocalUpdate(i, GW, E_i)**:**
$W_0 = GW$
while each epoch e from 1 to E_i **do**
  while each batch k from 1 to B_i **do**
    $W_k := W_{k-1} - \eta \nabla h_k(W_{k-1})$
  end while
end while
$LW_i = W_k$
Return LW_i

Algorithm 1. (DLE - FedAvg)

Proposed FedEff algorithm

In federated learning, client and global federated models should travel on similar trajectories to attain global minima. To achieve this, the progressions of local and global models must follow the same pattern after each training round. In addition, the average deviation between the local and global models should be minimal. The smaller the progression difference and deviation in the case of heterogeneous local updates, the better the convergence of the global model. In our study, an experimental analysis is performed to analyze the progression and divergence of local versus global models in heterogeneous federated learning. Our empirical findings show that consistent optimal local epochs by each participating client in every training round reduce the divergence of the local models and results in a quick and stable convergence of the global federated model. It is also observed that a smaller count of local epochs results in slower convergence of the global model and reduced testing accuracy. Thus, the number of local epochs must be chosen optimally for each client, so that it does not lead to divergence of its local model but at the same time contributes more to a quicker convergence of the federated model by doing a sufficient number of local updates. This, in turn, would improve efficiency in federated training, as the global model converges in a few rounds. Hence, an epoch selection mechanism is incorporated on the server side, where the server decides an appropriate number of local epochs for each client based on its available system resources and network speed. This enables all clients to perform consistently in all rounds and efficiently utilize their resources.

In FedEff (Algorithm 2), the server determines and assigns the optimal number of local epochs for each client, based on the computation and communication times of all clients. Hence, in FedEff, an initialization round is added to estimate these values in which the local models are pre-trained by the clients for a single epoch. This round provides two significant information to the server: the local model parameters transmitted by the clients, which can be used to initialize the global parameters, and the time taken for training the model for a single epoch by each client. Generally, the server initializes the global parameters with random values before the first round. To speed up the training process and achieve faster convergence, better initial values are needed. In the initialization round, the weighted sum of local parameters from all clients is used to initialize the global parameters, and the server computes an optimal number of local epochs for each client. These numbers are used by the respective clients in subsequent training rounds to ensure consistent optimal local training.

In the initialization round, all participating clients train local models for a single epoch. After training, all clients send their local model parameters along with the number of training samples used to train the models to the server. The server obtains the local parameters LW_i from each client i and measures the total time taken TT_i to receive them. The total training time is equal to the sum of the computation, upload, and download times. The server calculates the compute time σ_i for a single epoch, the upload time β_i, and the download time δ_i from TT_i for each client i using Equation (9).

$$TT_i = \sigma_i + \beta_i + \delta_i \tag{9}$$

The single epoch training time σ_i of any client *i* depends on the number of local updates in a single epoch($\frac{n_i}{b}$) and the time taken for a single update as shown in Equation (10), considering the CPU capability(CP_i), number of CPU cycles(*t*) needed to find the gradient and local update time γ_i.

$$\sigma_i = \frac{n_i}{b}(\frac{t}{CP_i} + \gamma_i) \tag{10}$$

The server finds the average computation time σ_a, upload time β_a, and download time δ_a of all clients. In the case of synchronous updating of global parameters, the server needs the local parameters of all participating clients to update the global parameters. However, the server cannot wait indefinitely for client updates; hence, the server will predefine a time window for a training round. This Estimated Round Time (ERT) must be determined optimally by the server based on all clients' mean communication and computation speeds. An optimal ERT will enable fair participation among heterogeneous clients, increase work efficiency, and minimize waiting times. To achieve this, all clients must train the local models for an optimal number of epochs, according to the availability of computational resources.

An Epoch Deciding Factor (EDF) τ, whose value ranges from (0,1], is optimally selected by the server based on the computation and communication speeds of all clients. With a suitable selection of the EDF, the ERT value can be varied accordingly. If the server wants a longer round time, it can choose higher values for τ closer to 1. Higher EDF values are meaningful in cases where the mean computation time is low and the difference between the maximum computation time (σ_{max}) and the minimum computation time (σ_{min}) is less. Lower EDF values are preferred if the server wants quick rounds. This value can be fixed or, if needed, can be adaptively chosen by the server, during training to decide the round time based on the availability of clients, stopping criteria of the federated training, run-time cost, etc. The EDF value is multiplied by the baseline epoch value (E_b) to find the approximate number of local epochs(*E*), which is used in the calculation of the ERT as shown in Equation (11).

$$E = \tau * E_b \quad where \quad \tau \in (0, 1] \tag{11}$$

The ERT is determined based on the assumption that the model is trained for *E* epochs, with the local training time σ_a and the two-way communication time ($\beta_a + \delta_a$) as shown in Equation (12). A single training round consists of a download of global parameters, local model training, and an upload of updated local parameters. Thus, the approximate number of local epochs (*E*) can be fine-tuned based on the suitable selection of EDF.

$$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil \tag{12}$$

After calculating ERT, the server calculates the available computation time CT_i for each client *i* using Equation (13).

$$CT_i = ERT - (\beta_i + \delta_i) \tag{13}$$

Then, the number of optimal local epochs for all clients is computed using their respective computation times using Equation (14).

$$E_i = \lfloor (CT_i/\sigma_i) \rfloor \tag{14}$$

Each client *i* executes E_i epochs in each training round. This ensures that clients consistently execute their assigned local epochs throughout the training rounds. This epoch-selection strategy incorporated by the server efficiently utilizes client resources. It also helps to mitigate the straggler effect caused by slow clients, as the idle times of all clients are minimized. The server also guarantees that the difference between the maximum and minimum computation times falls within a threshold value. The actual completion time (ACT) will deviate slightly from the ERT, for which the server always adds a threshold time to the ERT to complete a round. If some of the clients' computational capabilities significantly change over time, the difference between average ACT and ERT will be higher. To adapt to these dynamic changes, the initialization round can be repeated or a suitable EDF value can be chosen if at least 50% of the clients' ACTs deviate significantly from the ERT during the training rounds. In this case, the ERT and the number of local epochs for each client are updated according to the current settings. Each client's waiting time (WT) is calculated as the difference between its ACT and the maximum ACT. Our proposed algorithm minimizes the average waiting time (WT_{avg}) and thus reduces the total federated training time.

In each training round *r*, every client *i*, performs $E_i \frac{n_i}{b}$ local updates with the current global parameters(*GW*) used to initialize the local model in the training process. The server fixes the mini-batch size *b* for all clients. Each client *i* calculates its respective number of mini-batches ($B_i = \frac{n_i}{b}$), which remains constant in all rounds. Local training is repeated for E_i epochs in every round. The corresponding local updates are sent to the server for aggregation. After obtaining the local parameters from all clients, the server does a weighted sum of them to update the global parameters. The above process gets repeated for R rounds. The steps in the initialization round of the proposed FedEff algorithm are shown in Fig. 2. The initialization and training rounds are illustrated in Fig. 3.

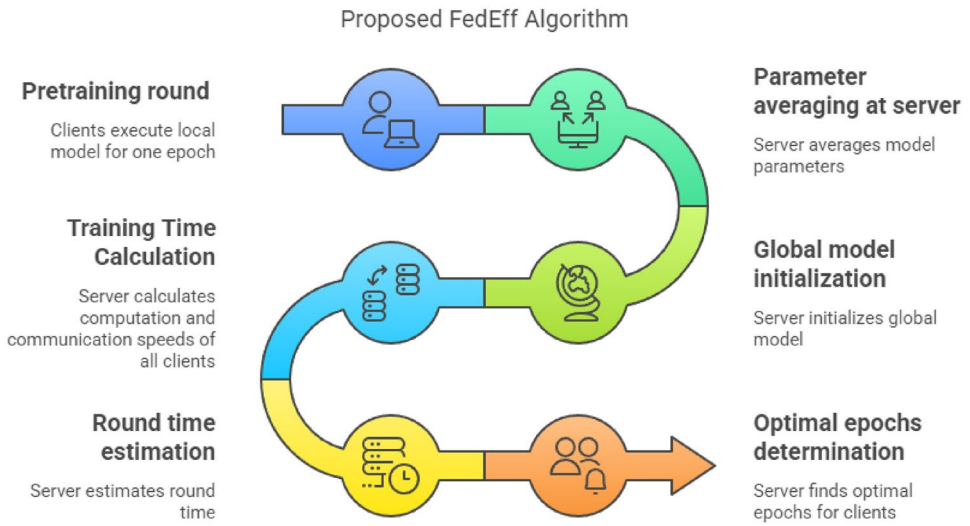

Fig. 2. Steps in initialization round of FedEff algorithm.

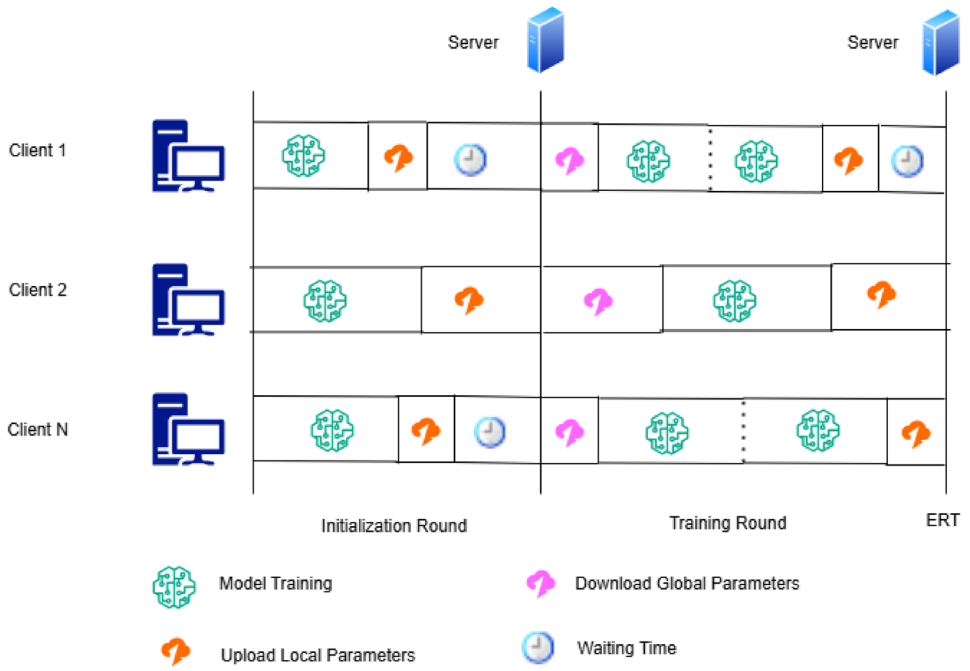

Fig. 3. Initialization and training round in FedEff algorithm.

To calculate GW_0, E_i; To estimate ERT
Initialization Round: Choose M, E_b, b, R, η
while each client i from 1 to M in parallel **do**
    $LW_i = Pretrainoneepoch(i, b)$
    $TT_i = \sigma_i + \beta_i + \delta_i$
    $\sigma_i = \frac{n_i}{b}(\frac{t}{CP_i} + \gamma_i)$
end while
$\sigma_a = \frac{1}{M}\sum_{i=1}^{M}\sigma_i$
$\beta_a = \frac{1}{M}\sum_{i=1}^{M}\beta_i$
$\delta_a = \frac{1}{M}\sum_{i=1}^{M}\delta_i$
$E = \tau * E_b$
$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil$
while each client i from 1 to M **do**
    $CT_i = ERT - (\beta_i + \delta_i)$
    $E_i = \lfloor (CT_i/\sigma_i) \rfloor$
end while
$GW_0 = \sum_{i=1}^{M}\frac{n_i}{n}LW_i$
Training Round:
while each round r from 1 to R **do**
    while each client i from 1 to M in parallel **do**
        $LW_i = ClientLocalUpdate(i, GW_{r-1}, E_i)$
    end while
    $GW_r = \sum_{i=1}^{M}\frac{n_i}{n}LW_i$
end while

Algorithm 2. (FedEff)

Our proposed algorithm FedEff, has several advantages over FedAvg and other existing algorithms. It achieves quick and stable global model convergence through consistent optimal local training and better initialization of global parameters. FedEff is highly efficient, as it minimizes idle times of all clients, thereby increasing their work efficiency. This, in turn, helps reduce both the total training time and the associated costs of federated learning. The advantages of our proposed FedEff algorithm are highlighted in Fig. 4.

Experiments

Simulation

DLE-FedAvg is experimentally analyzed by implementing multinomial Logistic Regression (LR) and Multi-layer Perceptron (MLP) models in Python with the MNIST dataset[36] used to classify handwritten digits. The MNIST dataset consists of images of handwritten numbers from 0 to 9. It has 60,000 training examples and 10,000 testing examples. The MLP model has a single hidden layer with 256 units of ReLu activation. The input layer consists of 784 units, as each image has 784 (28*28) pixels. The output layer has 10 units corresponding to 10 digits. Both models are implemented in federated settings with four classes of local epochs (EC, UEC, EIC, UEIC) and under three data distributions: balanced IID (Independent and Identically Distributed), unbalanced IID, and

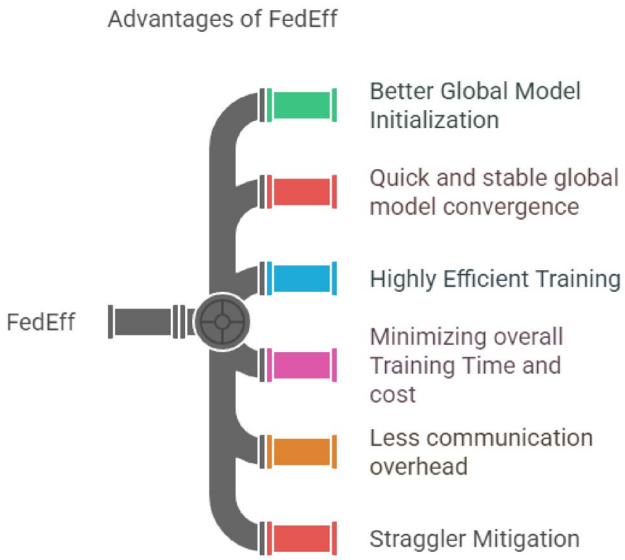

Fig. 4. Advantages of FedEff.

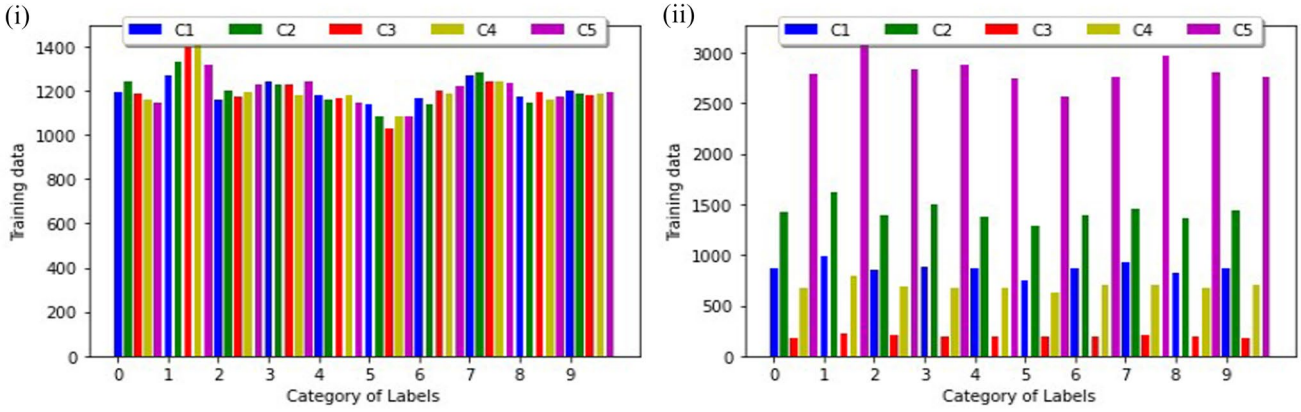

Fig. 5. Data distribution (**i**) Bal IID (**ii**) Unbal IID.

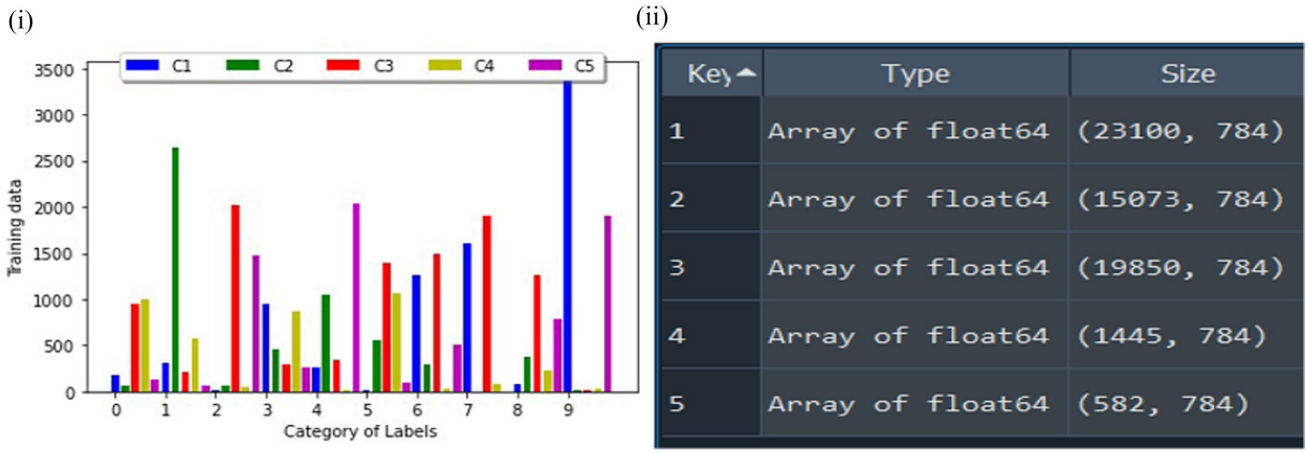

Fig. 6. Data distribution (**i**)non-IID (**ii**)Unbal IID split(α=0.5).

non-IID. In balanced IID, training and testing datasets are equally divided among the total number of clients. Figure 5i shows this distribution for five clients. In all figures, (i) refers to the left plot, and (ii) denotes the right. For generating unbalanced IID data, the Dirichlet distribution with parameter α is used, which produces the probability split values among the total number of clients. Each client is assigned a probability value between zero and one. Thus, each client gets its individual training and testing data share from the training and testing datasets, respectively, based on the assigned probability. The distribution of data among five clients under the unbalanced IID category is shown in Fig. 5ii. This is a quantity-based skewed distribution with clients having datasets of varying sizes. However, each client has training samples with a uniform distribution of labels. Figure 6ii shows the unbalanced division of the training dataset between five clients with α=0.5.

A typical non-IID data split, as shown in Fig. 6i, is obtained by dividing training examples among clients based on the labels in them. The MNIST dataset has training images of ten categories corresponding to digits 0 to 9. The Dirichlet distribution is used to generate the probability split for all categories for each client. For α =0.1, the number of training samples with a specific label can vary significantly between clients. Some clients may have fewer than ten samples, whereas others may have more than a hundred samples of that specific label. This will yield a label-based, highly skewed distribution. Thus, the non-IID data distribution is generated among the clients, with each client having an uneven distribution of labels in its training dataset. All experiments are carried out with SGD as the local optimizer and a constant learning rate of 0.001. The mini-batch size is fixed to ten in both the Logistic Regression model and the MLP model.

The number of local epochs is varied according to four classes with different baseline values. Figure 7i shows the epochs distribution in the EC class for five clients in five rounds with the baseline value 10. Each of the five clients trains the model for ten epochs in each round in the EC class. The distribution of epochs in the UEC class is shown in Fig. 7ii, where clients are consistent and assigned the same number of epochs in five rounds. For instance, the first client trains the model for five epochs in every round, whereas the third client trains only for one epoch in each round. Figure 8i shows the distribution of epochs in the EIC class, where all clients choose the same number of epochs in a round. In the first round, all clients chose ten epochs, and in the second round, all clients chose eight epochs. The distribution of epochs in the UEIC class is illustrated in Fig. 8ii, where each client chooses a different epoch value in each round within the baseline value of 10.

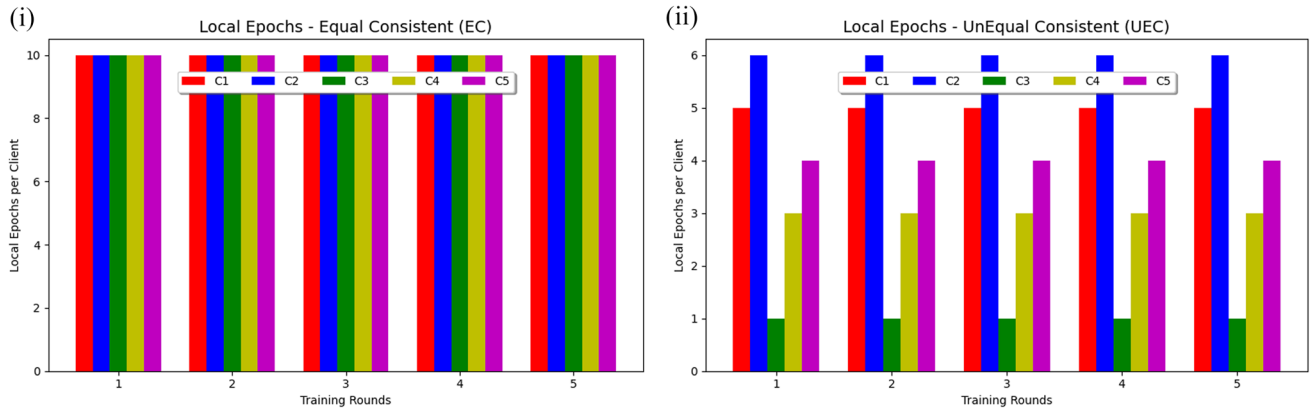

Fig. 7. Local Epochs for Clients(C1,C2,C3,C4,C5) (**i**) Equal consistent (EC) (**ii**) Unequal consistent (UEC).

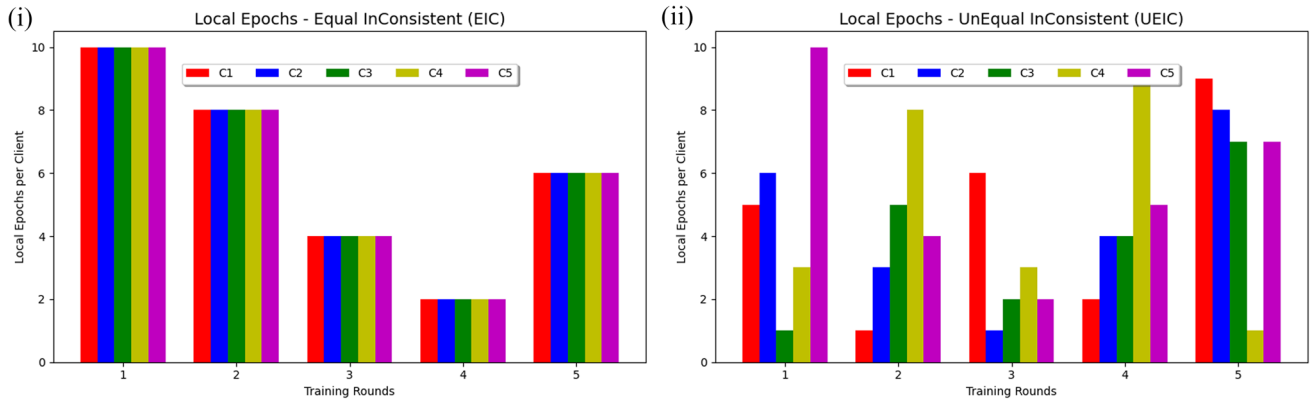

Fig. 8. Local epochs for clients(C1,C2,C3,C4,C5) (**i**) Equal inconsistent (EIC) (**ii**) Unequal inconsistent (UEIC).

Results and discussion

Analysis of DLE-FedAvg

Existing work has highlighted that FedAvg can diverge in the case of a large number of local epochs. However, sufficient local epochs are needed to achieve better accuracy and quick convergence in a few rounds. The impact of varying local epochs between clients and/or training rounds has not been explored much in the existing literature. The four metrics used in our analysis are test accuracy of the global model, global training loss, local vs. global parameter progression difference (PRD) and parameter deviation (PAD). Logistic regression and MLP models show similar patterns in progression difference and parameter deviation metrics. For a baseline epoch of 10, the PAD value shows a stable and smooth decrease without fluctuations in the EC and UEC classes in the LR model, as shown in Fig. 9i. The other two classes show fluctuations with higher variations in the EIC class. Significant deviations are present, as the variation between total epochs is greater in the EIC class. Figure 9i displays PAD patterns in the balanced IID case. Similar patterns are noticed for the non-IID and unbalanced IID distributions, also with higher PAD values being obtained for them. As in the case of PAD, PRD also shows a smooth decrease in the EC, UEC classes, and minor deviations are present in the other two classes, as shown in Fig. 9ii, under the unbalanced IID distribution. PRD value moves closer to zero as the global model approaches convergence. Similar results are obtained for the MLP model. Figure 10ii shows the PAD comparison in the MLP model for different data distributions. It is observed that non-IID has higher PAD values than balanced IID.

The experiments are carried out with baseline epoch values: 1,5,10, and 20. Only the EC class is implemented with a baseline epoch value of one. The local vs. global model progression difference decreases with decreasing baseline value. Considering the baseline epoch value of one, the difference is least, indicating that both the global and local models' progressions are equal. The parameter deviation is the least when the baseline epoch value is one. It increases when the baseline value increases, as shown in Fig. 10i. The increase is observed in all distributions and in all classes. When the baseline value increases beyond 20, the parameter deviation gradually increases as the training progresses. Our empirical results also demonstrate that FedAvg diverges, that is, the mean divergence between local and global models increases in the case of a large number of epochs. This divergence is present in both types of heterogeneity. This, in turn, will lead to overfitting of local models and results in lower global test accuracy due to poor generalization of clients' data.

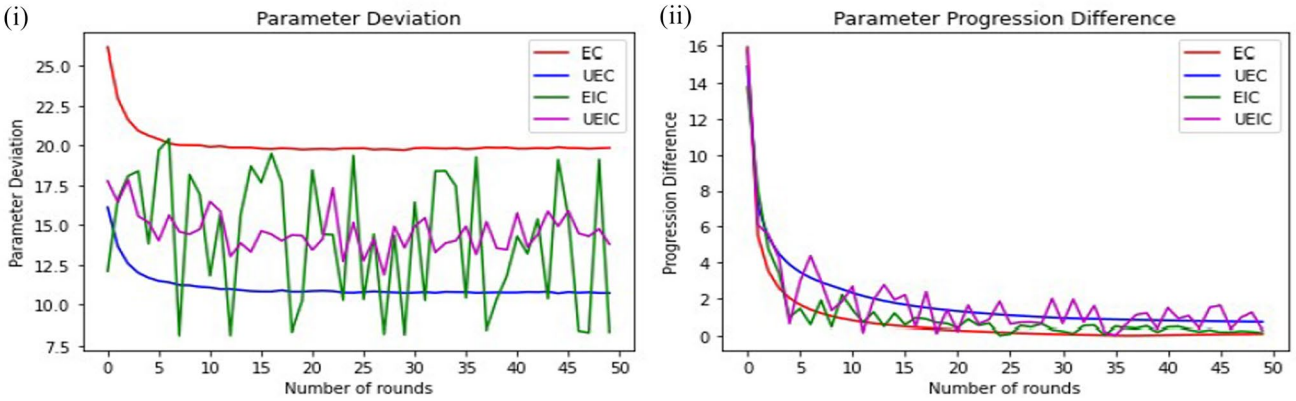

Fig. 9. Logistic Regression (**i**) PAD (Bal IID) (**ii**) PRD (Unbal IID).

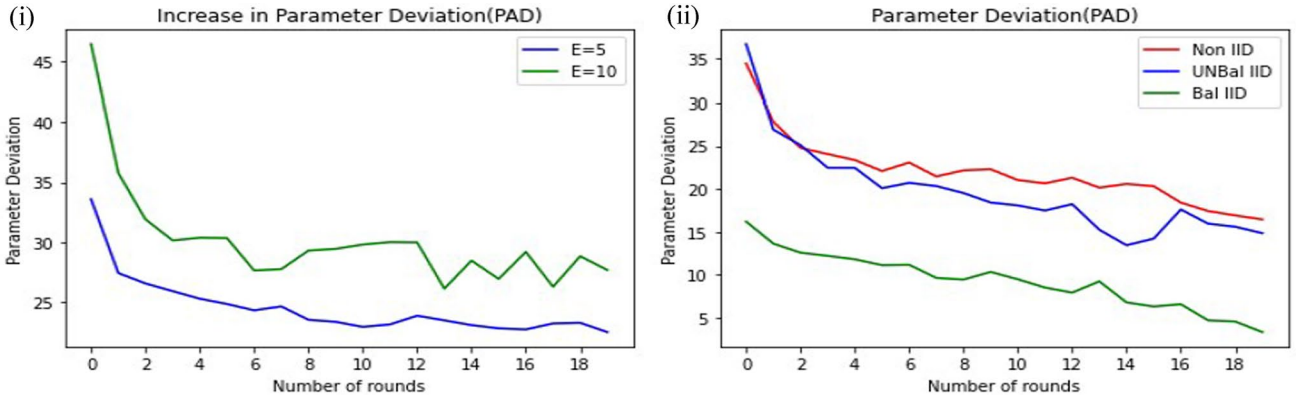

Fig. 10. MLP (**i**) PAD Increase (**ii**) PAD Comparison.

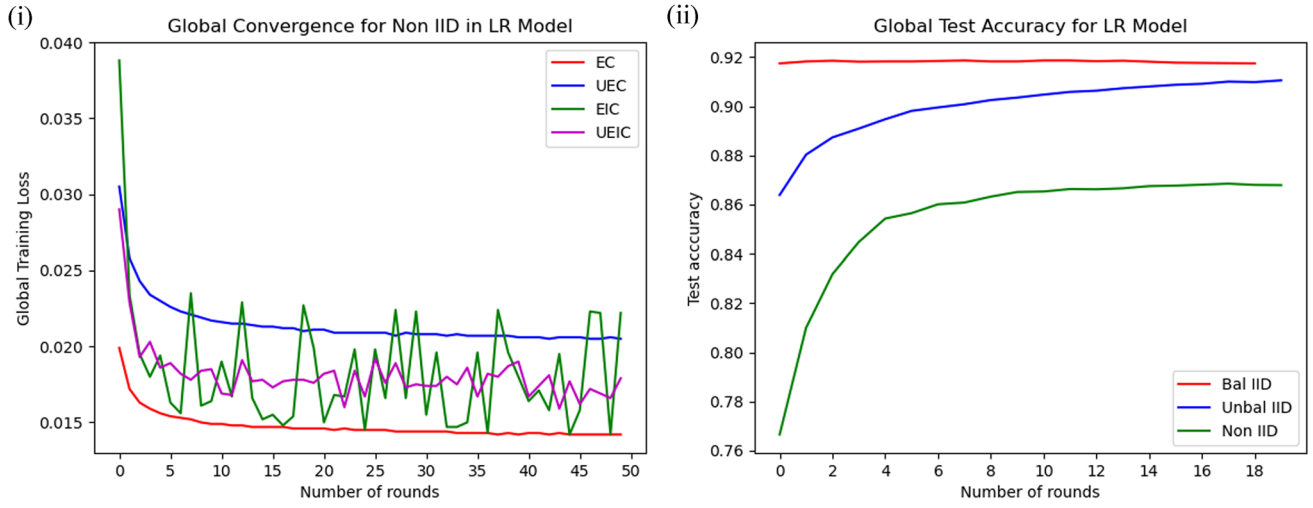

Fig. 11. Logistic regression (**i**) Global training loss (non-IID) (**ii**) Test accuracy (Bal IID vs. Unbal IID vs. Non IID).

The global accuracy of the trained federated model is higher for IID distributions than for non-IID distributions. In the Logistic Regression model, the global test accuracy is 92.6% for balanced IID, 90.8% for unbalanced IID, and 87.3% for non-IID. The test accuracy gradually increases as training progresses, as shown in Fig. 11ii, for all three distributions. Similarly, in MLP, the test accuracy ranges from 96% to 98% for balanced

and unbalanced IID distributions and is approximately 94% for the non-IID distribution, as shown in Fig. 12i. As in the case of PRD and PAD metrics, the baseline epoch value plays a significant role in deciding the test accuracy of the global model. Our results show that a sufficient and optimal number of epochs per round is necessary to achieve the desired target accuracy within a few rounds. A smaller number of epochs will yield a lower test accuracy, whereas a larger number of epochs will result in a decrease in the accuracy after a few initial rounds due to over-fitting of the local models. The test accuracy is lower by 2-3% when the model is trained with fewer epochs compared to the accuracy obtained with a larger number of epochs in the same number of rounds, as shown in Fig. 12ii.

The global training loss pattern indicates the nature and speed of convergence of the global model. Clients need to train their local models only for a sufficient number of epochs in each round to yield generalized models that will not memorize their data. Thus, client models need to have stable and smooth convergence. This, in turn, would result in a stable and faster convergence of the global federated model. Since the global training loss is calculated from the average training loss of the local models, convergence patterns in the local training loss will also be reflected in the global loss. Figure 11i depicts the global training loss of the Logistic Regression model in the case of the non-IID distribution. Figure 11i shows a stable and smooth decrease in the training loss curves for the EC and UEC classes, where the total epochs are consistent in all rounds.

Our empirical analysis shows that local models must be trained for the optimal number of epochs in each round, to get desired accuracy in fewer communication rounds and a quick, stable convergence. The total number of epochs in each round should not be too small or too large. In addition, consistent local training mitigates the drift between server and client models, which results in better alignment of the models during training. Therefore, in our proposed algorithm FedEff, the training of local models with optimal local epochs is incorporated, so that consistency can be achieved across training rounds in heterogeneous federated settings.

Performance analysis of FedEff

This study performs model training on real hardware with specified datasets, while the federated learning setting is simulated by partitioning the dataset between clients. The temporal aspects of the system, including the computation time, waiting time, download time, and upload time, are not measured directly from the physical hardware. Instead, these time units are simulated according to the parameters and assumptions defined in our experimental setup. This approach allows us to analyze performance trends under controlled and reproducible settings, while ensuring that all reported improvements in time-related metrics are interpreted as simulated values rather than real-world absolute times. Thus, all time-related performance metrics in this study, such as Estimated Round Time (ERT), waiting time (WT), actual completion time (ACT), and total training time, reflect simulated times based on the assumptions and configuration of our experimental setup.

Our proposed algorithm, FedEff, achieves efficiency by selecting optimal epochs for each client, so that clients with varying system heterogeneity can perform consistent local training, which in turn results in fast convergence of the global model and mitigates the waiting times of all clients under the given simulation settings. This consistency in local training can be maintained in all training rounds if the computational capabilities of the clients do not vary significantly as training progresses. Unlike existing work, our approach does not compute epochs for each client in each round, thus reducing the additional overhead in the training process. Since the waiting times of all clients are minimized, the total federated training time is reduced within the simulation environment, thus improving the efficiency in model training. The epochs calculation in FedEff is empirically validated using different ranges of simulated computation times and for different EDF values. The experimental results are tabulated in Table 2. The total number of simulated clients M is chosen as 20, and the baseline epochs E_b as 10. For the sake of simplicity, the simulated computation times of the clients when training the model for a single epoch are uniformly chosen from three different ranges: Type A [1 (σ_{min}) to 3 (σ_{max})], Type B (1 to 5) and Type C (1 to 7). In Type A, the computation time of the slowest client is at most three times that of the fastest. Type A denotes the lowest level of heterogeneity, and Type C represents the highest level. The simulated computation times of the clients are randomly sampled from the specified range. It is assumed that

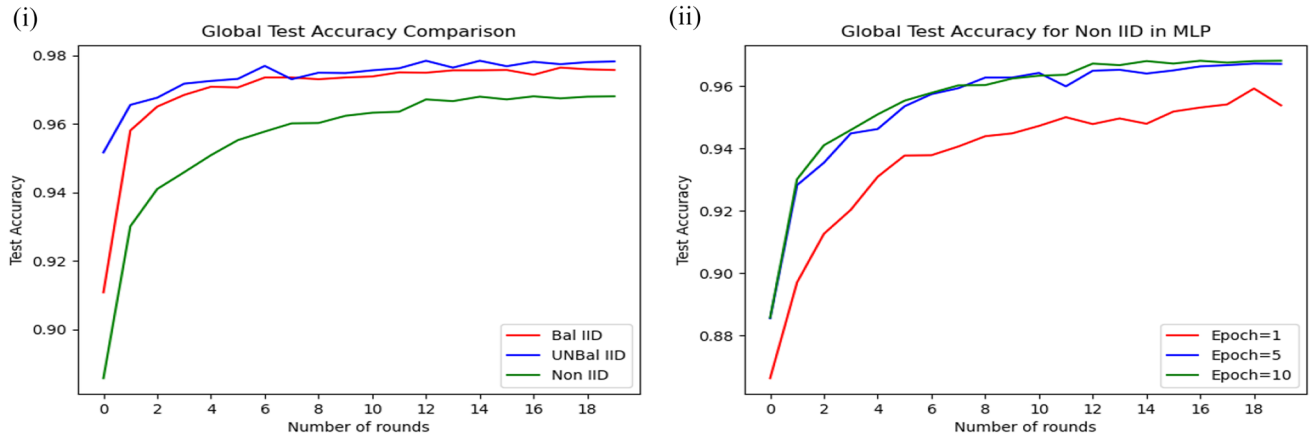

Fig. 12. MLP (**i**) Test accuracy (Bal IID vs. Unbal IID vs. Non IID) (**ii**) Global test accuracy vs. Epochs.

Range(σ)	σ_a	EDF(τ)	E_{min}	E_{max}	E_{avg}	ERT	ACT_{avg}	WT_{avg}
Type A	2.07	0.3	3	6	3.9	8	7.96	0.82
Type A	1.92	0.5	4	11	5.7	11	11.08	0.91
Type A	2.01	0.7	5	12	7.65	15	14.84	0.92
Type B	2.92	0.3	2	7	3.6	10	10.02	1.34
Type B	3.10	0.5	3	12	6.2	16	15.8	1.44
Type B	3.24	0.7	5	15	8	23	23.02	1.5
Type C	4.10	0.3	2	9	3.85	14	13.66	1.48
Type C	3.83	0.5	3	13	6.2	19	19.11	1.62
Type C	4.26	0.7	4	23	8.85	28	27.42	1.89

Table 2. Local epochs and ERT calculation in FedEff.

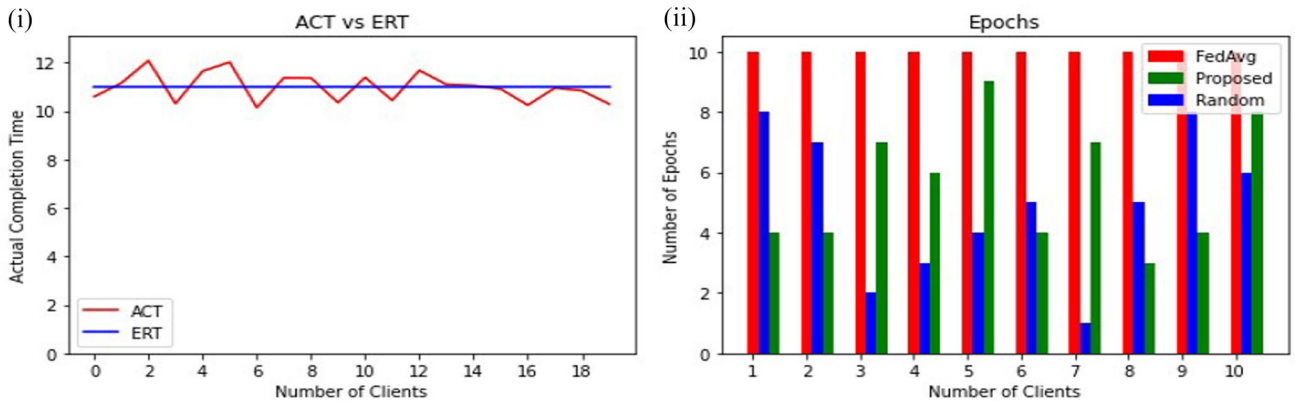

Fig. 13. (**i**) ACT vs ERT (**ii**) Number of Epochs.

the upload times are higher than the download times. The simulated upload times of the clients are randomly chosen between 0.1 (β_{min}) and 1(β_{max}). The simulated download times are randomly selected between 0.1 (δ_{min}) and 0.5 (δ_{max}).

Table 2 shows that the difference between Estimated Round Time (ERT) and average actual completion time (ACT_{avg}) is minimal in all cases. This shows that our approach effectively estimates the round time as shown in Fig. 13i. From the figure, we can observe that for some clients the simulated ACTs are almost the same as ERT. The difference between ACT and ERT is also minimal. The variation ranges between -1 and 1 and most of the values are closer to zero. Figure 13i presents results for 20 clients with Type A computation time, baseline epoch value of 10 and EDF of 0.5. Under the given simulation settings, the ACTs will deviate from ERT within a threshold limit. The significance of the Epoch Deciding Factor (EDF) is that it can be used to decide the value of E, which in turn determines the duration of ERT. The server chooses the EDF value based on the average computation time of all clients. If the difference between σ_{min} and σ_{max}, is less, the EDF value can be higher to enable clients to train the model for more epochs in a round. Since the variation in the simulated computation times of the clients is minimal, the average simulated waiting time is also minimal. The variation between the minimum epochs E_{min} and the maximum epochs E_{max} is also minimum, as shown in Table 2. When the heterogeneity level increases, the average waiting time WT_{avg} increases under the assumed simulation setup, as shown in Table 2. It is highest for the Type C category, for an EDF value of 0.7. The difference between E_{min} and E_{max} is also maximum in this category, mainly due to the significant variation in the simulated computation times of the clients. Moreover, a high EDF value will lead to an increased number of epochs, and since the differences in client computation times are high, the variation among epochs is also high. Table 2 indicates that the variation between E_{min} and E_{max} is the highest for the Type C category, for an EDF of 0.7.

In our study, the EDF was not adaptively tuned during the experimentation. Instead, it was manually assigned with fixed values in the range (0,1]. Specifically, we validated the results by considering representative values of EDF = 0.3, 0.5 and 0.7. These values were chosen to capture the low, moderate and relatively high influence of the EDF, ensuring a balanced evaluation under different operating conditions. In future work, we will explore adaptive strategies to determine the optimal value of EDF during real-time training through different optimization methods. An EDF of 0.5 is suitable in many cases. However, the server can increase or decrease it based on the computation and communication times of all clients. Longer training times per round are beneficial if the communication times of the clients are relatively higher than the computation times. In this case, an EDF of 0.7 can be chosen, leading to an increase in ERT as shown in Table 2. In the Type C computation range, selecting lower EDF values will reduce waiting times under the given simulation setup, as the difference between the minimum and maximum epochs is less. The average simulated waiting time increases as the heterogeneity level

	Type A			Type B			Type C		
EDF	0.3	0.5	0.7	0.3	0.5	0.7	0.3	0.5	0.7
Client 1	3	4	6	5	8	11	4	6	8
Client 2	2	4	5	3	4	6	4	6	8
Client 3	5	8	12	4	5	7	7	12	17
Client 4	5	9	14	3	5	7	4	6	8
Client 5	2	4	6	2	4	5	2	3	4
Client 6	4	6	9	7	11	16	3	5	8
Client 7	3	5	7	2	4	5	2	4	5
Client 8	4	6	10	7	11	16	6	10	14
Client 9	3	5	8	4	7	9	2	3	5
Client 10	3	4	6	2	3	4	3	6	8

Table 3. Epochs obtained for different EDF values.

Method	E_{min}	E_{avg}	E_{max}	ACT_{min}	ACT_{avg}	ACT_{max}	WT_{min}	WT_{avg}	WT_{max}
FedAvg	10	10	10	12	20.8	30	0	9.5	18
Random	1	5	9	2.5	11.25	23	0	11	20.5
FedEff	3	5.7	9	10	10.5	11.9	0	1.2	1.9

Table 4. Comparison of different methods.

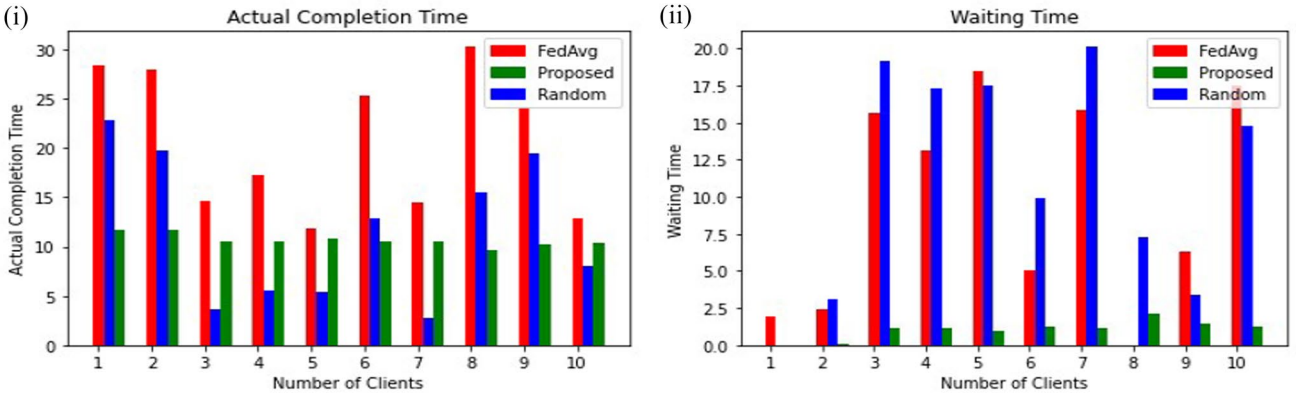

Fig. 14. (**i**) Actual completion time (**ii**) Waiting time.

increases. When the category changes from Type A to Type B, the average waiting time increases approximately from 0.9 to 1.5. And, for a change from Type B to Type C, the average waiting time increases to 1.9 under the given simulation setup, as shown in Table 2.

The number of epochs obtained through simulation experiments for 10 clients with representative values for the EDF as 0.3, 0.5 and 0.7 with the three types of heterogeneity (Type A, Type B, Type C) is specified in Table 3. From the table values, we can observe that as the EDF increases, the number of epochs increases for all clients in the three types of heterogeneity. As EDF approaches 1, the difference between the minimum epochs E_{min} and the maximum epochs E_{max} increases in three types of heterogeneity. We compared our proposed approach with the equal-epoch approach, as in FedAvg, and a random approach, where the epochs are selected randomly for each client within the baseline value. Table 4 summarizes the detailed results under the given simulation settings, with respect to the epochs, ACTs, and WTs for the three approaches for a baseline epoch value of 10, with Type A heterogeneity, the number of clients as 10 and the EDF of 0.5. The number of epochs obtained for each client in all approaches through simulated experiments is shown in Fig. 13ii. In FedAvg, each client trains the model for the same number of epochs, i.e., 10 epochs. In our proposed approach, the server selects an optimal number of epochs for each client. In the case of the random approach, each client uniformly chooses the number of epochs between one and ten. Our proposed approach shows less variation between the minimum epochs and the maximum epochs compared to the random approach, as shown in Table 4. Figure 14i shows the simulated actual completion time (ACT) for all clients in three approaches. It is observed that there is a significant variation in the clients' ACTs in FedAvg and the random approaches. The variation is higher in the random approach compared to FedAvg. In contrast, in our proposed approach, there is no significant variation

in the clients' ACTs. This is mainly due to the optimal allocation of epochs to each client based on the average computation time of all clients. In addition, the results in Table 4 show that the difference between ACT_{min} and ACT_{max} is the least in our approach, and it is almost 86-89% lower than the other two approaches within the specified simulation setup.

The simulated waiting times of the clients in the three approaches are shown in Fig. 14ii, which shows that the waiting times are significantly lower in our proposed approach compared to other approaches under the given simulation settings. The variation in simulated waiting times of all clients is minimal in our proposed approach. At least a few clients have waiting times closer to zero in our proposed algorithm. FedAvg and other approaches have longer waiting times for heterogeneous clients. If heterogeneity increases, that is, the variation in computation times among clients increases, the average waiting time also increases. However, in our approach, there is a minimum variation in the average waiting time, when computational heterogeneity increases between clients, as observed in Table 2. As shown in Table 4, the average waiting time WT_{avg} and the maximum waiting time WT_{max} in FedEff are approximately 85-87% lower than those of the FedAvg and random methods. It is also observed that the difference between WT_{min} and WT_{max} is the least in our proposed algorithm and is almost 86-88% lower than the other two approaches within the specified simulation setup.

Three classification models - Logistic Regression, MLP classifier with MNIST dataset and Convolutional Neural Network (CNN) model with CIFAR-10[37] dataset are used to validate the performance of our proposed algorithm. The CNN model has two convolutional layers of (3x3) kernel with 32 and 64 channels, respectively. A (2x2) maxpooling layer follows each convolutional layer. The CIFAR-10 dataset has 50,000 samples as training data and 10,000 samples as testing data. Each image in the dataset is an RGB image of 32*32 pixels. The labels in the images belong to 10 classes. In all models, our proposed algorithm achieves efficiency by attaining the target accuracy in fewer rounds compared to FedAvg, FedProx, and FedNova. All three models were individually implemented using the four FL algorithms with the number of clients set to 10, a baseline epoch value of 10, and an EDF of 0.5. In Logistic Regression, clients are modeled with Type A heterogeneity, in MLP Type B is considered, whereas in CNN the clients are modeled with Type C heterogeneity. In our experiments, we consider a synchronous FL setting in which the same set of clients participates in each round. Since client computation and communication times are assumed to remain constant across rounds, the round duration is determined by the slowest client. Consequently, the total federated training time is calculated as the product of the number of rounds required to reach the target accuracy and the per-round time. The training time per round is ACT_{max}, which is the training time of the slowest client. Table 5 specifies the number of rounds and the total time taken to attain target accuracy on the IID data for all three models using four approaches.

In FedAvg, each client is assigned an equal number of epochs, i.e., 10 epochs. In the case of FedProx and FedNova, a random number of epochs is assigned to the clients. In our proposed FedEff algorithm, optimal epochs are assigned to each client. In the case of the LR model, clients are modeled with Type A heterogeneity, thus, the maximum actual completion times ACT_{max} in FedAvg, FedProx, FedNova and FedEff are considered to be around 30, 20, 22, and 12, respectively, under the given simulation settings. These values were obtained from our simulation experiments. In the case of the MLP model, clients are modeled with Type B heterogeneity, and the simulated ACT_{max} in FedAvg, FedProx, FedNova and FedEff is around 47, 36, 38, and 18, respectively. In the case of the CNN model, clients are modeled with Type C heterogeneity, and the simulated ACT_{max} in FedAvg, FedProx, FedNova and FedEff is around 62, 46, 50, and 24, respectively. The total federated training time for the LR model with IID data using FedAvg is the product of the number of rounds needed to attain a target accuracy of 92% and the simulated ACT_{max} of FedAvg. Thus, total time = 25*30 = 750 time units. Similarly, all the other total training times in Tables 5 and 6 are computed. Table 6 specifies the number of training rounds and the total time taken to attain a target accuracy with non-IID data for the three models using four approaches.

In FedEff, assignment of optimal epochs to each client helps to reduce model divergence, and the global federated model is initialized with the average weights of local models that are obtained from clients in the initialization round. These two factors contribute to achieving convergence and the target accuracy in fewer rounds. FedEff achieves a target accuracy of 92% in the Logistic Regression model in around 18 rounds compared to other algorithms, which require more rounds to obtain the same accuracy. Moreover, our proposed algorithm significantly reduces the total federated training time under the given simulation setup. FedEff needs around 216 simulated time units in the Logistic Regression model, 396 time units in the MLP model, and 648 time units in the CNN model to complete the training with IID data. FedAvg has the highest training time as it uses equal epochs in all the rounds and the round completion time is longer. As shown in Table 6, the training times for the non-IID distribution are longer compared to the IID distribution in all models, since the number of rounds required to attain target accuracy is larger. Even in the case of non-IID data, our proposed algorithm has the least simulated total training time compared to other algorithms within the specified simulation setup. The

	LR-MNIST	MLP-MNIST	CNN-CIFAR10
Target ACC	92%	98%	70%
FedAvg	25 (750)	29 (1363)	34 (2108)
FedProx	21 (420)	27 (972)	33 (1518)
FedNova	20 (440)	25 (950)	30 (1500)
FedEff	18 (216)	22 (396)	27 (648)

Table 5. Number of rounds and total time taken (mentioned within brackets in time units) to attain target accuracy with IID data.

	LR-MNIST	MLP-MNIST	CNN-CIFAR10
Target ACC	88%	94%	64%
FedAvg	29 (870)	34 (1598)	41 (2542)
FedProx	24 (480)	31 (1116)	39 (1794)
FedNova	25 (550)	30 (1140)	37 (1850)
FedEff	22 (264)	26 (468)	35 (840)

Table 6. Number of rounds and total time taken (mentioned within brackets in time units) to attain target accuracy with non-IID data.

Client(i)	$\sigma_i(1-5)$	β_i (0.1-1)	δ_i (0.1-0.5)	TT_i
1	4.95	0.36	0.12	5.43
2	2.52	0.33	0.24	3.09
3	1.28	0.33	0.46	2.07
4	1.54	0.54	0.47	2.55
5	2.14	0.60	0.14	2.88
6	3.40	0.64	0.43	4.47
7	3.90	0.27	0.26	4.43
8	3.55	0.52	0.22	4.29
9	2.89	0.32	0.22	3.43
10	1.33	0.63	0.42	2.38

Table 7. Specification of clients.

results from both tables indicate that our proposed algorithm, FedEff, outperforms other existing algorithms in achieving efficiency by mitigating the total federated training time in heterogeneous federated settings. Although the findings demonstrate the effectiveness of the proposed method, the study does not model variations in client computation resources or client availability during training. Thus, the proposed method assumes that the relative computation speeds of the clients do not vary between or within the training rounds. This assumption may not hold in highly dynamic ad hoc mobile environments, where resource availability can fluctuate. We acknowledge this limitation and will address it by incorporating such real-world dynamics in our future work.

Case study

In this section, a separate case study is presented to explicitly explain the calculation of the number of epochs, waiting time (*WT*) and actual completion time (*ACT*) for each client in the three methods: FedAvg, Random, and FedEff. The number of clients *M* is 10, with computation times sampled from Type B(1-5) heterogeneity level, and baseline epochs E_b is 10. The computation time (σ_i) needed to train the model for a single epoch, the upload time (β_i) and the download time (δ_i) for each client *i* are specified in Table 7. These values are uniformly sampled from the specified ranges. In all three methods, the number of epochs, waiting times, and actual completion times of all clients are derived from the values specified in Table 7. The average computation time (σ_a) is 2.69, the average upload time (β_a) is 0.57, and the average download time (δ_a) is 0.28. The Epoch Deciding Factor(τ) is chosen as 0.5 for this case.

FedAvg (Equal Epochs)

In the FedAvg method, all clients execute the same number of epochs in each round. In our case, every client trains the model for 10 epochs. The calculation of *ACT*, *WT* for client 1 and client 2 is shown in Equations (15)-(18). Following the same procedure, *ACT* and *WT* for the remaining clients are calculated, and all values are tabulated in Table 8. The actual completion time for a client *i* is the time it takes to complete the model training in a round. This includes the time required to download the parameters from the server, the computation time needed to train the model for E_i epochs, and the time utilized to upload the model parameters.

$$ACT_1 = E_1 * \sigma_1 + \beta_1 + \delta_1 = 10 * 4.95 + 0.36 + 0.12 = 49.98 \tag{15}$$

$$ACT_2 = E_2 * \sigma_2 + \beta_2 + \delta_2 = 10 * 2.52 + 0.33 + 0.24 = 25.77 \tag{16}$$

The waiting time for each client *i* is calculated as the difference between its ACT_i and the maximum actual completion time ACT_{max}, considering all clients.

$$WT_1 = ACT_{max} - ACT_1 = 49.98 - 49.98 = 0 \tag{17}$$

$$WT_2 = ACT_{max} - ACT_2 = 49.98 - 25.77 = 24.21 \tag{18}$$

	FedAvg			Random			FedEff		
Client(i)	E_i	ACT_i	WT_i	E_i	ACT_i	WT_i	E_i	ACT_i	WT_i
1	10	49.98	0	9	45.03	0	3	15.33	0.8
2	10	25.77	24.21	7	18.21	26.82	6	15.69	0.44
3	10	13.59	36.39	10	13.59	31.44	11	14.87	1.26
4	10	16.41	33.57	1	2.55	42.48	9	14.87	1.26
5	10	22.14	27.84	6	13.58	31.45	7	15.72	0.41
6	10	35.07	14.91	2	7.87	37.16	4	14.67	0.46
7	10	39.53	10.45	5	20.03	25.00	4	16.13	0
8	10	36.24	13.74	7	25.59	19.44	4	14.94	1.19
9	10	29.44	20.54	10	29.44	15.59	5	14.99	1.14
10	10	14.35	35.63	10	14.35	30.68	10	14.35	1.78

Table 8. Comparison of epochs, ACT, WT in different methods.

Random epochs

In this method, each client *i* chooses a random number between 1 and 10 as its epoch(E_i) value. The calculation of *ACT*, *WT* for client 1 and client 2 is shown in Equations (19)-(22). Following the same procedure, *ACT* and *WT* for the remaining clients are calculated, and all values are tabulated in Table 8.

$$ACT_1 = E_1 * \sigma_1 + \beta_1 + \delta_1 = 9 * 4.95 + 0.36 + 0.12 = 45.03 \tag{19}$$

$$ACT_2 = E_2 * \sigma_2 + \beta_2 + \delta_2 = 7 * 2.52 + 0.33 + 0.24 = 18.21 \tag{20}$$

$$WT_1 = ACT_{max} - ACT_1 = 45.03 - 45.03 = 0 \tag{21}$$

$$WT_2 = ACT_{max} - ACT_2 = 45.03 - 18.21 = 26.82 \tag{22}$$

FedEff (optimal epochs)

In FedEff, the approximate number of epochs(E) which is considered for calculating ERT is derived from E_b and EDF as shown in Equation (23). The ERT calculation is shown in Equation (24).

$$E = \tau * E_b = 0.5 * 10 = 5 \tag{23}$$

$$ERT = \lceil (E * \sigma_a + \beta_a + \delta_a) \rceil = \lceil (5 * 2.69 + 0.57 + 0.28) \rceil = 15 \tag{24}$$

The calculation of the available computation time (CT_1, CT_2), the optimal number of epochs (E_1, E_2) for client 1 and client 2 is shown in Equations (25)-(28). The same procedure is followed for other clients and the optimal number of epochs (E_i) for all clients is specified in Table 8.

$$CT_1 = ERT - (\beta_1 + \delta_1) = 15 - (0.36 + 0.12) = 14.52 \tag{25}$$

$$E_1 = \lfloor (CT_1/\sigma_1) \rfloor = \lfloor (14.52/4.95) \rfloor = 3 \tag{26}$$

$$CT_2 = ERT - (\beta_2 + \delta_2) = 15 - (0.33 + 0.24) = 14.43 \tag{27}$$

$$E_2 = \lfloor (CT_2/\sigma_2) \rfloor = \lfloor (14.43/2.52) \rfloor = 6 \tag{28}$$

The calculation of *ACT*, *WT* for client 1 and client 2 is shown in Equations (29)-(32). Following the same procedure, *ACT* and *WT* for the remaining clients are calculated, and all values are tabulated in Table 8.

$$ACT_1 = E_1 * \sigma_1 + \beta_1 + \delta_1 = 3 * 4.95 + 0.36 + 0.12 = 15.33 \tag{29}$$

$$ACT_2 = E_2 * \sigma_2 + \beta_2 + \delta_2 = 6 * 2.52 + 0.33 + 0.24 = 15.69 \tag{30}$$

$$WT_1 = ACT_{max} - ACT_1 = 16.13 - 15.33 = 0.8 \tag{31}$$

$$WT_2 = ACT_{max} - ACT_2 = 16.13 - 15.69 = 0.44 \tag{32}$$

Conclusion and future work

This research work addressed the challenge of improving efficiency in heterogeneous federated learning by introducing a novel optimal epoch selection mechanism. In our proposed algorithm (FedEff), the server estimates the optimal time needed for a training round using the average computation and communication times of the clients. This Estimated Round Time (ERT) is used by the server to find the optimal count of local epochs for each client. Experimental evaluations conducted in simulated heterogeneous environments demonstrate that the proposed approach substantially reduces client waiting times and overall training duration within our simulation settings. The results of a comparative analysis indicate that our proposed method provides greater efficiency than FedAvg and random epoch selection strategies, highlighting its effectiveness in improving performance

in heterogeneous federated settings. These findings highlight the importance of system-aware optimal epoch allocation in mitigating the effects of heterogeneity in federated learning systems. This work assumes static client heterogeneity and availability, while real-world federated systems often operate under dynamic conditions, with clients experiencing fluctuating computing and communication capacities, as well as intermittent participation or dropouts.

An important extension of this work is to adapt the proposed algorithm to dynamic settings, leveraging reinforcement learning for adaptive scheduling, predictive modeling to estimate client availability, and online optimization to adjust local training strategies in real time. In future work, we intend to investigate resource-aware aggregation mechanisms and client selection policies that effectively mitigate the impact of stragglers while preserving fairness, with the goal of further enhancing the robustness and scalability of federated learning systems. These directions would increase the practicality of the proposed method, making it more suitable for deployment in heterogeneous and unpredictable real-world settings.

Data availibility

The datasets used in this work – MNIST (Modified National Institute of Standards and Technology) and CIFAR-10 (Canadian Institute For Advanced Research) are publicly available through the TensorFlow/Keras datasets library. https://www.tensorflow.org/datasets/catalog/mnist, https://www.tensorflow.org/datasets/catalog/cifar10

Received: 12 August 2025; Accepted: 30 September 2025
Published online: 06 November 2025

References

1. Yang, Q., Liu, Y., Chen, T. & Tong, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol. (TIST)* **10**(2), 1–19 (2019).
2. Li, T., Sahu, A. K., Talwalkar, A. & Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **37**(3), 50–60 (2020).
3. Kairouz, P. et al. Advances and open problems in federated learning. *Found. Trends Mach. Learn.* **14**(1–2), 1–210 (2021).
4. Wang, J., Charles, Z., Xu, Z., Joshi, G., McMahan, H.B., Al-Shedivat, M.,Andrew, G., Avestimehr, S., Daly, K., Data, D. et al. A field guide to federated optimization, arXiv:2107.06917, (2021).
5. McMahan, B., Moore, E., Ramage, D., Hampson, S. & Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. *Artif. Intell. Stat.*, PMLR, 1273–1282, (2017).
6. Li, T. et al. Federated optimization in heterogeneous networks. *Proc. Mach. Learn. Syst.* **2**, 429–450 (2020).
7. Park, J., Yoon, D., Yeo, S. & Oh, S. AMBLE: Adjusting mini-batch and local epoch for federated learning with heterogeneous devices. *J. Parall. Distrib. Comput.* **170**, 13–23 (2022).
8. Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S. & Suresh, A. T. SCAFFOLD: Stochastic controlled averaging for federated learning. In: Proceedings of the 37th International Conference on Machine Learning, PMLR, 5132–5143, (2020).
9. Wang, J., Liu, Q., Liang, H., Joshi, G. & Vincent, P. H. A novel framework for the analysis and design of heterogeneous federated learning. *IEEE Trans. Signal Process.* **69**, 5234–5249 (2021).
10. Johnson, R., Zhang, T., Burges, C. J., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q. Accelerating stochastic gradient descent using predictive variance reduction. Adv. Neural Inform. Process. Syst. 315–323., (2013).
11. Shalev-Shwartz, S. & Zhang, Tong. Stochastic dual coordinate ascent methods for regularized loss. *J. Mach. Learn. Res.* **14**, 567–599 (2013).
12. J. Reddi, S., Konecny, J., Richtarik, P., Poczos, B. & Smola, A. Aide: Fast and communication efficient distributed optimization, 1–23, (2016).
13. Shamir, O., Srebro, N., Zhang, T., Xing, E. P. & Jebara, T. Communication-efficient distributed optimization using an approximate newton-type method. In: Proceedings of the 31st International Conference on Machine Learning, PMLR, 1000–1008, (2014)
14. Shi, Y., Zhang, Y., Zhang, P., Xiao, Y. & Niu, L. Federated learning with l1 regularization. *Pattern Recognit. Lett.* **172**, 15–21 (2023).
15. Mendieta, M., Yang, T., Wang, P., Lee, M., Ding, Z. & Chen, C. Local learning matters: Rethinking data heterogeneity in federated learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 8397–8406, (2022).
16. Wang, J., Tantia, V., Ballas, N. & Rabbat, M. Slowmo: Improving communication-efficient distributed sgd with slow momentum, arXiv:1910.00643, (2019).
17. Hsu, T. H., Qi, H. & Brown, M. Measuring the effects of non-identical data distribution for federated visual classification, CoRR abs/1909.06335, (2019).
18. Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konecný, J., Kumar, S. & McMahan, H. B. Adaptive federated optimization, CoRR abs/2003.00295, (2020).
19. Wang, J., Xu, Z., Garrett, Z., Charles, Z., Liu, L. & Joshi, G. Local adaptivity in federated learning: Convergence and consistency, CoRR abs/2106.02305, (2021).
20. Yu, H., Jin, R. & Yang, S. On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization. In: Proceedings of the 36th International Conference on Machine Learning, PMLR, 7184–7193, (2019).
21. Yuan, H. & Ma, T. Federated accelerated stochastic gradient descent. *Adv. Neural Inform. Process. Syst.* **33**, 5332–5344 (2020).
22. Gao, L., Fu, H., Li, L., Chen, Y., Xu, M. & Xu, C. -Z. Feddc: Federated learning with non-IID data via local drift decoupling and correction, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10112–10121, 2022.
23. Mills, J., Hu, J. & Min, G. Client-side optimization strategies for communication-efficient federated learning. *IEEE Commun. Mag.* **60**(7), 60–66 (2022).
24. Yang, C., Wang, Q., Xu, M., Chen, Z., Bian, K., Liu, Y. & Liu, X. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data, arXiv:2006.06983, (2021).
25. Abdelmoniem, A. M., Ho, C. Y., Papageorgiou, P. & Canini, M. A comprehensive empirical study of heterogeneity in federated learning. *IEEE Internet Things J* **10**(16), 14071–14083 (2023).
26. Nishio, T. & Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge, (2019).
27. Zhang, J., Cheng, X., Wang, C., Wang, Y., Shi, Z., Jin, J., Song, A., Zhao, W., Wen, L. & Zhang, T. Fedada: Fast-convergent adaptive federated learning in heterogeneous mobile edge computing environment, World Wide Web, 1971–1998, (2022).
28. Zeng, M., Wang, X., Pan, W. & Zhou, P. Heterogeneous training intensity for federated learning: A deep reinforcement learning approach. *IEEE Trans. Netw. Sci. Eng.* **10**(2), 990–1002 (2023).
29. Xia, F. et.al. ChronusFed: Reinforcement-Based Adaptive Partial Training for Heterogeneous Federated Learning. In: Proceedings of the 53rd International Conference on Parallel Processing, 464–473, (2024).
30. Wang, Y., Wang, S., Lu, S. & Chen, J. FADAS: Towards Federated Adaptive Asynchronous Optimization, arXiv:2407.18365, (2024).

31. Peng, Z. et al. VFChain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Trans. Netw. Sci. Eng.* **9**(1), 173–186 (2021).
32. Yang, C., Zhu, M., Liu, Y. & Yuan, Y. FedPD: Federated Open Set Recognition with Parameter Disentanglement. In: IEEE/CVF International Conference on Computer Vision (ICCV), 4859-4868, (2023).
33. Chen, Z., Yang, C., Zhu, M., Peng, Z. & Yuan, Y. Personalized retrogress-resilient federated learning toward imbalanced medical data. *IEEE Trans. Med. Imaging* **41**(12), 3663–3674 (2022).
34. Zhu, M., Liao, J., Liu, J. & Yuan, Y. Fedoss: Federated open set recognition via inter-client discrepancy and collaboration. *IEEE Trans. Med. Imaging* **43**(1), 190–202 (2023).
35. Yang, Q. et al. Relation-guided versatile regularization for federated semi-supervised learning. *Int. J. Comput. Vis* **133**, 3312–3326 (2025).
36. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998).
37. Krizhevsky, A. Learning multiple layers of features from tiny images, University of Toronto, (2012).

Author contributions
K. Narmadha was responsible for the conceptualization, methodology, and writing of the work, while P. Varalakshmi contributed through supervision and review.

Funding
This work did not receive any specific grant from any funding agency.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to K.N.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.