A Joint Time and Energy-Efficient Federated Learning-based Computation Offloading Method for Mobile Edge Computing

Anwesha Mukherjee[1,2], Rajkumar Buyya[1]
[1] *Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Australia.*
[2] *Department of Computer Science, Mahishadal Raj College, Mahishadal, West Bengal, 721628, India.*
Email: anweshamukherjee2011@gmail.com
Email: rbuyya@unimelb.edu.au

Abstract

Computation offloading at lower time and lower energy consumption is crucial for resource-constrained mobile devices. This paper proposes an offloading decision-making model using federated learning. Based on the device configuration, task type, and input, the proposed decision-making model predicts whether the task is computationally intensive or not. If the predicted result is *computationally intensive*, then based on the network parameters the proposed decision-making model predicts *whether to offload or locally execute* the task. The experimental results show that the proposed method achieves above 90% prediction accuracy in offloading decision-making, and reduces the response time and energy consumption of the user device by ∼11-31%. A secure partial computation offloading method for federated learning is also proposed to deal with the Straggler effect of federated learning. The results present that the proposed partial computation offloading method for federated learning has achieved a prediction accuracy of above 98% for the global model.

Keywords: Federated learning, Offloading, Training Time, Encryption, Energy consumption.

1. Introduction

The energy and latency-aware computation offloading has gained significant research interest in the last few years with the rapid growth in wireless technology and the growing popularity of the Internet of Things (IoT). The IoT devices and the mobile devices, including laptops, tablets, and smartphones, have limited battery life and computational resources. In such a case, execution of computationally intensive tasks may not be feasible inside these mobile devices. However, if the device can execute the tasks, then it may also take more time than offloading the task to the edge server or cloud. On the other hand, offloading all the tasks may not be fruitful from the perspective of latency and energy consumption. For the tasks with less computations, offloading can consume more time than local execution. Hence, the decision of *offload or local execution* is important. To address this issue, this paper proposes an offloading decision-making model based on federated learning.

Nowadays, mobile devices have various recommendation applications, where model training is important. Many applications use reinforcement learning, machine learning, or deep learning algorithms to train the model. Further, local model training may not be always possible for resource-constrained mobile devices. However, the user's data is confidential, and the user may not like to share the data with the server. Also, the large amount of data transmission to the server for analysis creates a huge overhead on the server as well as causes high network traffic. Federated learning (FL) [1, 2, 3] is an emerging technology where the devices locally train models using their local datasets, and exchange model updates with the server. Finally, a global model is developed, and each device that serves as a client has a personalized model and the global model. In centralized federated learning (CFL), the devices working as clients train local models with their own datasets and exchange model updates with the server to build a global model [1]. In decentralized federated learning (DFL), the devices form a collaborative network among themselves to exchange model updates for developing a generalized model [1].

The use of FL protects data privacy as no data sharing takes place, and through collaborative learning, a global model with high prediction accuracy can be obtained. However, the mobile devices may not be always able to locally train a deep learning model using a huge number of data samples. Straggler effect is a major issue of FL, where the devices slow down while executing large-scale data analysis [4]. In such a case, conventional CFL and DFL models may not be feasible. To overcome the problem of resource limitation without compromising with data privacy, we propose a federated offloading approach that performs partial computation offloading along with data encryption.

1.1. Motivation and Contributions

A mobile device has a computational task to execute, and a decision-making model is required to decide *whether to offload* *or locally execute the task*, at minimal time and energy consumption. However, the mobile device may not have sufficient data samples for training to build an accurate decision-making model. The first motivation of this work is to develop an accurate model to decide *whether to offload or locally execute a task.*

The second motivation of this work is to mitigate the issue of the Straggler effect, and propose a secure partial computation offloading framework for FL by overcoming the resource limitation issue of participating mobile devices.

To address these issues, our paper makes the following key contributions:

- To address the first motivation, a CFL method is proposed to build a model for deciding *whether to offload or locally execute a task.* The proposed model is named as *FL-based Offloading Decision-Making Model* (*FLDec*).
    - At first, *FLDec* is used to decide whether a task is computationally intensive or not, based on the device configuration, requested computation, and user input. Here, as the underlying data analysis model, Multi-layer Perceptron (MLP) is used, which is suitable for nonlinear function classification tasks.
    - If the prediction result is *computationally intensive*, then the *FLDec* is used to decide whether to offload the task or locally execute it, based on the network parameters such as uplink and downlink traffic, network throughput, etc. Here, Long Short-Term Memory (LSTM) network is used as the underlying data analysis model, which is suitable for sequential data analysis. As the network parameters like throughput, uplink traffic, and downlink traffic are dynamic and change over time, we use LSTM in this case to capture the sequential pattern.
- To address the second motivation, a secure partial computation offloading method for FL is proposed and implemented, which is referred to as *Federated Offloading* (*FedOff*).
    - The user devices have large data samples, which are split into two parts. One part is used as the local dataset to train the local model, and the other part is offloaded to the edge server after encryption. Here, symmetric key cryptography is used for data encryption during transmission to the edge server. The keys of individual devices are different, and for secure key exchange with the server, public key cryptography is used.
    - The devices train their local models and exchange model updates with the edge server. The edge server, after receiving data from the devices, trains a model with the received dataset after decryption, and stores the model update. After receiving model updates from the devices, i.e., clients, the server performs aggregation considering the received updates as well as the stored update. The updated global model is then shared with the devices.

The rest of the paper is organized as follows. Section 2 presents the existing works on offloading and FL. Section 3 demonstrates the proposed offloading method. Section 4 evaluates the performance of the proposed approach. Finally, Section 5 concludes the paper with directions for future work.

2. Related Work

The existing research works on computation offloading, FL, and federated offloading are briefly discussed in this section. A summary of the uniqueness of our approach compared to the existing schemes is presented in Table 1.

Computation offloading is a significant research area of mobile cloud computing (MCC). Service delay minimization [19] is one of the primary issues for mobile devices. The computation offloading from a mobile device to the remote cloud suffered from connectivity interruption, high latency, etc. [20, 21]. The use of edge computing (EC) overcomes this problem by bringing resources to the network edge. The IoT/mobile devices are connected with the edge server, and the edge server is connected with the cloud [22]. The mobile devices offload the resource-intensive computations to the edge server. The edge server executes the computation and sends the result to the mobile device or forwards the request to the cloud server. The cloud server, after executing the computation, sends the result. The computation offloading in mobile edge computing (MEC) was discussed in [23]. The computation offloading in EC networks was illustrated in [24] along with a discussion on optimization methods. Task offloading in the MEC environment, along with future research directions, was illustrated in [25]. A joint task offloading approach with resource allocation was proposed for the MEC environment in [5]. In [20], a user mobility-based computation offloading method was proposed using game theory. In [6], a non-cooperative game-based optimal data offloading strategy was proposed. However, none of these methods focused on building an accurate decision-making model for offloading.

In [26], the use of machine learning (ML) in computation offloading was reviewed. The reinforcement learning (RL) has also gained popularity in computation offloading. In [27], the use of RL in computation offloading was discussed in detail. The use of deep reinforcement learning (DRL) for task offloading was discussed in [7]. A DRL-based method for action space recursive decomposition was proposed in [8]. In [9], a dynamic switching algorithm was proposed for intelligent task offloading in an edge-cloud environment. In [28], task offloading for serverless computing in an edge-cloud environment was proposed. In [10], the authors focused on the issue of offloading serverless functions in an edge-cloud environment. Though they considered response time, the energy consumption was not measured in [10]. As we observe, various offloading strategies have been proposed. However, none of these approaches considered the use of FL to build personalized as well as global models for accurate offloading decision-making.

Table 1: Comparison of proposed work with existing offloading methods

Work	Offloading decision-making using FL	Partial computation offloading for FL	Task details are provided	Accuracy/ MSE/ Loss in prediction measured	Time/ Delay is measured	Energy consumption is measured
Jiang et al. [5]	✗	✗	✓	✗	✓	✓
Apostolopoulos et al. [6]	✗	✗	✗	✗	✓	✓
Tang et al. [7]	✗	✗	✗	✗	✓	✗
Ho et al. [8]	✗	✗	✓	✗	✓	✓
Wang et al. [9]	✗	✗	✗	✗	✓	✓
Russo et al. [10]	✗	✗	✓	✗	✓	✗
Liu et al. [11]	✓	✗	✗	✓	✓	✗
Zhang et al. [12]	✓	✗	✗	✗	✓	✓
Han et al. [13]	✓	✗	✗	✓	✗	✗
Tang et al. [14]	✓	✗	✓	✗	✓	✓
Peng et al. [15]	✓	✗	✓	✓	✓	✓
Ji et al. [16]	✗	✓	✓	✓	✓	✗
Wu et al. [17]	✗	✓	✓	✓	✓	✗
Liaq et al. [18]	✗	✓	✓	✓	✓	✓ (power was measured)
Proposed work	✓	✓	✓	✓	✓	✓

FL with edge computing has gained popularity in recent years [29, 30]. In [31], FL was used for task offloading in an EC-supported vehicular network. In [32], a multi-agent-based collaborative strategy was proposed for vehicular task offloading using DRL-based FL. In [33], a hierarchical computation offloading method was proposed using federated reinforcement learning in the domain of Internet of Medical Things (IoMT). In [34], an edge-based FL was proposed where the edge servers train their local models and exchange updates with the cloud to build the global model. However, in MEC-based FL, the mobile devices can also train their local models with their own datasets and exchange model updates with the edge server that is connected to the cloud. The edge server performs aggregation in that case. The edge servers can also exchange their model updates with the cloud for developing a global model, and in that case cloud server acts as the aggregator. In [11], FL with DRL was used for computation offloading. Bidirectional LSTM (Bi-LSTM)-based FL was proposed for offloading decision-making and resource allocation in [12]. In [13], FL was used for computation offloading in edge-IoT systems. In [14], FL-based task offloading was proposed based on multi-feature matching. In [15], a security-aware computation offloading method was proposed based on federated reinforcement learning. They used differential privacy to achieve data security and DRL for selecting offloading decisions [15]. To mitigate the straggler effect of FL, an offloading method was proposed in [16]. To deal with the straggler effect and accelerate the local training in resource-constrained devices, offloading in FL was discussed in [17]. In [18], the unmanned aerial vehicle (UAV) was used with FL and MEC to deal with the straggler effect. The computations from the resource-limited devices were transferred to the MEC server in [18]. In the existing FL-based approaches, either offloading decision-making using FL or partial offloading for FL was explored. However, both issues, along with data security, were not simultaneously addressed in the existing works.

2.1. *Limitations of existing approaches and comparison with proposed work*

In the existing FL-based offloading approaches, either the offloading decision-making using FL or partial computation offloading in FL, was explored. In our work, we propose a computation offloading decision-making framework using FL, as well as a partial computation offloading method for FL. The comparison of the proposed and existing offloading approaches is presented in Table 1.

In [5], task offloading in MEC environment was discussed. In [6], data offloading in multi-access edge computing was discussed. However, the accuracy of the offloading decision was not measured in [5, 6]. Further, the offloading in FL was not explored in [5, 6]. In [7], DRL was used for task offloading in MEC. However, the accuracy in decision-making was not measured. Further, energy consumption, which is crucial in offloading, was not considered in [7]. In [8], a cooperative offloading in a multi-access edge computing environment was discussed, based on DRL. However, the prediction accuracy in offloading decision-making was not measured. In [9], an intelligent offloading method was proposed, however, the accuracy of offloading decision-making was not measured. In [10], a quality of service-aware offloading strategy was proposed for an edge-cloud environment, with a focus on serverless functions. The authors did not measure the energy consumption. The accuracy of decision-making was also not measured in [10]. In [11], FL and DRL were used for computation offloading. However, the energy consumption, which is vital in offloading, was not considered in [11]. In [12], computation offloading using FL based on Bi-LSTM was discussed, however, the prediction accuracy in offloading decision-making was not measured. In [13], computation offloading using FL was discussed, however, the accuracy in offloading decision-making was not measured. In [14], FL-based computation offloading in an edge-cloud environment was discussed. However, energy consumption was not considered in [13]. In [15], offloading decision was selected using FL with DRL. However, the authors did not consider the partial computation offloading aspect for FL to address the Straggler effect. In [16], computation offloading for FL was discussed. In [17], adaptive offloading in FL was discussed. However, the energy consumption was not considered in [16, 17]. Unlike the existing works on offloading decision-making [5, 7, 8, 12, 14], we have not only measured the time and energy consumption for the proposed methods but also the accuracy and model loss for the proposed approaches.

In [11] and [13], the authors measured the model loss, but the energy consumption was not measured. In [18], computation offloading in FL was proposed using UAV and EC. Though the authors measured system delay and processing power of the IoT devices, they did not focus on secure data offloading. Compared to existing computation offloading methods [16, 17, 18], the uniqueness of the proposed work is the use of cryptography for secure data transmission to the server. As we observe from Table 1, the proposed work is unique compared to the state-of-the-art.

3. Proposed Offloading Methods

The proposed work is divided into two approaches: (i) Approach 1: An FL-based model is proposed to decide whether to offload a task or not. (ii) Approach 2: A secure partial computation offloading method is proposed for FL to deal with the straggler effect of resource-constrained mobile devices.

3.1. Proposed Offloading Decision-making Method

We propose a decision-making method regarding whether to offload a task or not, based on its computational intensiveness, network parameters, etc. The mathematical notations used in our approach FLDec are summarized in Table 2.

3.1.1. Problem statement

A device has a computational task C that is to be executed locally or remotely.

- *Objective 1:* Decide whether C is computationally intensive or not.
- *Objective 2:* If it is computationally intensive, then decide whether to offload C or not.

Table 2: Mathematical notations used in FLDec

Notation	Definition
$\mathcal{K}$	Set of connected devices
$\mathcal{D}_k$	Local dataset of a device k
B	Batch size into which $\mathcal{D}_k$ is split
$\mathcal{E}$	Number of epochs
$\mathcal{R}$	Number of rounds
$\mathcal{M}_f$	Final global model
$\mathcal{M}_k$	Local model of device k
α_r	Fraction of devices participating in round r
N_k	Maximum number of devices participating in FL
$\mathcal{M}_{k_{exh}}$	Local model of a device k to predict whether a task is computationally intensive or not
$\mathcal{M}_{k_{off}}$	Local model of a device k to predict whether to offload or not

Algorithm 1: FL-based model for decision-making

```
Input: $\mathcal{D}_k, \mathcal{K}, \mathcal{R}, B, \mathcal{E}, N_k$
Output: Updated Global model ($\mathcal{M}_f$), Local model ($\mathcal{M}_k$)
Decision-making Model ($\mathcal{D}_k, \mathcal{K}, \mathcal{R}$):
    $j \leftarrow 0$
    while ($j < N_k$) do
        $listen()$
        $establishConnection()$
        $j = j + 1$
    end
    $\mathcal{K} \leftarrow ConnectedDevices()$
    start Client-side process() and Server-side process()
Client-side process():
    $\mathcal{D}_k \leftarrow preprocess(\mathcal{D}_k)$
    $Accuracy \leftarrow 0$
    while ($connectedtoServer() == TRUE$) do
        $\mathcal{M}_{up} \leftarrow get(\mathcal{M})$
        $\mathcal{D}_{k_{train}}, \mathcal{D}_{k_{test}} \leftarrow split(\mathcal{D}_k)$
        $\mathcal{M}_{up}.fit(\mathcal{D}_{k_{train}}, \mathcal{E}, B)$
        $Accuracy_{up} \leftarrow accuracy\_score(\mathcal{M}_{up}, \mathcal{D}_{k_{test}})$
        if ($Accuracy < Accuracy_{up}$) then
            $\mathcal{M}_k \leftarrow \mathcal{M}_{up}$
        end
        $send(\mathcal{M}_k)$
    end
    $save(\mathcal{M}_k)$
Server-side process():
    $\mathcal{M}_{in} \leftarrow initmodel()$
    $\mathcal{M} \leftarrow \mathcal{M}_{in}$
    for $r = 1$ to $\mathcal{R}$ do
        initialize $Model_{all} = []$
        $\mathcal{K}_r \leftarrow Subset(\mathcal{K}, max(\alpha_r * |\mathcal{K}|, 1), \text{"random"})$
        for $k = 1$ to $|\mathcal{K}_r|$ do
            $receive(\mathcal{M}_k)$
            $Model_{all}.append(\mathcal{M}_k)$
        end
        $w \leftarrow \sum_{k=1}^{|\mathcal{K}_r|} getweights(\mathcal{M}_k)/|\mathcal{K}_r|$
        $\mathcal{M}.setweights(w)$
        $send(\mathcal{M})$
    end
    $\mathcal{M}_f \leftarrow \mathcal{M}$
    $save(\mathcal{M}_f)$
    $releaseConnectedDevices()$
```

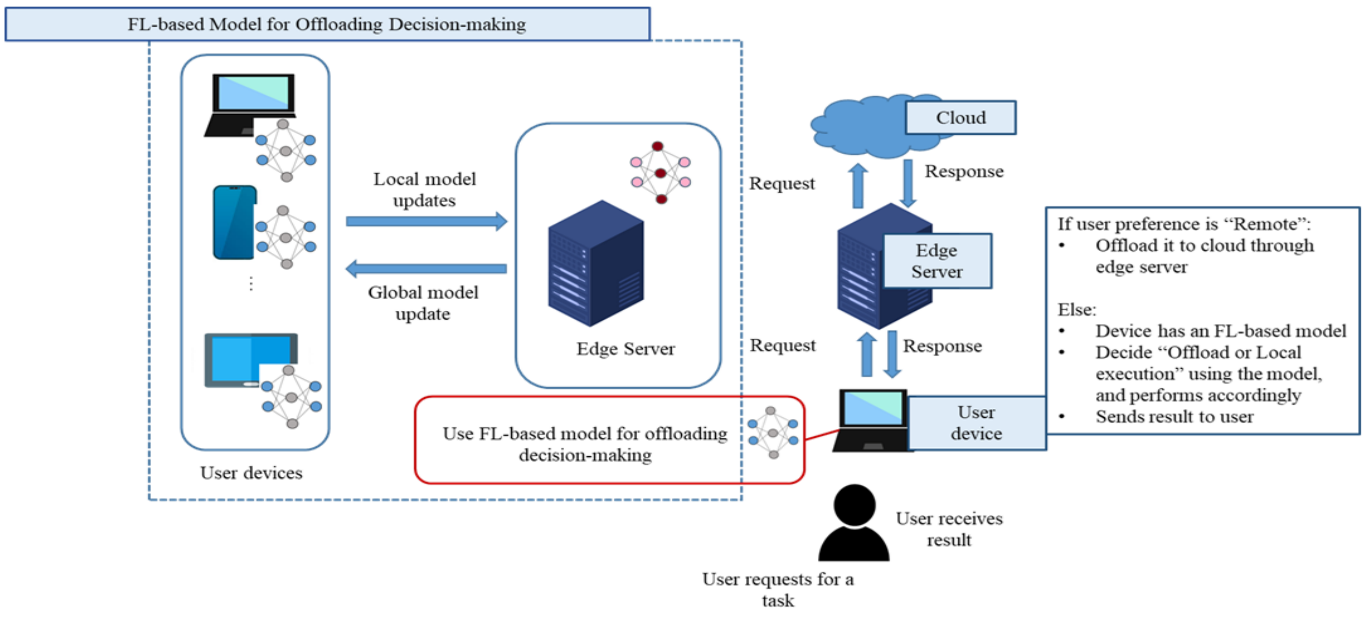

FL-based Model for Offloading Decision-making
Local model updates
Global model update
Edge Server
User devices
Use FL-based model for offloading decision-making
Cloud
Request
Response
Edge Server
Request
Response
User device
If user preference is "Remote":
• Offload it to cloud through edge server
Else:
• Device has an FL-based model
• Decide "Offload or Local execution" using the model, and performs accordingly
• Sends result to user
User receives result
User requests for a task

Figure 1: System model of the proposed FL-based offloading decision-making process

3.1.2. Proposed framework

For offloading decision-making, we use the FL-based model. The steps of developing a decision-making model using FL is presented in Algorithm 1. Here, the clients are the participating user devices, and the server is the edge server. At first, the connection between the server and the clients is established. Then, the clients receive an initial model from the server, and train their local models using their local datasets. The server selects a subset of clients to participate in the FL process. Each participating client splits its dataset (D_k) into train and test datasets to train the local model. In our experiment, we have split the dataset into a train dataset containing 80% data samples, and a test dataset containing the remaining 20% data samples. Then, based on the batch size (B) the train dataset is split into a number of batches, and the training takes place depending on the number of epochs ($\mathcal{E}$). After developing the local model at the client-side, each participating client sends its local model update ($\mathcal{M}_k$) to the server. After receiving local model updates from all participating clients, the server builds the global model ($\mathcal{M}$) using aggregation. Here, we have used federated averaging for aggregation. After all rounds ($\mathcal{R}$), the final global model ($\mathcal{M}_f$) is obtained. The proposed offloading method is presented in Algorithm 2, and the proposed system model is pictorially depicted in Fig. 1. In the proposed offloading method, at first, it is checked that whether the task is computationally intensive or not. To decide *whether a task is computationally intensive or not*, MLP is used as the underlying data analysis model. MLP contains an input layer, hidden layers, and an output layer. MLP is suitable for modeling complex non-linear relationships in data. Based on the *requested computational task*, *respective input*, and *configuration of the device (RAM size, processing speed)*, it is predicted whether the task would be computationally intensive or not. For a computational task, along with the device configuration, input also plays an important role. For ex-

Algorithm 2: Proposed offloading algorithm

```
Input: Computational Task ($C$), Model to check
       computational intensiveness ($\mathcal{M}_{k_{exh}}$), Model to check
       offload or local execution ($\mathcal{M}_{k_{off}}$)
Output: Result ($R$)

Offloading method ($C$):
    $category(C) \leftarrow \mathcal{M}_{k_{exh}}.predict(inputfeatures_{exh})$
    if $category(C)$ is $NotIntensive$ then
        if ($userPreference ==$ "$RemoteAccess$") then
            $R \leftarrow offloadtoCloudthroughEdgeServer(C)$
        else
            $R \leftarrow executeLocally(C)$
        end
    else
        $decision \leftarrow \mathcal{M}_{k_{off}}.predict(inputfeatures_{off})$
        if ($decision ==$ "$Offload$") then
            $R \leftarrow offloadtoEdgeServer(C)$
        else
            $R \leftarrow executeLocally(C)$
        end
    end
```

ample, the multiplication of two 3x3 matrices takes much less time to compute than the multiplication of two 300x300 matrices. Thus, based on the device configuration, input as well as the task type (matrix operation, sorting, searching, etc.), it is decided whether it can be computationally intensive or not. If the prediction is "Not Intensive", then, based on the user preference of local or remote access, the task is executed either locally or offloaded to the cloud. Otherwise, if the prediction is "computationally intensive", then it is decided whether to offload it or not. To decide *whether to offload or locally execute* the task, we use LSTM as the underlying model. LSTM is one type of recurrent neural network (RNN). The difference between LSTM and conventional RNN is that LSTM maintains a memory cell to hold information for an extended period. LSTM is able to capture long-term dependencies, hence, suitable for sequence prediction tasks. Based on the network parameters such as *network throughput*, *latency*, *uplink* and *downlink traffic*, etc., it is decided whether to offload the task or not, using LSTM-based FL. Based on the decision, the task C is either offloaded or locally executed. If the decision is *offload*, C is offloaded to the edge server. If the edge server is unable to offload C, then it forwards the request to the cloud. The cloud server executes the task and sends the result to the device.

3.1.3. Computational complexity

The time complexity of the FL process depends on the time complexity of model initialization, local model training, exchange of model updates, and aggregation. The time complexity of model initialization is $O(1)$. The computational complexity of model initialization is given as $O(w_{\mathcal{M}_{in}})$, where $w_{\mathcal{M}_{in}}$ denotes the weights of the initial model. The time complexity of local model training is given as $O(\mathcal{R} \cdot \mathcal{E} \cdot (\mathcal{D}_k/B) \cdot w_k)$, where w_k denotes model weights for client k. The computational complexity of local model training is given as $O(\mathcal{R} \cdot \mathcal{E} \cdot (\mathcal{D}_k/B) \cdot w_k \cdot |\mathcal{K}|)$. The time complexity for model aggregation is given as $O(\mathcal{R} \cdot |\mathcal{K}| \cdot w_k)$. The computational complexity for model aggregation is also given as $O(\mathcal{R} \cdot |\mathcal{K}| \cdot w_k)$. The time complexity and computational complexity for model updates exchange both are given as $O(w_{\mathcal{M}} + |\mathcal{K}| \cdot w_k)$, where w_k denotes the model weights of client k and $w_{\mathcal{M}}$ denotes model weights of the server.

3.2. Proposed Federated Offloading Method

In this section, we propose a partial computation offloading method for FL, which is referred to as federated offloading or FedOff. In the proposed method, the user devices are the clients and the server is the edge server. The mathematical notations used in FedOff are summarized in Table 3.

3.2.1. Problem statement

Let the model of an application A needs to be developed using FL, each device has a respective local dataset $\mathcal{D}_{k_A}$, and the number of rounds is $\mathcal{R}_A$. In FL, each of the participating devices needs to locally train the model with its local dataset. However, model training with a large number of data samples may not be feasible for devices with resource limitations. In that case, the devices put a portion of their datasets to the server. The server trains the model with the offloaded datasets and stores the model. During aggregation, the server considers this trained model also along with the received model updates from the devices. However, how much portion of data will be offloaded that is significant. Further, data security during offloading to the server is another issue. The objective is to propose a secure federated offloading method with good prediction accuracy and minimal loss.

Table 3: Mathematical notations used in FedOff

Notation	Definition
$\mathcal{K}$	Set of connected devices
$\mathcal{D}_{k_A}$	Local dataset of a device k for application A
B_A	Batchsize into which $\mathcal{D}_{k_A}$ is split
$\mathcal{E}$	Number of epochs
$\mathcal{R}_A$	Number of rounds in federated offloading
$\mathcal{M}_{f_A}$	Final global model for application A
$\mathcal{M}_{k_A}$	Local model of device k for application A
α_r	Fraction of devices participating in round r
N_k	Maximum number of devices for participating in FL
T_{total}	Total time consumption in FedOff
T_{train}	Total time consumption for training in FedOff
T_{init}	Time consumption for model initialization
T_{tr}	Time consumption for data transmission from clients to server
T_{loc}	Time consumption for local model training
T_{exm}	Time consumption for model updates exchange
T_{ser}	Time consumption for model training inside the server using received local datasets
T_{agg}	Time consumption for aggregation of model updates
T_{crypt}	Time consumption for data encryption and decryption

3.2.2. Proposed framework

The proposed federated offloading method is stated in Algorithm 3. In the proposed federated offloading method, each participating user device k splits its local dataset (D_{k_A}) into two parts: one part is processed locally ($D_{k_{loc_A}}$) and the other part is sent to the edge server ($D_{k_{ser_A}}$). Let the ratio of partitioning D_{k_A} is denoted as ρ_{k_A}, where $0 < \rho_{k_A} < 1$. Then, $D_{k_{loc_A}} = \rho_{k_A} \cdot D_{k_A}$ and $D_{k_{ser_A}} = (1-\rho_{k_A}) \cdot D_{k_A}$. The value of ρ_{k_A} depends on the ability of the device k to process the dataset without performance degradation. The ability of the device is considered as a function of the present load ($load_k$), memory capacity ($memory_k$), processing speed ($speed_k$), and available network bandwidth ($bandwidth_k$), given as:

$$Ability(k) = f(load_k, memory_k, speed_k, bandwidth_k) \qquad (1)$$

After splitting the dataset, the client sends the second part ($D_{k_{ser_A}}$) to the server. The server receives $D_{k_{ser_A}}$ from each participating device k, accumulates the collected data, and fits its model with the dataset. Each device k trains its model with the local dataset ($D_{k_{loc_A}}$) and sends the model update to the server. The server,

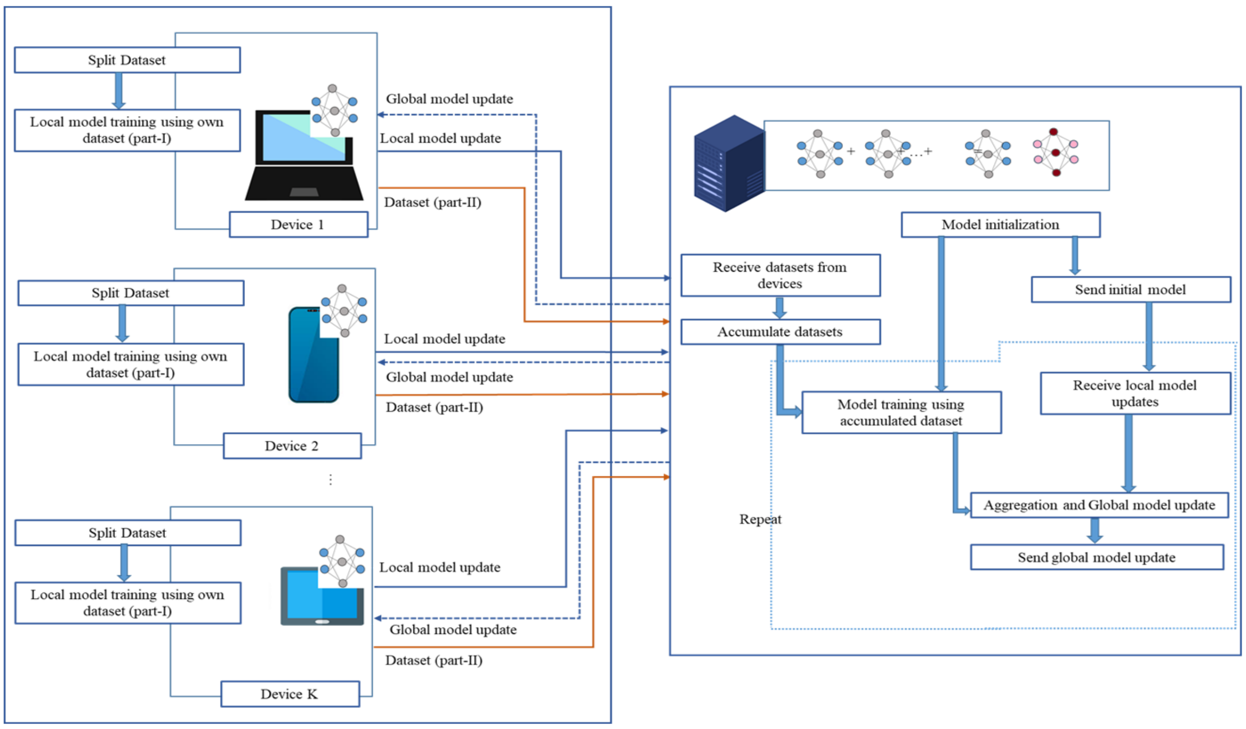


Figure 2: The proposed federated offloading method

after receiving model updates from the clients, performs aggregation along with the model update for the offloaded data. The process of model update is repeated for the number of rounds $\mathcal{R}_A$. The proposed federated offloading method is presented in Fig. 2. During data transmission from the clients to the server, encryption is used for protecting data privacy, and preventing unauthorized access to the data. Here, we have used Advanced Encryption Standard (AES)-128 for data encryption. AES [35, 36] is a well-known symmetric encryption algorithm used for data encryption. As symmetric cryptography is used, the same key is used for encryption and decryption. Hence, secure key sharing is vital. Here, for secure key sharing, a well-known public key cryptography approach Rivest, Shamir, and Adleman (RSA) algorithm [37] is used. Each client shares the encrypted key with the server, and the server decrypts it with the private key to obtain the original key to be used for data decryption.

3.2.3. *Computational complexity*

The time complexity of the proposed federated offloading method depends on the time complexity of model initialization, data transmission from clients to the server, local model training, exchange of model updates, model training at the server, and aggregation. The time complexity of model initialization is $O(1)$. The computational complexity of model initialization is given as $O(w_{\mathcal{M}_{in}})$, where $w_{\mathcal{M}_{in}}$ denotes the weights of the initial model. The time complexity and computational complexity of data transmission from clients to the server, both are given as $O(|\mathcal{K}| \cdot (\mathcal{D}_{k_{ser_A}}/S))$, where S denotes the data transmission speed. The time complexity of local model training is given as $O(\mathcal{R}_A \cdot \mathcal{E} \cdot (\mathcal{D}_{k_{loc_A}}/B_A) \cdot w_{k_A})$, where w_{k_A} denotes model weights for client k for application A. The computational complexity of local model training is given as $O(\mathcal{R}_A \cdot \mathcal{E} \cdot (\mathcal{D}_{k_{loc_A}}/B_A) \cdot w_{k_A} \cdot |\mathcal{K}|)$. The time complexity and computational complexity for model updates exchange both are given as $O(w_{\mathcal{M}_A} + |\mathcal{K}| \cdot w_{k_A})$, where w_{k_A} denotes the model weights of client k and $w_{\mathcal{M}_A}$ denotes model weights of the server for application A. The time complexity and computational complexity of model training at the server is given as $O(\mathcal{R}_A \cdot \mathcal{E} \cdot (\mathcal{D}_{loc_A}/B_A) \cdot w_{loc_u})$, where w_{loc_u} denotes the model weights of the server for application A while training the offloaded data. The time complexity and computational complexity for model aggregation both are given as $O(\mathcal{R}_A \cdot N_{model} \cdot w_i)$, where w_i denotes model weights of $\mathcal{M}_i$ and $\mathcal{M}_i \in Model_{all}$.

3.2.4. *Total time consumption*

The total time consumption of the proposed federated offloading method is determined as follows:

$$T_{foff} = T_{train} + T_{crypt} \tag{2}$$

where $T_{train} = T_{init} + T_{tr} + T_{loc} + T_{exm} + T_{ser} + T_{agg}$, where $T_{init} = O(1)$, $T_{tr} = O(|\mathcal{K}| \cdot (\mathcal{D}_{k_{ser_A}}/S))$, $T_{loc} = O(\mathcal{R}_A \cdot \mathcal{E} \cdot (\mathcal{D}_{k_{loc_A}}/B_A) \cdot w_{k_A})$, $T_{exm} = O(w_{\mathcal{M}_A} + |\mathcal{K}| \cdot w_{k_A})$, $T_{ser} = O(\mathcal{R}_A \cdot \mathcal{E} \cdot (\mathcal{D}_{loc_A}/B_A) \cdot w_{loc_u})$, and $T_{agg} = O(\mathcal{R}_A \cdot N_{model} \cdot w_i)$, where w_i denotes model weights of $\mathcal{M}_i$ and $\mathcal{M}_i \in Model_{all}$.

4. Performance Evaluation

This section discusses implementation of our proposed offloading decision-making method and federated offloading frame-

Algorithm 3: Federated offloading method

```
Input: $\mathcal{D}_{k_A}, \mathcal{K}, \mathcal{R}_A, B_A, \mathcal{E}, N_k$
Output: Updated Global model ($\mathcal{M}_{f_A}$), Local model ($\mathcal{M}_{k_A}$)
Update Model ($\mathcal{D}_{k_A}, \mathcal{K}, \mathcal{R}_A$):
    $j \leftarrow 0$
    while ($j < N_k$) do
        $listen()$
        $establishConnection()$
        $j = j + 1$
    end
    $\mathcal{K} \leftarrow ConnectedDevices()$
    start Client-side process() and Server-side process()
Client-side process():
    $\mathcal{D}_{k_A} \leftarrow preprocess(\mathcal{D}_{k_A})$
    $\mathcal{D}_{k_{loc_A}}, \mathcal{D}_{k_{ser_A}} \leftarrow split(\mathcal{D}_{k_A})$
    $Accuracy \leftarrow 0$
    $t \leftarrow 1$
    while ($connectedtoServer() == TRUE$) do
        if ($t == 1$) then
            $\mathcal{D}_{k_{ser_A}} \leftarrow encrypt(\mathcal{D}_{k_{ser_A}})$
            $sendtoServer(\mathcal{D}_{k_{ser_A}})$
            $t \leftarrow t + 1$
        end
        $\mathcal{M}_{up} \leftarrow get(\mathcal{M}_A)$
        $\mathcal{D}_{k_{train_A}}, \mathcal{D}_{k_{test_A}} \leftarrow split(\mathcal{D}_{k_{loc_A}})$
        $\mathcal{M}_{up}.fit(\mathcal{D}_{k_{train_A}}, \mathcal{E}, B_A)$
        $Accuracy_{up} \leftarrow accuracy\_score(\mathcal{M}_{up}, \mathcal{D}_{k_{test_A}})$
        if ($Accuracy < Accuracy_{up}$) then
            $\mathcal{M}_{k_A} \leftarrow \mathcal{M}_{up}$
        end
        $send(\mathcal{M}_{k_A})$
    end
    $save(\mathcal{M}_{k_A})$
Server-side process():
    $\mathcal{M}_{in} \leftarrow initmodel()$
    $\mathcal{M}_A \leftarrow \mathcal{M}_{in}$
    initialize $\mathcal{D}_{loc_A} = []$
    for $k = 1$ to $|\mathcal{K}|$ do
        $receive(\mathcal{D}_{k_{ser_A}})$
        $\mathcal{D}_{k_{ser_A}} \leftarrow decrypt(\mathcal{D}_{k_{ser_A}})$
        $\mathcal{D}_{loc_A}.append(\mathcal{D}_{k_{ser_A}})$
    end
    $\mathcal{D}_{l_{train_A}}, \mathcal{D}_{l_{test_A}} \leftarrow split(\mathcal{D}_{loc_A})$
    $Accuracy_l \leftarrow 0$
    for $r = 1$ to $\mathcal{R}_A$ do
        initialize $Model_{all} = []$
        $\mathcal{K}_r \leftarrow Subset(\mathcal{K}, max(\alpha_r * |\mathcal{K}|, 1), \text{“}random\text{”})$
        for $k = 1$ to $|\mathcal{K}_r|$ do
            $collect(\mathcal{M}_{k_A})$
            $Model_{all}.append(\mathcal{M}_{k_A})$
        end
        $\mathcal{M}_{loc_u} \leftarrow \mathcal{M}_A$
        $\mathcal{M}_{loc_u}.fit(\mathcal{D}_{l_{train_A}}, \mathcal{E}, B_A)$
        $Accuracy_{loc} \leftarrow accuracy\_score(\mathcal{M}_{loc_u}, \mathcal{D}_{l_{test_A}})$
        if ($Accuracy_l < Accuracy_{loc}$) then
            $\mathcal{M}_{loc} \leftarrow \mathcal{M}_{loc_u}$
        end
        $Model_{all}.append(\mathcal{M}_{loc})$
        $N_{model} \leftarrow length(Model_{all})$
        $w \leftarrow \frac{1}{N_{model}} \sum_{\mathcal{M}_i \in Model_{all}} getweights(\mathcal{M}_i)$
        $\mathcal{M}_A.setweights(w)$
        $send(\mathcal{M}_A)$
    end
    $\mathcal{M}_{f_A} \leftarrow \mathcal{M}_A$
    $save(\mathcal{M}_{f_A})$
    $releaseConnectedDevices()$
```

work[1], to analyze their performance. We have performed an experimental analysis in the CLOUD lab, The University of Melbourne, to evaluate the performance of the proposed approaches. Python 3.8.10 has been used for implementation, and Tensorflow has been used for deep learning-based data analysis. For exchange of model updates in FL, *MLSocket* has been used. The accuracy, precision, recall, and F1-score, considered in performance evaluation are mathematically defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

where TP, TN, FP, and FN denotes true positives, true negatives, false positives, and false negatives, respectively. Along with the accuracy metrics, the time and energy consumption are also considered for performance evaluation.

4.1. Performance of FLDec

The proposed offloading decision-making model, *FLDec*, is evaluated based on prediction accuracy, training time, training energy consumption, and response time and energy consumption of the user device during that period. Different real-time offloading case studies are considered. The time consumption is measured in seconds (s), and the energy consumption is measured in joule (J).

4.1.1. Experimental setup

For the experiment, we have provisioned four virtual machines from the RONIN cloud environment of Amazon AWS to act as four user nodes. Each of the user nodes has 4GB of memory. We also have provisioned two virtual machines from the RONIN cloud for using as the edge server and the cloud server instance. The edge server has 4GB of memory and the processor is Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz. The cloud server instance has 4GB of memory and 2vCPUs. The number of rounds in FL has been set to 10, and $\alpha_r = 1$.

4.1.2. Prediction accuracy, Time, and Energy consumption of user device

In the proposed framework, CFL is used. For predicting the computational intensiveness of a task, we use MLP as the underlying model, and for offloading decision-making, we use LSTM as the underlying model. The parameter values used in MLP and LSTM are presented in Table 4. As we could not find a suitable dataset for deciding whether a task is computationally intensive or not, we have prepared a dataset[1], by executing several computational tasks such as matrix multiplication, sorting,

[1]https://github.com/AnuTuli/OffFed

Table 4: The values of the parameters in MLP and LSTM

Classifier	Parameter	Value
LSTM	Activation	softmax
	Optimizer	Adam
	Epochs	10
	Batch size	200
MLP	Activation	ReLU
	Maximum iteration	100
	Solver	Adam

searching, basic calculator design, file creation, etc., using different input values, and recording the respective time consumption. The dataset contains task type, respective input, memory capacity of the device, and processing speed of the device, which are considered as the input features to predict whether the task is computationally intensive or not. The output label is Yes (if computationally intensive) or No (not intensive). The dataset is split into two parts: (i) one part (80% of the samples) is divided into the number of participating devices, and (ii) the other part (remaining 20% of the samples) is stored in the server as the global dataset. We have obtained the prediction accuracy of 94.74% for the global model using the FL-based decision-making model, using MLP. The results are presented in Fig. 3. For the local models we have achieved an accuracy of 91.17-94.74% after round three. The global model loss after round 10 is presented in Fig. 4. As we observe from the figure, the model converges when the loss becomes minimal. After predicting the computational intensiveness, we have used another dataset[2] that contains network parameters, which are used to predict whether to offload or locally execute the task, using the proposed FL-based decision-making model, using LSTM. The dataset contains cell identity, uplink traffic, downlink traffic, location area code, time duration of data transfer, radio access type, latency, and throughput as the features, and offloading decision (offload or locally execute) as the class label. In this case also, the dataset is split into two parts: (i) one part (80% of the number of samples) is divided into the number of participating devices as their local datasets, and (ii) other part (20%) is stored in the server as the global dataset. Each local dataset is partitioned into the training and testing datasets by the participating devices during local model training using LSTM. After local model training, each device sends its model update to the edge server. The edge server aggregates the received updates to build the global model. This process is repeated for ten rounds in our experiment to build the final global model.

The accuracy, precision, recall, and F1-score, achieved by the local models inside the participating devices for their test datasets are measured, and presented in Fig. 5. The accuracy, precision, recall, and F1-score, achieved by the global model for its test dataset, are also measured and presented in Fig. 5. We observe that the global model has achieved an accuracy of >99% and the local models have achieved >97% accuracy. The

[2]`https://www.kaggle.com/datasets/ucimachinelearning/task-offloading-dataset/data`

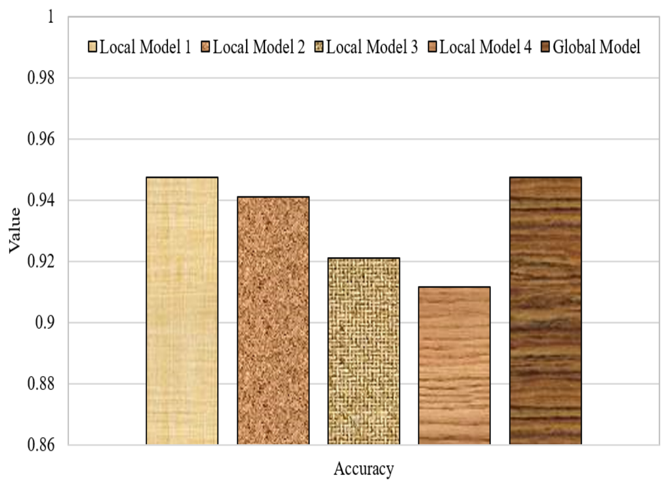


Figure 3: Performance of the local and global models for predicting the task is computationally intensive or not

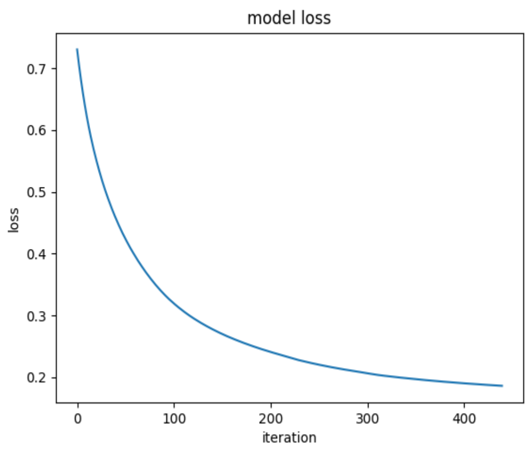


Figure 4: Global model loss while predicting the task is computationally intensive or not

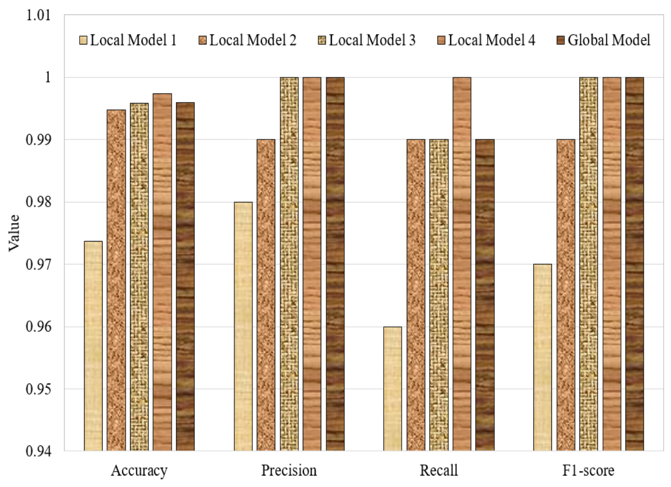


Figure 5: Performance of the local and global models for predicting whether to offload or locally execute the task

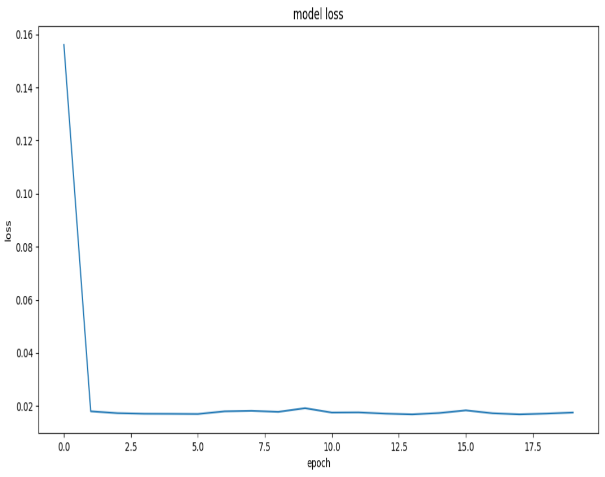
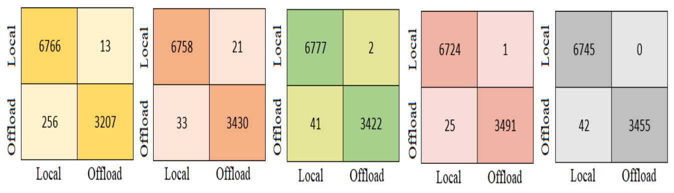


Figure 6: Global model loss while predicting whether to offload or locally execute the task



(a) Confusion matrix for Local Model 1 (b) Confusion matrix for Local Model 2 (c) Confusion matrix for Local Model 3 (d) Confusion matrix for Local Model 4 (e) Confusion matrix for Global Model

Figure 7: Confusion matrix for local and global models

precision, recall, and F1-score are ≥0.99 for the global model, and ≥0.96 for the local models. The global model loss after round 10 is presented in Fig. 6, and we observe that the model converges when the loss becomes minimal. The confusion matrices for the local models for their test datasets are presented in Fig. 7. The confusion matrix for the global model for its test dataset is presented in Fig. 7. As there are two classes for decision-making ((1) locally execute, (2) offload), the confusion matrices show the results for two classes only. Here, the test accuracy for the local and global models are presented. The test datasets for the devices and the edge server are considered for evaluating the models' prediction accuracies.

The total time consumption for the edge server is determined by summing up the model initialization time, aggregation time, and communication time for exchanging model updates. For the participating devices the total time consumption is determined by adding the local model training time and communication time for exchanging model updates. The total time consumption for the edge server and participating devices are measured from the experiment, and presented in Fig. 8. In the figure, the time consumption of the local model indicates the total time consumption of the respective device, and the time consumption of the global model denotes the total time consumption of the edge server. We observe that the time consumption for the local models ranges from 200s to 350s, and the time consumption for the edge server is 110.36s. The energy consumption of the edge server is calculated as the product of the total time consumption of the edge server and its power

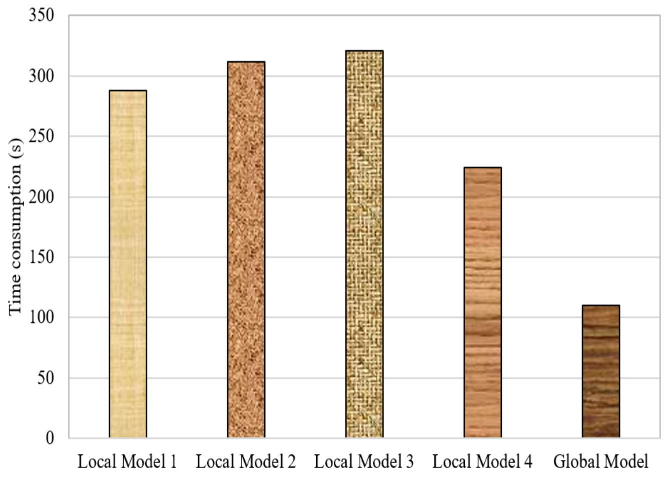


Figure 8: Time consumption in proposed approach FLDec

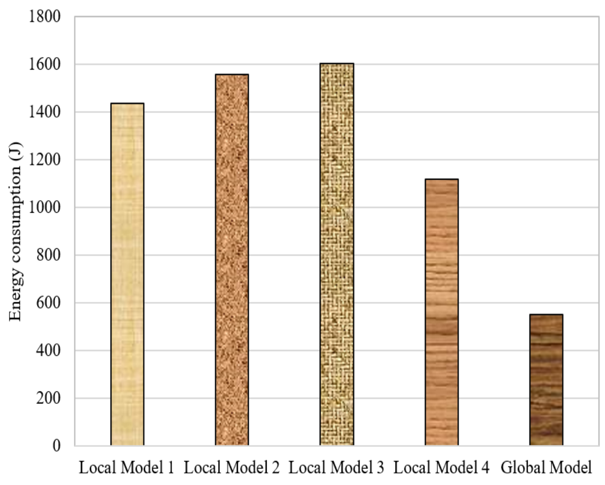


Figure 9: Energy consumption in proposed approach FLDec

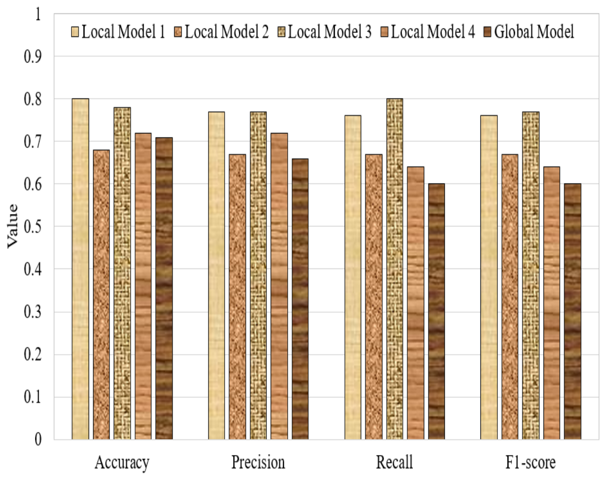


Figure 10: Performance of local and global models without using FL

consumption per unit time (second). The energy consumption of each participating device is calculated as the product of the total time consumption of the device and its power consumption per unit time (second). The energy consumption for the participating devices and the edge server are presented in Fig. 9. In the figure, the energy consumption of the local model indicates the energy consumption of the respective device, and the energy consumption of the global model denotes the energy consumption of the edge server. The energy consumption of the edge server is 551.8J, and the energy consumption of the devices ranges from 1000J to 1600J.

Comparison with decision-making without FL: The accuracy, precision, recall, and F1-score achieved by the local and global models using LSTM without FL are also measured, and we have observed that the prediction accuracy is above 65% for all the local models (trained using local datasets) and above 70% for the global model (trained using global dataset). The precision, recall, and F1-score for all local models are above 0.6, and for the global model it is ≥0.6. The results are presented in Fig. 10. As we observe, the accuracy, precision, recall, and F1-score using FL are better for the global model as well as for all the local models.

4.1.3. Task offloading, Response time, and Energy consumption of user device

We have considered different tasks in our experiment, and the FL-based model is used for offloading decision-making. The response time for task execution scenarios and the energy consumption of the user device during the period are measured and presented in Table 5. The response time in computational task offloading is determined as the difference between the time stamp of submitting a request and the time stamp of receiving the response. If the time stamp of request submission for task execution is T_{req} and the time stamp of receiving response is T_{res}, then the response time is given as $(T_{res} - T_{req})$. The energy consumption is measured as the power consumption per unit time multiplied by the response time. If the power consumption per unit time is Pow_{dev}, then the energy consumption is measured as $(Pow_{dev} \cdot (T_{res} - T_{req}))$.

As the *first task type*, we have considered a basic calculator design that performs four operations: addition, subtraction, multiplication, and division. Most of the user devices contain this basic application, which can work irrespective of the Internet connectivity. The user inputs numbers, and the calculator generates the results according to the requested operation. The response time and energy consumption of the device during the period are measured and presented in Table 5. As the *second task type*, we have considered matrix multiplication, where, based on the prediction result of the *FL-based decision-making model*, the task is either offloaded or locally executed. Here, we have considered four different cases. For the first case, the two matrices are of order 50x50, and based on the prediction result, the task is executed locally. The response time and energy consumption of the device during the period are measured, and presented in Table 5. For the other three cases, based on the prediction results, the tasks are offloaded. The response time and energy consumption of the device during the period are measured, and presented in Table 5. As the *third task type*, we have considered file creation. The file is created locally or remotely, according to the user's preference. The response time and energy consumption of the device during the period are measured and presented in Table 5.

For evaluating the performance for a wide range of users in a realistic environment, we have performed a simulation in MATLAB2024. The data transmission speed considered in the simulation is 500-1500Mbps with an average of 1000Mbps, and the task size is randomly chosen from the range 10,000-50,000 bits. The processing speed of the task executing node is considered 5GHz. The link failure rate is considered from 0.1-0.5. The task waiting delay is also considered during simulation. In Fig. 11, the average response time for 100-1000 users' requests for task execution are presented. The average energy consumption of the user device during the periods are also computed and presented in Fig. 12. We observe that the average response time lies in the range of 3-4.25s, and the average energy consumption lies in the range of 14-22J.

Comparison of task offloading using FLDec with baselines: As the baseline for comparison, we considered two scenarios: (i) all tasks are offloaded, (ii) all tasks are locally executed. We observe from Table 5 that for the task *Calculator*, the proposed scheme *FLDec* suggests *local execution*. If the task was offloaded, then the response time and energy consumption for addition, subtraction, multiplication, and division operations would be ~90%, ~95%, ~85%, and ~96%, higher than the local execution. For the task matrix multiplication, four scenarios are considered as presented in Table 5. For the first scenario, the FLDec decides *local execution*, and it has ~57% lower time and energy consumption than offloading it. For the other three scenarios, FLDec suggests *offload*, and we observe that the response time and energy consumption both are reduced by ~11-31% than local execution. Hence, we observe that if all the considered tasks were locally executed, then for higher order matrix multiplication, the response time and energy consumption would be higher. Similarly, if all the considered tasks were offloaded, the response time and energy consumption would be higher for all of the considered operations for the calculator, and for matrix multiplication, the first scenario would have higher response time and energy consumption. Thus, from the results, we observe that the proposed FL-based decision-making model takes appropriate decisions for the considered case studies, and reduces the response time and energy consumption.

4.2. Performance of FedOff

The performance of the proposed partial computation offloading method for FL which is referred to as federated offloading or *FedOff*, is evaluated in terms of prediction accuracy, precision, recall, and F1-score of the global as well as local models, total time consumption, and energy consumption of the user device during the period. The implementation diagram of the proposed framework is presented in Fig. 13.

4.2.1. Experimental setup

To evaluate the performance of *FedOff*, we have provisioned six virtual machines from the RONIN cloud environment. Among

Table 5: Response time and energy consumption of the user device in FLDec-based task execution

Task		Decision (Offload/ Local)	Response Time (s)	Energy consumption (J) of user device
(1) Calculator	Addition: Five digit numbers	Local	0.01 (Offload: 0.1)	0.05 (Offload: 0.5)
	Subtraction: Five digit numbers	Local	0.0065 (Offload: 0.125)	0.0325 (Offload: 0.625)
	Multiplication: Five digit numbers	Local	0.018 (Offload: 0.128)	0.09 (Offload: 0.64)
	Division: Five digit numbers	Local	0.005 (Offload: 0.127)	0.025 (Offload: 0.635)
(2) Matrix multiplication	Order: 50x50, 50x50	Local	0.06 (Offload: 0.14)	0.3 (Offload: 0.7)
	Order: 100x100, 100x100	Offload	0.46 (Local: 0.6684)	2.3 (Local: 3.342)
	Order: 200x200, 200x200	Offload	4.55 (Local: 5.78)	22.75 (Local: 28.9)
	Order: 300x300, 300x300	Offload	23.476 (Local: 26.43)	117.38 (Local: 132.15)
(3) File creation	User preference: Local, size: 5KB	Local	1.92	9.6
	User preference: Remote, size: 5KB	Offload	1.37	6.85
	User preference: Local, size: 9KB	Local	2.07	10.35
	User preference: Remote, size: 9KB	Offload	1.88	9.4
	User preference: Local, size: 28KB	Local	2.72	13.6
	User preference: Remote, size: 28KB	Offload	2.61	13.05

them, five act as the clients, and one as the edge server. Each of the clients has 4GB of memory. The edge server has 4GB of memory and the processor is Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz. The number of rounds was set to 5, and $\alpha_r = 1$. To evaluate the performance of FedOff, we have considered an activity recognition dataset[3]. Activity recognition is one of the popular application used by mobile users. The dataset is split into two parts: one part (80% of the samples) is split into the five clients, and the other part (remaining 20% of the samples) is stored inside the server as the global dataset.

4.2.2. Prediction accuracy

The clients, i.e., devices, split their datasets into the following ratios: (i) local-25%, offload-75%, (ii) local-50%, offload-50%, and (iii) local-75%, offload-25%. The portion of data to be offloaded is encrypted using AES-128 [36], and Hash-based Message Authentication Code (HMAC) using Secure Hash Algorithm (SHA)-256 [38] is used for authentication, in the proposed partial computation offloading method for FL (federated offloading or FedOff). As symmetric key cryptography is used, the same key is used for encryption and decryption. However, all the devices have different keys. For secure key exchange, the RSA algorithm [37] is used. We have measured the data transmission latency and the latency for exchanging keys, encryption, and decryption, from the experiment.

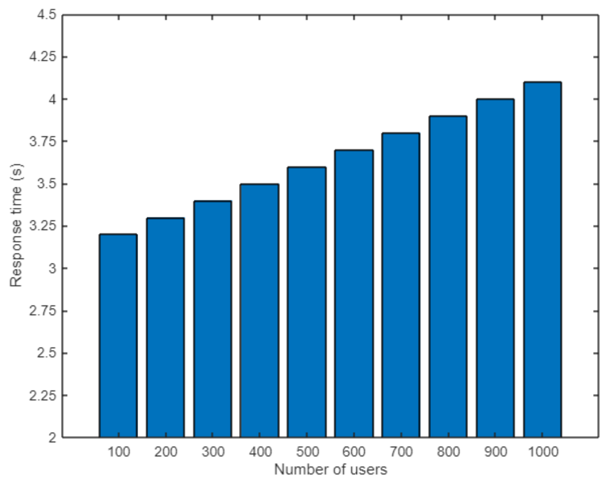


Figure 11: Average response time in FLDec

[3]https://www.kaggle.com/datasets/nurulaminchoudhury/harsense-datatset

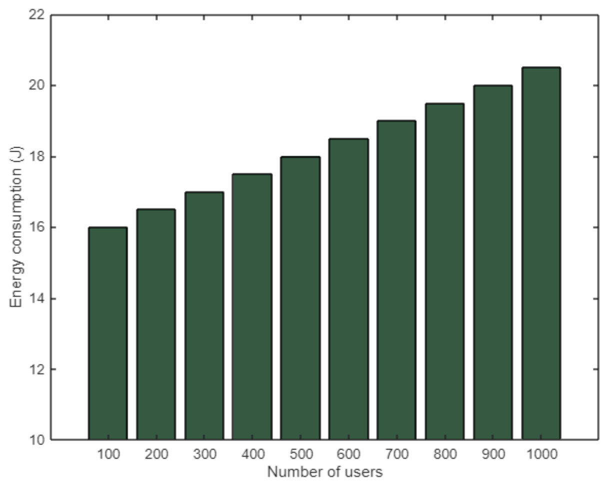


Figure 12: Average energy consumption of user device during response time period in FLDec

The accuracy, precision, recall, and F1-score of the global model, achieved by the proposed approach FedOff are presented in Fig. 14. The average accuracy, precision, recall, and F1-score of the local models, using the proposed approach FedOff are presented in Fig. 15. The results show that FedOff has achieved >98% accuracy for the global model and >94% accuracy for the local models. The precision, recall, and F1-score are 0.98 for the global model, and >0.92 for the local models. We have also conducted the experiment for conventional FL, where no data offloading takes place, and only model updates are exchanged between the clients and the server. We have also conducted the experiment for the case where the devices are unable to train models, and the data from the devices are fully offloaded to the edge server. The edge server trains the model for individual datasets, and then performs aggregation. After developing the model, the server sends it to the clients so that they can perform predictions using the model in the future. The accuracy, precision, recall, and F1-score of the global model for conventional FL and full offloading are also presented along with the proposed approach FedOff in Fig. 14. The average accuracy, precision, recall, and F1-score of the local models for conventional FL are presented along with the proposed approach FedOff in Fig. 15. As we observe, the accuracy, precision, recall, and F1-score of the global model in the proposed method FedOff are almost the same as those of conventional FL, and better than the case of full offloading.

The global model loss for FedOff for the considered three scenarios ((i) local-25%, offload-75%, (ii) local-50%, offload-50%, and (iii) local-75%, offload-25%) are presented in Fig. 16. We observe that for each of the case, the loss tends to 0, which, along with the nature of the curve, indicates that the model has converged.

4.2.3. Time and Energy consumption

The total time for FedOff for the three considered scenarios ((i) local-25%, offload-75%, (ii) local-50%, offload-50%, and (iii) local-75%, offload-25%) are measured and presented in Fig. 17. The total time includes the total training time (training time + communication time for model updates exchange model updates + aggregation time) considering all rounds, the time consumption for data transmission, and the time consumption for data encryption and decryption. The total time consumptions for the conventional FL and full offloading are also measured, and presented in the figure. The average energy consumption of the devices during the total period are measured as the product of the time period and the average power consumption per unit time, and presented in Fig. 18. As we observe, the total time and energy consumption are lowest in conventional FL, where the devices locally analyze the full dataset, and exchange model updates with the server. However, if the devices cannot analyze the full dataset, then case 1 consumes the lowest time and energy consumption. Case 2 can also be adopted, where 50% data is locally analyzed, and rest 50% is offloaded to the server. If the device is unable to analyze the dataset at all, then the entire dataset is offloaded to the server. We observe from Figs. 17 and 18 that the use of cryptography for privacy protection during data transmission consumes a very less amount of time and energy. We observe that the use of encryption and decryption increases the time and average energy consumption by only 0.05-0.16% but ensures data privacy protection during transmission. The data size encrypted in the experiment ranges from 1-1.5 MB. AES is a faster encryption algorithm, i.e., the time consumption is very less. The time complexity of AES-128 is $O(Num_{block})$, where Num_{block} is the number of blocks of size 128 bits. As data privacy and time consumption both are vital, we have used AES-128 for encryption due to its fast encryption process.

4.3. Comparison with Existing Offloading Methods

This section compares the proposed approaches with the existing offloading schemes.

Comparison with existing offloading schemes using simulation: In Fig. 19, the total time in task execution using FLDec is compared with the task execution time in existing offloading schemes [5, 8, 9, 14], with respect to the task size. The data transmission speed was considered 500-1500Mbps, with an average of 1000Mbps. The processing speed of the task executing node was considered 5GHz. For comparison, we have performed a simulation based on different task sizes (1000-3000 bits), while performing task offloading or local execution using FLDec, and determined the time consumption in task execution. We have performed simulation for the existing works also to obtain the results. The total energy consumption (device and edge server) during task execution period is also determined for FLDec and compared with the existing offloading schemes in Fig. 20, with respect to the task size. We observe from the results that FLDec outperforms the existing offloading approaches with respect to total time and energy consumption in task execution. FLDec has ~75-79%, ~47-57%, ~66-74%, and ~38-50%, less time consumption than joint task offloading and resource allocation method (JORA) [5], multi-feature matching scheme (MFM) [7], action space recursive decomposition method (ASRD) [8], and dynamic switching algorithm (DSA) [9], respectively. We also observe that FLDec has

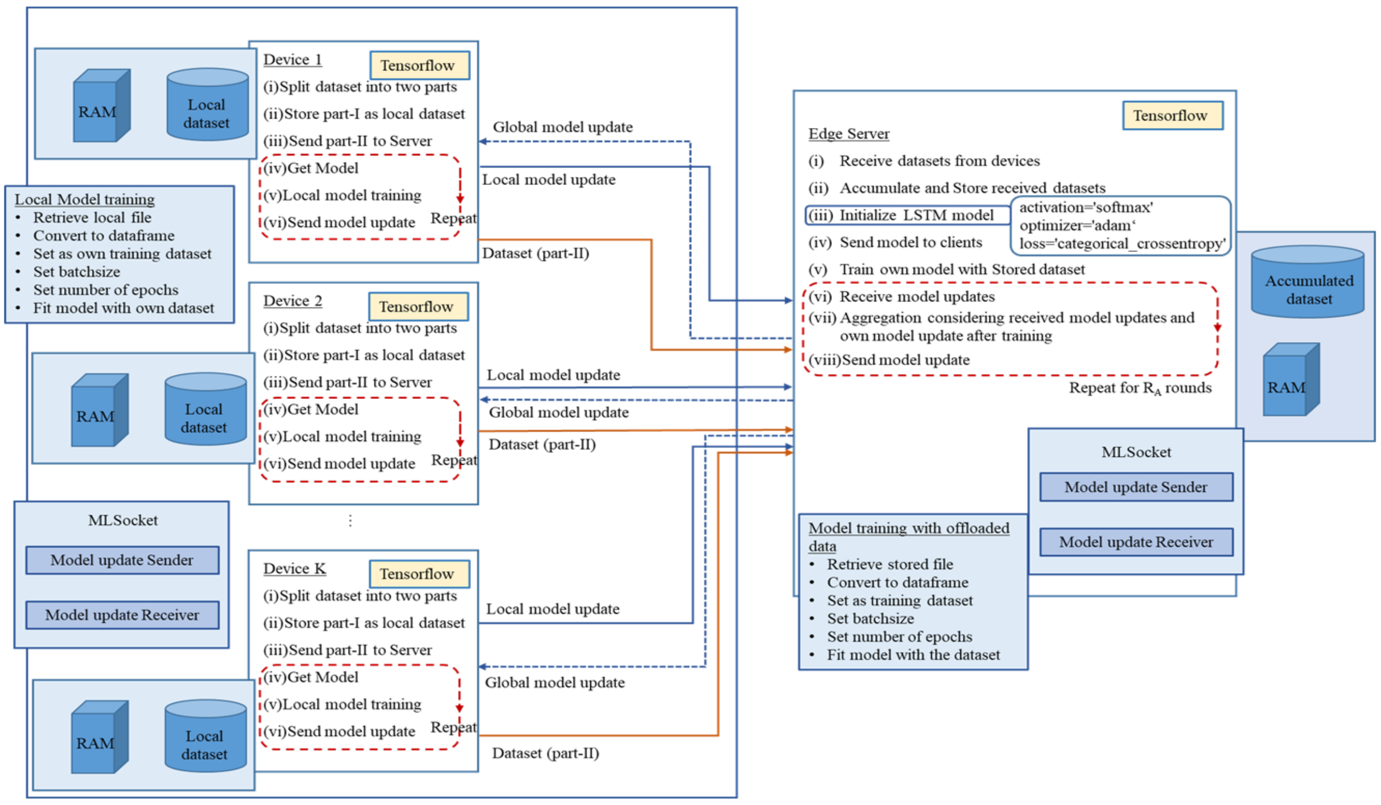

Device 1
Tensorflow
(i)Split dataset into two parts
(ii)Store part-I as local dataset
(iii)Send part-II to Server
(iv)Get Model
(v)Local model training
(vi)Send model update
Repeat
RAM
Local dataset
Local Model training
Retrieve local file
Convert to dataframe
Set as own training dataset
Set batchsize
Set number of epochs
Fit model with own dataset
Global model update
Local model update
Dataset (part-II)
Device 2
Device K
MLSocket
Model update Sender
Model update Receiver
Edge Server
(i) Receive datasets from devices
(ii) Accumulate and Store received datasets
(iii) Initialize LSTM model
activation='softmax'
optimizer='adam'
loss='categorical_crossentropy'
(iv) Send model to clients
(v) Train own model with Stored dataset
(vi) Receive model updates
(vii) Aggregation considering received model updates and own model update after training
(viii)Send model update
Repeat for R_A rounds
Accumulated dataset
Model training with offloaded data
Retrieve stored file
Convert to dataframe
Set as training dataset
Set batchsize
Set number of epochs
Fit model with the dataset

Figure 13: Implementation diagram of the proposed federated offloading framework

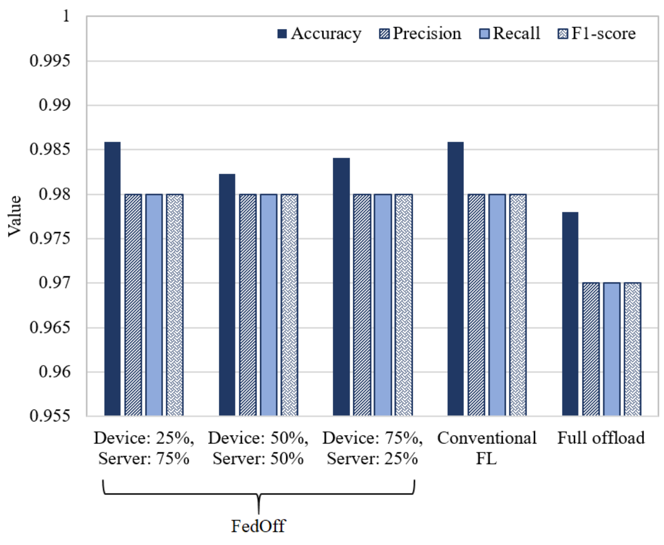


Figure 14: Accuracy, precision, recall, and F1-score of the global model in FedOff

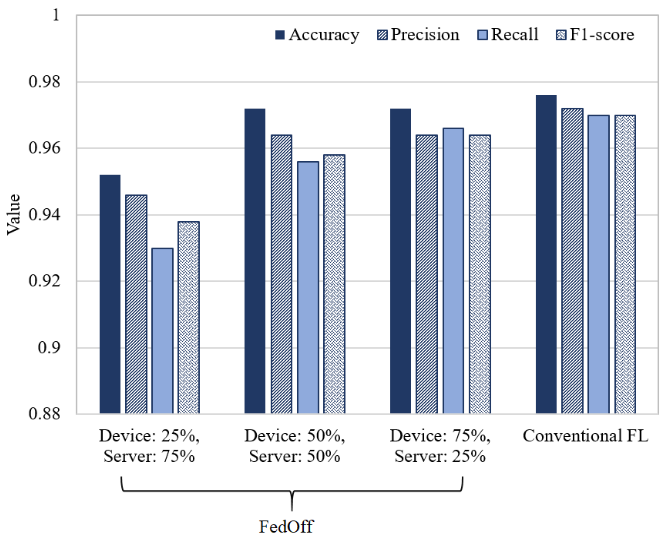
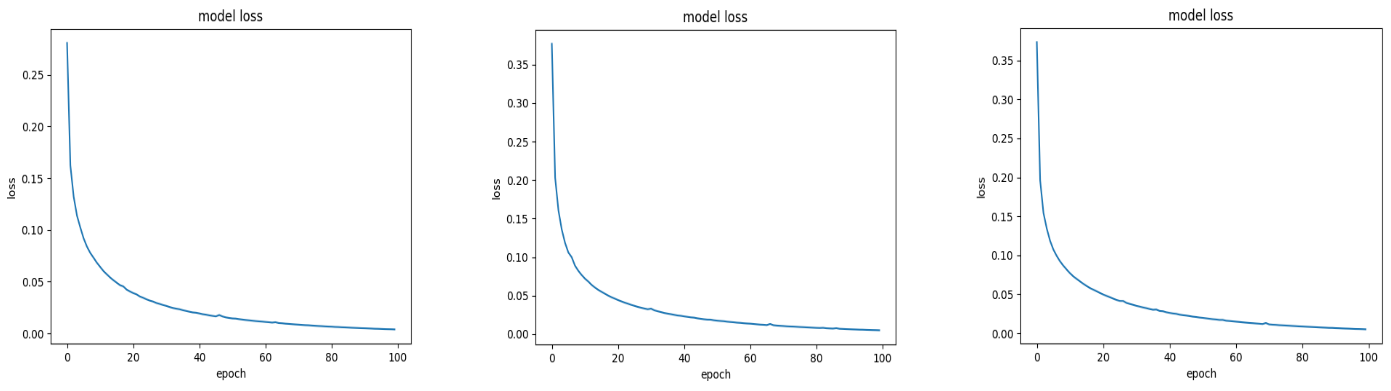


Figure 15: Average accuracy, precision, recall, and F1-score of the local models in FedOff



(a) Global model loss for case 1 (local-25%, offload-75%)

(b) Global model loss for case 2 (local-50%, offload-50%)

(c) Global model loss for case 3 (local-75%, offload-25%)

Figure 16: Global model loss in the proposed federated offloading method

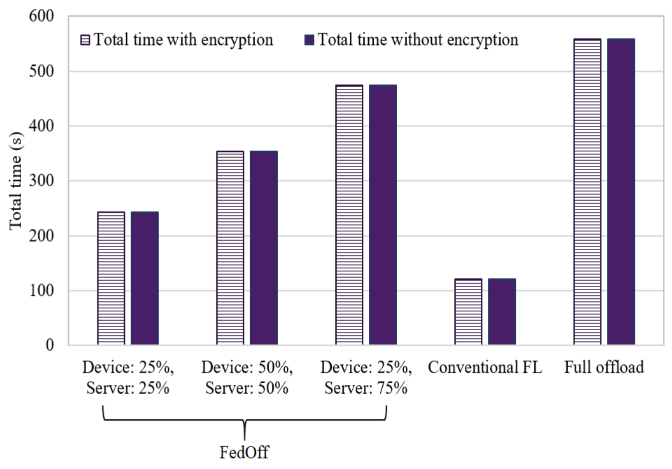


Figure 17: Total time consumption in FedOff

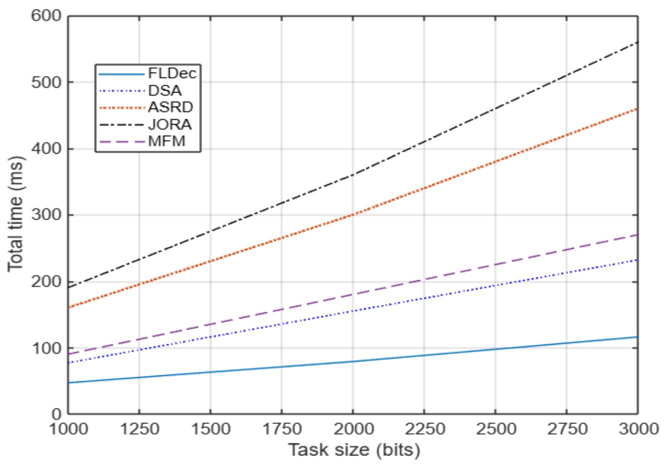


Figure 19: Total time in task offloading in FLDec and existing offloading schemes

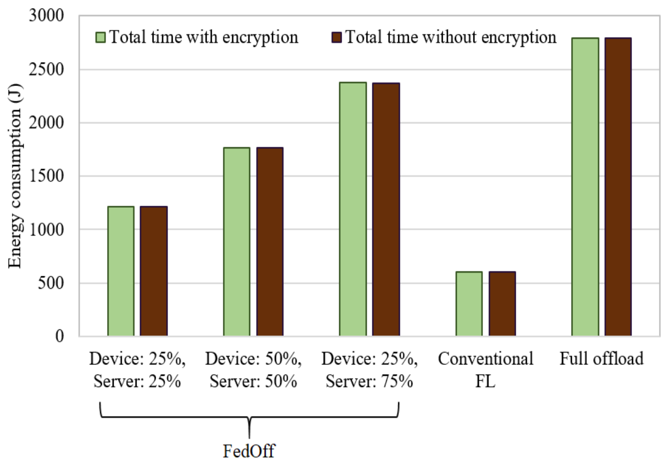


Figure 18: Average energy consumption of the participating devices during the total time in FedOff

~10-15%, ~33-42%, ~45-58%, and ~65-68% less energy consumption than DSA, MFM, JORA, and ASRD, respectively. In FLDec, as we have used an FL-based decision-making model to decide whether to offload or not based on task type, user input, device configuration, and network parameters, accurate decision-making is achieved, and the time consumption as well as energy consumption in task execution are minimal.

Comparison with existing schemes for offloading in FL using experiments: We carried out a comparative study between the proposed partial computation offloading method FedOff and the existing offloading methods for FL. The comparative study is presented in Table 6. In [16], convolutional neural network (CNN) was used, and the model loss was 0.2-0.5. The system delay was 10-50s per round [16]. The number of rounds in [16] was 100 to converge and get minimum loss. Hence, the total delay was 1000-5000s in [16], considering all rounds. In [17], deep neural network (DNN) was used, and the prediction accuracy was 80%. The training time was 343-701s per round [17]. The total number of rounds in [17] was 100 to converge. Hence, the total training time was 34300-70100s in [16], considering all rounds. Unlike our proposed method, [16, 17] did

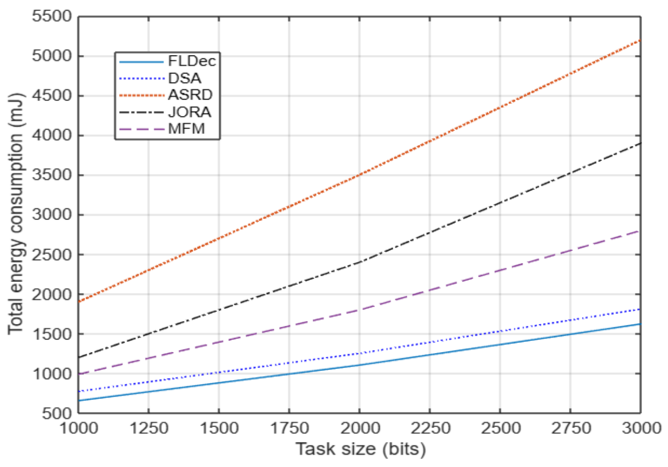


Figure 20: Total energy consumption in FLDec and existing offloading schemes

not consider the energy consumption and secure data offloading aspects. In [18], DRL was used, and the authors set a target training accuracy of 98%. The number of iterations was 2000 to converge [18]. The authors measured the system delay from simulation, and it was 10-28s [18]. They also measured the processing power, and it was 4-15 watts [18]. However, the authors did not use any cryptographic approach for data privacy protection during offloading of dataset [18]. As we observe, FedOff achieved >98% accuracy for the global model and >94% accuracy for the local models. The number of rounds to converge with minimum loss in FedOff was 5. The total time consumption was 200-500s in FedOff, and the average time consumption per round was 40-100s. The total time consumption in FedOff includes the total training time (training time + communication time for model updates exchange model updates + aggregation time) considering all rounds, the time consumption for data transmission, and the time consumption for data encryption and decryption. The average energy consumption of the participating user devices during the total period was 1000-2500J. As we observe, the achieved prediction accuracy is highest in FedOff, and it has acheived secure data offloading by using cryptography. Thus, considering all the aspects as presented in Table 6, we observe that FedOff outperforms the existing approaches.

5. Conclusion and Future Work

This paper proposes an FL-based offloading decision-making model for mobile devices, and a partial computation offloading method for FL to deal with the Straggler effect. Based on the device configuration, task type, and respective input, the FL-based decision-making model predicts whether a task is computationally intensive or not, using MLP. If the predicted result is *computationally intensive*, then the proposed FL-based decision-making model predicts *whether to offload or locally execute* the task based on the network parameters, using LSTM. Based on the predicted result, the task is either locally executed or offloaded to the edge server. If the edge server is unable to execute it or if the user preference is remote execution, then, the task is offloaded to the cloud. The experimental results present that the proposed offloading decision-making method has achieved above 90% prediction accuracy, and the considered case scenarios show that the proposed method provides the result at a lower response time and lower energy consumption of the user device. The results also present that the proposed offloading method has reduced the response time by ~11-31% for computationally intensive tasks. In the proposed partial computation offloading method for FL, the resource-constrained devices offload a part of their local dataset to the edge server if they are unable to analyze the whole dataset. For secure data transmission during offloading to the edge server, AES-128 is used. As time consumption and data privacy both are vital, we have used AES-128 for fast encryption. The experimental results show that the proposed partial computation offloading method for FL has achieved >98% accuracy for the global model. The results also present that the use of AES-128 for encryption increases the time by 0.05-0.16% but ensures data privacy protection during transmission.

As part of our future work, we will extend the proposed work to a differential privacy-based FL model for task offloading to further enhance the data privacy and security of model updates. The class imbalance and data scarcity issues are also to be dealt with to introduce an accurate prediction model. To deal with these issues, generative artificial intelligence can be used to generate realistic samples for training purpose. The integration of generative adversarial networks with FL is another future research direction of this work.

Acknowledgements

This work is partially supported by ARC Discovery Project (DP240102088) and ARC LIEF grants.

References

[1] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[2] Zhiyuan Wu, Sheng Sun, Yuwei Wang, Min Liu, Ke Xu, Wen Wang, Xuefeng Jiang, Bo Gao, and Jinda Lu. Fedcache: A knowledge cache-driven federated learning architecture for personalized edge intelligence. *IEEE Transactions on Mobile Computing*, 2024.

[3] Zhiwei Yao, Jianchun Liu, Hongli Xu, Lun Wang, Chen Qian, and Yunming Liao. Ferrari: A personalized federated learning framework for heterogeneous edge clients. *IEEE Transactions on Mobile Computing*, 2024.

[4] Anwesha Mukherjee and Rajkumar Buyya. Federated learning architectures: A performance evaluation with crop yield prediction application. *Software: Practice and Experience*, 2024.

[5] Hongbo Jiang, Xingxia Dai, Zhu Xiao, and Arun Iyengar. Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Transactions on Mobile Computing*, 22(7):4000–4015, 2022.

[6] Pavlos Athanasios Apostolopoulos, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. Risk-aware data offloading in multi-server multi-access edge computing environment. *IEEE/ACM Transactions on Networking*, 28(3):1405–1418, 2020.

Table 6: Comparison of FedOff with existing offloading methods for FL

Work	**Classifier**	**Accuracy/ Loss**	**Time/ Delay**	**Energy consumption of user device**	**Use of cryptography for data privacy protection**	**Number of rounds to converge**
Ji et al. [16]	CNN	Loss: 0.2-0.5	10-50s (per round) Total: 1000-5000s (100 rounds)	Not measured	✗	100
Wu et al. [17]	DNN	Accuracy: 80%	343-701s (per round) Total: 34300-70100s (100 rounds)	Not measured	✗	100
Liaq et al. [18]	Deep Reinforcement Learning (DRL)	Accuracy: 98%	System delay: 10-28s	Processing power: 4-15 watts	✗	2000
FedOff (proposed)	LSTM	Accuracy: >98% (global model), Loss: <0.35	40-100s (per round), Total: 200-500s (5 rounds)	1000-2500J	✓	5

[7] Ming Tang and Vincent WS Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 21(6):1985–1997, 2020.

[8] Tai Manh Ho and Kim-Khoa Nguyen. Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach. *IEEE Transactions on Mobile Computing*, 21(7):2421–2435, 2020.

[9] Tian Wang, Yuzhu Liang, Yilin Zhang, Xi Zheng, Muhammad Arif, Jin Wang, and Qun Jin. An intelligent dynamic offloading from cloud to edge for smart iot systems with big data. *IEEE Transactions on Network Science and Engineering*, 7(4):2598–2607, 2020.

[10] Gabriele Russo Russo, Daniele Ferrarelli, Diana Pasquali, Valeria Cardellini, and Francesco Lo Presti. Qos-aware offloading policies for serverless functions in the cloud-to-edge continuum. *Future Generation Computer Systems*, 156:1–15, 2024.

[11] Song Liu, Shiyuan Yang, Hanze Zhang, and Weiguo Wu. A federated learning and deep reinforcement learning-based method with two types of agents for computation offload. *Sensors*, 23(4):2243, 2023.

[12] Xiangjun Zhang, Weiguo Wu, Jinyu Wang, and Song Liu. Bilstm-based federated learning computation offloading and resource allocation algorithm in mec. *ACM Transactions on Sensor Networks*, 19(3):1–20, 2023.

[13] Yiwen Han, Ding Li, Haotian Qi, Jianji Ren, and Xiaofei Wang. Federated learning-based computation offloading optimization in edge computing-supported internet of things. In *Proceedings of the ACM Turing Celebration Conference-China*, pages 1–5, 2019.

[14] Jine Tang, Sen Wang, Song Yang, Yong Xiang, and Zhangbing Zhou. Federated learning-assisted task offloading based on feature matching and caching in collaborative device-edge-cloud networks. *IEEE Transactions on Mobile Computing*, 2024.

[15] Kai Peng, Peiyun Xiao, Shangguang Wang, and Victor CM Leung. Scof: Security-aware computation offloading using federated reinforcement learning in industrial internet of things with edge computing. *IEEE Transactions on Services Computing*, 2024.

[16] Zhongming Ji, Li Chen, Nan Zhao, Yunfei Chen, Guo Wei, and F Richard Yu. Computation offloading for edge-assisted federated learning. *IEEE Transactions on Vehicular Technology*, 70(9):9330–9344, 2021.

[17] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. Fedadapt: Adaptive offloading for iot devices in federated learning. *IEEE Internet of Things Journal*, 9(21):20889–20901, 2022.

[18] Mudassar Liaq and Waleed Ejaz. Minimizing delay in uav-aided federated learning for iot applications with straggling devices. *IEEE Open Journal of the Communications Society*, 2024.

[19] Rui Chen, Dian Shi, Xiaoqi Qin, Dongjie Liu, Miao Pan, and Shuguang Cui. Service delay minimization for federated learning over mobile devices. *IEEE Journal on Selected Areas in Communications*, 41(4):990–1006, 2023.

[20] Anwesha Mukherjee, Shreya Ghosh, Debashis De, and Soumya K Ghosh. Mcg: mobility-aware computation offloading in edge using weighted majority game. *IEEE Transactions on Network Science and Engineering*, 9(6):4310–4321, 2022.

[21] Shreya Ghosh, Anwesha Mukherjee, Soumya K Ghosh, and Rajkumar Buyya. Mobi-iost: mobility-aware cloud-fog-edge-iot collaborative framework for time-critical applications. *IEEE Transactions on Network Science and Engineering*, 7(4):2271–2285, 2019.

[22] Anwesha Mukherjee, Debashis De, Soumya K Ghosh, and Rajkumar Buyya. Introduction to mobile edge computing. *Mobile Edge Computing*, pages 3–19, 2021.

[23] Chuan Feng, Pengchao Han, Xu Zhang, Bowen Yang, Yejun Liu, and Lei Guo. Computation offloading in mobile edge computing networks: A survey. *Journal of Network and Computer Applications*, 202:103366, 2022.

[24] Kuanishbay Sadatdiynov, Laizhong Cui, Lei Zhang, Joshua Zhexue Huang, Salman Salloum, and Mohammad Sultan Mahmud. A review of optimization methods for computation offloading in edge computing networks. *Digital Communications and Networks*, 9(2):450–461, 2023.

[25] Mohammad Yahya Akhlaqi and Zurina Binti Mohd Hanapi. Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications*, 212:103568, 2023.

[26] Sanaz Taheri-abed, Amir Masoud Eftekhari Moghadam, and Mohammad Hossein Rezvani. Machine learning-based computation offloading in edge and fog: a systematic review. *Cluster Computing*, 26(5):3113–3144, 2023.

[27] Zeinab Zabihi, Amir Masoud Eftekhari Moghadam, and Mohammad Hossein Rezvani. Reinforcement learning methods for computation offloading: a systematic review. *ACM Computing Surveys*, 56(1):1–41, 2023.

[28] Xin Li, Long Chen, Zian Yuan, and Guangrui Liu. Aiho: Enhancing task offloading and reducing latency in serverless multi-edge-to-cloud systems. *Future Generation Computer Systems*, 165:107607, 2025.

[29] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

[30] Yunfan Ye, Shen Li, Fang Liu, Yonghao Tang, and Wanting Hu. Edgefed: Optimized federated learning based on edge computing. *IEEE Access*, 8:209191–209198, 2020.

[31] Xu Zhao, Yichuan Wu, Tianhao Zhao, Feiyu Wang, and Maozhen Li. Federated deep reinforcement learning for task offloading and resource allocation in mobile edge computing-assisted vehicular networks. *Journal of Network and Computer Applications*, 229:103941, 2024.

[32] Xing Chen, Bohuai Xiao, Xinyu Lin, Zheyi Chen, and Geyong Min. Multi-agent collaboration for vehicular task offloading using federated deep reinforcement learning. *IEEE Transactions on Mobile Computing*, 2025.

[33] Wei Zou, Rongqian Zhang, and Yijie Xun. On a federated learning based computation offloading strategy for non-terrestrial network assisted internet of medical things. *IEEE Internet of Things Journal*, 2025.

[34] Somnath Bera, Tanushree Dey, Anwesha Mukherjee, and Debashis De. Flag: Federated learning for sustainable irrigation in agriculture 5.0. *IEEE Transactions on Consumer Electronics*, 2024.

[35] Manish Bharat, Ritesh Dash, K Jyotheeswara Reddy, ASR Murty, C Dhanamjayulu, and SM Muyeen. Secure and efficient prediction of electric vehicle charging demand using α2-lstm and aes-128 cryptography. *Energy and AI*, 16:100307, 2024.

[36] Mohammed N Alenezi, Haneen Alabdulrazzaq, Hajed M Alhatlani, and Faisal A Alobaid. On the performance of aes algorithm variants. *International Journal of Information and Computer Security*, 23(3):322–337, 2024.

[37] Toufik Ghrib, Ahmed Ghali, Fouzi Benbrahim, and Yusef Awad Abusal. Utilizing the rsa algorithm to enhance the security of real-world applications. *Informatica*, 49(9), 2025.

[38] Ankush Soni, Sanjay K Sahay, and Vaishali Soni. Hash based message authentication code performance with different secure hash functions. In *2025 10th International Conference on Signal Processing and Communication (ICSC)*, pages 104–109. IEEE, 2025.