Zone-based Federated Learning for Mobile Sensing Data

Xiaopeng Jiang*§ Thinh On*§ NhatHai Phan* Hessamaldin Mohammadi* Vijaya Datta Mayyuri† An Chen†
Ruoming Jin‡ Cristian Borcea*
New Jersey Institute of Technology* Qualcomm Incorporated† Kent State University‡
Email:{xj8,to58,phan,hm385,borcea}@njit.edu, {vmayyuri,anc}@qualcomm.com, rjin1@kent.edu

***Abstract*—This paper proposes Zone-based Federated Learning (ZoneFL) to simultaneously achieve good model accuracy while adapting to user mobility behavior, scaling well as the number of users increases, and protecting user data privacy. ZoneFL divides the physical space into geographical zones mapped to a mobile-edge-cloud system architecture for good model accuracy and scalability. Each zone has a federated training model, called a zone model, which adapts well to data and behaviors of users in that zone. Benefiting from the FL design, the user data privacy is protected during the ZoneFL training. We propose two novel zone-based federated training algorithms to optimize zone models to user mobility behavior: Zone Merge and Split (ZMS) and Zone Gradient Diffusion (ZGD). ZMS optimizes zone models by adapting the zone geographical partitions through merging of neighboring zones or splitting of large zones into smaller ones. Different from ZMS, ZGD maintains fixed zones and optimizes a zone model by incorporating the gradients derived from neighboring zones' data. ZGD uses a self-attention mechanism to dynamically control the impact of one zone on its neighbors. Extensive analysis and experimental results demonstrate that ZoneFL significantly outperforms traditional FL in two models for heart rate prediction and human activity recognition. In addition, we developed a ZoneFL system using Android phones and AWS cloud. The system was used in a heart rate prediction field study with 63 users for 4 months, which demonstrated the feasibility of ZoneFL in real-life.**

***Index Terms*—federated learning, smart phones, mobile sensing, edge computing**

I. Introduction

Sensing data collected on mobile devices (e.g., from wearable sensors) can be employed in many practical deep learning (DL) application domains, such as mobile health and wellness. An effective DL model requires data from many users, and it needs to protect the privacy of sensitive mobile sensing data. Furthermore, it needs to adapt to user behavior, which is location-dependent. The aim of this paper is to build an effective DL system for mobile sensing data that works efficiently on smart phones and satisfies the following requirements: (i) *Privacy-preserving*: learn from data provided by many users, while protecting user data privacy; (ii) *Mobility-awareness*: achieve good model accuracy by adapting to user mobility behavior, and (iii) *Scalability*: scale well as the number of users increases. We propose **Zone-based Federated Learning (ZoneFL)**, a novel federated learning (FL) architecture that builds and manages different models for different geographical zones, to satisfy these requirements. By design, ZoneFL satisfies the privacy-preserving requirement because Federated Learning (FL) [1] learns from data collected by many users, while protecting the user data privacy during training. In FL, the models are trained on mobile devices with their local data, and the server aggregates the models received from mobile devices. The users' local data never leave the mobile devices.

Vehicular traffic prediction and heart health notification are two motivating examples for ZoneFL. For traffic prediction, the traffic patterns are different across zones due to zone-dependent user behavior (e.g., shopping vs. business). For an app that sends alerts about the level of cardiovascular risk associated with users' current activity, the altitude and the climate of different zones will impact the notifications received by users. Therefore, zone-based models built by ZoneFL are expected to outperform a global FL model.

The main novelty of ZoneFL is its zone-based approach to satisfy requirements for mobility-awareness and scalability. To adapt DL models to user mobility for higher accuracy and to achieve good scalability, ZoneFL divides the physical space into geographically non-overlapping zones mapped to a mobile-edge-cloud architecture. Each zone trains its own zone model, which adapts to the data and behaviors of the users who spend time in that zone. As users move from one zone to another, they collect data and participate in model training for different zones. For inference, their mobile devices switch from one zone model to another, based on their current location. Thus, zone models achieve higher accuracy than globally trained FL models, satisfying the mobility-awareness requirement. In ZoneFL, edge nodes manage the FL training within their zones and host the latest models for their zones. Mobile devices can download these models when they enter a new zone. The cloud collaborates with the edge nodes to dynamically maintain the zone partitions for the entire space, but it is not involved in training. Compared to the traditional FL mobile-cloud architecture, the mobile-edge-cloud architecture of ZoneFL is more scalable because model aggregation is done distributedly at the edge (satisfying the scalability requirement), has lower latency for mobile users who interact with the edge instead of the cloud, and results in less bandwidth consumption in the network core [2, 3].

A major challenge in ZoneFL is how to ensure the zone models adapt to user mobility behavior changes over time.

§Equal contribution

978-1-6654-3902-2/21/$31.00 2021 © IEEE

To solve this challenge, we propose two novel zone-based federated training algorithms: Zone Merge and Split (ZMS) and Zone Gradient Diffusion (ZGD). ZMS optimizes zone models by adapting the zone geographical partitions through merging of neighboring zones or splitting of large zones back to previously merged smaller zones. The algorithm ensures that merging and splitting results in better model accuracy in each new zone. ZMS can be used to gradually improve the zone partitions, when they are initially suboptimal. Different from ZMS, ZGD maintains fixed zones and optimizes a zone model by leveraging concepts from graph neural networks to incorporate the gradients derived from neighboring zones' data. ZGD uses a self-attention mechanism to dynamically control the impact of one zone on its neighbors. ZGD can be used to further optimize zone models when the zone partitions are relatively stable according to ZMS.

ZoneFL was evaluated in terms of model accuracy and system performance using two models and two real-world datasets: Human Activity Prediction (HAR) with mobile sensing data collected in the wild, and Heart Rate Prediction (HRP) with the FitRec dataset [4][1]. The results demonstrate that models using ZoneFL without the optimization performed by ZMS and ZGD significantly outperform their counterpart models using traditional FL for zones that have enough training data. ZoneFL with ZGD and ZMS further improve the model performance, with ZMS improving the performance in the initial rounds and ZGD after that.

We implemented a ZoneFL system using Android phones and AWS cloud. The results show that ZoneFL achieves low training and inference latency, as well as low memory and battery consumption on the phones. ZoneFL scales better, because a zone edge server handles only 34.98% to 37.26% of the communication and computation load handled by a global FL server. We also observed multiple zone merges and splits in the field study, when the model utility improved significantly. These results show the feasibility of ZoneFL in real-life.

II. Background and Related Work

Federated learning (FL) is a multi-round communication protocol between a coordination server and a set of N clients to jointly train a model f_θ, where θ is a vector of model parameters (also called weights). The training proceeds in rounds. At each round t, the server sends the latest model weights θ_t to a randomly sampled subset of clients S_t. Upon receiving θ_t, each client $u \in S_t$ uses θ_t to train its local model and generate the weights θ_t^u. Client u computes its local gradient $\nabla\theta_t^u = \theta_t^u - \theta_t$, and sends it back to the server. After receiving the local gradients from all the clients in S_t, the server updates the model weights by aggregating all the received local gradients using an aggregation function $\mathcal{G} : \mathbb{R}^{|S_t| \times n} \rightarrow \mathbb{R}^n$, where n is the size of $\nabla\theta_t^u$. A typical function $\mathcal{G}$ is weighted average, called FedAvg [5].

By joining the FL protocol, clients minimize the average of their loss functions as follows: $\theta^* = \arg\min_\theta \frac{1}{N} \sum_{u=1}^{N} \mathcal{L}_u(\theta)$, where $\mathcal{L}_u$ is the loss function of client u on their local training dataset D_u. $\mathcal{L}_u$ is defined as $\mathcal{L}_u(\theta) = \frac{1}{|D_u|} \sum_{x \in D_u} \mathcal{L}\big(f_\theta(x), y\big)$, where $|D_u|$ denotes the number of data samples in D_u, and $\mathcal{L}$ is a loss function (e.g., cross-entropy) penalizing the mismatch between the predicted values $f_\theta(x)$ of an input x and its associated ground-truth label y.

Location Embedding in FL. To adapt to user mobility behavior, a naive approach in FL could be to incorporate the user location in the model input [6, 7]. This approach, however, increases the model size and the computation overhead on the mobiles. Differently, ZoneFL balances the trade-offs between model utility, resource constraints, and scalability by developing novel federated training algorithms seamlessly integrated into a scalable mobile-edge-cloud system architecture.

Clustering and Personalization in FL. Although ZoneFL shares the idea of training models over groups of users with clustering in FL [8–10], there is no efficient clustering method to group users by their mobility behavior without violating users' location privacy. ZoneFL optimizes models to user mobility behavior and does not require centralized model updates or privacy-sensitive user information. The edge nodes do not have access to users' locations; they just know that the user has been in a possibly large zone. Furthermore, ZoneFL provides a solution that can be naturally deployed at the edge for better scalability, which is a further advantage compared to clustering approaches. Note that ZoneFL is orthogonal to personalized FL [11], which can be leveraged in ZoneFL to produce personalized models for each user in each zone.

Deep Learning at the Edge. DL has been employed in edge computing for a broad range of applications, such as video analytics, speech recognition, and autonomous vehicles [12–14]. However, these solutions still collect device data at the edge, and hence present privacy problems. Also, ZoneFL achieves better scalability because the training is done at the edge, without any involvement from the cloud.

III. ZoneFL Training

This section presents zone partitioning, ZoneFL training, and our two federated training algorithms.

A. Zone Partitioning

The physical space (e.g., a city) is partitioned into non-overlapping zones, based on administrative boundaries or other knowledge about their characteristics (e.g., shopping district, park, etc.). The zones are model-specific. For example, a heart rate prediction model has different zones compared with a vehicular traffic prediction model. In this way, ZoneFL can achieve better model performance by targeting training to zones in which the user behavior is more homogeneous for a given type of mobile sensing data. For example, the user mobility behavior in a park (e.g., exercising) is different from the behavior in a shopping districts (e.g., leisurely walking).

The granularity of zones is defined based on the target application and the size of the user pool, i.e., each zone shall be small enough to capture behavior differences among zones,

[1]Datasets were downloaded and evaluated by the NJIT team.

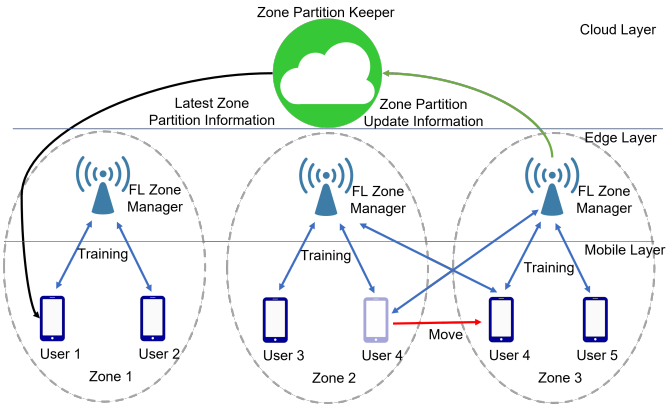

Fig. 1. ZoneFL Training Architecture.

and big enough to have sufficient users for good scalability-utility balance. The zone topology is a graph defined by neighboring relations of zones. By default, the neighboring relation is adjacency (i.e., two zones are neighbors if their borders touch), but this could be modified, for example, to define two zones as neighbors if they are geographically closer than a given threshold.

B. ZoneFL Training Overview

ZoneFL uses a mobile-edge-cloud architecture and trains separate zone models (i.e., separate instances of the same base model) on data collected in each zone, as shown in Figure 1. Each zone is managed by an FL Zone Manager at the edge, which maintains the latest models for its zone. A mobile is not tied to a zone, but collects data from all the zones visited by the user and performs training for each of these zones. For example, *User 4* moves from *Zone 2* to *Zone 3* and collects data in both zones (Figure 1). For each zone, the mobiles that collected data in the zone train the zone models jointly with its FL Zone Manager. Mobiles download the updated models from the edge managers when they need inference in a new zone (e.g., *User 4* in *Zone 3*). The FL Zone Manager does not know when and where the user was in a zone. It only knows its participation in zone training and the downloaded zone models. Therefore, the potential private information known by the edge is limited to coarse-grained zone information, while the fine-grained user location data is protected.

The cloud collaborates with the edge to update the zone partitions for the entire space, as the geographical coordinates of the zones may change over time due to changes in user behavior, but it is not involved in training. The mobile devices download the zone information and the identifiers of the edge managers from the cloud every time new zone configuration information is available. The zone partition information is used to associate data with different zones, perform local training, and send the weights to the corresponding edge manager, which will aggregate the zone model.

ZoneFL allows for the FL Zone Managers to be located in the cloud or at the edge, because their software is decoupled from the hardware infrastructure. This enables an incremental deployment over time, given that the edge infrastructure is currently available only in major cities. The FL Zone Managers for certain zones can start in the cloud and be migrated to the edge, when the edge becomes available. The mobile-edge-cloud architecture provides better scalability than a mobile-cloud architecture because edge nodes in ZoneFL have a lower communication and computation load than the cloud server in traditional FL. Also, the edge allows for faster interaction with the mobiles and for less bandwidth consumption in the network core. If there are multiple edge nodes in a zone, only one runs the FL Zone Manager, while the others act as relays between the mobile devices and the FL Zone Manager.

A major challenge in ZoneFL training is how to adapt the models to changes in user mobility behavior over time. We present two federated training algorithms that address this challenge. First, Zone Merge Split (ZMS) dynamically adapts the zone partitions by either (i) merging two neighboring zones into a larger zone, whose model performs better than each of the individual zone models, or (ii) splitting a larger zone back to previously merged smaller zones, whose individual models perform better than the model of the larger zone. Second, Zone Gradient Diffusion (ZGD) improves a zone model by aggregating contextual information derived from local gradients of neighboring zones. In ZGD, the zones do not change, but the user mobility behavior change is captured through the diffusion of information from neighboring zones. A self-attention mechanism is applied in ZGD to dynamically quantify the impact of each zone on its neighbors. Different deployments of ZoneFL may use either ZMS or ZGD or a combination of both based on trade-offs between model utility, scalability, and user mobility behavior.

C. Zone Merge and Split (ZMS)

ZMS is a dynamic zone management protocol that optimizes model utility across zones. Next, we define zone merging and splitting, and approximate the merging and splitting problem using novel greedy algorithms for the two operations.

Zone Merging. Given a set of N non-overlapping zones $Z = \{Z_i\}_{i\in[0,N]}$ and its complete set of possible combinations of zones Θ, merging a zone Z_i with its neighboring zones in Z is to find the set of non-overlapping and merged zones $\mathcal{Z} = \{\mathcal{Z}_j\}_{j\in[0,|\mathcal{Z}|]}$ where $\mathcal{Z} \in \Theta$, $|\mathcal{Z}|$ is the number of non-overlapping and merged zones in $\mathcal{Z}$, and $\cup_j \mathcal{Z}_j = \cup_i Z_i$ so that: **(1)** The model utility across merged zones $\sum_{\mathcal{Z}_j\in\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}_j, \mathcal{Z}_j)$ is optimized (Eq. 1); and **(2)** Every zone Z_i achieves better model utility after merging (Eq. 2). Note that $\mathcal{L}(\boldsymbol{\theta}_j, \mathcal{Z}_j)$ is the loss function of a zone $\mathcal{Z}_j$ with the model parameters $\boldsymbol{\theta}_j$.

$$\mathcal{Z}^*, \{\boldsymbol{\theta}_j^*\} = \underset{\{\boldsymbol{\theta}_j\},\mathcal{Z}\in\Theta}{\arg\min} \sum_{\mathcal{Z}_j\in\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}_j, \mathcal{Z}_j) \quad (1)$$

$$\text{s.t. } \forall Z_i \in \mathcal{Z}_j : \mathcal{L}(\boldsymbol{\theta}_j^*, Z_i) \leq \mathcal{L}(\boldsymbol{\theta}_i^*, Z_i) \quad (2)$$

where the loss of a zone $\mathcal{Z}_j$ is an average loss over all the users' local data in $\mathcal{Z}_j$: $\mathcal{L}(\boldsymbol{\theta}_j, \mathcal{Z}_j) = \frac{1}{|U_j|}\sum_{u\in U_j} \mathcal{L}(\boldsymbol{\theta}_j, u)$ where $|U_j|$ is the number of users in the zone $\mathcal{Z}_j$.

Zone Splitting. Splitting a large zone into a set of smaller sub-zones is the reverse process of merging zones. Given a

Algorithm 1 Zone Merging Algorithm

Input: Zone Z_i

1: $C \leftarrow \emptyset$ # initialize a list of zone merging candidates
2: $\mathcal{N} \leftarrow$ ***getNeighbors***(Z_i) # get neighboring zones of Z_i
3: **for** each neighboring zone $Z_n \in \mathcal{N}$ **do**
4: $\quad \boldsymbol{\theta}_{in}^{t} \leftarrow (\boldsymbol{\theta}_i^t + \boldsymbol{\theta}_n^t)/2$ # average of two zone models
5: $\quad \boldsymbol{\theta}_{in}^{t+1} \leftarrow \arg\min_{\boldsymbol{\theta}_{in}} \mathcal{L}(\boldsymbol{\theta}_{in}^t, Z_i \cup Z_n)$ # Eq. 1
6: $\quad$ **if** $\mathcal{L}(\boldsymbol{\theta}_{in}^{t+1}, Z_i) < \mathcal{L}(\boldsymbol{\theta}_i^{t+1}, Z_i)$ and $\mathcal{L}(\boldsymbol{\theta}_{in}^{t+1}, Z_n) < \mathcal{L}(\boldsymbol{\theta}_n^{t+1}, Z_n)$ # satisfying Eq. 2 **then**
7: $\quad\quad C \leftarrow C \cup Z_n$ # add Z_n into a list of candidates
8: **if** $C \neq \emptyset$ **then**
9: $\quad Z_n^* \leftarrow \arg\max_{Z_n \in C} \left[\mathcal{L}\left(\boldsymbol{\theta}_{in}^{t+1}, Z_i\right) - \mathcal{L}(\boldsymbol{\theta}_i^{t+1}, Z_i)\right] + \left[\mathcal{L}\left(\boldsymbol{\theta}_{in}^{t+1}, Z_n\right) - \mathcal{L}(\boldsymbol{\theta}_n^{t+1}, Z_n)\right]$ # get the best neighboring zone
10: $\quad$ ***Merge***(Z_i, Z_n^*)

large zone $Z = \cup_{i\in[0,N]} Z_i$ formed by merging smaller sub-zones $\{Z_i\}_{i\in[0,N]}$ and $\boldsymbol{\Theta}$ is the set of all possible combinations of sub-zones $\{Z_i\}_{i\in[0,N]}$, splitting Z is to find the set of sub-zones $\mathcal{S} \in \boldsymbol{\Theta}$, such that: **(1)** The model utility across sub-zones is optimized; and **(2)** Every sub-zone Z_i achieves better model utility after the zone splitting.

$$\mathcal{S}^*, \{\boldsymbol{\theta}_j^*\} = \underset{\mathcal{S}\in\boldsymbol{\Theta}, \{\boldsymbol{\theta}_j^*\}}{\arg\max} \frac{1}{|\mathcal{S}|} \sum_{\mathcal{Z}_j \in \mathcal{S}} [\mathcal{L}(\boldsymbol{\theta}_Z^*, Z) - \mathcal{L}(\boldsymbol{\theta}_j^*, \mathcal{Z}_j)] \quad (3)$$

Eq. 3 indicates S^* is the set of zones which has the maximal utility gain from the federated training of the original zone Z, i.e., $1/|\mathcal{S}| \sum_{\mathcal{Z}_j \in \mathcal{S}} [\mathcal{L}(\boldsymbol{\theta}_Z^*, Z) - \mathcal{L}(\boldsymbol{\theta}_j^*, \mathcal{Z}_j)]$.

Zone Merge and Split (ZMS) Algorithm. We proved that merge and split are NP-Hard by reducing them to the set cover problem. For brevity, we skip the proofs. Therefore, we propose greedy algorithms for merge and split. In simple terms, ZMS merges two zones when the model performance of the merged zone is better than the performance of each of the models of the individual zones to be merged. Each FL Zone Manager decides when to run the zone merging or splitting, based on the conditions of each zone. For instance, the users in some zones may collect more data than the users in other zones, which may result in more frequent training. Also, the user behavior may change in some zones, while remaining similar in others. While running the zone merging and splitting in every training round may result in the best zone partitioning, such a solution has too much overhead for both mobiles and FL Zone Managers. Therefore, ZoneFL balances zone partitioning efficiency with the overhead.

Algorithm 1 shows the pseudo code for zone merging. This algorithm is executed by the FL Zone Manager of zone Z_i in collaboration with the managers of its neighboring zones. ZMS merges Z_i with its best neighboring zone Z_n^*, optimizing the zone merging objectives in Eqs. 1 and 2 (Alg. 1, Lines 2-7). The number of neighbors in line 2 is typically a small constant in practice, and lines 4-7 repeat over it. The additional round of training in line 5 trades computation cost for better model utility. It can be omitted, and $\boldsymbol{\theta}^{t+1}$ becomes $\boldsymbol{\theta}^t$ in lines 6 and 9. The best neighboring zone Z_n^* is the zone that provides the maximal utility gain after the zone merging among all potential merges (Alg. 1, Line 9). Line 10 merges Z_i and Z_n^*.

Algorithm 2 Zone Splitting Algorithm

Input: Zone $\mathcal{Z}_j = \cup_i Z_i$, level l

1: $C \leftarrow$ ***getCandidates***$(\mathcal{Z}_j, l)$
2: **for** each zone $Z_c \in$ top-$k(C)$ **do**
3: $\quad \boldsymbol{\theta}_c^{t+1} \leftarrow \arg\min_{\boldsymbol{\theta}_c} \mathcal{L}(\boldsymbol{\theta}_j^t, Z_c)$
4: $\quad$ **if** $\mathcal{L}(\boldsymbol{\theta}_c^{t+1}, Z_c) < \mathcal{L}(\boldsymbol{\theta}_j^{t+1}, Z_c)$ **then**
5: $\quad\quad$ ***split***$(\mathcal{Z}_j, Z_c)$ # split the sub-zone Z_c from the merged zone $\mathcal{Z}_j$
6: $\quad\quad$ **break**
Function ***getCandidates***$(\mathcal{Z}_j, l)$:
7: $C \leftarrow \emptyset$ # initialize a list of worst sub-zones
8: **for** $Z_c \in$ ***subZones***$(\mathcal{Z}_j, l)$ **do**
9: $\quad$ **if** $\mathcal{L}(\boldsymbol{\theta}_j^t, Z_c) > \mathcal{L}(\boldsymbol{\theta}_j^t, \mathcal{Z}_j)$ **then**
10: $\quad\quad C \leftarrow C \cup Z_c$
11: **return** $sorted(C)$ # descending $\mathcal{L}(\boldsymbol{\theta}_j^t, Z_c)$

The zone models are trained and validated in the background by the mobiles in their respective zones. Mobiles retain a small validation dataset to validate the zone models, and send the validation results to their zone manager to be used in merge decisions. Thus, these operations do not incur latency during merges. The only operation that needs to be done specifically for a merge is the validation of the model over the two zones. To reduce the overhead, the zone manager selects only a percentage p of the phones in its zone to perform training and validation in this case.

In addition to adapting zones to changes in user mobility behavior, merging in ZMS also handles the case when the original zones set during bootstrapping do not have enough data for adequate training. In this situation, ZMS merges such zones with neighboring zones, therefore improving performance.

ZMS repeats the merging process across different zones and training rounds to create a set of merged zones, denoted $\mathcal{Z}$. However, over time, in response to user mobility behavior changes, some of the merged zones may need to be split.

The key idea of zone splitting (Algorithm 2) is to identify the sub-zone that performs worst in terms of model utility and split it from its merged zone, such that the zones after splitting perform better than the original zone. ZMS recursively splits sub-zones of a merged zone, which have the worst model utility, to optimize model utility across all sub-zones. Each of the merged zones $\mathcal{Z}_j \in \mathcal{Z}$ is a set of sub-zones $\{Z_i\}_{i\in[1,N]}$ represented by a binary tree of zone merging history, as illustrated in Figure 2. Each internal node in the tree represents a merged zone from its two sub-zones (children). Each leaf node is an indivisible zone. Periodically, the FL Zone Manager of a merged zone $\mathcal{Z}_j$ checks for a potential split. ZMS considers all the internal nodes of the binary tree of zone $\mathcal{Z}_j$ up to level l, a parameter that balances the utility benefit of splitting with its overhead, as potential sub-zones to split. ZMS keeps the best merges after zone splitting; thus approximating the zone splitting objective (Eq. 3) without affecting the zone merging objectives (Eqs. 1, 2). The training and validation at mobiles for split are similar to the ones for merge.

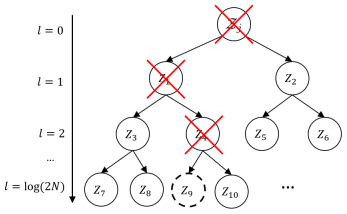

Fig. 2. Binary Tree and Zone Splitting

Algorithm 3 Zone Gradient Diffusion with Self-Attention

Input: Zone Z_i

1: $\mathcal{N}_i \leftarrow$ ***getNeighbors***(Z_i)
2: **for** $Z_n \in \mathcal{N}_i$ **do**
3: $\quad e_{in} \leftarrow \sigma\big(\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_i) \bullet \boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_n)\big)$ # where "$\bullet$" is an inner product
4: $\forall Z_n \in \mathcal{N}_i : \beta_{in} \leftarrow \dfrac{\exp(e_{in})}{\sum_{Z_j \in \mathcal{N}_i} \exp(e_{ij})}$ # computing coefficients
5: $\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t + \boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_i) + \sum_{Z_n \in \mathcal{N}_i} \beta_{in} \boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_n)$ # aggregating gradients from neighboring zones

D. Zone Gradient Diffusion (ZGD)

In addition to ZMS, we propose ZGD, an algorithm that keeps the zones fixed but adapts the model by aggregating contextual information derived from local gradients of neighboring zones (Algorithm 3). We found that contextual information captures changes in mobility patterns and significantly improves the utility of zone models. In ZGD, at round t, the managers of the neighboring zones Z_n of a zone Z_i derive their local gradients using the model parameters $\boldsymbol{\theta}_i^t$ from the zone Z_i by using local data D_u from their users u, as follows: $\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_n) = 1/|U_n| \sum_{u \in U_n} \boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, D_u)$. For data privacy protection, the mobiles of users u compute the gradients $\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, D_u)$ and send them to the managers of Z_n.

Intuitively, the more similar the gradients of a zone $(\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t Z_i))$ are with the gradients of a neighboring zone $(\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_n))$, the higher the impact of the neighboring zone Z_n on Z_i will be. We quantify this impact through self-attention coefficients β_{in} by normalizing the inner product of the local gradients of zone Z_i and its neighboring zones $Z_n \in \mathcal{N}_i$:

$$\forall Z_n \in \mathcal{N}_i : \beta_{in} \leftarrow \frac{\exp(e_{in})}{\sum_{Z_j \in \mathcal{N}_i} \exp(e_{ij})} \tag{4}$$

where $e_{in} = \sigma\big(\boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_i) \bullet \boldsymbol{\nabla}(\boldsymbol{\theta}_i^t, Z_n)\big)$, σ is the sigmoid function, and "$\bullet$" is an inner product.

Finally, we aggregate the gradients from neighboring zones to update the zone model $\boldsymbol{\theta}_i^t$ at round t:

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t + \boldsymbol{\nabla}\big(\boldsymbol{\theta}_i^t, Z_i\big) + \sum_{Z_n \in \mathcal{N}_i} \beta_{in} \boldsymbol{\nabla}\big(\boldsymbol{\theta}_i^t, Z_n\big) \tag{5}$$

By doing so, ZGD updates the zone models to diffuse contextual information from one zone to all the remaining zones across training rounds. This operation significantly enriches the information used to optimize zone models in ZoneFL, compared with existing FL algorithms.

IV. System Design and Implementation

A. System Architecture

The ZoneFL architecture has three main components, as shown in Figure 3: *(1) FL Phone Manager* coordinates the ZoneFL activities on the mobiles/phones; *(2) FL Zone Manager* coordinates the ZoneFL activities at the edge; and *(3) Zone Partition Keeper* maintains and provisions the latest zone partition information in the cloud.

The software components work together to support the six phases of ZoneFL: data collection and preprocessing, privacy protection, model training and aggregation, mobile apps using

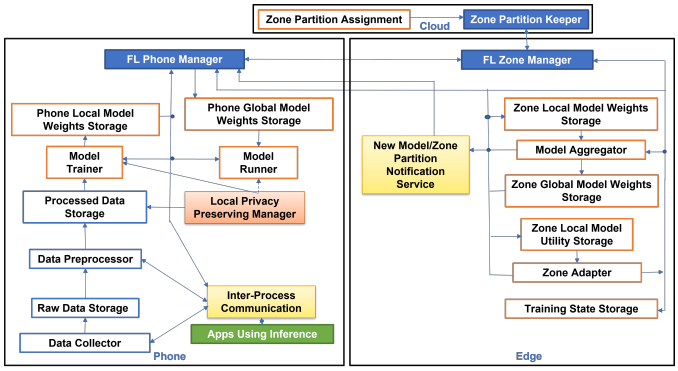

Fig. 3. ZoneFL System Architecture.

models for inference, zone partition maintenance, and zone training adaptation to user mobility. The first four phases follow traditional FL, and their details can be referred from FLSys [15] that ZoneFL is built on. In the following, we explain the two phases that are specific to ZoneFL.

Zone Partition Maintenance. The Zone Partition Keeper maintains the latest zone partition information in the system, which is represented as a graph. Each non-overlapping zone is a vertex, and each edge connects two neighboring zones. The initial zone partition information is bootstrapped by the administrator of the system based on administrative divisions of a region. The Zone Partition Keeper is also responsible for maintaining information about the identity (e.g., IP addresses) of the FL Zone Managers at the edge.

Initially, a phone receives the zone partition information from the Zone Partition Keeper. Then, it maps its data to different zones, based on the geographic locations where the data were collected. This determines the list of zones to which the phone subscribes for training. The phone communicates with the FL Zone Managers of these zones to jointly train the zone models. For inference, a phone may use a zone model even if the phone did not participate in the training of the given zone. This allows new users to quickly benefit from ZoneFL.

Zone Training Adaptation. The Zone Adapter of each zone exchanges model weights with its neighboring zones for dynamic adaptation of zone training (i.e., ZMS and ZGD) To perform merge and split in ZMS, the system needs to perform zone level model validation. This operation is done through the cooperation of the phones and the edge manager. The FL Zone Manager maintains a Zone Local Model Utility Storage for phones to report the model utility computed on their validation datasets, periodically aggregates the validation results, and updates the zone partition information accordingly. This process involves extra communication between phones and the FL Zone Manager, but it mitigates potential privacy issues, since data never leaves the phone.

ZGD exploits the fixed and stable zone partition information from the Zone Partition Keeper. The model aggregator aggregates clients gradients incorporating the gradients with regard to neighboring zones' model weights (received from the Zone Adapter) to further optimize a zone model (Eq. 5).

B. ZoneFL Prototype Implementation

We implemented an end-to-end ZoneFL prototype on Android phones and AWS cloud. This prototype, with ZMS for dynamic adaptation, was used in our field study, described in Section V. AWS offers AWS Local Zones as its edge computing service. However, it is not available yet in the area of our field study, and therefore the edge components of ZoneFL are deployed in the AWS cloud. We chose Deep Learning for Java (DL4J) as the underlying framework for DL-related operations, because it is a mature framework that supports model training on Android devices.

Deployment and Operation Scripts. The system administrator prepares the initial zone partition information as a geojson file, which defines the zones' geometry as polygon coordinates. We implement Python scripts to deploy and operate the system. The deployment script reads the geojson file provided by the system administrator to create an independent FL Zone Manager for each zone in AWS. The operation scrips are used to collect performance and reliability data.

FL Zone Manager. Most components are inherited from FLSys [15]. The new Zone Adapter component is implemented and deployed as AWS Lambda functions for low overhead and fast start time, and it communicates with its counterparts in neighboring zones to perform ZMS.

Zone Partition Keeper. We use an AWS S3 bucket as the Zone Partition Keeper of all zones. This is the only shared AWS resource in the system. All the other AWS resources are independent among different zones. In this way, once edge computing becomes more widespread, the FL Zone Manager can be migrated from the cloud to the edge. The S3 bucket makes the latest zone partition information available to phones for download. The previous partition information is also stored for the Zone Adapter to help with the split operation in ZMS.

Android Implementation. In the FL Phone Manager, the Data Collector collects heart rate (HR) sensing data, used in the field study, from a Bluetooth Polar HR tracking wrist band [16]. The Data Preprocessor uses the geojson file with Android Google Map API to check the zone where each data point belongs to, and generates the model input for training. The rest of the Android implementation is based on FLSys.

V. Evaluation

The evaluation presents results for model utility and system performance. The model utility experiments have two goals: **(i)** Compare the performance of ZoneFL with Global FL (i.e., traditional FL trained with all users globally); **(ii)** Quantify the benefit of ZGD and ZMS. The system experiments have three goals: **(i)** Demonstrate the feasibility of ZoneFL in a real-life deployment; **(ii)** Investigate ZoneFL scalability; and **(iii)** Quantify the ZoneFL phone training time overhead.

A. Datasets, Models, and Metrics

We use two datasets collected in the wild to evaluate two types of ZoneFL models: **(1)** Human activity recognition (HAR) [15]; and **(2)** Heart rate prediction (HRP) [4]. While other models can benefit even more from ZoneFL (e.g., vehicular traffic prediction), we chose these two models because we had access to suitable mobile sensing datasets for them.

TABLE I
ZoneFL vs. Global FL Performance

Application	Metrics	Global FL	Static ZoneFL	Improvement Gain
HAR	Accuracy (%)	65.27	69.63	6.67%
HRP	RMSE	21.20	19.86	6.74%

HAR Model. The dataset for this model has data from 51 users, moving in a region larger than 20,000 km^2. In the experiments, we start with 9 non-overlapping zones over the region covered by the dataset, based on GPS coordinates. The zones are diverse and include a university campus zone, several suburban residential zones, a riverside urban zone, a metro zone, etc. On average, each user have 1,995 data samples for each zone. The preprocessing and the CNN-based model architecture follow the work in [15]. We use accuracy as the main metric for model performance.

Heart Rate Prediction (HRP). The dataset for this model contains 167,373 workout records for 956 users in 33 countries. Among the countries, we select the top 6 countries having at least 10 zones with a reasonable number of average data samples per zone to assess ZoneFL's performance effectively. We use an LSTM-based model [4] to predict the heart rate given input features consisting of the workouts' altitude, distance, and time elapsed (or speed). We use the root mean squared error (RMSE) metric for this prediction task.

B. Model Utility Results

ZoneFL vs. Global FL. Table I shows the performance comparison between ZoneFL and Global FL. In this experiment, ZoneFL works only with the zones defined at the beginning of the experiment (called Static ZoneFL), without employing ZMS or ZGD to adapt the models to user mobility behavior over time. Thus, it provides a lower bound on ZoneFL's performance, which is expected to improve with ZMS and ZGD. Global FL trains with all the users in the datasets. Zone FL trains a different model for each zone in the respective dataset. Some users have data and participate in training in more than one zone. The metrics are computed per sample in the test data set and then averaged. ZoneFL models outperform the Global FL models by 6.67% for HAR and by 6.74% for HRP. This performance gain is significant given that it is very challenging for DL models to achieve even 1% improvement in HAR and HRP tasks, as illustrated in recent studies [4, 17]. As shown next, we observe further improvements with the dynamic adaptation algorithms.

ZGD Performance. Although both ZGD and ZMS adapt zone models to user mobility behavior changes, they serve different purposes. Hence, we present the performance of ZGD and ZMS separately. ZGD is designed to work with fixed zones that have enough data for training. ZMS is designed to adapt the zone partitions until all of them achieve reasonable model performance. In practical terms, ZMS is generally used for the beginning rounds of ZoneFL, while ZGD is used once the zone model performance is relatively stable. For both

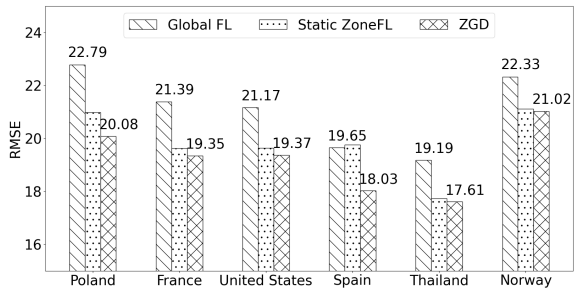

Fig. 4. Performance Improvements of ZGD for HRP.

TABLE II
PERFORMANCE IMPROVEMENTS OF ZMS FOR HRP.

	Before (RMSE)	After (RMSE)	Improvement Gain (%) Mean / SD	Occurrence Per 100 Rounds
Merge	23.79	21.44	9.87 / 3.11	4
Split	23.04	20.71	11.10 / 3.63	3

algorithms, we show just the results for HRP because its dataset is more suitable for dynamic adaptation, since it has a sufficient number of zones and users, while the HAR dataset does not.

Figure 4 shows the performance of ZGD for the top-6 countries in the HRP dataset. ZoneFL with ZGD performs better than Static ZoneFL for each country, and it clearly outperforms Global FL (by as much as 11.89% for Poland). We also observe that Static ZoneFL performs better than Global FL for 5 countries, and slightly worse for one country. The reason for the worse performance for Spain is that the static zones do not capture well the changes in user mobility behavior. ZoneFL with ZGD is able to alleviate this problem and results in better performance than Global FL.

ZMS Performance. Table II shows the average model performance improvement for zone merge and split in HRP. In merge, the improvement gain is calculated as follows: $\frac{L_1+L_2}{2} - L_{12}$, where L_1 and L_2 are RMSE losses evaluated on the two constituent zones, and L_{12} is RMSE loss computed on the merged zone. The reverse formula is used for splitting a larger zone in two sub-zones. The results demonstrate that ZMS can significantly improve the model performance. On average, 4 merges and 3 splits occur every 100 rounds of training, which shows that dynamic adaptation needs to happen about once a month in a scenario where users train once a day.

C. System Results

To showcase the feasibility and benefits of ZoneFL in a real-life deployment, we conducted an HRP field study with 63 users for 4 months. Along with phone sensor data, the users were tasked to collect heart rate data from a Bluetooth-connected heart rate tracking wrist band for daily activities. The region of the study is larger than 20,000 km^2, and it was originally divided in 9 zones. The study ran the prototype of ZoneFL with ZMS. In the field study, the ZMS split operation is performed for only one level ($l = 1$). The prototype worked reliably throughout the duration of the field study.

1) ZoneFL Feasibility on Smart Phones: We benchmarked ZoneFL with HAR and HRP on Android phones using a testing app to evaluate training and inference performance. We also assessed the resource consumption on the phones, with different specs (Nexus 6P and Google Pixels 3). The results demonstrate the on-device feasibility of ZoneFL, even for the Nexus 6P phone, unveiled in 2015 and running Android 7. Since ZoneFL works well on such a low-end phone, we expect ZoneFL to work well on most of today's phones.

TABLE III
TRAINING ON PHONES: RESOURCE CONSUMPTION AND LATENCY

Application	Phone	Max RAM Usage (MB)	Foreground Training Time Mean/SD (min)	Background Training Time Mean/SD (min)	Battery Consumption per Round (mAh)	Number of Training Rounds for Full Battery
HAR	Nexus 6P	232	15.21/2.89	59.99/4.06	53.86	64
	Google Pixel 3	228	2.13/0.24	9.32/0.09	9.91	294
HRP	Nexus 6P	266	3.09/0.39	10.97/1.08	33.18	104
	Google Pixel 3	230	0.40/0.10	5.07/0.37	4.66	625

TABLE IV
INFERENCE ON PHONES: RESOURCE CONSUMPTION AND LATENCY

Application	Phone	Max RAM Usage (MB)	Foreground Inference Time Mean/SD (millisecond)	Background Inference Time Mean/SD (millisecond)	Battery Consumption per prediction (μAh)	Millions of inferences for Full Battery
HAR	Nexus 6P	161	54.65/16.36	1963.04/1540.29	4.49	0.77
	Google Pixel 3	177	36.59/6.43	99.60/33.69	1.94	1.50
HRP	Nexus 6P	232	528.93/53.53	1809.71/700.96	45.47	0.08
	Google Pixel 3	229	167.71/6.83	669.88/112.01	5.74	0.51

Training Performance. Table III shows the ZoneFL training time and resource consumption on the phones. The training time is recorded by training on average 86 samples samples per zone per user in 5 epochs for HAR and HRP, respectively. Foreground training (screen turned on) provides a lower bound for the training time by using the full single core capacity. In reality, we expect training to be done in the background, while the phone is being charged. We take 10 measurements and report the mean and standard deviation.

Training for one round is fast on the phones. The foreground training time on Pixel 3 is just 2.13 min for HAR, and 0.4 min for HRP. The background training time is also good for any practical situation. The background training time is notably longer compared with foreground training, since Android attempts to balance computation with battery savings.

The results also show training is feasible in terms of resource consumption. The maximum RAM usage of the app is less than 266MB, and modern phones are equipped with sufficient RAM to handle it. The phones could easily perform hundreds of rounds of training on a fully charged battery. It is worth noting that, typically, one round of training per day is enough, as the users need enough time to collect new data.

Inference Performance. The results in Table IV demonstrate that ZoneFL can be used efficiently by third-party apps working in real-time. The measured time consists of the inference computation time and the inter-process communication time. We continuously perform predictions/classifications for 30 minutes and report the average values. The inference time for the two scenarios, foreground and background, follows a similar trend as training.

2) Scalability: ZoneFL utilizes multiple FL Zone Managers to receive and aggregate the gradients from the users. Compared with a single server in Global FL, the communication

TABLE V
SERVER LOAD IN ZONEFL VS. GLOBAL FL

Application	HAR	HRP
ZoneFL server load	37.26%	34.98%

TABLE VI
ZMS PERFORMANCE IN THE FIELD STUDY

Merge Time	Zone X/ RMSE	Zone Y/ RMSE	Merged Zone RMSE
2022-04-09 13:57	A/13.96	B/18.40	12.56
2022-05-29 12:53	C/44.53	D/11.86	10.84
2022-06-05 13:07	E/18.48	A/15.28	13.30
2022-07-29 21:56	F/17.40	G/39.23	14.78

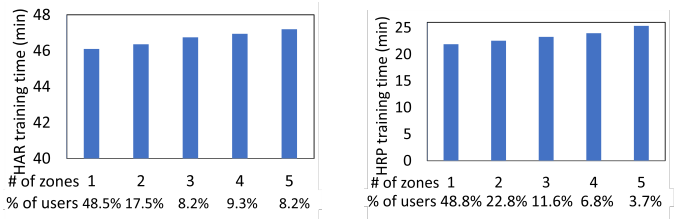

Fig. 5. User Training Time vs. Number of Zones in the Datasets.

and computation load in ZoneFL is distributed among multiple zone servers. Considering a user may send gradients to multiple zone servers, Table V computes the average ZoneFL server load savings based on the user percentage distribution over the number of zones in Figure 5. The results demonstrate ZoneFL scales better than Global FL because the server load is 34.98% to 37.26% of the one in Global FL.

3) ZMS Performance in the Field Study: Table VI depicts zone merge time and model utility gains in the field study. At the end of the field study, the number of zones changed from 9 to 7 after several merges and splits. For merge, the highest model utility gain observed is to improve RMSE from 44.53 to 10.84. This is because the original zones did not have enough users and data. For split, the highest RMSE improvement is from 16.38 to 11.20. These observations showcase the ZMS benefits in real-life.

4) ZoneFL User Training Time Overhead: In ZoneFL, the phones may train in multiple zones, which may introduce overhead when compared with Global FL. For every round in Global FL, a phone trains once for all its data. In ZoneFL, a phone may train once per zone from where it has data, but for a smaller fraction of data. Figure 5 illustrates the background training time when the phone trains on the same amount of data, while varying the number of zones from which the data is collected. The percentage of users shown under the X axis represents the fraction of users that have data in [1, 5] zones (e.g., 8.2% of users have data in 5 zones). The number of samples trained per zone follows the average reported in section V-C1. For the 49% of the users that have data in a single zone, there is no overhead compared to Global FL. For the rest of the users, we observe a small overhead, which increases with the number of zones but never exceeds 3.5 minutes. Considering that the training occurs in the background, this is an acceptable overhead for the benefits of ZoneFL in terms of model utility and server scalability.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed ZoneFL, a mobile-edge-cloud FL system, that distributes training across geographical zones to improve model utility and scalability. We augmented ZoneFL with two federated training algorithms, ZMS and ZGD, enabling zone models to adapt to changes in user mobility behavior. Using two FL models and mobile sensing datasets collected in the wild, we showed that ZoneFL outperforms traditional FL in terms of model utility and server scalability. We implemented an Android/AWS prototype of ZoneFL with ZMS and demonstrated the feasibility of ZoneFL in real-life conditions. As future work, we will investigate how ZGD and ZMS work together to further improve model performance.

REFERENCES

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282, 2017.

[2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE IoT Journal*, 5(1):450–465, 2017.

[3] MG Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *arXiv preprint arXiv:1908.00080*, 2019.

[4] Jianmo Ni, Larry Muhlstein, and Julian McAuley. Modeling heart rate and activity data for personalized fitness recommendation. In *WWW*, page 1343–1353, 2019.

[5] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[6] Xiaopeng Jiang, et al. Federated meta-location learning for fine-grained location prediction. In *IEEE Big Data*, pages 446–456, 2021.

[7] Anliang Li, Shuang Wang, Wenzhu Li, Shengnan Liu, and Siyuan Zhang. Predicting human mobility with federated learning. In *ACM SIGSpatial*, pages 441–444, 2020.

[8] Sannara EK, François PORTET, Philippe LALANDA, and German VEGA. A federated learning aggregation algorithm for pervasive computing: Evaluation and comparison. In *IEEE PerCom*, pages 1–10, 2021.

[9] Riccardo Presotto, Gabriele Civitarese, and Claudio Bettini. Fedclar: Federated clustering for personalized sensor-based human activity recognition. In *IEEE PerCom*, pages 227–236, 2022.

[10] Yang Qin and Masaaki Kondo. Mlmg: Multi-local and multi-global model aggregation for federated learning. In *IEEE PerCom Workshops*, pages 565–571, 2021.

[11] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *ICML'21*, pages 6357–6368.

[12] Q. Cui, Z. Gong, W. Ni, Y. Hou, X. Chen, X. Tao, and P. Zhang. Stochastic online learning for mobile edge computing: Learning from changes. *IEEE Communications Magazine*, 57(3):63–69, 2019.

[13] Mondher Bouazizi, Chen Ye, and Tomoaki Ohtsuki. Low-resolution infrared array sensor for counting and localizing people indoors: When low end technology meets cutting edge deep learning techniques. *Information*, 13(3):132, 2022.

[14] He Li, Kaoru Ota, and Mianxiong Dong. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[15] Xiaopeng Jiang, et al. Flsys: Toward an open ecosystem for federated learning mobile apps. *IEEE Transactions on Mobile Computing*, pages 1–18, 2022.

[16] Polar. SDK for Polar sensors. https://github.com/polarofficial/polar-ble-sdk, 2022.

[17] Kaixuan Chen, Dalin Zhang, Lina Yao, Bin Guo, Zhiwen Yu, and Yunhao Liu. Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities. *ACM Computing Surveys (CSUR)*, 54(4):1–40, 2021.