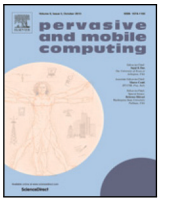
TCP-HAR: On-Device Transferable and Copyright-Preserving Human Activity Recognition

Pervasive and Mobile Computing 117 (2026) 102163

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc



TCP-HAR: On-Device Transferable and Copyright-Preserving Human Activity Recognition

Alessio Sacco [a,*], Bruno Palermo [b], Giulio Figliolino [a], Chiara Contoli [c], Guido Marchetto [a], Flavio Esposito [d]

[a] DAUIN, Politecnico di Torino, Turin, Italy
[b] Amazon, Dublin, Ireland
[c] Department of Pure and Applied Sciences, University of Urbino, Urbino, Italy
[d] Department of Computer Science, Saint Louis University, Saint Louis, USA

ARTICLE INFO

Keywords:
Human activity recognition
Federated learning
Android smartphones
Watermarking

ABSTRACT

Teaching a machine to accurately identify human activities from sensor data poses a significant challenge, which is further compounded by considerations of data privacy, resource costs, and responsiveness, particularly within the constraints of devices like smartphones. While current solutions efficiently identify activities, trained models are barely portable in scenarios composed of diverse activities and limited battery life devices, such as smartphones. This paper introduces Transferable and Copyright-Preserving Human Activity Recognition (TCP-HAR), a mobile-based HAR system that integrates digital watermarking, Federated Learning (FL), Transfer Learning (TL), and compression techniques to provide efficient human activity recognition while providing copyright protection of deep neural network models over Android smartphones. Our solution optimizes the utilization of FL, TL, and their combination (FTL) by extensively testing standalone TL models in offline contexts and comparing these results with FL across a network of mobile devices. Our findings highlight the benefits of TCP-HAR for mobile environments in terms of accuracy, F1-score, and training time. In addition, our proposed watermarking mechanism is robust yet computationally efficient, ensuring ownership verification without compromising the scalability of the TFL process.

1. Introduction

Human Activity Recognition (HAR) on smartphones enables pervasive health monitoring, fitness tracking, and context-aware applications. However, deploying robust HAR models on smartphones remains challenging due to three key limitations: (1) ensuring user data privacy, (2) operating within strict energy and computational constraints, and (3) preserving ownership rights over trained models. Over the past decades, HAR, coupled with the growing penetration of smartphones, has gained significant attention in fields such as healthcare monitoring [1–3], surveillance [4], sport and fitness tracking [5,6], and smart environments [7]. Since the initial studies conducted in 2008, advancements in storage and processing technologies have enabled the direct collection of data from smartphones [8]. Machine learning (ML) and deep learning (DL) approaches have proved to provide reliable results in the recognition process [9], but these approaches share two main concerns, namely privacy [10] and energy (or more generally resource) consumption [11].

* Corresponding author.
E-mail address: alessio_sacco@polito.it (A. Sacco).

https://doi.org/10.1016/j.pmcj.2026.102163
Received 6 June 2025; Received in revised form 26 December 2025; Accepted 12 January 2026
Available online 13 January 2026

To address the privacy issue, Federated Learning (FL) appears as a promising approach by ensuring that models are trained locally. By eliminating the need to transfer sensitive data to a centralized server on distributed users' devices, FL can safeguard user privacy [12]. This approach is particularly relevant for HAR, where user activity data is inherently personal and sensitive [7,13]. To address the resource consumption issue, an emerging approach is to use a combination of compression techniques and Transfer Learning (TL). In TL, a deep learning model developed for one task is reused as the starting point for a model on a second, related task. This significantly reduces training time and computational demand by leveraging prior knowledge. For example, a model trained on walking data collected in one region can quickly adapt to users in another region using TL, saving training time and energy.

Many libraries, such as TensorFlow-Federated, PySyft, and LEAF [14–16], and frameworks, such as Flower and FedML [17,18], have been proposed to test, simulate, and emulate FL algorithms, both in centralized and distributed scenarios. While several solutions to use FL or TL in HAR have been proposed [19,20], two main challenges remain still unsolved. The first is the model privacy or copyright, which is becoming an important requirement for the ML/DL domain. The second is a practical deployment of the FL framework that can effectively run on common Android smartphones. To summarize our framework's contribution compared to the aforementioned platforms, our system enables, firstly, the direct handling of the entire federated learning and recognition process on Android smartphones, thereby providing a real-world deployment experience; secondly, it guarantees ownership of the neural network model. Those two features, coupled together, are not provided by any other platforms. To achieve this result, our solution builds upon Flower to leverage its capabilities of executing large-scale FL experiments, while shifting from simulated to real devices.

To cope with the two challenges, we propose Transferable and Copyright-Preserving Human Activity Recognition (TCP-HAR), a solution that integrates watermarking tailored to guarantee neural network model ownership in a centralized FL scenario of HAR and that can run smoothly over Android smartphones. We design a procedure to embed watermarks via backdooring [21], a commonly used technique in the context of computer vision. We adapted this algorithm to the sensor-based human activity recognition domain, since data exhibits diverse patterns. Then, we provide a thorough exploration of the performance of an FL scenario tested on an Android smartphone. In particular, our Android client application makes it possible to customize the whole FL process: from model selection, training from scratch, testing, recognizing activities, to data collection, loading pre-trained model to reduce the training time and improve generalization capabilities via TL, notifying users on ongoing process, and testing different pre-processing strategies and well-known quantization techniques to fit the hardware constraint. We thus compare the inevitable performance loss against the full Python implementation.

We utilized two publicly available datasets, i.e., MotionSense and MobiAct, to measure performance in terms of standard metrics, such as recognition accuracy, F1-score, training time, and model size. The findings emphasize how the combination of TL and model quantization [22], along with a refactoring of the learning process to meet the Android implementation, made the FL setting of the HAR system possible in mobile environments. We also prove the successful embedding and verification of the watermark and that the ownership verification task does not interfere with the primary learning objective.

The contributions of the paper are as follows:

- We propose a watermarking technique integrated into a federated learning scenario to offer neural network model ownership verification for sensor-based human activity recognition. This is the first attempt at applying watermarking in a sensor-based HAR context.
- We implement and test our approach in a real mobile federated environment. Specifically, we developed an Android application that enables the management of the entire federated learning process, encompassing sensor monitoring, federated training, and testing configurations.
- We perform a thorough experimental evaluation on two publicly available datasets under different settings. In particular, we tested different data pre-processing techniques, dataset partitioning, and we also tested the application of two well-known and widely used quantization methods, i.e., Dynamic Range Quantization (DRQ) and Full Integer Quantization (FIQ), combined with federated and transfer learning.

The rest of the paper is organized as follows: Section 2 reports background on HAR-related literature and recent federated and transfer learning work; a detailed description of our system with our proposed watermarking algorithm is provided in Section 3; our Android application implementation is described in Section 4, whereas the experimental evaluation is described in Section 5. We finally draw some conclusions in Section 6.

2. Related work

2.1. Human activity recognition

Human activity recognition (HAR) has been explored for years [23–25] due to its broad importance in various applications, from healthcare monitoring to entertainment. HAR consists of recognizing daily activities (whether simple or complex), such as walking and brushing teeth or gestures, and if they are carried out outdoors or indoors. Their corresponding data may be collected via vision-based systems, such as video cameras, inertial measurement unit systems, including accelerometers, or other types of sensors that collect environmental or vital signal information.

The literature highlighted that the accuracy of the recognition process is influenced by multiple factors spanning data collection, sensors, data pre-processing, feature engineering, and learning strategy. For example, collecting data in controlled environments,

such as laboratories, compared to collecting data in real-world conditions, can lead to different performance [26,27]. The sampling frequency of sensors needs to be carefully selected, not only because of its impact on capturing activity patterns, but also due to its impact on battery duration [28–30]. Moreover, sensor modalities and their placement influence performance [31–33]. The most common sensors used in HAR are accelerometers, gyroscopes, and, occasionally, magnetometers [34]. The multiplicity of data sources necessitates normalization techniques to standardize data and address anomalies (e.g., outliers) [23,35,36]. Indeed, proper data re-scaling improves computational efficiency and model performance [37]. Data segmentation, i.e., dividing data streams into subgroups with common characteristics in HAR, is primarily realized via time-based sliding windowing due to its simplicity in implementation. The optimal time window, which also influences the outcome of feature engineering, depends on the specific activity to be recognized and the attributes considered [38–41]. So-called shallow machine learning (e.g., decision trees (DT), Support Vector Machines (SVMs), Naive Bayes (NB) to cite a few) and deep learning (e.g., Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to name a few) approaches have been investigated to study the ability and efficiency of recognizing the activities showing promising results also in real-time applications [42,43].

Smartphone-based activity recognition has gained momentum due to the pervasiveness of the device, its advanced sensor equipment (e.g., accelerometers and gyroscopes), portability, and computational capabilities. However, their resources remain limited in computation and battery, especially compared to other devices (such as servers or PCs). It is, therefore, clear that there is a need to adopt and adequately combine efficient strategies to perform the recognition process on smartphones. One of the pioneering studies in HAR using smartphone sensors was conducted by Saponas et al. who introduced iLearn, a system using the iPhone's accelerometer and Nike+iPod Sport Kit for real-time activity classification [8]. In this paper, we aim to investigate how the aforementioned factors, combined with federated and transfer learning strategies, affect the activity recognition process via a smartphone device.

2.2. Federated transfer learning

Federated Learning (FL) is defined as the practice of learning a global model starting from trained local machine learning models whose data are distributed and owned by different parties, e.g., organizations [44,45]. This enhances information privacy and global model accuracy by aggregating model updates from models trained on local (otherwise inaccessible) data.

FL has been previously studied in HAR and has gained momentum because of its promising results in privacy-preserving [7,46], personalized, and collaborative learning [47,48]. Sozinov et al. showed that FL achieves slightly worse, but acceptable, accuracy compared to traditional centralized model training while improving communication costs and privacy concerns [49]. They simulated the adoption of mobile devices using a server with hardware comparable to that of a smartphone. Concone et al. also compared the distributed FL training approach against the centralized one by proposing a general FL framework that considers a wide range of scenarios of distributed activity recognition scenario [50].

Most of the literature focuses on proposing either a framework or an algorithm tailored to improve the performance of FL state-of-the-art approaches [46,51–53]. Consequently, it accounts only for how those approaches perform on datasets, neglecting the feasibility of implementing and deploying such approaches on smartphones. Few works focus on the study of implementing and deploying FL algorithms on smartphones for HAR purposes. For example, Dayakaran and Kadiresan recently experimented with the Flower framework on top of an Android smartphone, two laptops as clients, and an AWS EC2 as a server [54]. They trained the three clients using the mHealth dataset. They provided measures of accuracy, precision, recall, f1-score, and the energy consumed by the smartphone as a function of the number of epochs. Ek et al. proposed a new FL algorithm and performed preprocessing using channel-wise z-normalization on three datasets [55]. They compared against three other FL algorithms (FedAvg, FedMA, and FedProx) using TensorFlow. Although the tests have been carried out on datasets whose data was collected via smartphones, the authors do not assess the TensorFlow implementation's performance on a smartphone.

Transfer Learning (TL) usage has been raised to address the need to create high-performance learners trained with more easily obtained data [56] and has found utility across various real-world applications, including text sentiment classification, image classification, HAR, and many other fields [57–59]. TL allows models to leverage knowledge gained from pre-trained networks (*base model*), reducing the need for large labeled datasets and extensive computational resources. By transferring knowledge among the parties, it improves performance and speeds up training by only training a smaller portion of the model (*head model*). For a formal definition of FL, TL, and FTL concepts, we refer the reader to the following literature [60,61].

In recent years, an emerging trend has combined Federated and Transfer Learning (FTL) to provide a new state-of-the-art in several fields. FTL is defined as the practice of leveraging datasets from different parties, e.g., organizations, but with a similar nature [44]. This means that, in principle, datasets may differ both in features and sample space. Those characteristics enable the ability to learn and leverage knowledge gained from a certain domain and transfer it to other, related domains.

Chen et al. presented FedHealth, an FTL framework designed to enhance customization based on user data and facilitate data aggregation while preserving privacy, evaluated in the contexts of HAR and Parkinson's disease [62]. Nutter et al. explored deep learning combined with computer vision-based transfer learning techniques for HAR, focusing on dimensionality reduction of learned feature sets and classifier size reduction to optimize storage, compute, and battery life on a mobile device [63]. However, the authors did not implement their proposal on a smartphone. Recently, Osorio et al. proposed a method to use transfer learning for FL on Android smartphones [64]. As highlighted by the authors, using previously learned knowledge in scenarios where the source and target domains, tasks, or data distributions are different reduces latencies during the FL phase while maintaining good accuracy.

Table 1 sums up a comparative analysis between existing literature and our work. With Personalized FL, we mean any strategy that allows for improving accuracy and generalization capability under user data heterogeneity conditions. Examples of these

Table 1
State-of-the-art comparison.

Literature	FL	Personalized FL	Android implementation	Watermarking
[7,49,50,55]	✓	✗	✗	✗
[46–48,51,62,63]	✓	✓	✗	✗
[54]	✓	✗	✓	✗
[64]	✓	✓	✓	✗
Our work	✓	✓	✓	✓

strategies are clustering-based FL, meta-learning, federated multi-task learning, and FTL. While the literature is rich in work proposing new F(T)L algorithms that leverage datasets collected by smartphones, the literature neglects to focus on the feasibility of implementing such algorithms on smartphones. Indeed, very few, e.g. [54,64], explore a real FL deployment on (Android) smartphones, but none of them explore the adoption and performance of providing network model ownership mechanisms. Digging into [54], the deployment on the Android smartphone is described as a Flower client participating in the network training together with two laptops, but no other detail is provided. Moreover, no specific personalization technique is mentioned; therefore, we assume the Android client participates in a standard FL configuration. The authors also provide the energy consumed by the smartphone during the training phase taken at 3, 7, and 10 epochs, but do not provide information about how this measure was performed. In [64], the authors use transfer learning for FL on Android smartphones by pre-training an off-line Python model and transferring it, in TensorFlow Lite format, to the mobile devices. With respect to [54], the work in [64] did not measure the smartphones' energy consumption but provided some metrics about RAM and CPU usage. Compared to the existing literature, we developed an Android client application that makes possible to customize the whole FL process: from model selection, training from scratch, testing, recognizing activities, to data collection, loading pre-trained model to reduce the training time and improve generalization capabilities via TL, notifying users on ongoing process, and testing different pre-processing strategies and well-known quantization techniques to fit the hardware constraint. This setup is enhanced with network model ownership guarantees via digital watermarking.

3. Proposed system

Our TCP-HAR system consists of two main components: client application(s) and a learning server. The client application is designed to be deployed on Android smartphones and is in charge of customizing and handling the whole federated activity recognition process. The learning server is responsible for coordinating the federated distributed learning process and managing client interactions. The system is also designed to provide a watermarking-based FL process for model ownership verification. The ownership is guaranteed by embedding a robust backdoor-based watermark at the aggregator level into the global model in a data-independent and secure manner. In the following, we provide further details of the system components and the entire federated watermark distributed learning process.

3.1. Client agent of the system

Our proposed approach involves multiple clients (agents) collaborating to train a model that can efficiently recognize human activities. The architecture of TCP-HAR follows the centralized federated learning framework, shown in Fig. 1, in which the clients (Android smartphones) train a local model with their local data and then share their parameters with an aggregation server. The server uses a weighted average algorithm to aggregate the model's parameters received from clients. Subsequently, it sends an updated global model to all the clients. These steps occur for several rounds until the model converges because a specific stop condition is met (e.g., a desired level of accuracy is reached).

In addition to the FL approach, TCP-HAR also employs a TL methodology to speed up training and make the model more generalizable. TL allows models to leverage knowledge gained from pre-trained networks, thus reducing the need for large labeled datasets and extensive computational resources. It improves performance and speeds up training by only training a smaller portion of the model. Combining these concepts in TCP-HAR, during training, the clients maintain such pre-trained models with fixed weights in the base model and only update weights in the head model using the server's generalized model weights.

The *TCP-HAR Client* is an Android application developed to support ML models that can support both TL and FL. Specific focus, and part of the contribution, resides on the implementation of libraries to develop and deploy these functionalities over Android smartphones. The app also features a notification system that informs users of testing statuses, allowing them to respond to issues promptly. The *TCP-HAR Client* is customizable to run with loaded pre-trained models, with options for quantization, and percentage of non-trainable layers. In the current version, users can load the model with the specific number of trainable layers, ranging from 0% (fully trainable) to 83% (all layers pre-trained except for the final dense layer used for activity classification). Utilizing pre-existing knowledge from the pre-trained models and adapting them to new datasets decreases the chances of overfitting or consuming excessive computational power. We consider two options for quantization: *(i) Dynamic Range Quantization (DRQ):* This method statically quantizes only the weights from floating-point to 8-bit integers (dynamically quantizing layers based on their range), and keeps activation layers as float during inference. *(ii) Full Integer Quantization (FIQ)*, a more aggressive and comprehensive form of quantization. Unlike DRQ, FIQ requires a calibration step using a representative dataset, typically consisting of a small subset of a few hundred samples of training or validation data. This dataset is used to determine the range (min/max) of activations during inference, so that proper scaling can be applied when converting float values to integers (see Section 5.7 for results).

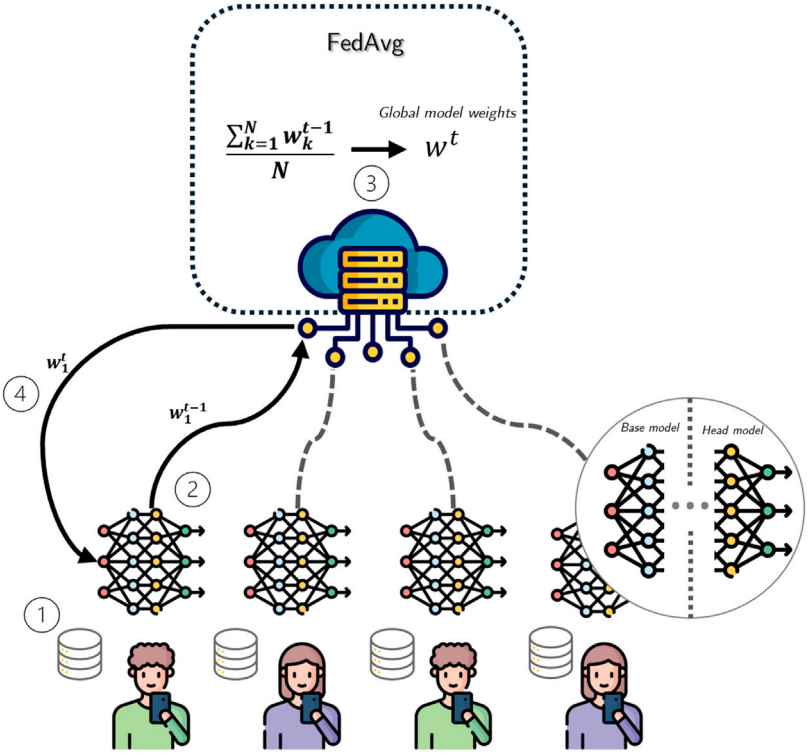


Fig. 1. System overview.

The main functions of the *TCP-HAR clients* include: (i) *Real-time sensor monitoring*, which enables live sensor data monitoring, ensuring validity and identifying potential issues before training, (ii) *Pre-trained model and dataset selection*, users can choose from various pre-loaded models and datasets, configuring quantization and trainable layers for TL and FL techniques, (iii) *Local training*, it supports local training with device-stored datasets, and (iv) *Federated Learning communication*, to support collaborative training and testing.

3.2. Learning server

The server is the node coordinating the learning process and is responsible for client interactions, model updates, and performance evaluations. It handles distributed learning tasks, tracks cumulative training time, and ensures consistent updates to the global model.

The server configuration is designed to manage several key aspects of the federated learning process. It defines the training and testing datasets, selects the model architecture, and specifies the number of sections involved in the current experimental trial. To facilitate distributed learning, the server coordinates the sampling of clients and aggregates their local updates to iteratively refine the global model. It monitors cumulative training time by collecting timing information from all participating clients and updating the total with the maximum reported value.

Additionally, the server establishes the minimum number of clients required for training and evaluation, determines the fraction of clients to be sampled in each communication round, and enforces an acceptable failure tolerance. The global model is periodically evaluated using standard performance metrics, including loss, accuracy, and F1 score. Finally, the server aggregates and updates the global model, reconstructing the weights by averaging the contributions from the clients.

In detail, the following steps outline how the smartphones collaborate in this federated setting. *(i)* The learning process begins with the server initializing the model parameters and distributing them to clients to initiate training. *(ii)* In each subsequent training round, the server configures the required components by generating client proxies and transmitting the current global model parameters to the selected clients. *(iii)* These clients then perform local training using their respective datasets and return the updated model parameters to the server. *(iv)* The server aggregates these updates through weighted averaging, thereby producing a new version of the global model, which is then redistributed to the clients. *(v)* Following this, the server conducts a centralized evaluation of the updated model using a predefined validation dataset. *(vi)* It also configures a federated evaluation phase by distributing evaluation metrics to clients. *(vii)* Clients subsequently assess the global model using their local validation data and transmit the evaluation results back to the server. *(viii)* These individual evaluation outcomes are then aggregated by the server to obtain a comprehensive performance assessment of the global model. *(ix)* This iterative cycle continues until the specified number of training rounds has been completed.

3.3. Preserving model copyright via watermarking

As discussed previously, FL preserves the privacy of clients' data and avoids incurring substantial costs for transferring training data from clients to a centralized server. However, despite its advantages, FL brings forth the issue of preserving ownership. In large-scale FL applications, there are multiple clients who are data owners and can use the model on their local devices, but there is only one model owner. This is particularly tricky in healthcare applications [13,65–67]. A side effect of the training process is that each client in FL can gain complete access to the global model in every round, including the final one, by reverse engineering the software application [68] or through on-device dynamic analysis [69]. Some approaches propose that the model owner deploy further mechanisms for hiding model parameters during the local training (e.g., homomorphic encryption), but this might lead to high latency, bandwidth costs, and additional requirements for benign clients [70]. Different watermarking techniques have been proven to be effective means of demonstrating intended ownership in cases where a malicious client uses the global model residing on their devices in unauthorized ways, such as monetizing the model or making it available to their own customers [21,71–73]. A recent approach is based on demonstrating ownership of DNN models. In this setting, the model owner first designs a secret watermark that consists of mislabeled input–output pairs. Then, the model owner trains the model with both the training dataset and the watermark in order to embed the watermark into the model. This watermark can be subsequently used to demonstrate ownership [74].

In this work, we design a procedure for effectively embedding watermarks into DNN models trained via centralized FL. Results will validate that we can achieve this without decreasing the accuracy of the resulting global model while minimizing the computational and communication overhead imposed on the distributed training process.

We can now summarize the watermarking-based FL process in our solution. The centralized FL process is composed of three main parties:

1. A number of clients $\mathcal{C} = \{c_j\}_{j=1}^{K}$ who are data owners and keep their datasets $\mathcal{D}_{c_j}$ private.
2. A model owner $\mathcal{O}$ providing a randomly initialized global model w_G at the beginning of federated learning and receiving the trained model at the end.
3. A secure aggregator Agg located between $\mathcal{O}$ and $\mathcal{C}$, as in [75].

Our model, a deep neural network (DNN), can be viewed as a function $F(x; w) : \mathbb{R}^n \to \mathbb{R}^m$ parameterized by w (e.g., w_G). The model owner $\mathcal{O}$ cannot obtain any information about $\mathcal{D}_{c_j}$ due to the secure aggregation protocol implemented by Agg [75].

In this work, we focus on federated learning using the common FederatedAveraging (FedAvg) algorithm [12], a widely used aggregation rule in which clients train their models using stochastic gradient descent (SGD). Before federated learning starts, w_G is initialized by $\mathcal{O}$. At the aggregation round t:

1. Agg sends $w_G^{(t)}$ to a subset of clients $\mathcal{C}_{\text{sub}} = \{c_i\}_{i=1}^{L}$, where $L \ll K$.
2. Each c_i updates $w_{c_i}^{(t-1)}$ with $w_G^{(t)}$, re-trains the updated $w_{c_i}^{(t)}$ using a pre-determined number of local passes over its local dataset $\mathcal{D}_{c_i}$, and sends the re-trained local model $w_{c_i}^{(t)}$ to Agg.
3. Agg averages all local models into a new $w_G^{(t+1)}$.

Several studies [21,71–73,76,77] have demonstrated the feasibility of embedding watermarks into DNNs for ownership verification. A watermark set $\mathcal{WM}_w$ consists of samples $\{x, B(x)\}$ designed by the model owner. The model owner embeds $\mathcal{WM}_w$ into w by optimizing w on both the training set and $\mathcal{WM}_w$ such that $B(x) = w^+(x)$ and $B(x) \neq w(x)$ for almost all $x \in \mathcal{WM}_w$, where $w^+(x)$ is the watermarked model. If the model owner suspects that another model w_{adv} may have been derived from w^+, the owner uses $\mathcal{WM}_w$ with a predefined verification algorithm `VERIFY` to demonstrate ownership. Verification is successful if $\texttt{VERIFY}(w_{\text{adv}}, \mathcal{WM}_w) = \texttt{True}$.

In this work, we investigate watermarking techniques based on *backdooring* [21,71,73]. A backdoor [78] typically consists of a *trigger set* composed of samples with intentionally incorrect labels. In the context of computer vision, the trigger set often includes specific patterns embedded into images [21] or entirely unrelated images [71], both of which serve as triggers. In the context of TCP-HAR, we adapted this approach to fit the tabular data that constitutes the HAR domain, applying a specific pattern in the trigger set. The backdoor is introduced into a DNN by training it on a combination of clean data and the trigger set. At inference time, the backdoored DNN behaves normally on clean inputs but predicts the designated incorrect label when presented with a trigger sample. To formally verify that a model w_{adv} is derived from a watermarked model w^+, i.e., to achieve $\texttt{VERIFY}(w_{\text{adv}}, \mathcal{WM}_w) = \texttt{True}$, the watermark $\mathcal{WM}_w$ must be kept secret and published in a timestamped public bulletin [71,79]. Additionally, the model's accuracy on the watermark set must exceed a threshold T_{acc}, i.e., $\text{Acc}(w_{\text{adv}}; \mathcal{WM}_w) \geq T_{\text{acc}}$. The value of T_{acc} is determined based on the size of the watermark set $|\mathcal{WM}_w|$ and the number of output classes m for the model w as in [79].

3.4. Overall algorithm

We now summarize in Algorithm 1 the main operation to embed a robust backdoor-based watermark into our FL setting, without requiring access to client data. This is achieved through two algorithmic components executed exclusively on the aggregator: *PRETRAIN* and *RETRAIN*. The described algorithm ensures that a watermark set $\mathcal{WM}_w$ is integrated into the global model in a data-independent and secure manner.

Algorithm 1 Federated Watermark Embedding in TCP-HAR

Initialize L, k, C, $\mathcal{P}$, w, Δ, η_G, η_l, $w_G^{(0)}$, tr, E, …
$\mathcal{WM}_w \leftarrow \textsc{GenerateTriggerset}(L, k, C, \mathcal{P}, w, \Delta)$
for $i = 1$ **to** E **do**
  for $b_G \in \mathcal{WM}_w$ **do**
    $w_G^{(0)} \leftarrow w_G^{(0)} - \eta_l \nabla \ell_G\big(w_G^{(0)}; b_G\big)$ ▷ PRETRAIN
  end for
end for
$w_G^{+(0)} \leftarrow w_G^{(0)}$
for $t = 0, 1, 2, \ldots$ **do** ▷ Federated Learning
  select $C_{\text{sub}}^{(t)} \subset C$ such that $|C_{\text{sub}}^{(t)}| = \lceil 0.1 \cdot |C| \rceil$
  broadcast $w_G^{+(t)} \rightarrow C_{\text{sub}}^{(t)}$
  $w_c^{-(t)} \leftarrow \textsc{LocalTrain}(w_G^{+(t)}) \quad \forall c \in C_{\text{sub}}^{(t)}$
  $w_G^{-(t+1)} \leftarrow \textsc{FedAvg}\big(\{w_c(t)\}_{c \in C_{\text{sub}}^{(t)}}\big)$
  while $\textsc{Acc}(w_G^{-(t+1)}, \mathcal{WM}_w) < T_{acc}$ **and** $ep < E$ **do** ▷ RETRAIN
    for $b_G \in \text{WM}_{w_G}$ **do**
      $w_G^{-(t+1)} \leftarrow w_G^{-(t+1)} - \eta_G \nabla \ell_G\big(w_G^{-(t+1)}; b_G\big)$
    end for
  end while
  $w_G^{+(t+1)} \leftarrow w_G^{-(t+1)}$
  $t \leftarrow t + 1$
end for
return w_G^{+}

Algorithm 2 Trigger Set Generation for HAR

procedure $\textsc{GenerateTriggerset}(L, k, C, \mathcal{P}, w, \Delta)$
  function $\textsc{PerturbedSignal}(L, \mathcal{P})$
    $sig \leftarrow \textsc{Uniform}(0, 1, L)$ ▷ base noise
    for $p \in \mathcal{P}$ **do**
      if $p =$ bump **then**
        sig += $\textsc{GaussianBumps}(L)$
      else if $p =$ step **then**
        sig += $\textsc{StepChanges}(L)$
      else if $p =$ wave **then**
        sig += $\textsc{Sinusoid}(L)$
      end if
    end for
    return $\textsc{Normalize}(sig)$
  end function
  for $c = 0$ **to** $C - 1$ **do**
    for $j = 1$ **to** k **do**
      $s_j \leftarrow \textsc{PerturbedSignal}(L, \mathcal{P})$
    end for
    $W_c \leftarrow [s_1; \ldots; s_k]$ ▷ $k \times L$ matrix
    $win \leftarrow \textsc{SlidingWindow}(W_c, w, \Delta)$
    $X \leftarrow X \cup win$
    $y \leftarrow y \cup \textsc{Repeat}(c, |win|)$
  end for
  return (X, y)
end procedure

In adapting the proposed watermarking procedure to the context of tabular time-series data (as in HAR), we proposed a new approach to the generation of the trigger set that preserves the core principles of the original method (*i.e.,* data-independence, visual consistency, and learnability) while tailoring the perturbations to suit temporal structures. Instead of using image-based patterns as in the original work, where watermark samples were constructed by superimposing class-consistent shapes onto noise-generated images, our design leverages time-domain perturbations that are more appropriate for sensor-based and sequential data representations. Specifically, to construct a data-independent trigger set suitable for watermarking models trained on time-series

or tabular data, we introduce a generative procedure based on synthetic perturbations, formalized in Algorithm 2. The algorithm takes as input the desired signal length L, the number of signal instances per class k, the number of watermark classes C, a set of perturbation types $\mathcal{P}$, and the parameters w and Δ defining the size and stride of a sliding window. At the core of the procedure lies the generation of perturbed signals, each initialized as a uniformly sampled random vector of length L. This base signal is iteratively modified by adding structured perturbations, selected from $\mathcal{P}$, which may include Gaussian bumps, discrete step changes, or sinusoidal waves. Each type of perturbation introduces distinct features that enhance the diversity and recognizability of the signal, while the normalization step ensures consistency in scale and learning from the model. For each watermark class label $c \in \{0, \ldots, C-1\}$, a set of k perturbed signals is generated independently and stacked into a matrix of shape $k \times L$, capturing intra-class variability. This matrix is then segmented using a sliding window mechanism to produce overlapping subsequences of fixed width w, spaced by stride Δ, emulating the input format of many time-series classification models. These windowed segments constitute the trigger inputs X, while the corresponding labels y are assigned by repeating the class identifier c for each window derived from the class-specific signals. The result is a labeled dataset of synthetic, but class-distinguishable, windows that can be used to embed a backdoor into the global model in a manner that is both task-agnostic and fully compatible with federated learning settings.

The $PRETRAIN(w_G^{(0)}, \mathcal{WM}_w)$ function embeds the output of Algorithm 2 (the watermark set) into the initial global model before the start of the FL rounds. It receives as input the randomly initialized global model $w_G^{(0)}$ and the watermark set designed by the model owner $\mathcal{WM}_w$, consisting of samples with specific features and assigned target labels. A loop runs for E epochs, allowing the model to include the watermark set. In each iteration, the watermark $\mathcal{WM}_w$ is divided into batches b_G and for each mini-batch: the gradient $\nabla\mathcal{L}_G(w_G^{(0)}; b_G)$ of the loss function $\mathcal{L}_G$ is computed with respect to the model parameters and the mini-batch and the gradient descent update is applied using learning rate η_i: $w_G^{+(0)} \leftarrow w_G^{(0)} - \eta_i \nabla\mathcal{L}_G(w_G^{(0)}; b_G)$. The watermarked model $w_G^{+(0)}$ is then broadcast to a subset of clients C_{sub} participating in the round, which compute a local training and send only the model weights to the aggregator. The aggregator computes the new global model $w_G^{-(t+1)}$ using the aggregation algorithm (*i.e.,* FedAvg) of all client updates: $w_G^{-(t+1)} \leftarrow \text{FedAvg}(w_{c_i}(t))$. At this point, the model $w_G^{-(t+1)}$ contains the watermark, but it has not yet been reinforced ($w^- \rightarrow w^+$). After that, the $RETRAIN(w_G^{(0)}, \mathcal{WM}_w)$ starts: the model is retrained on the watermark set until one of two termination conditions is met: either the model's accuracy on the watermark set $\text{Acc}(w_G^{-(t+1)}, \mathcal{WM}_w)$ reaches or exceeds the threshold T_{acc} or the maximum number of retraining rounds E_r is reached, to avoid a big computational overhead. This function embeds and reinforces the watermark in the global model after each aggregation round. Once the stopping condition is satisfied, the final watermarked global model $w_G^{+(t+1)}$ is returned and broadcast to clients for the next round, ensuring that ownership can be demonstrated at any time.

4. Android implementation

We developed an Android application whose look is represented in Fig. 2(a). It consists of Android clients created with *Android Studio* and Java (version 1.8), and it utilizes the *Transfer API* library (version 2.16.1) for ML operations through TensorFlow Lite (TFlite). The communication between the client and the server is performed using the *Flower* framework [17], which automates much of the communication processes required for FL, including tasks such as model updates and aggregation. This automation is necessary for maintaining synchronization between multiple clients and the central server. The server is again an Android device that coordinates the FL process through `FedAvgAndroid` strategy to aggregate the gradients by averaging the received weights.

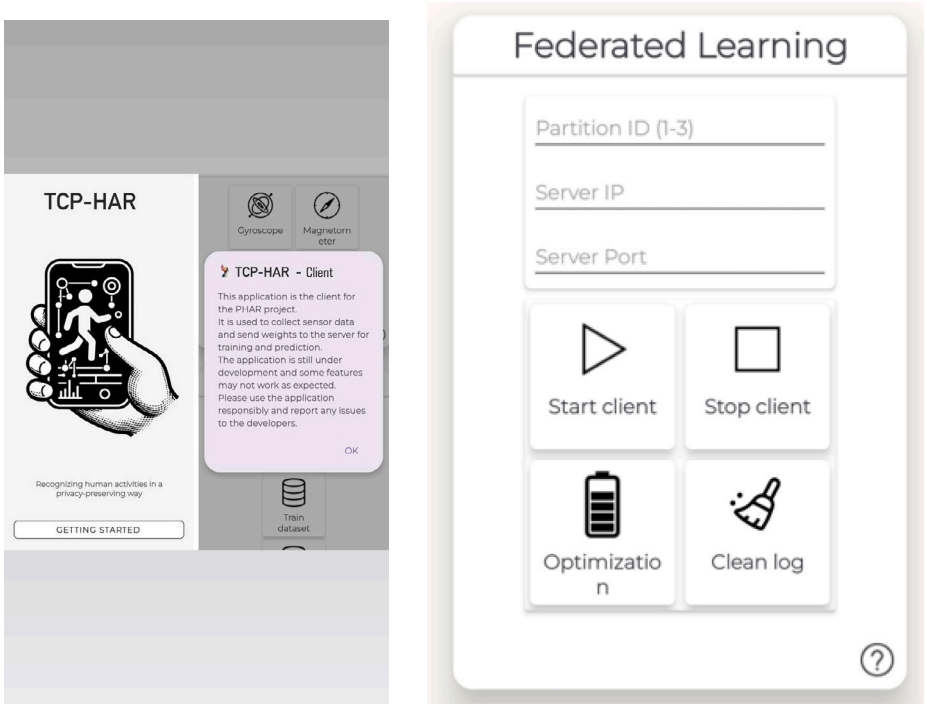
4.1. Notification system, memory management, and application permissions

To improve the user experience, the *TCP-HAR Client* integrates a sophisticated notification system. This feature is designed to keep users informed about the ongoing status of tests, whether they are being conducted locally on the device or through FL with the server. Users are continuously updated on the progress of their tasks and can respond promptly to any issues that arise. When tests demand significant storage resources, there is a risk of causing application crashes. To mitigate this, the notification system alerts users to potential issues. Indeed, if a crash occurs, the system automatically restarts and resumes the tests from the point where they were interrupted. Users are notified if a problem arises, indicated by the disappearance of the notification, which signals that the testing thread has been suspended.

The application requires specific permissions to access device storage and data. These permissions are necessary for several functions, including running background threads, posting notifications, enabling the FL connection, and storing test results in CSV format. The requested permissions include access to: network connectivity status, task and app management, device management, battery optimization, file and storage management, data synchronization, notifications, and job scheduler.

4.2. Application features

The **Real-Time Sensor Monitoring** feature provides sensor data dynamic display, enabling users to verify sensor readings as they are collected. This feature is useful for ensuring that the application effectively captures real-time data, reflecting the actual sensor inputs, status, and data collection. Upon application startup, a `SensorManager` instance registers listeners for the available sensors, including the gyroscope, accelerometer, magnetometer, and rotation vector sensors. The data from these sensors is stored in a matrix designed to hold up to 50 records, each containing multidimensional sensor inputs. This matrix enables tracking and



(a) (b)

Fig. 2. Overview of the Adroid application interface highliting (a) the landing page and (b) Federated learning settings.

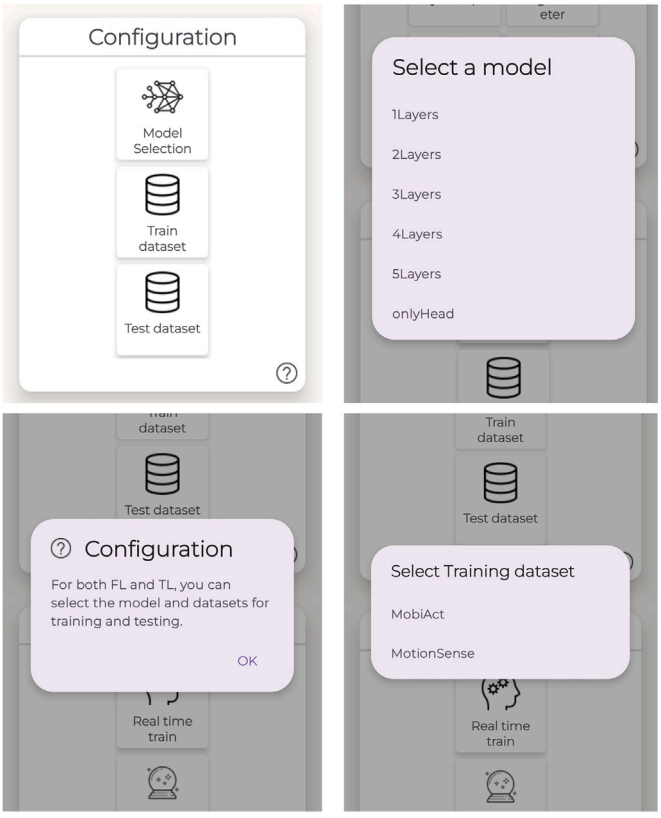


Fig. 3. Model selection overview.

recording the most recent sensor data. The data is continuously updated in the matrix, and once the matrix reaches its capacity of 50 records, the data is saved in a record that can then be utilized for real-time training or analysis.

The **Model Transferring** feature enables users to select an ML model from those available within the application. By recursively scanning the model folder, it displays a nested directory structure (Fig. 3) that reflects the path to each model, along with its associated configurations. Current options include model quantization and the number of frozen pre-trained layers. Users can also choose between different datasets for training and testing, where models can also be selected in an FL setup.

4.3. Data pre-processing, training, and testing

The datasets used have already undergone pre-processing. This includes normalization, the removal of outliers, and the synchronization of sampling frequencies, ensuring that the data is ready for the next stage. Each dataset is split into four portions, simplifying data selection for federated learning. The `NormalizedPreprocessing` class handles the extraction of samples and splits them into time-series sequences. Subsequently, these sequences are divided into smaller segments, referred to as "sections", which serve as input data for the model. The datasets considered include options such as *MS* and *MA*. The selected samples are divided into two parts: 80% of the data is allocated for training, while the remaining 20% is used for evaluation. When using a single dataset, this 80–20 split is maintained. In scenarios involving two datasets, 80% of the data is taken from the first dataset for training, while the remaining 20% is sourced from the second dataset for evaluation. The models differ in their quantization levels and layer configurations, offering a wide range of configurations for testing.

A sliding window technique is used to segment the datasets. Each window, or segment, contains a fixed number of samples (e.g., 50), and the sliding window moves with a predetermined step size (e.g., 10 samples), creating overlapping segments. This method is crucial in capturing both short-term and long-term temporal dependencies in the data. The resulting sections are structured so that 80% are used for training, while the remaining 20% are reserved for evaluation. Therefore, each segment is shaped as a 9×50 matrix, representing 50 time steps for each of the 9 sensor dimensions. This ensures that for every sensor dimension, 50 consecutive records are preserved per segment, providing a comprehensive view of temporal patterns in the input data. Evaluation is performed by assessing the model's performance on the testing set, with metrics such as accuracy, loss, and F1-score being computed and recorded for further analysis.

4.4. Federated learning setting

Fig. 2(b) shows the federated learning user interface: users can enter the server's IP address, port number, dataset partition, and model details. After inputting this information, a validation process starts to ensure the accuracy of these settings. Upon successful validation, the client establishes a gRPC connection with the server, which then confirms the connection. Once the connection is established, the Flower worker begins its operation by opening a gRPC channel to receive messages from the server. The worker plays a central role in managing the FL tasks. It facilitates communication with the server through gRPC, handling various messages that may include requests for model parameters, training instructions, and evaluation commands. Additionally, the worker is configured to run in the foreground to prevent the system from terminating it during long-running tasks.

A robust logging mechanism is also integrated into the worker. If an error occurs or if the task is canceled, detailed logs are recorded. These logs capture the status and outcomes of operations. Users can disconnect from the network at any time. These logs include details about the connection status and FL operations, complete with timestamps for each action. They also document training and evaluation metrics such as accuracy, loss, and training time.

5. Experimental evaluation

This section presents the core analysis of the paper, outlining the experiments, results, and insights gained. The tests followed an incremental approach, refining the model step by step by exploring new configurations and techniques based on prior results. The FL setting is tested among three Android smartphones running Android 11, 4 cores x2.2 GHz Cortex-A55, 6GB RAM, connected to a centralized server.

5.1. Datasets

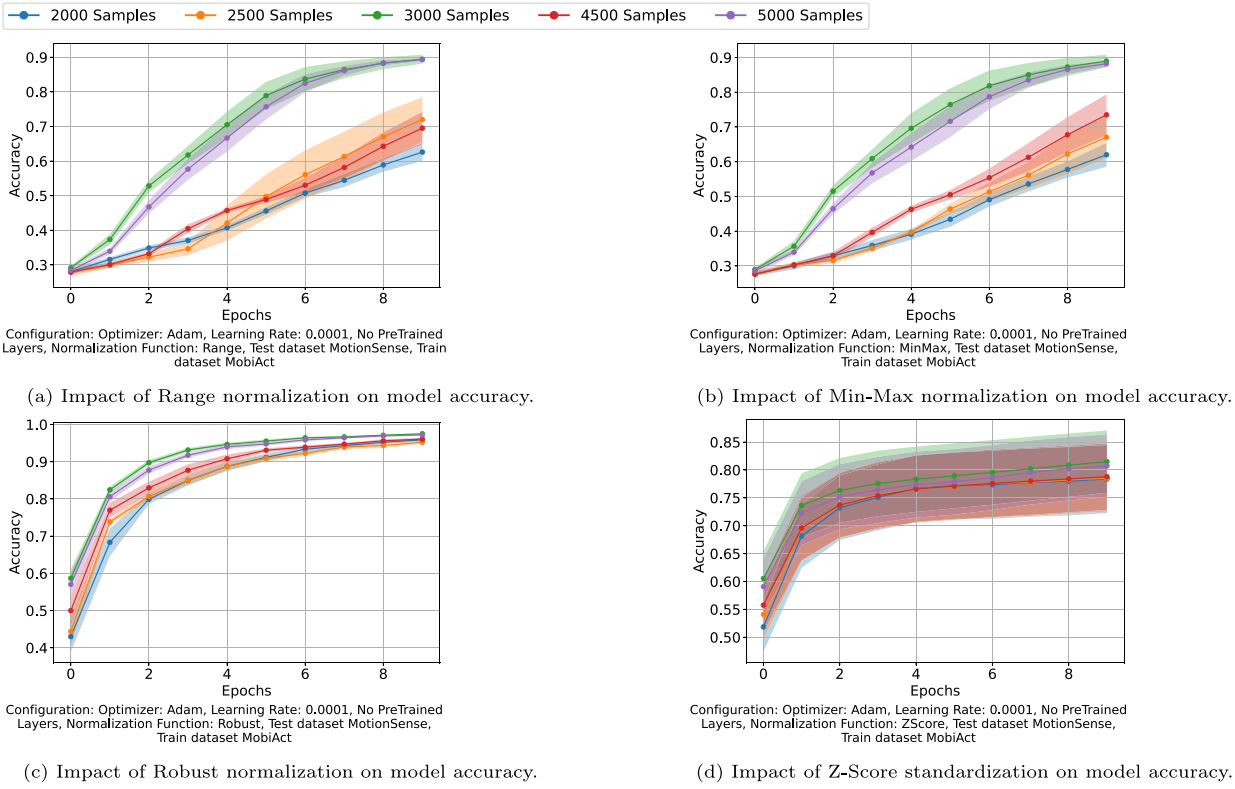
We reviewed widely used publicly available datasets, focusing on standard smartphone sensors, i.e., accelerometers, gyroscopes, and magnetometers, typically involved in data collection for daily activity recognition purposes [39]. We chose the MotionSense (MS) [80] and MobiAct (MA) [81] datasets for our analysis. Table 2 summarizes dataset characteristics.

MS: Data were collected from an iPhone 6s placed in participants' front trouser pockets, using an accelerometer and gyroscope at 50 Hz. The dataset includes supervised trials from 24 participants performing activities like walking, jogging, sitting, standing, and climbing stairs, with participants labeling the activities.

MA: Data were collected using a Samsung Galaxy S3 equipped with the LSM330DLC inertial module (accelerometer, gyroscope, and orientation sensors) at 200 Hz using the "SENSOR_DELAY_FASTEST" setting. The smartphone was placed in the participant's trouser pocket, with data recorded without fixed orientation to simulate everyday use. The dataset includes supervised recordings from 57 participants performing activities like walking, jogging, climbing stairs, sitting, and standing.

Table 2
Datasets overview. A: Accelerometer, G: Gyroscope, M: Magnetometer.

Dataset	Records	Classes	Participants	Sampling Frequency	Sensors
MS	1,412,865	6	24	50 Hz	A, G, M
MA	16,756,325	20	57	200 Hz	A, G



(a) Impact of Range normalization on model accuracy.

(b) Impact of Min-Max normalization on model accuracy.

(c) Impact of Robust normalization on model accuracy.

(d) Impact of Z-Score standardization on model accuracy.

Fig. 4. Comparison of the effects of different normalization techniques on model accuracy over multiple sample settings.

For our analysis, we considered the four activities that are shared between the two datasets, i.e., sitting (SIT), standing (STD), walking (WLK), and jogging (JOG). We considered data collected from all the three sensors, i.e., accelerometer, gyroscope, and magnetometer. The class distribution of the selected activities is imbalanced for the MA dataset, making the F1 Score a valuable metric [82]. It effectively measures classification performance by considering false positives and negatives rather than merely counting incorrect predictions.

5.2. Data pre-processing

Studies suggest the optimal range for sampling such activities is between 20 Hz and 50 Hz [28,38]. Consequently, reducing the sampling frequency of the MA dataset is important not only to minimize data storage but also to ensure compatibility with the MS dataset.

After resampling, the sensor data must be adjusted to account for differences in sensor characteristics, environmental conditions, and calibration settings of each sensor. Normalization options were considered based on outliers and variance. Analysis revealed a Gaussian distribution in most attributes, making Z-score standardization effective for removing outliers within the range of [−3, 3]. This step aims to prevent outliers from distorting model training and to reduce class imbalance by selecting a defined range. Although this may exclude high-intensity activities like running, the goal is to create a more balanced and clear dataset.

Evaluating four normalization functions would require extensive testing of normalized datasets across all hyper-parameter combinations to validate this choice. Therefore, we decided to experiment with the Adam optimizer at a learning rate of 0.0001 (additional extensive analysis exploring various configurations is available on GitHub [83]) and compare the performance of the normalization functions of the study, as shown in Fig. 4. Robust normalization (Fig. 4(c)) and Z-score standardization (Fig. 4(d)) achieve the highest training accuracy. We chose Z-score standardization.

Table 3
Baseline performance metrics when using the same data for Training–Testing evaluation.

Dataset	Accuracy	Loss	F1 Score	Training time (s)
MA-MA	0.9969	0.0094	0.9969	1158.82
MS-MS	0.9958	0.0150	0.9958	824.93

Table 4
Resulting metrics when combining different datasets for Training–Testing evaluation.

Dataset	Accuracy	Loss	F1 Score	Training Time (s)
MA-MS	0.9964	0.0105	0.9964	1179.90
MS-MA	0.9970	0.0110	0.9970	780.88

5.3. Dataset partitioning and segmentation

This subsection addresses a key challenge of the paper: applying FTL in a limited data storage environment. To handle this challenge, we divided each datasets into four partitions after standardization and the removal of outliers. Indeed, on the client side, data is accessed by partition. Attributes are combined with activity labels using a one-hot encoding scheme. The time-series data is then organized into sliding window vectors of 50 samples (equivalent to one second at 50 Hz) with a step size of 10 (equivalent to a 20% overlap). This process continues until the maximum number of sections is retrieved for input into the model.

5.4. Model selection

We customized the model proposed in [80]. The model is a sequential CNN that accepts an input tensor of shape $(9, 50)$, where 9 is the number of features and 50 is the sliding window size. The model processes it through several layers, and features an output layer with four neurons for classification, using softmax activation to generate probability distributions over the classes, with categorical cross-entropy as the loss function during training.

We used TensorFlow to implement our customized model, which we subsequently converted to its TFLite version to prepare the model for Android deployment. This process divides the model into a base and a head section. The base model consists of frozen, untrainable layers preloaded with weights from a previously trained MS dataset, while the head model remains trainable. A significant challenge during the conversion was ensuring layer compatibility to avoid issues during conversion and monitoring the converted operations.

To explore the model's capabilities, all possible combinations of configurations for the architecture's pre-trained layers were considered for conversion. The converted models were further compacted by considering post-training quantization.

5.5. Classifications and evaluations

The final stage of the HAR process evaluates the performance of the CNN across different configurations, using the accuracy described as Accuracy $= \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$, where TP means *True positive*, TN stands for *True negative*, FP means *False positive*, and FN stands for *False negative*. The F1-Score described as F1 $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$ as the primary comparison metrics, where Precision $= \frac{TP}{TP+FP}$ and Recall $= \frac{TP}{TP+FN}$. The categorical cross-entropy as the loss metric is then defined as Loss $= -\sum_{i=1}^{C} y_i \log(p_i)$ where y_i is the actual label and p_i is the predicted probability for class i.

Training time is recorded from the beginning of the first epoch to the end of the last. For FL, the cumulative training time is defined as the total time taken across all training rounds. The maximum training time observed among clients in each round is also recorded. The maximum training time is defined as follows: Cumulative Training Time $= T_1 + \sum_{i=1}^{N} T_i$ where T_1 is the worst training time for the first epoch round, T_i is the worst training time for the ith round, and N is the total number of rounds. The maximum training time observed among clients in round x is defined as:

$T_x = \max(T_{x,1}, T_{x,2}, \ldots, T_{x,n})$ where $T_{x,i}$ is the training time of the ith client in round x.

5.6. Transfer learning analysis

Before implementing techniques that may impact performance, it is essential to evaluate the model's baseline performance.

Table 3 presents the results from training the model on 80% of the total samples, with the remaining 20% set aside for testing, thereby establishing a baseline for future experiments. Next, we tested a combination of the two datasets. This approach yielded comparable results despite considering factors such as outlier selection, normalization, and methodological considerations, as shown in Table 4.

The impact of previously discussed factors on overall performance was minimal. A comparison of the model's performance on the Android and Python clients highlights the effects of model conversion and sample reduction, simulating real-world scenarios with storage constraints. Fig. 5 displays performance metrics for various training and testing sample configurations on both clients. It illustrates the impact of sample reduction on model performance, emphasizing the trade-offs associated with limited resources. The

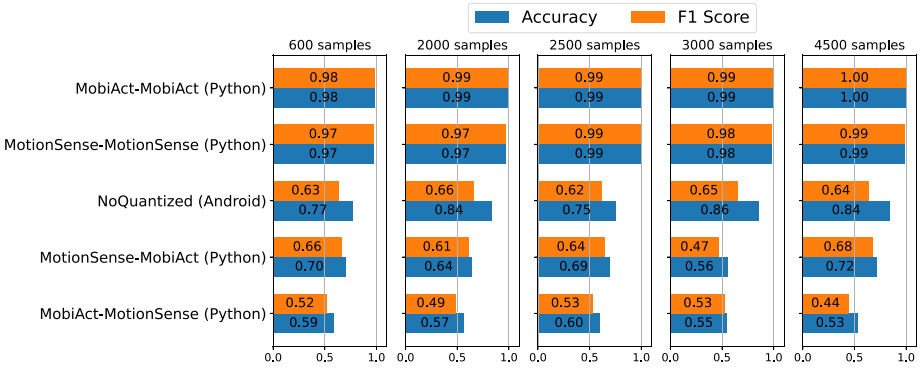
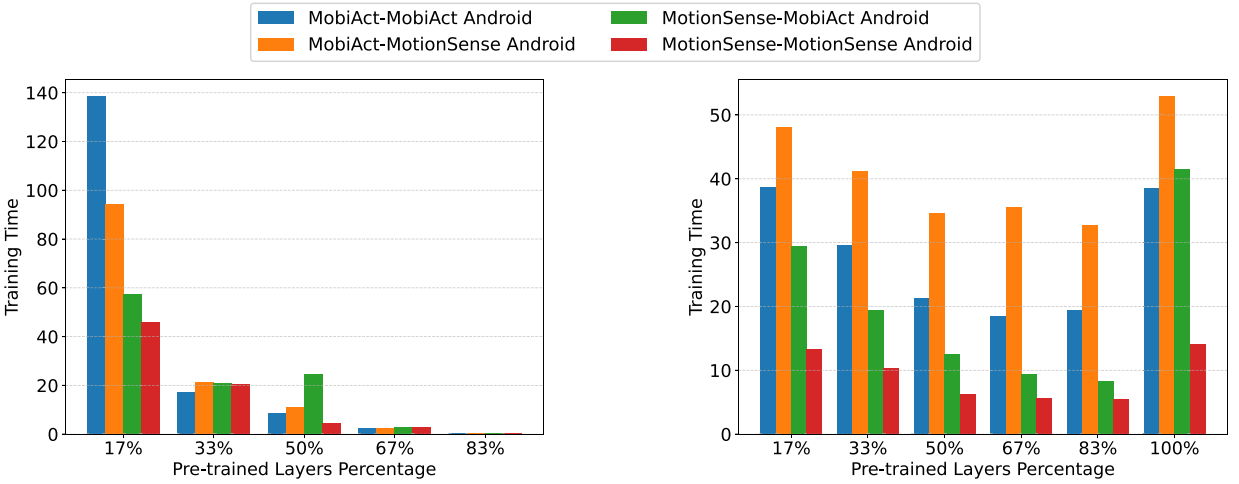


Fig. 5. Comparison of model performance across different sample configurations and clients (Android vs. Python).



(a) TL: Training time vs. number of pre-trained models for Android.

(b) TL: Training time vs. number of pre-trained models for Python.

Fig. 6. Comparison of the effects of pre-trained layers on training time (in seconds).

accuracy drop is linked to the dataset combination and class imbalance. For instance, the 'SIT' class in the MA dataset comprises only 6509 samples, representing just 1.54% of the total. This limited representation hinders the model's ability to generalize effectively, as achieving a balanced test scenario would necessitate 600 samples for each class, which is not feasible for our experiment.

Fig. 6 shows that training time decreases as the number of pre-trained layers increases. Notably, there is a significant difference in training times between the Android models (Fig. 6(a)) and the Python models (Fig. 6(b)), due to the different implementations of the TF library and the computational limitations of Android devices. These findings demonstrate the feasibility of training models on Android and provide a benchmark against the Python version.

5.7. Quantization

Three well-known (and widely adopted in the field of HAR) compression techniques are pruning, quantization, and knowledge distillation [22,84,85]. In pruning, parameters, neurons, and connections that do not contribute to (significant) accuracy are removed. In quantization, the weight representation is reduced by lowering bit width numbers, typically from floating-point values to integer values. Knowledge distillation consists of a pre-trained complex model, i.e., "the teacher", that transfers the learning to a smaller model, i.e., "the student", which mimics the teacher's behavior. The smaller nature of the student model is a form of model compression (of the more complex one), which allows easier deployment on mobile or embedded devices. Quantization can reduce model size, latency, and training time with minimal impact on accuracy [86].

We analyzed two specific quantization techniques: Dynamic Range Quantization (DRQ) and Full Integer Quantization (FIQ). The memory usage of the quantized models is illustrated in Fig. 7. This figure visually compares how different quantization methods affect model size against the non-quantized (NQ) versions.

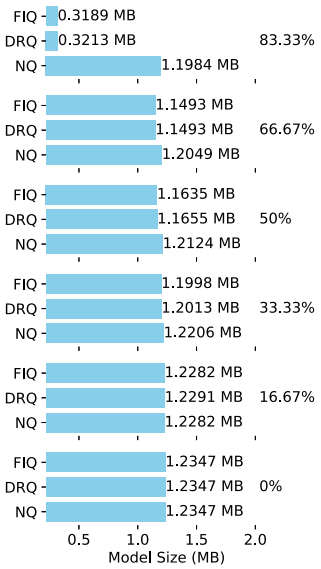


Fig. 7. Effect of quantization on the model size.

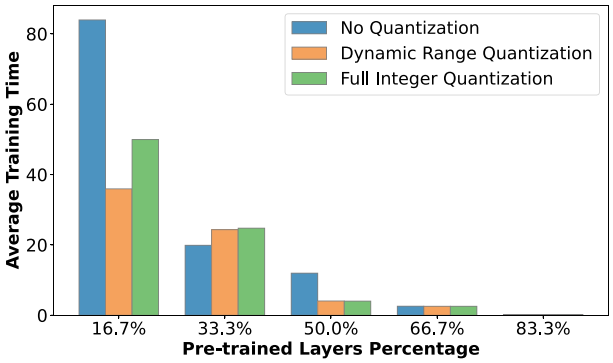


Fig. 8. TL: Training time for quantization method averaged on dataset combinations using pre-trained layers for Android version.

Quantization, as shown in Fig. 7, significantly reduces model size, particularly when more layers are pretrained and frozen. Quantization is applied to the base model, which is not used in later training phases. Higher levels of quantization lead to greater size reductions. However, pre-training the entire model does not necessarily ensure optimal accuracy or F1-score.

Fig. 8 shows that training time is affected by the quantization method. Training time decreases exponentially with increased quantization of early layers, while later layers exhibit similar times across methods. Though quantization generally has little effect on accuracy, results from the 83% pretrained layers test reveal that combining it with additional datasets can complicate generalization to unseen data, influenced by both the quantization methods and the selection of pre-trained layers. It is important to emphasize that the same comparison could not be performed when transitioning from a single partition analysis to a full dataset analysis in the Android version. This is because loading the entire dataset into memory is not computationally feasible on a standard smartphone.

5.8. Federated learning analysis

In this test benchmark, the first three dataset partitions were used for client training, while the final one was set aside for server-side testing. Training time is replaced by cumulative training time to account for FL iterations. It is crucial to identify trainable layers and deserialize weights before loading them into the server model for testing, as client-side training errors can affect model integrity. The migration from local testing to FL began by analyzing the Python implementation. The initial goal is to analyze the differences in training time between the TL and FTL approaches and subsequently evaluate the model performance in predicting human activities.

Table 5 provides a comparative analysis of training and cumulative times for the Python implementation, emphasizing the percentage change in training time between the two approaches across models with pre-trained layers. This table highlights the

Table 5
Comparison of Training Time (TT) and RAM usage between No-FL and FL for Python version.

% Layers	No-FL TT (s)	FL TT (s)	No-FL RAM (MB)	FL RAM (MB)
0.00%	99.59	63.04	605.3	599.0
16.67%	86.88	50.85	542.9	540.2
33.33%	50.90	30.52	485.6	478.2
50.00%	37.10	22.79	422.9	413.5
66.67%	32.75	20.82	359.7	330.8
83.33%	29.45	19.51	300.4	268.8

Table 6
Comparison of accuracy and F1-Score between No-FL and FL for Python version.

% Layers	Accuracy			F1-Score		
	No-FL	FL	% Change	No-FL	FL	% Change
0.00%	0.90	0.87	−3.47%	0.90	0.86	−5.23%
16.67%	0.89	0.84	−6.12%	0.88	0.81	−7.82%
33.33%	0.90	0.84	−6.37%	0.89	0.81	−8.78%
50.00%	0.90	0.85	−5.61%	0.90	0.82	−7.91%
66.67%	0.90	0.85	−5.46%	0.89	0.82	−7.63%
83.33%	0.91	0.89	−2.71%	0.90	0.88	−2.90%

performance of both techniques across all dataset partitions, reporting the average across the diverse settings. The table also reports the RAM consumption during model training for both the standard TL and FL settings. For the FL configuration, the values represent the average RAM usage across all participating client devices (agents) during local training.

The analysis considers the ratio of total training time to the number of epochs, where, for FL, the effective epochs are calculated as # epochs × # rounds. The results underscore that the FL approach yields significantly faster responses during the training phase. Similar results yield in RAM usage. This reduction occurs because in FL, each client trains on a smaller subset of data and typically updates only a portion of the model parameters, while the remaining layers remain frozen or are shared globally. As a result, the memory required for optimizer states, gradient storage, and intermediate activations is reduced.

Transitioning from a local to a federated approach not only promotes privacy and collaboration among devices but also optimizes the utilization of computational resources. The findings reveal that these advantages are further enhanced by a reduction in overall training time. Performance metrics shown in Table 6, including accuracy and F1-score, indicate that the quality of the model remains largely unaffected. Although we did not measure Android smartphones' energy consumption, results obtained in Figs. 7 and 8, with a training time in the order of around 1–2 min in the worst case, it is reasonable to think that this would not affect smartphones' battery level too badly. This claim is also supported by the results obtained for the Python implementation in terms of RAM consumption during the training phase. Although the Android implementation would show different RAM consumption, we expect a trend similar to the one for the Python implementation.

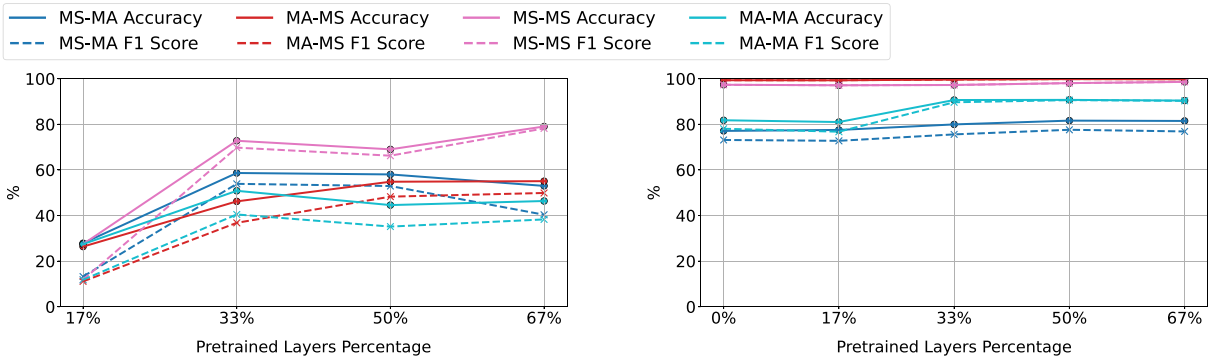
Limiting the samples to consider per test to 2500, Fig. 9 illustrates the impact of varying the number of pre-trained models on accuracy and F1-score metrics in the Android and Python versions using FTL. The plots highlight significant potential for enhancing generalization across datasets. Given that the pre-trained layers are derived from the MS dataset, the model demonstrates higher performance on MS-MS test sets. However, comparisons between the Android version, i.e., Fig. 9(a), and the Python version, i.e., Fig. 9(b), are limited by their performance disparity. Python employs FedAvg while Android utilizes FedAvgAndroid; additionally, different computational resources and training libraries are being used.

Focusing on the Android test configurations, Fig. 10 shows a deeper analysis with limited sample sizes, revealing no significant gains in accuracy and F1-score with the applied quantization techniques. These findings indicate that adopting the FTL approach does not result in substantial performance improvements, suggesting limitations in the model's generalization across different dataset partitions. The figures present averaged results from models with pre-trained layers across various dataset combinations.

5.9. Model copyright

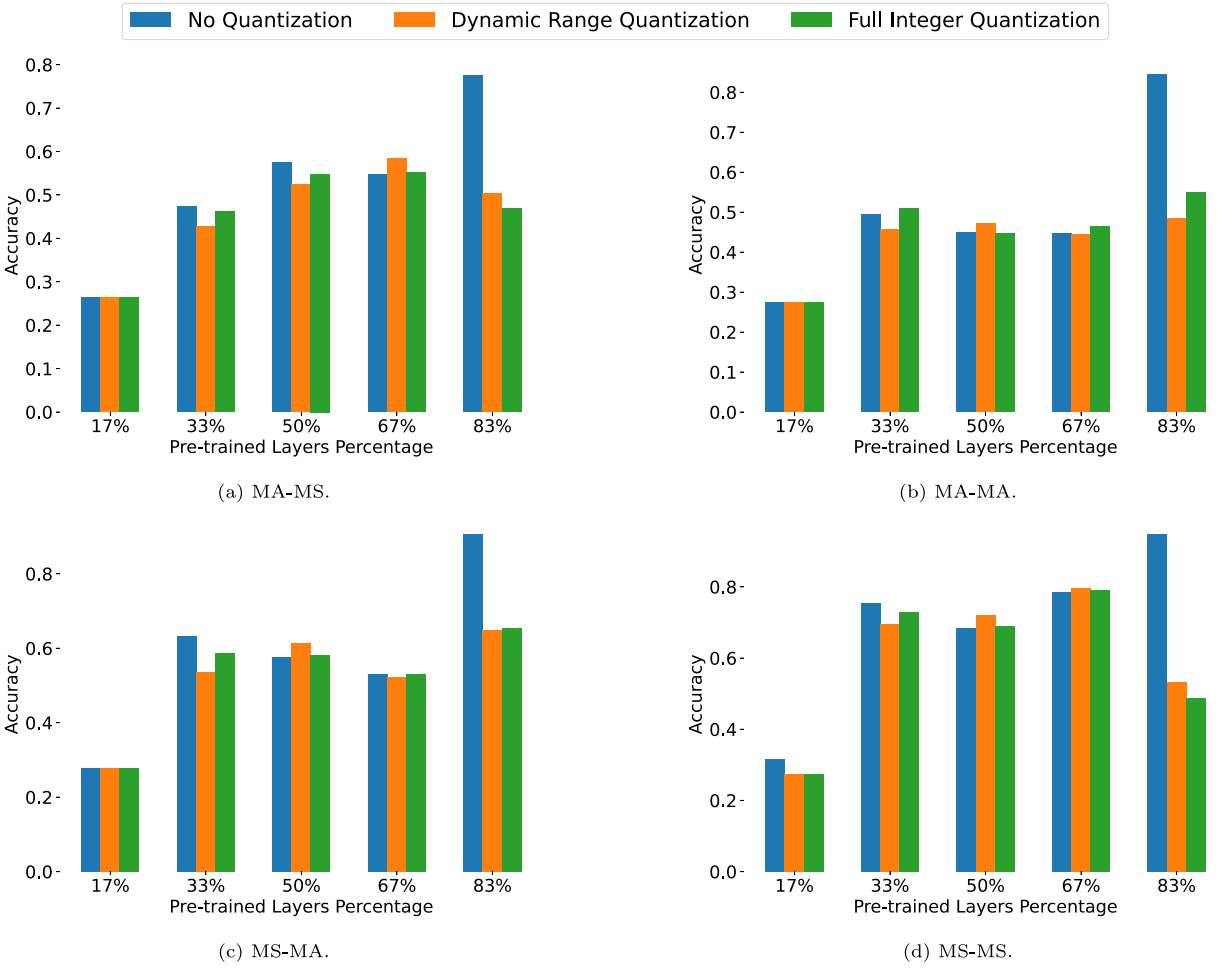
To prevent unauthorized use of the model distributed to clients for the training phase, we extended the WAFFLE framework to embed a backdoor into the model, serving as a watermark to certify the server's ownership. This approach aims to deter the unauthorized resale or misuse of the functional model. Fig. 11 illustrates the successful embedding and verification of the watermark throughout each FL round compared to the baseline model. In particular, Fig. 11(a) shows a consistently high accuracy, demonstrating that the ownership verification task does not interfere with the primary learning objective, as well as the robustness and reliability of the embedded signature throughout the training process. As shown in Fig. 11(b), the model maintains high accuracy and F1 score despite the presence of the watermark, indicating that the insertion does not degrade the main task performance, maintaining stable convergence patterns.

To evaluate the additional computational load caused by watermark embedding, the resource consumption of the federated server during training is reported in Fig. 12. It is worth noting that the watermark operation is only performed on the server, so



(a) Android (b) Python

Fig. 9. FTL: Comparison of classification metrics (accuracy and F1) vs. number of pre-trained models for (a) Android and (b) Python versions.

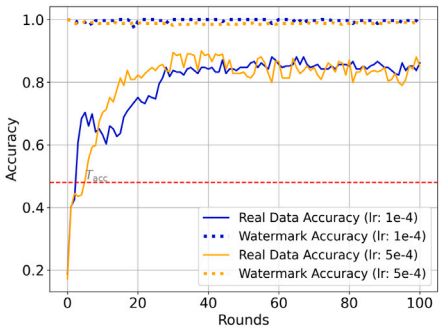


(a) MA-MS. (b) MA-MA.

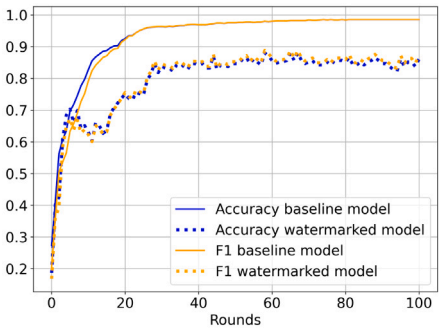
(c) MS-MA. (d) MS-MS.

Fig. 10. FTL: Accuracy for different quantization methods across dataset combinations with pre-trained layers for Android.

no overhead is introduced on the clients. Fig. 12(a) illustrates the peak CPU utilization per round where the baseline configuration and the watermark-embedded one are compared. The CPU workload of the watermark integration is mildly increased because the additional retraining phase on the trigger set is performed after each aggregation step. However, the general trend remains

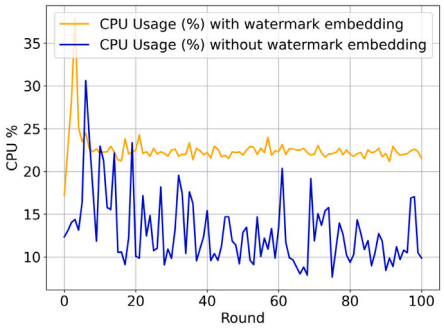


(a) Accuracy of the watermark-embedded model on both real data and watermark verification sets.

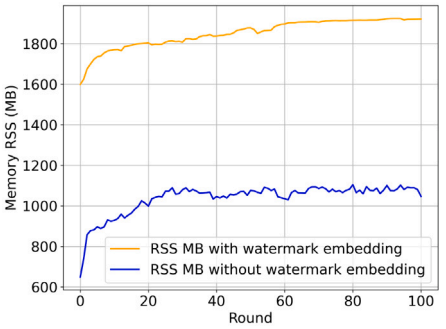


(b) Accuracy and F1 score for the baseline model and the watermarked one.

Fig. 11. Implementation of watermark on the model to prevent unauthorized usage by clients. (a) Our watermark integration achieves strong verification capability even for different learning rates (b) while preserving model accuracy, showing less than a 10% degradation compared to the baseline.

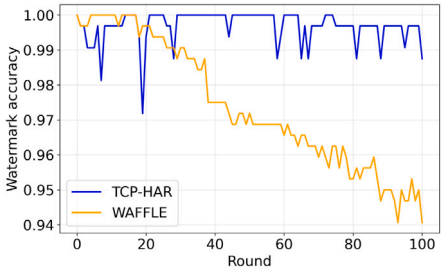


(a) Peak CPU usage (%).

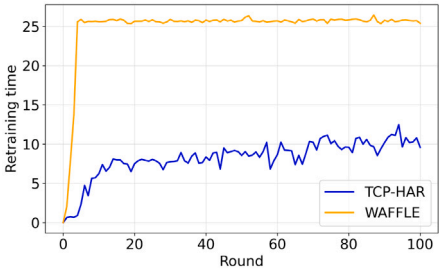


(b) Memory RSS (MB).

Fig. 12. Resource consumption analysis at each federated round. (a) CPU and (b) memory usage of the federated server with and without watermark embedding. The results indicate that the watermarking process introduces a moderate overhead on computational resources while maintaining stable behavior across all rounds.



(a) Watermark accuracy.



(b) Retraining time (s).

Fig. 13. Comparison between WAFFLE and TCP-HAR method on (a) watermark accuracy and (b) retraining time. As shown, TCP-HAR achieves significantly lower retraining times per round, thus reducing the overhead introduced by the watermarking process. At the same time, TCP-HAR maintains nearly perfect watermark accuracy throughout the entire training, while WAFFLE exhibits a gradual degradation across rounds.

unchanged; therefore, the watermarking process incurs only a limited and well-controlled computational overhead. Along with that, Fig. 12(b) depicts the memory usage (RSS in MB) of the server per round. The watermarked configuration exhibits a consistently higher but stable memory footprint, primarily due to the temporary storage of the trigger set and additional model buffers during retraining. These results confirm the scalability and practicality of the watermark, which serves as a robust and reliable solution for ownership verification and is also computationally sustainable; thus, it can be practically deployed in real-world federated learning scenarios, as evidenced by these findings.

To evaluate the trade-off between computational overhead and watermark reliability, we then compared the performance of TCP-HAR with the baseline WAFFLE framework in terms of retraining time and watermark accuracy, as illustrated in Fig. 13. The results in Fig. 13(b) clearly show that TCP-HAR achieves a remarkable improvement in efficiency, maintaining an average retraining time below 12 s per round, whereas WAFFLE rapidly saturates at approximately 25 s. This reduction of more than 50% in computational cost demonstrates that TCP-HAR introduces a significantly lighter retraining phase while still ensuring effective watermark embedding. In parallel, Fig. 13 also shows that TCP-HAR exhibits superior watermark stability. Throughout the entire federated training process shown in Fig. 13(a), TCP-HAR watermark accuracy remains close to 100%, confirming the persistent detectability of the embedded trigger set. Conversely, WAFFLE shows a progressive decrease in watermark accuracy over successive rounds, starting a decreasing trend. This degradation indicates a weaker retention of watermark characteristics under repeated aggregation and model updates. These findings confirm that TCP-HAR provides a robust yet computationally efficient watermarking mechanism. It not only minimizes retraining overhead but also guarantees a durable watermark signature, thereby ensuring ownership verification without compromising the scalability of the federated learning process.

6. Conclusions

This paper focuses on exploring and implementing FL and FTL techniques to provide a comprehensive understanding of their application in smartphone-based systems for HAR. The key contributions of this work are as follows:

- Development of *TCP-HAR*, a privacy-preserving smartphone application designed for HAR, integrating federated and transfer learning, digital watermarking embedding, and quantization techniques.
- Exploration of these techniques in the context of CNN, demonstrating the incremental integration of each approach, and tracking the progress and impact of these techniques throughout the paper.
- Demonstrating the potential for privacy gains with the showed relative loss in model performance metrics (e.g., accuracy and F1-score) by applying FL on edge devices.
- Addressing the challenges of deploying large-scale models on resource-constrained mobile devices while maintaining model performance and scalability of the Flower framework.

Findings suggest that properly combining FL with TL, coupled with quantization techniques, on mobile devices can have a significant impact on model performance. The developed architecture has demonstrated scalability and adaptability across various datasets, highlighting its potential for widespread applicability and promising results in terms of resource consumption. Moreover, TCP-HAR provides a robust, yet computationally efficient, watermarking mechanism, thereby ensuring ownership verification without compromising the scalability of the federated learning process.

Key challenges and future opportunities identified include improving model generalization and achieving real-time prediction capabilities within the TCP-HAR application. The scalable architecture proposed in this work shows considerable promise for continuous learning and live predictions in practical, user-centric applications. Furthermore, there is potential for further exploration of: (i) energy profiling, monitoring, and measurements for the Android smartphone; (ii) integrating and testing alternative watermarking strategies; (iii) FL by incorporating alternative strategies such as FedAdagrad, FedAdam, FedSDG, and FedYogi.

This paper contributes to the research by addressing critical aspects of privacy in FL environments and optimizing memory usage for resource-constrained devices, while reducing training times and model sizes.

CRediT authorship contribution statement

Alessio Sacco: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Bruno Palermo:** Software, Methodology, Conceptualization. **Giulio Figliolino:** Software. **Chiara Contoli:** Writing – review & editing, Visualization. **Guido Marchetto:** Supervision. **Flavio Esposito:** Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Alessio Sacco reports was provided by Polytechnic of Turin Department of Control and Computer Engineering. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data used are already publicly available.

References

[1] J. Fontecha, F.J. Navarro, R. Hervás, J. Bravo, Elderly frailty detection by using accelerometer-enabled smartphones and clinical information records, Pers. Ubiquitous Comput. 17 (6) (2013) 1073–1083.
[2] J. Dai, X. Bai, Z. Yang, Z. Shen, D. Xuan, PerFallD: A pervasive fall detection system using mobile phones, in: 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops, IEEE, 2010, pp. 292–297.
[3] S. Majumder, M.J. Deen, Smartphone sensors for health monitoring and diagnosis, Sensors 19 (9) (2019) 2164.
[4] W. Niu, J. Long, D. Han, Y.-F. Wang, Human activity detection and recognition for video surveillance, in: 2004 IEEE International Conference on Multimedia and Expo (ICME)(IEEE Cat. No. 04TH8763), vol. 1, IEEE, 2004, pp. 719–722.
[5] B. Fu, F. Kirchbuchner, A. Kuijper, A. Braun, D. Vaithyalingam Gangatharan, Fitness activity recognition on smartphones using doppler measurements, Informatics 5 (2) (2018) 24.
[6] M. Straczkiewicz, P. James, J.-P. Onnela, A systematic review of smartphone-based human activity recognition methods for health research, NPJ Digit. Med. 4 (1) (2021) 148.
[7] S. Zehtabian, S. Khodadadeh, L. Bölöni, D. Turgut, Privacy-preserving learning of human activity predictors in smart environments, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021, pp. 1–10.
[8] T. Saponas, J. Lester, J. Froehlich, J. Fogarty, J. Landay, iLearn on the Iphone: Real-Time Human Activity Classification on Commodity Mobile Phones, University of Washington CSE Tech Report UW-CSE-08-04-02 2008, 2008.
[9] F. Gu, M.-H. Chung, M. Chignell, S. Valaee, B. Zhou, X. Liu, A survey on deep learning for human activity recognition, ACM Comput. Surv. 54 (8) (2021) 1–34.
[10] Z. Xiao, X. Xu, H. Xing, F. Song, X. Wang, B. Zhao, A federated learning system with enhanced feature extraction for human activity recognition, Knowl.-Based Syst. 229 (2021) 107338.
[11] W. Qi, H. Su, A. Aliverti, A smartphone-based adaptive recognition and real-time monitoring system for human activities, IEEE Trans. Human-Machine Syst. 50 (5) (2020) 414–423.
[12] 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.
[13] O. Aouedi, A. Sacco, L.U. Khan, D.C. Nguyen, M. Guizani, Federated learning for human activity recognition: Overview, advances, and challenges, IEEE Open J. Commun. Soc. 5 (2024) 7341–7367.
[14] K.O. Alex Ingerman, Introducing TensorFlow federated, 2019, URL https://blog.tensorflow.org/2019/03/introducing-tensorflow-federated.html. (Accessed 24 September 2024).
[15] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, et al., Pysyft: A library for easy federated learning, in: Federated Learning Systems: Towards Next-Generation AI, Springer, 2021, pp. 111–139.
[16] S. Caldas, S.M.K. Duddu, P. Wu, T. Li, J. Konečnỳ, H.B. McMahan, V. Smith, A. Talwalkar, Leaf: A benchmark for federated settings, 2018, arXiv preprint arXiv:1812.01097.
[17] D.J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K.H. Li, T. Parcollet, P.P.B. de Gusmão, et al., Flower: A friendly federated learning research framework, 2020, arXiv preprint arXiv:2007.14390.
[18] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, et al., Fedml: A research library and benchmark for federated machine learning, 2020, arXiv preprint arXiv:2007.13518.
[19] M. Li, M. Gjoreski, P. Barbiero, G. Slapničar, M. Luštrek, N.D. Lane, M. Langheinrich, A survey on federated learning in human sensing, 2025, arXiv preprint arXiv:2501.04000.
[20] S.G. Dhekane, T. Ploetz, Transfer learning in human activity recognition: A survey, 2024, arXiv preprint arXiv:2401.10185.
[21] J. Guo, M. Potkonjak, Watermarking deep neural networks for embedded systems, in: 2018 IEEE/ACM International Conference on Computer-Aided Design, ICCAD, IEEE, 2018, pp. 1–8.
[22] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: A survey, Int. J. Comput. Vis. 129 (6) (2021) 1789–1819.
[23] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (Mar) (2003) 1157–1182.
[24] O.D. Lara, M.A. Labrador, A survey on human activity recognition using wearable sensors, IEEE Commun. Surv. Tutor. 15 (3) (2012) 1192–1209.
[25] S. García, J. Luengo, F. Herrera, et al., Data preprocessing in data mining, vol. 72, Springer, 2015.
[26] H. Gjoreski, M. Gams, M. Lutrek, Human activity recognition: From controlled lab experiments to competitive live evaluation, in: 2015 IEEE International Conference on Data Mining Workshop, ICDMW, IEEE, 2015, pp. 139–145.
[27] M. Stojchevska, M. De Brouwer, M. Courteaux, F. Ongenae, S. Van Hoecke, From lab to real world: Assessing the effectiveness of human activity recognition and optimization through personalization, Sensors 23 (10) (2023) 4606.
[28] M. Shoaib, S. Bosch, O.D. Incel, H. Scholten, P.J. Havinga, Complex human activity recognition using smartphone and wrist-worn motion sensors, Sensors 16 (4) (2016).
[29] S. Chung, J. Lim, K.J. Noh, G. Kim, H. Jeong, Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning, Sensors 19 (7) (2019) 1716.
[30] S. Agac, O.D. Incel, Resource-efficient, sensor-based human activity recognition with lightweight deep models boosted with attention, Comput. Electr. Eng. 117 (2024) 109274.
[31] E. McAdams, C. Gehin, N. Noury, C. Ramon, R. Nocua, B. Massot, A. Oliveira, A. Dittmar, C. Nugent, J. McLaughlin, Biomedical sensors for ambient assisted living, in: Advances in Biomedical Sensing, Measurements, Instrumentation and Systems, Springer, Berlin, Germany, 2010, pp. 240–262.
[32] R. Khusainov, D. Azzi, I.E. Achumba, S.D. Bersch, Real-time human ambulation, activity, and physiological monitoring: Taxonomy of issues, techniques, applications, challenges and limitations, Sensors 13 (10) (2013) 12852–12902.
[33] S.K. Yadav, K. Tiwari, H.M. Pandey, S.A. Akbar, A review of multimodal human activity recognition with special emphasis on classification, applications, challenges and future directions, Knowl.-Based Syst. 223 (2021) 106970.
[34] S. Zhang, Y. Li, S. Zhang, F. Shahabi, S. Xia, Y. Deng, N. Alshurafa, Deep learning in human activity recognition with wearable sensors: A review on advances, Sensors 22 (4) (2022) 1476.
[35] K. Xia, J. Huang, H. Wang, LSTM-CNN architecture for human activity recognition, IEEE Access 8 (2020) 56855–56866.
[36] N. Dua, S.N. Singh, V.B. Semwal, Multi-input CNN-gru based human activity recognition using wearable sensors, Computing 103 (7) (2021) 1461–1478.
[37] I. Goodfellow, Deep learning, 2016.
[38] O. Banos, J.-M. Galvez, M. Damas, H. Pomares, I. Rojas, Window size impact in human activity recognition, Sensors 14 (4) (2014) 6474–6499.
[39] W. Sousa, E. Souto, J. Rodrigres, P. Sadarc, R. Jalali, K. El-Khatib, A comparative analysis of the impact of features on human activity recognition with smartphone sensors, in: Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web, 2017, pp. 397–404.
[40] W. Sousa Lima, E. Souto, K. El-Khatib, R. Jalali, J. Gama, Human activity recognition using inertial sensors in a smartphone: An overview, Sensors 19 (14) (2019) 3213.
[41] M. Jaén-Vargas, K.M.R. Leiva, F. Fernandes, S.B. Gonçalves, M.T. Silva, D.S. Lopes, J.J.S. Olmedo, Effects of sliding window variation in the performance of acceleration-based human activity recognition using deep learning models, PeerJ Comput. Sci. 8 (2022) e1052.

[42] J. Suto, S. Oniga, Efficiency investigation from shallow to deep neural network techniques in human activity recognition, Cogn. Syst. Res. 54 (2019) 37–49.
[43] A.I. Middya, S. Kumar, S. Roy, Activity recognition based on smartphone sensor data using shallow and deep learning techniques: A comparative study, Multimedia Tools Appl. 83 (3) (2024) 9033–9066.
[44] S. Saha, T. Ahmad, Federated transfer learning: Concept and applications, Intell. Artif. 15 (1) (2021) 35–44.
[45] L. Pappone, A. Sacco, F. Esposito, On traffic matrix estimation via super-resolution and federated learning, IEEE Trans. Netw. Serv. Manag. 23 (2024) 544–554, http://dx.doi.org/10.1109/TNSM.2024.3516472.
[46] Y. Li, X. Wang, L. An, Hierarchical clustering-based personalized federated learning for robust and fair human activity recognition, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 7 (1) (2023) 1–38.
[47] C. Bettini, G. Civitarese, R. Presotto, Personalized semi-supervised federated learning for human activity recognition. arxiv 2021, 2021, arXiv preprint arXiv:2104.08094.
[48] P. Zhou, Y. He, Y. Zhai, K. Gao, C. Chen, Z. Qin, C. Zhang, S. Guo, FedAH: Aggregated head for personalized federated learning, 2024, arXiv preprint arXiv:2412.01295.
[49] K. Sozinov, V. Vlassov, S. Girdzijauskas, Human activity recognition using federated learning, in: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom, IEEE, 2018, pp. 1103–1111.
[50] F. Concone, C. Ferdico, G.L. Re, M. Morana, A federated learning approach for distributed human activity recognition, in: 2022 IEEE International Conference on Smart Computing, SMARTCOMP, IEEE, 2022, pp. 269–274.
[51] D. Cheng, L. Zhang, C. Bu, X. Wang, H. Wu, A. Song, ProtoHAR: Prototype guided personalized federated learning for human activity recognition, IEEE J. Biomed. Health Inform. 27 (8) (2023) 3900–3911.
[52] H. Jamil, Y. Jian, F. Jamil, S. Ahmad, Swarm learning empowered federated deep learning for seamless smartphone-based activity recognition, IEEE Trans. Consum. Electron. (2024).
[53] A. Sacco, A. Angi, G. Marchetto, F. Esposito, P4FL: An architecture for federating learning with in-network processing, IEEE Access 11 (2023) 103650–103658.
[54] D. Dayakaran, N. Kadiresan, Federated learning framework for human activity recognition using smartphones, Procedia Comput. Sci. 235 (2024) 2069–2078.
[55] S. Ek, F. Portet, P. Lalanda, G. Vega, Evaluation and comparison of federated learning algorithms for human activity recognition on smartphones, Pervasive Mob. Comput. 87 (2022) 101714.
[56] K. Weiss, T.M. Khoshgoftaar, D. Wang, A survey of transfer learning, J. Big Data 3 (2016) 1–40.
[57] C. Wang, S. Mahadevan, Heterogeneous domain adaptation using manifold alignment, in: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, 22, (1) Citeseer, 2011, p. 1541.
[58] L. Duan, D. Xu, I. Tsang, Learning with augmented features for heterogeneous domain adaptation, 2012, arXiv preprint arXiv:1206.4660.
[59] M. Harel, S. Mannor, Learning from multiple outlooks, 2010, arXiv preprint arXiv:1005.0027.
[60] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. (TIST) 10 (2) (2019) 1–19.
[61] Y. Kang, T. Fan, H. Gu, X. Zhang, L. Fan, Q. Yang, Grounding foundation models through federated transfer learning: A general framework, ACM Trans. Intell. Syst. Technol. 16 (4) (2025) 1–54.
[62] Y. Chen, X. Qin, J. Wang, C. Yu, W. Gao, Fedhealth: A federated transfer learning framework for wearable healthcare, IEEE Intell. Syst. 35 (4) (2020) 83–93.
[63] M. Nutter, C.H. Crawford, J. Ortiz, Design of novel deep learning models for real-time human activity recognition with mobile phones, in: 2018 International Joint Conference on Neural Networks, IJCNN, IEEE, 2018, pp. 1–8.
[64] A.F. Osorio, F. Grassiotto, S.A. Moraes, A. Muñoz, S.F.G. Neto, W. Gibaut, Transfer learning for human activity recognition in federated learning on android smartphones with highly imbalanced datasets, in: 2024 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2024, pp. 1–4.
[65] S. Shang, J. Yang, Z. Sun, P. Fua, Datacook: Crafting anti-adversarial examples for healthcare data copyright protection, 2024, arXiv preprint arXiv:2403.17755.
[66] W.N. Price, I.G. Cohen, Privacy in the age of medical big data, Nature Med. 25 (1) (2019) 37–43.
[67] M. Ienca, Don't pause giant AI for the wrong reasons, Nat. Mach. Intell. 5 (5) (2023) 470–471.
[68] S.J. Oh, B. Schiele, M. Fritz, Towards reverse-engineering black-box neural networks, Explain. AI: Interpret. Explain. Vis. Deep. Learn. (2019) 121–144.
[69] Z. Sun, R. Sun, L. Lu, A. Mislove, Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps, in: 30th USENIX Security Symposium, Security 21, USENIX, 2021, pp. 1955–1972.
[70] P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, Found. Trends® Mach. Learn. 14 (1–2) (2021) 1–210.
[71] Y. Adi, C. Baum, M. Cisse, B. Pinkas, J. Keshet, Turning your weakness into a strength: Watermarking deep neural networks by backdooring, in: 27th USENIX Security Symposium, Security 18, USENIX, 2018, pp. 1615–1631.
[72] Y. Uchida, Y. Nagai, S. Sakazawa, S. Satoh, Embedding watermarks into deep neural networks, in: Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, ICMR, ACM, 2017, pp. 269–277.
[73] J. Zhang, Z. Gu, J. Jang, H. Wu, M.P. Stoecklin, H. Huang, I. Molloy, Protecting intellectual property of deep neural networks with watermarking, in: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS, ACM, 2018, pp. 159–172.
[74] B.G. Tekgul, Y. Xia, S. Marchal, N. Asokan, Waffle: Watermarking in federated learning, in: 40th International Symposium on Reliable Distributed Systems, SRDS, IEEE, 2021, pp. 310–320.
[75] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for privacy-preserving machine learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS, 2017, pp. 1175–1191.
[76] B. Darvish Rouhani, H. Chen, F. Koushanfar, Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, ACM, 2019, pp. 485–497.
[77] Z. Li, C. Hu, Y. Zhang, S. Guo, How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN, in: Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC, ACM, 2019, pp. 126–137.
[78] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, X. Zhang, Trojaning attack on neural networks, in: 25th Annual Network and Distributed System Security Symposium, NDSS 2018, Internet Society, 2018.
[79] S. Szyller, B.G. Atli, S. Marchal, N. Asokan, Dawn: Dynamic adversarial watermarking of neural networks, in: Proceedings of the 29th ACM International Conference on Multimedia, MM '21', ACM, 2021, pp. 4417–4425.
[80] M. Malekzadeh, R.G. Clegg, A. Cavallaro, H. Haddadi, Protecting sensory data against sensitive inferences, in: Proceedings of the 1st Workshop on Privacy By Design in Distributed Systems, W-P2DS '18, ACM, 2018, pp. 1–6.
[81] G. Vavoulas, C. Chatzaki, T. Malliotakis, M. Pediaditis, M. Tsiknakis, The mobiact dataset: Recognition of activities of daily living using smartphones, in: International Conference on Information and Communication Technologies for Ageing Well and E-Health, vol. 2, SciTePress, 2016, pp. 143–151.
[82] H. Narasimhan, W. Pan, P. Kar, P. Protopapas, H.G. Ramaswamy, Optimizing the multiclass F-measure via biconcave programming, in: 2016 IEEE 16th International Conference on Data Mining, ICDM, IEEE, 2016, pp. 1101–1106.

[83] Frirhos, PHAR repository, 2024, URL https://github.com/Frirhos-he/PHAR. (Accessed 24 September 2024).
[84] T. Liang, J. Glossner, L. Wang, S. Shi, X. Zhang, Pruning and quantization for deep neural network acceleration: A survey, Neurocomputing 461 (2021) 370–403.
[85] A. Pinto, A. Masci, A. Sacco, G. Marchetto, F. Esposito, Optimizing model pruning in decentralized learning networks with DFL-trim, in: 2025 IEEE 11th International Conference on Network Softwarization, NetSoft, IEEE, 2025, pp. 189–193.
[86] I.L. Orăşan, C. Seiculescu, C.D. Caleanu, Benchmarking TensorFlow lite quantization algorithms for deep neural networks, in: IEEE 16th International Symposium on Applied Computational Intelligence and Informatics, SACI, IEEE, 2022, pp. 000221–000226.