FEDge-HAR: An Optimized Private Mobile Edge-enabled IoT paradigm for Privacy of Human Activity Recognition

Ateeq ur Rehman, Mahnoor Farooq, Fazlullah Khan*, Gautam Srivastava*, Rakan Ameen Aldmour, Ryan Alturki, Bandar Alshawi

***Abstract*—Federated Learning (FL) has emerged as a pivotal technology for the Internet of Things (IoT) that models distributed client data without compromising privacy. The IoT-based wearable generates data and FL running on a private edge performing Human Activity Recognition (HAR). In this paper, we proposed a novel technique to protect sensitive data during the training process and ensure the confidentiality of model updates before transmission to the edge server. The proposed technique integrates the El-Gamal encryption technique for data protection, and the FL process is rigorously optimized using Pruning, Quantization, and Network Slicing. Pruning removes redundant connections, which reduces model complexity and communication delays. On the other hand, Quantization decreases the bit precision of model parameters, and Network Slicing strategically allocates resources solely for FL resulting in low latency and optimal bandwidth utilization. The results are evaluated in terms of accuracy and communication overhead, which is highly required in real-world applications. Furthermore, the HAR system within PEC shows better results by achieving an accuracy of 99% at 300 epochs that outperformed existing Machine Learning (ML) algorithms.**

***Index Terms*—Internet of Things; Federated Learning; Private Edge Computing; Human Activity Recognition; El-Gamal; Quantization; Pruning; Network Slicing; Privacy Protection.**

I. Introduction

The widespread adoption of mobile devices and the rapid rise of the Internet of Things (IoT) has resulted in an exponential increase in data generation recently. A huge amount of data is generated in healthcare, particularly by Human Activity Recognition (HAR) systems. The HAR has developed as a crucial field with significant practical consequences among the various applications and domains that benefit from this data. The HAR systems detect and classify human actions using sensor data gathered from wearable devices, cellphones, or IoT sensors. These systems have many applications such as healthcare monitoring, smart homes, security surveillance and fitness tracking. Studies such as [1], [2] underline the significance of HAR in healthcare monitoring, allowing for continuous patient monitoring and personalized treatment plans. Traditional HAR techniques often rely on centralized data processing, in which all sensor data is collected and analyzed on a central server or in the cloud. However, in terms of privacy, communication delay, and bandwidth consumption, this centralized paradigm presents considerable issues. As user data is transmitted and stored in a centralized server, privacy concerns develop, raising the possibility of unauthorized access and misuse. Furthermore, sending substantial volumes of unprocessed sensor data to the cloud incurs considerable communication latency and consumes significant bandwidth, particularly in resource-constrained mobile situations. Research such as [3], [4] discusses the challenges of privacy and latency in IoT and mobile computing environments.

HAR has been widely investigated using various Machine Learning (ML) paradigms in [5]–[7]. For instance in [5], the authors employed a Recurrent Neural Network (RNN), leveraging Long Short-Term Memory (LSTM) RNN and smart home sensors for HAR. However, it was not studied in the context of preserving privacy. Similarly, the authors in [6] addressed the challenge of monitoring physical activities using smartphone sensors, covering actions like running, walking, climbing, biking and driving. Their evaluation utilized algorithms such as Naive Bayes, Decision Trees, K-Nearest Neighbors (KNN), and Support Vector Machines (SVM), achieving remarkable results with a true positive rate of over 95 percent and a false positive rate of less than 1 percent. Despite this success, the broader issue of privacy still lurks in the shadows of their investigation. In [7], the authors reviewed HAR with Smartphone and Wearable Sensors using Deep Learning Techniques. All research in this article is carried out just for the feature extraction of Human activity, not focused on privacy issues. Furthermore, despite its potential, existing studies have highlighted the challenges of optimizing Federated Learning (FL) for privacy and efficiency in PEC environments, particularly for applications like HAR [8], [9]. To address these challenges, recent research has begun to explore

A. U. Rehman and R. A. Aldmour are with the Department of Computing, Stoke-on-Trent, Staffordshire University, United Kingdom (email: ateequr.rehman@staffs.ac.uk, rakan.aldmour@staffs.ac.uk)

M. Farooq is with the Department of Information Technology, University of Haripur, KPK, Pakistan (email: mahnoor@uoh.edu.pk)

F. Khan is with the School of Computer Science, Faculty of Science and Engineering, University of Nottingham Ningbo China, Ningbo 315104, Zhejiang, China (email:fazlullah@ieee.org)

G. Srivastava is with the Department of Math and Computer Science, Brandon University, Brandon, R7A 6A9, Manitoba, Canada, and the Research Centre for Interneural Computing, China Medical University, Taichung, 40402, Taiwan as well as the Centre for Research Impact & Outcome, Chitkara University Institute of Engineering and Technology, Chitkara University, Rajpura, 140401, Punjab, India (email: srivastavag@brandonu.ca)

R. Alturki is with the Department of Software Engineering, College of Computing, Umm Al-Qura University, Makkah, Saudi Arabia (email: rmturki@uqu.edu.sa)

B. Alshawi is with the Department of Computer and Network Engineering, College of Computing, Umm Al-Qura University, Makkah, Saudi Arabia (email: bmhshawi@uqu.edu.sa)

Corresponding author: Fazlullah Khan and Gautam Srivastava

the integration of advanced techniques such as encryption, pruning, quantization, and network slicing.

FL has emerged as a promising technique for HAR systems to address these difficulties. FL is a learning model tackling data governance and privacy issues through collaborative training of ML models [10], [11]. These models work across distributed devices such as smartphones or edge devices while keeping data localized and user privacy protected. FL reduces the risks involved with communicating sensitive user data to a centralized server by decentralizing the learning process, hence improving data privacy and security. Particularly, the authors in [12]–[14] highlight the potential of FL in addressing privacy concerns in distributed systems and IoT environments.

Despite significant advancements in Deep Neural Networks (DNNs), optimizing these systems in resource-constrained PEC environments, particularly for FL and HAR, remains challenging. To address these complexities, our research leverages pruning, quantization, and network slicing within private edge computing. In contrast to the proposed combination, techniques such as model compression, efficient network design, and weight sharing have been explored as alternatives. However, model compression can lead to significant loss of accuracy [15], efficient network design often requires substantial manual intervention [16], and weight sharing can introduce unwanted model biases [17]. These drawbacks highlight the necessity and effectiveness of combining pruning, quantization, and network slicing for optimizing FL in PEC environments and offer a more comprehensive solution. Pruning reduces model size, simplifies the model to speed up training, and reduces the need for storage and transmission, which is critical in PEC settings [18], [19]. Quantization optimizes data representation by lowering the bit precision of model parameters, thereby cutting down the memory footprint and computational demands without significantly sacrificing accuracy [19], [20]. Finally, network slicing ensures effective resource management by creating dedicated virtual networks tailored to specific application needs [21]. This ensures efficient resource allocation and minimizes latency. This multilayered strategy not only enhances the efficiency and privacy of FL environments but also addresses multiple challenges concurrently [19], [21], [22].

In contrast to the above, this study proposes an FL technique in the PEC environment to overcome the concerns of privacy, communication latency, and bandwidth usage in HAR systems. It intends to achieve privacy protection, communication efficiency, and computational performance by exploiting the benefits of FL, El-Gamal encryption, and optimization approaches. Our methodology draws inspiration from recent research efforts such as [23], [24] that explore encryption and optimization techniques in FL scenarios. The following is a brief overview of the contributions:

- Data breach is the main problem especially the leakage of sensitive information. Existing studies mostly used ML algorithms for activity recognition, but the ML model training process has vulnerabilities and lacks privacy. To protect sensitive information, we propose the FL integrated with the EL-Gamel technique and analyze sensitive information with high accuracy and high confidentiality.
- The El-Gamal multiplicative encryption is used to achieve a secure aggregation protocol for privacy. This has been analyzed using a mathematical model and proved by Theorem 1. The privacy and security concerns are addressed by implementing homomorphic encryption as a secure aggregation mechanism.
- A PEC architecture is proposed as a solution to reduce the latency issue and increase bandwidth between users and the edge server for instant results. Additionally, pruning, quantization, and network slicing techniques are implemented to optimize the model in terms of network resources, transmission, and overall network efficiency. These are supported by a mathematical model for each technique and proved by Theorem 2.

TABLE I
LITERATURE REVIEW OF HAR RELATED TO PRIVACY AND OPTIMIZATION ISSUES.

Year	Proposed work
2012	The current state of HAR based on wearable sensors was studied focusing on comparing between classifiers [25]
2014	HAR-related sensing technologies such as Depth sensors and RGB cameras are investigated [26]
2017	Proposed channel state information based model for HAR [27]
2019	Proposed SmartSense, a device-free HAR system using commercial WiFi routers, however, failed to investigate optimization and privacy issues [28]
2021	Proposed an efficient Convolutional Neural Network (CNN) architecture for human activity classification in mobile devices but did not consider privacy issues [29].
2023	Reviewed the privacy issues in HAR [13].
2024	Introduced a WiFi signal-based WiLDAR network for HAR which shows promise with its lightweight design and high accuracy. However, it has potential limitations in handling diverse and dynamic real-world environments, which can impact its robustness and generalizability [30]

The subsections that follow present system modeling and analysis, detailing dataset, simulation and results discussions demonstrate the suggested approach's efficiency and effectiveness in the context of HAR systems.

II. SYSTEM MODEL

The proposed model consists of FL with a PEC system. To ensure that training data stays on individual respective devices and to enable ML modes across distributed devices, a decentralized ML approach known as FL is proposed in [31]. In FL, instead of centralizing all data on a server or the cloud, participants perform the learning task and generate the training data locally. We used an extended Artificial Neural Network (ANN) model for training and updating the model because it was the best fit for our algorithm based on our data set. CNN also has a high accuracy rate, but it is best for the images and large data sets, whereas we have numerical data. The SVM also provides the best training for HAR, but because we are using a PEC system, the SVM is unable to perform high computation and consumes a lot of battery power during the training process. In the second step, we use an encrypted technique to secure the model as it is sent to the edge server. The third phase involves transmitting the training weights to the edge server through the pruning and quantization technique

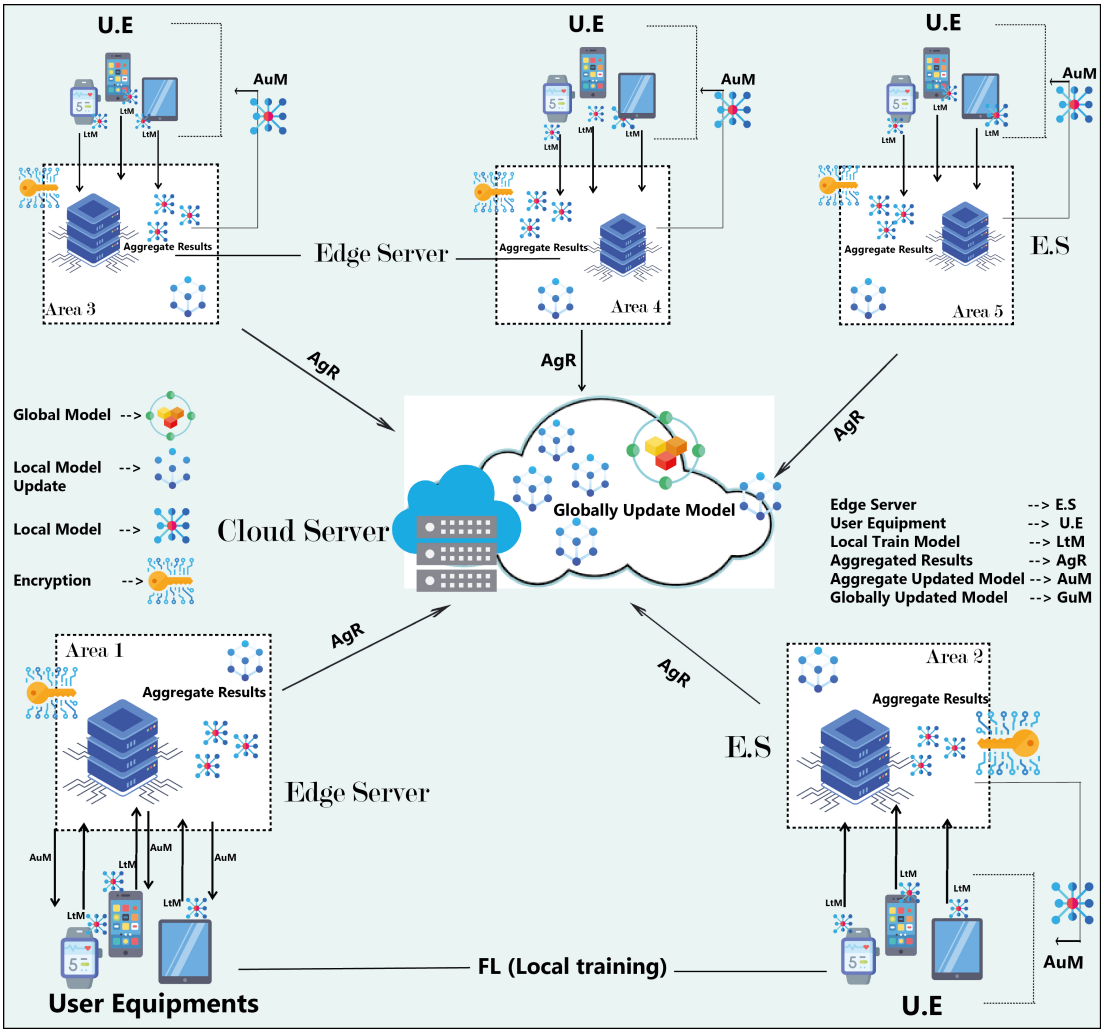

Fig. 1. Proposed Model of the System

which computes the weights after several training loops before aggregating the results and storing them in the cloud server.

A. System Architecture:

To keep the sensitive information of the user participants confidential while training the model, the model is first trained locally. As is well known, ANN outperforms existing ML algorithms by automatically recognizing and learning features from data. As a result, we used the ANN with three layers, the input hidden and output layer. Additionally, the activation functions Softmax, Relu, and Tanh, and the Stochastic Gradient Descent (SGD) is used for optimization. This process aids in passing a weighted and bias-adjusted input value through a non-linear activation function within a feed-forward neural network to generate an output.

The dataset is initially split into the training and test datasets for various clients before training. Then, the training dataset is used as input information for the ANN's weights to be optimized. The learning rate Lr, which determines the gradient descent step size for each iteration, and the partial derivative of the loss function F concerning the weight w are used to calibrate the weights using SGD. The following is the SGD formula:

$$w = w - Lr\frac{\partial F}{\partial w} \tag{1}$$

$$\frac{\partial F}{\partial w} \approx \frac{1}{x}\sum_{i=1}^{x}\frac{\partial F^{(i)}}{\partial w} \tag{2}$$

SGD formula shown in Eq. (1) is a mini-batch SGD. The Eq. (2) is calculated by averaging the gradient matrices from a batch, where each batch consists of a random subset containing x training samples. Since the full batch SGD can result in delayed training and batch memorization, this is favored over the full batch SGD, in which the complete training set is used to compute the partial derivative. Backpropagation is used to create the gradient matrices from the input gradient k as shown in Algorithm 1 [32].

The System Architecture presented in Fig. 1 consists of User Equipment (U.E), Edge Servers (E.S), Cloud Server (CS), Local Train Model (LtM), Aggregated Results (AgR), Aggregate Updated Model (AuM) and Globally Updated Model (GuM). For a clear understanding, the proposed model is organized in phases which are explained below.

1) Phase 1: The FL approach keeps training data locally on the device rather than having to upload and store it on a central server. Many different approaches were suggested to solve various HAR problems. However, preventing the leakage of sensitive data and protecting privacy are difficult concerns. In this approach, we address the privacy concerns of data owners by introducing FL. By enabling users to jointly train a common model while maintaining their data on their devices, FL allays clients' privacy concerns. In Fl, secure aggregation is utilized to keep client changes confidential. We manually split the dataset into sections for multiple clients, trained the model independently, and sent the weights to the edge. The server is unable to determine the significance or final position of any model update provided by the user as a result. Attacks using inference and data attribution are therefore less probable. The working phenomena of the FL model are illustrated in Fig. 1.

Let's break down the process into mathematical steps.
Notations:

N : Total number of user equipment

K : Number of activity classes

D_i : Dataset of the i^{th} device, containing sensor data and labels for HAR.

$D_i = \{(x_{ij}, y_{ij})\}_{j=1}^{m_i}$

where x_{ij} is the sensor data and y_{ij} is the label for the j^{th} sample on the i^{th} user equipment.

a) Local Activity Recognition: Every mobile device i trains a local model w_i with its respective dataset D_i. The training aims to minimize a local loss function $L_i(w)$, which quantifies the discrepancy between the predicted activity probabilities and the true labels.

$$w_i = \arg\min_w \frac{1}{m_i} \sum_{j=1}^{m_i} L(w, x_{ij}, y_{ij}) \tag{3}$$

b) Model Aggregation Result (AgR): The edge server aggregates the local model updates received from the devices to update the global model w. This can be done using a simple averaging method:

$$w_{\text{new}} = \frac{1}{N} \sum_{i=1}^{N} w_i \tag{4}$$

c) Globally Updated Model (GuM): The updated global model w_{new} is then sent back to each device for the next round of training.

Iteration: Steps 2-4 are repeated for multiple rounds, allowing the global model to improve through collaboration with the devices. The local loss function $L_i(w)$ in step 2 depends on the specific algorithm used for HAR. For instance, it could be a softmax cross-entropy loss for multi-class classification:

$$L_i(w) = \frac{1}{m_i} \sum_{j=1}^{m_i} \sum_{k=1}^{K} y_{ij}^k \log(P(w, x_{ij})_k) \tag{5}$$

Where y_{ij}^k is the indicator function representing whether the label y_{ij} corresponds to activity class k given the model w, and input data x_{ij}.

2) Phase 2: El-Gamal homomorphic encryption is implemented for the encryption technique once the model has undergone distributed training. It is possible to think of the well-known El-Gamal cryptosystem as an additively homomorphic version of our approach. El-Gamal is a more dependable and simple-to-use cryptosystem. Let p and q represent the two highest-order elements in M. It should be noted that if p is calculated as q^k, then k is likely to contain elements of order (m), and p is therefore of maximum order. Here, the message space is Z_N. Below is an explanation of the El-Gamal model's mechanism which is presented in the Algorithm 1.

The FL system is composed of a Key Generation Centre (KGC), a Server, and several Participants.

Key Generation:

Let $\alpha \in Z_N^*$ be a random element, and let $a \in \mathbb{Z}_{N^2}$ be a random value. Define $q = \alpha^2 \mod N^2$ and $p = q^a \mod N^2$. The public key is then (N, q, p), and the associated secret key is denoted as k.

Encryption:

For a given message $m \in Z_N$, select a random pad r uniformly from Z_{N^2}. The ciphertext (A, B) is computed as follows:

$$A = q^r \mod N^2 \tag{6}$$

$$B = p^r(1 + mN) \mod N^2 \tag{7}$$

First Decryption Procedure:

With knowledge of the secret key k, the message m can be recovered using the following formula:

$$m = \frac{\frac{B}{A^a} - 1 \mod N^2}{N} \tag{8}$$

Alternate Decryption Procedure:

If the factorization of the modulus N is known, it is possible to compute $k \mod N$ and $r \mod N$ as described previously. Let ar mod ord$(M) = r_1 + r_2 N$, where $r_1 = ar \mod N$ can be efficiently calculated [33]. The decryption is then given by:

$$D = \left(\frac{B}{q^{r_1}}\right)^{\lambda(N)} = \frac{q^{\text{ar}}(1 + mN)}{q^{r_1}\lambda(N)} 1 + m\lambda(N)N \mod N^2 \tag{9}$$

Let π be the inverse of $\lambda(N)$ in Z_N^*. The message m can be recovered by:

$$m = \left(\frac{D - 1 \mod N^2}{N}\right)\pi \mod N \tag{10}$$

It should be noted that, while both decryption procedures produce the same result when decrypting a correctly generated ciphertext, they differ in their computational methods. Specifically, if the discrete logarithm a of h with respect to the base g in Z_N^* is known, it allows for decryption of any ciphertext that was generated using g and h as the public key. This method is also known as exponential El-Gamal.

As discussed above, we use the Exponential El-Gamal encryption method to ensure secure computation. The detailed steps of this process are outlined in Algorithm 1. This method involves initializing values, applying the generator's self-apply method, encrypting the values using the public key, combining

Algorithm 1 Exponential El-Gamal Encryption, Decryption Process and Verification

Require: Messages m_1, m_2, Public Key (p, g), Secret Key k

Choose a random value $r_1, r_2 \in \mathbb{Z}_p$
Apply generator's self-apply method to the first message: $v1 \leftarrow g^{m_1} \mod p$
Apply generator's self-apply method to the second message: $v2 \leftarrow g^{m_2} \mod p$
Encrypt the first value: $A_1 \leftarrow g^{r_1} \mod p$, $B_1 \leftarrow v1^{r_1} \mod p$
Encrypt the second value: $A_2 \leftarrow g^{r_2} \mod p$, $B_2 \leftarrow v2^{r_2} \mod p$
Combine the encrypted values: $C \leftarrow (A_1 \times A_2, B_1 \times B_2) \mod p$
Decrypt the combined result:
$C_1 \leftarrow C[1]^{k^{-1}} \mod p$
$C_2 \leftarrow C[2]^{k^{-1}} \mod p$
Results $\leftarrow (C_1, C_2)$
Verify equality in the message space by checking if $m_1 = m_2$:
Compare $C_1 \times C_2 \mod p$ to see if it matches the original messages' operation in the encrypted space
Output the verification result and decrypted message
End

the encrypted values, decrypting the result with the private key, and verifying equality in the message space.

Similarly, the multiplicative homomorphic property of the standard El-Gamal encryption scheme is utilized in voting systems that employ a re-encryption mix net. In these systems, each mixing authority shuffles and re-encrypts the votes before forwarding them to the next authority. Each vote must be re-encrypted; otherwise, a simple permutation could be used to identify votes by matching ciphertexts [34].

$$E(1) \times E(m) \tag{11}$$

In the ciphertext space, this operation adds 1 to the original vote. This allows the ciphertext to be modified without altering the vote it represents.

$$E(1) \times E(m) = E(1 \times m) = E(m) \tag{12}$$

Using the multiplicative homomorphic feature, the vote remains unaltered. When utilizing El-Gamal encryption, we recently demonstrated how multiplication operations can be performed in the encrypted ciphertext space. With a slight modification, it turns out that you can enable addition instead of multiplication.

$$E : Z_q \rightarrow G_q \times G_q \tag{13}$$

As before, let the public key h, generator g, message m, and randomness r then we have a different encryption function:

$$E(m) = (g^r, g^m \times h^r) \qquad (9) \tag{14}$$

When encrypting a message m, it is first transformed into g^m before proceeding with the encryption. Essentially, the system operates in the same manner as before, but the message is now represented as g^m instead of m.

3) Phase 3: In the next step, we apply the PEC system, after which the model is trained and its encrypted weights are forwarded to the edge server through the pruning and quantization technique. This technique computes the weights after several training loops before aggregating the results and storing them in the cloud server. It decreases our latency and bandwidth issues. We implement the model with the help of FL and PEC to assist data owners in securing their data while training the model and applying encryption and decryption techniques while uploading the highlight weights on the edge server and resolving bandwidth and latency issues. With the suggested approach, we may protect user data while training the model, implement PEC to minimize core network traffic load, relieve processing stress on the central server, reduce response time and latency of end-to-end activities, and improve wireless network performance overall.

a. **Pruning**

Pruning involves introducing a sparsity constraint to encourage certain weights or connections to become zero or be removed entirely. Mathematically, this can be formulated as; *Loss (W) denotes the loss function, λ is a hyper-parameter that controls the importance of the sparsity constraint, and $\|W\|_0$ denotes the L0 norm or the number of non-zero weights in the network. The prime aim is to reduce the loss function while encouraging sparsity in the weights. The sparsity constraints can be implemented using various techniques such as L1 regularization, group sparsity, or thresholding.*

Pruning can lessen the number of weights in a model by up to 90% with little or no loss in accuracy, depending on the specific architecture and dataset. Pruning is particularly effective in this context, as it reduces the model's weight count, leading to a significant decrease in data transfer between mobile devices and the edge server. This can contribute to lower latency and improved system performance. Moreover, pruning can be done locally on the client devices, which can reduce the communication overhead required for transmitting the weights to the edge server.

b. **Quantization**

Quantization is the process of lowering the precision of weights and activations in a neural network. Mathematically, this can be represented as:

Minimize: Loss(Q(W))

Here, Loss(Q(W)) denotes the loss function of the quantized neural network, which may incorporate additional terms to account for the quantization error. The aim is to minimize the loss function while adapting the weights to fit the quantization levels. Quantization can be implemented by mapping the continuous weights to a finite set of discrete levels. This can be expressed as;

$$Q(W) = Q(W_{\text{unquantized}}) \tag{15}$$

Here, Q(W) represents the quantized weights, and $Q(W_{\text{unquantized}})$ denotes the quantization function applied to the unquantized weights. The quantization function can be defined using various methods such as uniform quantization, where weights are mapped to a fixed number of levels, or

non-uniform quantization, where different levels have different probabilities of occurrence. The quantization error can be incorporated into the loss function to guide the optimization process. Quantization can also be useful because it reduces the precision of the weights, which can lead to a reduction in the size of the model. This can help to reduce memory usage and accelerate inference time on mobile devices, which is particularly important in the resource-constrained environment of PEC.

Notations:

N : Total number of mobile devices
K : Number of activity classes
D_i : Dataset of the i^{th} device, $D_i = \{(x_{ij}, y_{ij})\}_{j=1}^{m_i}$
w : Global model's parameters
w_i : Local model's parameters on the i^{th} device

Local Activity Recognition, Pruning, and Quantization: Each mobile device i trains a local model w_i using its local dataset D_i. Model pruning is applied to the local model w_i to obtain a pruned local model w_i':

$$w_i' = \text{Prune}(w_i). \tag{16}$$

Additionally, model quantization is applied to the pruned local model w_i' to further reduce the precision of its parameters:

$$w_i'' = \text{Quantize}(w_i'). \tag{17}$$

Model Aggregation with Pruned and Quantized Models: The edge server aggregates the pruned and quantized local model updates received from the devices to update the global model w:

$$w_{\text{new}} = \frac{1}{N} \sum_{i=1}^{N} w_i'' \tag{18}$$

Global Model Pruning and Quantization: The edge server can further prune and quantize the global model w_{new} to ensure its size is controlled and it remains lightweight:

$$w_{\text{pruned\_quantized}} = \text{Quantize}(\text{Prune}(w_{\text{new}})) \tag{19}$$

Global Model Update and Iteration: The pruned and quantized global model $w_{\text{pruned\_quantized}}$ is sent back to each device for the next round of training, pruning, and quantization.

c. **Network Slicing**

Network slicing in PEC can be represented mathematically using various notations and formulations.

- **Network Resource Allocation:** Network slicing involves the allocation of network resources to different slices. This can be represented as an optimization problem, where the goal is to allocate resources to maximize the utility or satisfaction of each slice, subject to resource constraints. Mathematically, it can be expressed as:

$$\text{Maximize: } \sum \text{U}(\text{slice}_i) \tag{20}$$

$$\text{Subject to: } \sum \text{Resource}(\text{slice}_i) \leq \text{Total\_Resource} \tag{21}$$

Here, $\text{U}(\text{slice}_i)$ represents the utility or satisfaction function for each network slice, and $\text{Resource}(\text{slice}_i)$ denotes the resource allocation for slice_i. The objective is to maximize the total utility of all slices while ensuring the total allocated resources do not exceed the available total resources.

- **Quality of Services (QoS) Constraints:** Network slicing involves providing specific Quality of Service (QoS) guarantees to different slices. This can be represented by QoS constraints that ensure certain performance metrics are met for each slice. Mathematically, it can be expressed as:

$$\text{Constraint: QoS}(\text{slice}_i) \geq \text{QoS\_Threshold} \tag{22}$$

Here, $\text{QoS}(\text{slice}_i)$ represents the QoS metric, such as latency, throughput, or reliability, for each network slice, and QoS_Threshold denotes the minimum required QoS level for the slice. The constraint ensures that the QoS of each slice meets or exceeds the specified threshold.

- **Slicing Configuration:** Network slicing also involves defining the configuration and parameters of each slice, such as network functions, security policies, or service-level agreements. This can be represented as a configuration specification, specifying the desired characteristics for each slice. Mathematically, it can be expressed as a set of constraints and specifications for each slice. Slice_i Configuration = {NF1, NF2, Security_Policy, SLA}. Here, NF1 and NF2 represent the required network functions for slice_i, Security_Policy represents the desired security measures, and SLA denotes the service-level agreement or specific requirements for the slice. This specification ensures that each slice is configured according to the desired parameters. Because of these devices' increasing processing capacity and concerns about sending private data, keeping data locally and shifting network computing to the edge is becoming increasingly desirable. Sensor-based HAR is the problem of identifying different types of human activities based on sensor data from smart devices. Having a significant impact on ubiquitous and professional computing. To the best of my knowledge, limited study has been focused on privacy issues in HAR that employs this technology. This is primarily due to its complexity, as well as a lack of technological expertise or IT competence.

All previous studies described simple ML models with other techniques or simply implemented the FL technique, which was not fully executed as well, whereas our research offers a fully implemented work.

The layout structure of the connection between an edge server and a client is depicted in the above figure. By sending a default "Echo Msg" to the edge server, the client first establishes a connection with it. If a connection cannot be made within 30 seconds, the session expires, and the client must resend the echo message. When a connection is established successfully, the client requests the model using the pre-set message "Listen." The model is provided to the client and the session is terminated. Once the model is updated, the session is initiated, and the weights are transmitted to the edge server.

The default "Done" message was given to the client and the connection was terminated when the revised model's specified training rate was achieved. Our system will be evaluated with the help of the accuracy rate we obtain as well as the latency rate and bandwidth comparison between the PEC system vs. the cloud server system.

In practice, both pruning and quantization can be used together to further reduce the size of the model. For example, the model can be pruned first to reduce the number of weights, and then the remaining weights can be quantized to reduce their precision. This can lead to a very compact model that can be efficiently transmitted between the client devices and the edge server.

III. Theorems for Privacy and Optimization

Theorem 1. *Given the proposed FL framework with ElGamal encryption, the privacy of individual participant data is preserved under differential privacy constraints.*

Proof. Each client generates a public-private key pair. The public key is exchanged with the edge server, while the private key remains secure on the client device. Each client encrypts the model updates using the edge server's public key. This ensures only the edge server can decrypt the updates with its private key. The edge server decrypts the updates, aggregates them, and re-encrypts the global model updates with each client's public key before dissemination. We use the Gaussian mechanism to add noise $N(0, \sigma^2)$ to the model gradients. The noise scale σ is chosen based on the desired privacy parameters (ϵ, δ) and the sensitivity Δf of the function.

$$\sigma = \frac{\Delta f \sqrt{2 \ln \left(\frac{1.25}{\delta}\right)}}{\epsilon} \tag{23}$$

For each communication round, the Gaussian mechanism ensures the privacy loss is bounded by (ϵ, δ). Over multiple communication rounds, the overall privacy loss is:

$$\epsilon_{\text{total}} = \sqrt{T} \cdot \epsilon \tag{24}$$

The mutual information $I(X;Y)$ measures information leakage.

$$I(X;Y) = H(X) - H(X|Y) \tag{25}$$

Thus,

$$\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(D') \in S] + \delta \tag{26}$$

$\Pr[M(D) \in S]$ is the probability that the mechanism M applied to dataset D results in an output in the set S.

The model ensures that differential privacy is maintained by adding Gaussian noise to the model gradients. The noise level σ is calculated to achieve the desired privacy parameters ϵ and δ. Over multiple rounds of communication, the privacy loss is controlled and the information leakage is minimized. □

Theorem 2. *The proposed FL framework with pruning and quantization techniques achieves significant model optimization while maintaining model accuracy.*

Proof. Introducing sparsity constraints encourages certain weights to become zero or be removed. Loss function with sparsity constraint may be formulated as,

$$\text{Loss}(W) + \lambda \|W\|_0 \tag{27}$$

This reduces the model size by up to 90% without significant accuracy loss. To reduce the precision of weights and activations. Quantized loss function may be expressed as,

$$\text{Minimize} : \text{Loss}(Q(W)) \tag{28}$$

Mapping continuous weights to a finite set of discrete levels:

$$Q(W) = Q(W_{\text{unquantized}}) \tag{29}$$

This reduces the model size and accelerates inference time. To aggregate pruned and quantized models at the edge server we have,

$$w_{\text{new}} = \frac{1}{N} \sum_{i=1}^{N} w_i'' \tag{30}$$

This results in updating the global model iteratively, with further pruning and quantization ensuring size control and lightweight characteristics which is mathematically expressed as,

$$w_{\text{pruned\_quantized}} = \text{Quantize}(\text{Prune}(w_{\text{new}})) \tag{31}$$

The model optimization is achieved through pruning, which introduces sparsity constraints to reduce model size, and quantization, which reduces the precision of weights and activations. These techniques are applied iteratively, ensuring that the global model remains efficient and lightweight while maintaining accuracy. □

IV. Data Set

The dataset was obtained from the UCI machine learning repository [35] and is available on the Kaggle website as well [36]. The collection includes data from 30 volunteers who were between the ages of 19 and 48. The dataset therefore has six labels to predict. The dataset consists of 561 feature vectors with the primary variables being frequency and time. There are no missing or duplicate values in the dataset. Each participant completes six tasks: Lying, Sitting, Walking, Walking upstairs, Walking downstairs, and Standing. These attributes result from the following information:

- **Acceleration gravity** along the x, y, z axes.
- **Body gyroscope** data files for the x, y, z axes.
- as well as **body acceleration** data files for x, y, z axes.

Accelerometer data from smartphones was used to generate mean features for two types of human activity: static (sitting, standing, laying) and dynamic (walking, walking downstairs, going upstairs). The distributions of these features were then analyzed. The two categories of activity can certainly be distinguished from one another. For every record, we also have the volunteer's unique identification number. There are 10299 records altogether, and the training and test data sets are split 73.3 per to 26.6 per. The six classes are then translated to numbers in the following order: [1, 2, 3, 4, 5, 6].

V. Results and Analysis

In our study, we evaluated the proposed FL framework for HAR within PEC environments using both synthetic and real-world datasets. The edge computing environment consisted of ARM Cortex-A53 edge devices, Intel Xeon E5-2670 edge servers, and an Intel Xeon E5-2690 cloud server. Key parameters included an initial learning rate of 0.01 (decayed to 0.001), a batch size of 64, 300 epochs, and a noise scale of 0.5 for differential privacy. The model performance was assessed based on accuracy, precision, recall, F1-score, latency, and throughput, with additional privacy guarantees evaluated using differential privacy metrics. Detailed hardware and software configurations, along with parameter tuning processes, were also considered to optimize the performance of our framework. Furthermore, we employed the Tensor-Flow Federated library and Socket library for the connection between edge server and clients, further used iFogsim for creating the model, and then created an IPs dataset and find out its latency for our endeavors and managed to capture the adequate results, describing the success of our research. The edge server and a client create a connection by sending a default "Echo Msg" to the edge server, the client first establishes a connection with it. If a connection cannot be made within 30 seconds, the session expires, and the client must resend the echo message. When a connection is established successfully, the client requests the model using the pre-set message "Listen." The model is provided to the client and the session is terminated. Once the model is updated, the session is initiated, and the weights are transmitted to the edge server. The default "Done" message was given to the client and the connection was terminated when the revised model's specified training rate was achieved.

1) Accuracy and Loss: FL allows for training without exposing sensitive raw data to a central server. We indicated that FL for HAR is sufficiently robust under a wide range of workloads and produces models with comparable accuracy to centralized learning. FL achieves up to 98.6 % accuracy in our studies on synthetic and real-world datasets. Our proposed

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 16)                9008
_________________________________________________________________
dense_27 (Dense)             (None, 32)                544
_________________________________________________________________
dense_28 (Dense)             (None, 6)                 198
=================================================================
Total params: 9,750
Trainable params: 9,750
Non-trainable params: 0
_________________________________________________________________
69/69 [==============================] - 0s 2ms/step - loss: 0.0332 - accuracy: 0.9869
Out[37]: [0.033178556710481644, 0.9868540167808533]
```

Fig. 2. Accuracy rate of the Extended ANN Model

model is evaluated with the help of accuracy rate, latency rate, and bandwidth comparison between the PEC system vs the cloud server system.

2) Comparison of Extended ANN with SVM: The comparative evaluation between the ANN and SVM models revealed notable distinctions in performance within the context of HAR conducted in the PEC environment. Upon rigorous experimentation and optimization, the ANN model showcased a remarkable accuracy of 99% at 300 epochs as illustrated

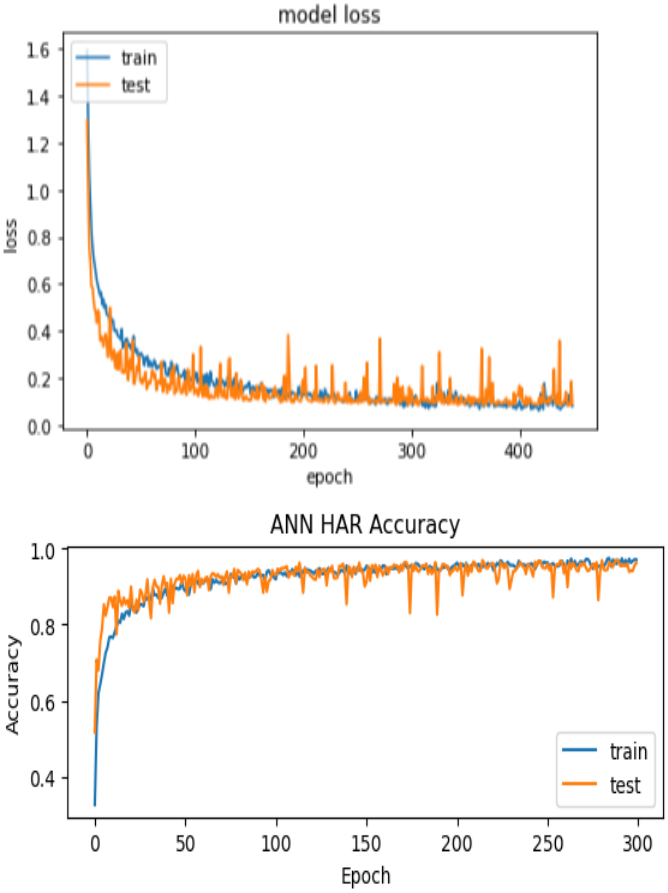

Fig. 3. Graphical representation of the Accuracy and Loss rate

Model	DataSet	Accuracy	Precision	Recall	F1-Score
ANN	Synthetic	99%	98.8%	99.2%	99%
ANN	Real-world	98.6%	98.5%	98.7%	98.6%
SVM	Synthetic	97.5%	97.3%	97.6%	97.4%
SVM	Real-world	97.2%	97.1%	97.3%	97.2%

TABLE II
Model Performance Metrics

in Table II. Leveraging an extended ANN architecture with an additional dropout layer notably enhanced the model's performance, demonstrating its capability to discern complex patterns within the distributed client data.

In contrast, the SVM model, while commendable, displayed a slightly lower accuracy of 97.5% at 300 epochs within the same experimental conditions. The SVM model effectively discerned patterns but was marginally outperformed by the extended ANN architecture in terms of accuracy. This observation substantiates the superiority of the extended ANN over the SVM in achieving higher accuracy for HAR tasks within the secure and privacy-preserving framework of PEC.

3) Precision: Precision is the measure of the accuracy of positive predictions. It quantifies how many of the instances predicted as positive are positive. For the class "Walking," let's define: True Positives (TP) : Instances correctly classified as "Walking"

False Positives (FP): Instances incorrectly classified as "Walking" (when they are not)

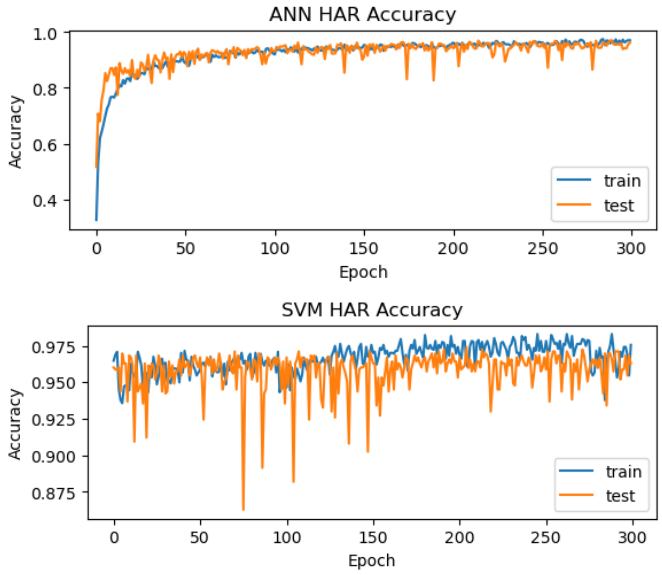

Fig. 4. ANN vs SVM HAR Model Accuracy

The precision formula for the ”Walking” class is:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$$

4) Recall (Sensitivity): Recall, also known as sensitivity or true positive rate, is the measure of how well the model captures all positive instances. It quantifies how well the model identifies positive instances out of all actual positive instances. For the class ”Walking,” let’s define:

True Positives (TP): Instances correctly classified as ”Walking”

False Negatives (FN): Instances incorrectly classified as not ”Walking” (when they are ”Walking”)

The recall formula for the ”Walking” class is:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

These metrics help you comprehensively evaluate your HAR model’s performance for different activity classes, accounting for both accuracy and completeness.

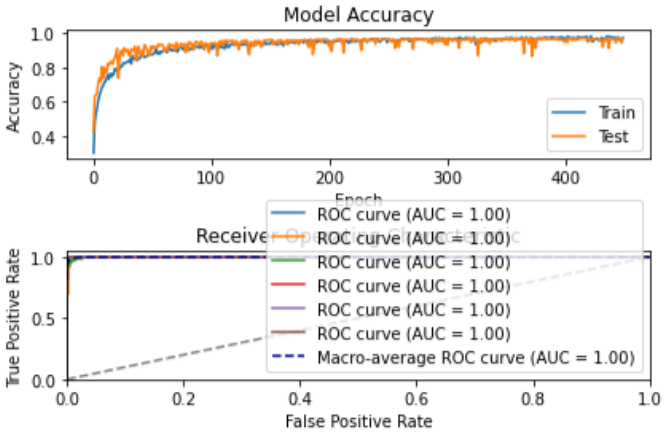

Fig. 5. Recall Graph ROC AUC Curve

5) Precision-Recall Curve in the Model: The Precision-Recall curve illustrates the trade-off between precision and recall in a classification model. HAR with PEC and FL, Precision-Recall helps balance the accuracy and completeness of activity predictions.

Precision measures the accuracy of positive predictions, while **Recall** quantifies the model’s ability to capture actual positive instances. The curve is created by adjusting the prediction threshold and plotting precision against recall. It starts with high recall and low precision (many predicted positives, some incorrect), and as the threshold increases, precision rises and recall decreases. The optimal point on the curve depends on the application’s needs, finding the threshold that achieves the right balance between precision and recall.

6) Confusion Matrix Results: Each row of the confusion matrix represents the actual classes or ground truth activities. For example, in HAR, these show the ”Walking,” ”Running,” and ”Sitting.” Each column of the confusion matrix represents the classes predicted by the model for the given samples. These are the classes that the model assigns to the data. Moreover, the cells along the diagonal of the confusion matrix represent cases where the actual class and the predicted class are the same. These are the correct predictions, often referred to as true positives. A higher value on the diagonal indicates accurate predictions. The cells off the diagonal represent misclassified samples. These are cases where the actual class is different from the predicted class. These misclassifications are divided into false positives (predicting a class that isn’t the actual class) and false negatives (not predicting the correct class). The sum of each row represents the total number of samples for that actual class. The sum of each column represents the total number of samples predicted as that class. These sums provide insights into the distribution of your dataset and the predictions.

7) Connection Built with Server: The connection between an edge server and a client is established by sending a default “Echo Msg” to the edge server. The client first establishes a connection with it. If a connection cannot be made within 30 seconds, the session expires, and the client must resend the echo message.

8) Latency of Cloud vs Edge Results: The key outcome of this study is the comparison of latency under various conditions, including both uploading and connected scenarios, with an expanded measurement of unloaded latency. It is crucial to evaluate the expected latency differences when preferring edge servers over cloud servers. Latency measurements were taken from clients to both edge servers and cloud servers.

As shown in Fig. 6, the latency measurements indicate that 62% of clients can reach a nearby edge server in less than 10 ms, while only 25% of clients can achieve a similar latency when connecting to a nearby cloud location. Furthermore, the latency distribution for connections to edge servers is entirely below 70 ms. In contrast, the latency distribution for connections to cloud servers shows that only 25% of clients experience latency below 10 ms, 50% experience latency below 100 ms, and 35% experience latency ranging from 200 to 800 ms. This disparity underscores the importance of

deploying edge servers to achieve lower latency and better load balancing.

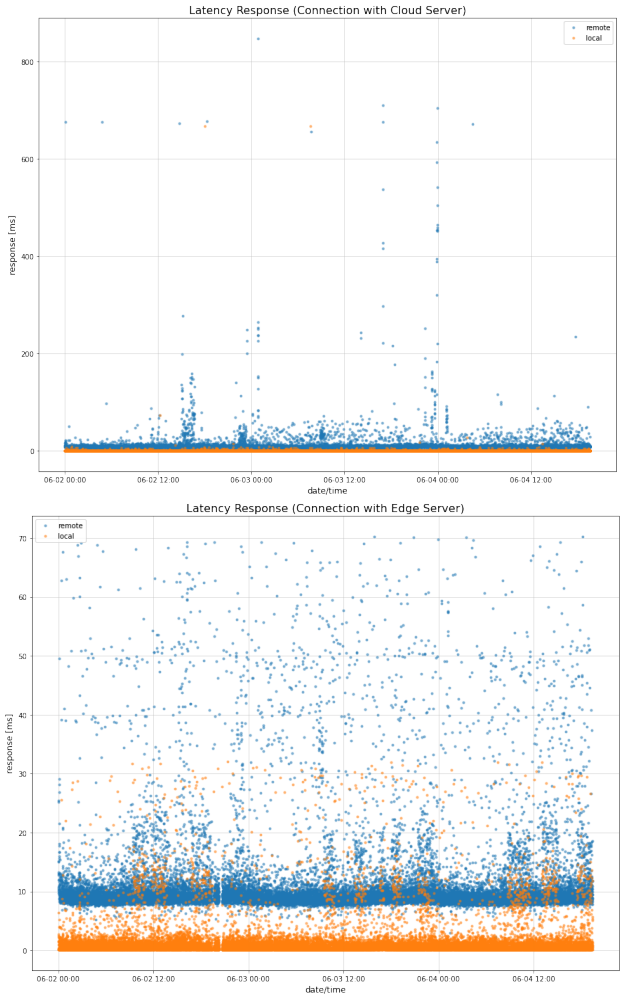

Fig. 6. Latency of Cloud & Edge Server with Client Connection, where blue color represents Remote Network while orange represents Local network.

The latency graphs in Fig. 6 clearly demonstrate that the edge server provides significantly lower latency compared to the cloud server. Specifically, the edge server setup shows a more uniform and lower latency distribution, with 62% of clients experiencing latency below 10 ms. In contrast, the cloud server setup reveals higher variability and longer latencies, with only 25% of clients achieving latency below 10 ms. These findings highlight the advantage of edge server deployment in reducing latency and improving response times in networked applications.

9) Edge vs Cloud Server Throughput: **Edge Server Throughput:** For the edge server:

- Data Transfer Time: D_{edge} (data size), $T_{\text{comm\_edge}}$ (communication time)
- Processing Time on Edge Server: $T_{\text{process\_edge}}$ (aggregation, updates)

Total time for a round of FL using edge server:

$$T_{\text{edge}} = T_{\text{comm\_edge}} + T_{\text{process\_edge}}$$

Cloud Server Throughput: For the cloud server:

- Data Transfer Time: D_{cloud} (data size), $T_{\text{comm\_cloud}}$ (communication time)
- Processing Time on Cloud Server: $T_{\text{process\_cloud}}$ (aggregation, updates)

Total time for a round of FL using cloud server:

$$T_{\text{cloud}} = T_{\text{comm\_cloud}} + T_{\text{process\_cloud}}$$

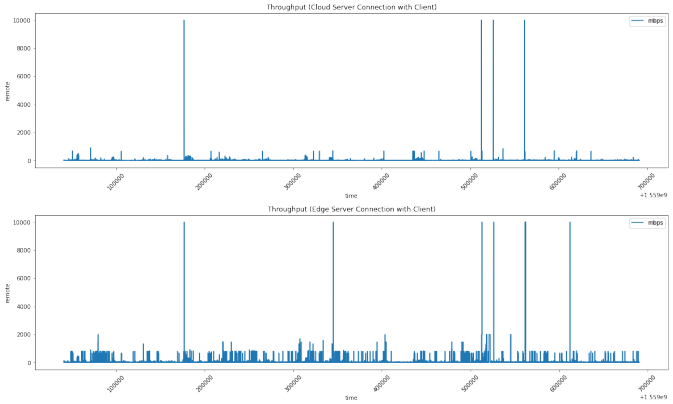

Fig. 7. Throughput of Cloud & Edge Server with Client Connection where x-axis represents time and y-axis represents throughput in mbps.

As shown in Fig. 7, the edge server consistently achieves higher throughput compared to the cloud server. This indicates more efficient data processing and communication in edge computing environments.

Comparison: Throughput for edge server:

$$\text{Throughput}_{\text{edge}} = \frac{1}{T_{\text{edge}}}$$

Throughput for cloud server:

$$\text{Throughput}_{\text{cloud}} = \frac{1}{T_{\text{cloud}}}$$

Hence, higher throughput indicates faster model updates and potentially better convergence in the edge server scenario.

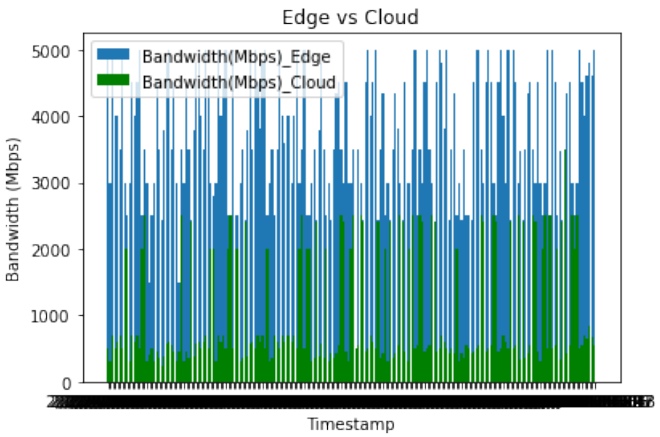

Fig. 8. Bandwidth Comparison Between Edge and Cloud Server

Fig. 8 illustrates that the edge server provides significantly higher bandwidth compared to the cloud server, demonstrating

the edge server's capability to handle larger data transfers more efficiently.

FL can reliably train and update models using edge servers. Real-time decisions, such as detection, localization, and activity recognition, can be made locally at edge nodes or end devices within the PEC (Processing at Edge Computing) framework. As a result, the latency is significantly shorter compared to making decisions in the cloud and then sending them to end devices. This is crucial for time-sensitive applications like health systems, surveillance, or other tracking systems where even the smallest delays could endanger lives. Figures 7 and 8 clearly show the throughput and bandwidth differences between edge server and cloud server connections.

VI. Conclusions

This research introduces a novel approach for FL in HAR within the context of PEC. The proposed methodology seamlessly integrates FL, El-Gamal encryption for privacy protection, and optimization techniques including pruning, quantization, and network slicing. Results and simulations corroborate the efficacy of this approach in mitigating challenges associated with privacy preservation, communication latency, and bandwidth consumption in HAR systems. The comparative evaluation between the ANN and SVM models unveils significant performance distinctions in the PEC environment. Rigorous experimentation and optimization revealed that the ANN model achieved a remarkable accuracy of 99% at 300 epochs. Leveraging an extended ANN architecture with an additional dropout layer notably enhanced the model's performance, showcasing its adeptness in discerning complex patterns within distributed client data. Contrasting, the SVM model, while commendable, demonstrated a slightly lower accuracy of 97.5% at 300 epochs under identical experimental conditions. Although proficient in pattern discernment, the SVM model was marginally outperformed by the extended ANN architecture in terms of accuracy. This observation substantiates the superiority of the ANN over the SVM in achieving higher accuracy for HAR tasks within the secure and privacy-preserving framework of PEC. The devised methodology, encompassing FL, HAR, El-Gamal encryption, pruning, quantization, and network slicing, facilitated the optimization of the ANN model. This optimization resulted in superior accuracy while ensuring data privacy and addressing challenges related to communication delay and bandwidth constraints. The resultant solution not only strikes a balance between accuracy and communication overhead but also stands ready for practical deployment in real-world applications requiring secure and private-edge environments. In the future, this study might incorporate enhanced encryption methods or privacy-preserving techniques, to maximize privacy guarantees while minimizing computational expense.

Acknowledgment

This work was supported by the Ningbo Municipal Bureau of Science and Technology under Grant 2023J194.

References

[1] J. Wu, W. Gao, Z. Zheng, D. Zhao, Y. Zeng, Study of human activity intensity from 2015 to 2020 based on remote sensing in anhui province, china, Remote Sensing 15 (8) (2023) 2029.

[2] M. A. Jan, W. Zhang, F. Khan, S. Abbas, R. Khan, Lightweight and smart data fusion approaches for wearable devices of the internet of medical things, Information Fusion 103 (2024) 102076.

[3] F. Khan, M. A. Jan, R. Alturki, M. D. Alshehri, S. T. Shah, A. ur Rehman, A secure ensemble learning-based fog-cloud approach for cyberattack detection in iomt, IEEE Transactions on Industrial Informatics (2023).

[4] B. Jin, F. Khan, R. Alturki, M. A. Ikram, Computational intelligence-enabled prediction and communication mechanism for iot-based autonomous systems, ISA transactions 132 (2023) 146–154.

[5] D. Singh, E. Merdivan, I. Psychoula, J. Kropf, S. Hanke, M. Geist, A. Holzinger, Human activity recognition using recurrent neural networks, CoRR abs/1804.07144 (2018). arXiv:1804.07144.
URL http://arxiv.org/abs/1804.07144

[6] A. Anjum, M. Ilyas, Activity recognition using smartphone sensors, 2013, pp. 914–919. doi:10.1109/CCNC.2013.6488584.

[7] R. Elangovan, T. Perumal, s. Padmavathi, Human activity recognition with smartphone and wearable sensors using deep learning techniques: A review, IEEE Sensors Journal PP (2021) 1–1. doi:10.1109/JSEN.2021.3069927.

[8] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (2) (2019) 1–19.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.

[10] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data (2023). arXiv:1602.05629.

[11] N. Waheed, A. Ur Rehman, A. Nehra, M. Farooq, N. Tariq, M. A. Jan, F. Khan, A. Z. Alalmaie, P. Nanda, Fedblockhealth: A synergistic approach to privacy and security in iot-enabled healthcare through federated learning and blockchain, in: GLOBECOM 2023 - 2023 IEEE Global Communications Conference, 2023, pp. 3855–3860. doi:10.1109/GLOBECOM54140.2023.10437356.

[12] K. A. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, J. Roselander, Towards federated learning at scale: System design, CoRR abs/1902.01046 (2019). arXiv:1902.01046.
URL http://arxiv.org/abs/1902.01046

[13] S. Kalabakov, B. Jovanovski, D. Denkovski, V. Rakovic, B. Pfitzner, O. Konak, B. Arnrich, H. Gjoreski, Federated learning for activity recognition: A system level perspective, IEEE Access 11 (2023) 64442–64457. doi:10.1109/ACCESS.2023.3289220.

[14] L. Zhang, W. Cui, B. Li, Z. Chen, M. Wu, T. S. Gee, Privacy-preserving cross-environment human activity recognition, IEEE Transactions on Cybernetics 53 (3) (2023) 1765–1775. doi:10.1109/TCYB.2021.3126831.

[15] Y. Cheng, D. Wang, P. Zhou, T. Zhang, A survey of model compression and acceleration for deep neural networks, arXiv preprint arXiv:1710.09282 (2017).

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861 (2017).

[17] Y. Wang, C. Xu, C. Xu, D. Tao, Adversarial learning of portable student networks, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 32, 2018.

[18] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, Advances in neural information processing systems 28 (2015).

[19] P. Prakash, J. Ding, R. Chen, X. Qin, M. Shu, Q. Cui, Y. Guo, M. Pan, Iot device friendly and communication-efficient federated learning via joint model pruning and quantization, IEEE Internet of Things Journal 9 (15) (2022) 13638–13650. doi:10.1109/JIOT.2022.3145865.

[20] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704–2713.

[21] S. D. A. Shah, M. A. Gregory, S. Li, Toward network-slicing-enabled edge computing: A cloud-native approach for slice mobility, IEEE Internet of Things Journal 11 (2) (2024) 2684–2700. doi:10.1109/JIOT.2023.3292520.
[22] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, S. Wright, Atomo: Communication-efficient learning via atomic sparsification, Advances in neural information processing systems 31 (2018).
[23] C. T. Dinh, N. H. Tran, M. N. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, V. Gramoli, Federated learning over wireless networks: Convergence analysis and resource allocation, IEEE/ACM Transactions on Networking 29 (1) (2020) 398–409.
[24] R. Chen, C. Huang, X. Qin, N. Ma, M. Pan, X. Shen, Energy efficient and differentially private federated learning via a piggyback approach, IEEE Transactions on Mobile Computing (2023).
[25] O. D. Lara, M. A. Labrador, A survey on human activity recognition using wearable sensors, IEEE communications surveys & tutorials 15 (3) (2012) 1192–1209.
[26] O. C. Ann, L. B. Theng, Human activity recognition: A review, in: 2014 IEEE international conference on control system, computing and engineering (ICCSCE 2014), IEEE, 2014, pp. 389–393.
[27] W. Wang, A. X. Liu, M. Shahzad, K. Ling, S. Lu, Device-free human activity recognition using commercial wifi devices, IEEE Journal on Selected Areas in Communications 35 (5) (2017) 1118–1131. doi:10.1109/JSAC.2017.2679658.
[28] H. Zou, Y. Zhou, R. Arghandeh, C. J. Spanos, Multiple kernel semi-representation learning with its application to device-free human activity recognition, IEEE Internet of Things Journal 6 (5) (2019) 7670–7680. doi:10.1109/JIOT.2019.2901927.
[29] J. Zhu, X. Lou, W. Ye, Lightweight deep learning model in mobile-edge computing for radar-based human activity recognition, IEEE Internet of Things Journal 8 (15) (2021) 12350–12359. doi:10.1109/JIOT.2021.3063504.
[30] F. Deng, E. Jovanov, H. Song, W. Shi, Y. Zhang, W. Xu, Wildar: Wifi signal-based lightweight deep learning model for human activity recognition, IEEE Internet of Things Journal 11 (2) (2024) 2899–2908. doi:10.1109/JIOT.2023.3294004.
[31] K. Yang, T. Fan, T. Chen, Y. Shi, Q. Yang, A quasi-newton method based vertical federated learning framework for logistic regression, arXiv preprint arXiv:1912.00513 (2019).
[32] S. Ek, F. Portet, P. Lalanda, G. Vega, Evaluation of federated learning aggregation algorithms: application to human activity recognition, in: Adjunct proceedings of the 2020 ACM international joint conference on pervasive and ubiquitous computing and proceedings of the 2020 ACM international symposium on wearable computers, 2020, pp. 638–643.
[33] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, M. Chen, In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning, Ieee Network 33 (5) (2019) 156–165.
[34] C. Fang, Y. Guo, Y. Hu, B. Ma, L. Feng, A. Yin, Privacy-preserving and communication-efficient federated learning in internet of things, Computers & Security 103 (2021) 102199. doi:https://doi.org/10.1016/j.cose.2021.102199.
URL https://www.sciencedirect.com/science/article/pii/S0167404821000237
[35] J. A. Reyes-Ortiz, G. A. Davide, L. Oneto, X. Parra, Human Activity Recognition Using Smartphones, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C54S4K (2012).
[36] G. Lin, W. Jiang, S. Xu, X. Zhou, X. Guo, Y. Zhu, X. He, Human activity recognition using smartphones with wifi signals, IEEE Transactions on Human-Machine Systems 53 (1) (2022) 142–153.