WiFederated: Scalable WiFi Sensing using Edge Based Federated Learning

Steven M. Hernandez *Student Member, IEEE,* and Eyuphan Bulut, *Senior Member, IEEE,*

***Abstract*—WiFi sensing using Channel State Information (CSI) offers a device-free and non-intrusive method for human activity monitoring. However, the data-hungry and location-specific training process hinders its scalable deployment at large sizes. In this work, we propose *WiFederated*, a federated learning (FL) approach to train machine learning models for WiFi sensing tasks. Using WiFederated, client devices can not only perform training in parallel at the edge instead of sequentially at a central server but can also collaboratively learn and share generalizable location-independent traits about physical actions being monitored. We demonstrate that an FL model trained on as few as 2-3 locations can provide high prediction accuracy in new locations even without any data available from them. We also demonstrate how new locations can achieve higher prediction accuracy even with a small number of available samples when using the pretrained FL model rather than training from scratch. The results show that the FL model can save local training epochs and reduce the need for large data collection at each new location. Thus, the proposed WiFederated system scales as more locations are added. We show that WiFederated provides a more accurate and time efficient solution compared to existing transfer learning and adversarial learning solutions thanks to the parallel training ability at multiple clients. By introducing new client selection methods during the FL process, we also show that accuracy can further increase. Finally, we evaluate the feasibility of training models at the edge and introduce continuous annotation to allow for continuous learning over time.**

***Index Terms*—WiFi sensing, federated learning, channel state information (CSI), device-free sensing.**

I. Introduction

WIFI sensing [1]–[3] aims to capture and analyze ambient WiFi signals to understand physical characteristics of a given environment. New applications of WiFi sensing have quickly gained popularity as it allows for low-cost (by leveraging existing WiFi infrastructure), device-free, and non-intrusive sensing of human presence and physical activities compared to sensor-based systems which typically require dedicated devices to be placed on the body. By leveraging existing infrastructure, it also adds more value for the owners of WiFi devices in several ways such as by providing physical occupancy analytics in retail shopping areas or other public buildings [4]–[6], reinforcing security through intrusion detection [7] and monitoring patients [8], [9] to protect from falls or other serious injuries.

Leveraging existing WiFi access points (AP) for sensing purposes provides a remarkable advantage compared to sensor-based methods by reducing costs associated with both hardware and labor for deploying the new sensors. However, typical WiFi sensing systems require large amounts of data to train their deep learning models [10]–[13], thus it is very cumbersome to build such systems. As the number of actions to be recognized increases for the model, the required amount of training data also increases. Moreover, the models produced in most existing works are specialized to a single given physical location. Thus, the physical actions must be performed over many repetitions for each individual location to train a model for it. This is time-consuming and impractical especially in cases where the locations are already occupied such as in the case of patient monitoring in hospital settings.

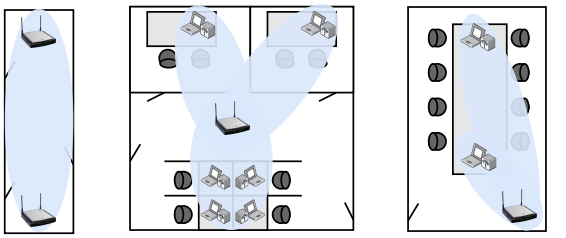
Consider the three physical environments illustrated in Fig. 1a-c for a typical office building. Each location has unique environmental properties (e.g., size, environmental clutter such as furniture), thus the characteristics of the signals received in one location will vary from that of each of the other locations. For example, in a less cluttered hallway environment such as in Fig. 1a, very few physical obstructions block the transmission path between WiFi devices. In some environments, the WiFi devices can only receive signals through the wall when performing sensing tasks such as in the private office rooms shown in Fig. 1b. Additionally, some other environments may have very dynamic physical features such as meeting rooms as shown in Fig. 1c where furniture like chairs and tables are constantly shifted around. As such, the core issue with deploying WiFi sensing systems is to develop generalizable models that can be used in new locations without requiring complete and extensive data recording and annotation steps for each new environment. Fig. 1d shows a typical WiFi sensing system diagram where for each environment, a technician must repeat many trials of each physical action before sending the manually labelled data to a server. Once at the server, the data must be run through data cleaning and preprocessing steps followed by extensive hyperparameter selection steps to determine the best model parameters for the given environment. After fully training, the location-specific model can finally be shared back to the edge for model inference.

There are some recent studies [14]–[18] looking into the creation of generalizable models that can also be used in new environments, however these studies mostly focus on creating

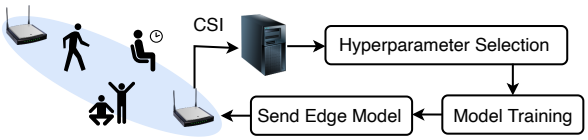
S. M. Hernandez and E. Bulut are with the the Department of Computer Science, Virginia Commonwealth University, Richmond, VA, USA (e-mail: hernandezsm, ebulut@vcu.edu).

This work is supported in part by National Science Foundation (NSF) Graduate Research Fellowship Program (GRFP) under Grant No. 1744624, Virginia Commonwealth University Presidential Research Quest Fund (PeRQ) and Commonwealth Cyber Initiative (CCI). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Manuscript received XXX XX, XXXX; revised XXX XX, XXXX.



(a) Hall (b) Offices and cubicles (c) Meeting Room



(d)

Fig. 1: (a)-(c) WiFi sensing environments in an office building. (a) Less cluttered environment. (b) Highly cluttered environment with through-wall sensing. (c) Highly cluttered dynamic environment. (d) Typical WiFi sensing system diagram.

a single model upfront rather than in designing a system which allows collaboration between new environments. Additionally, these studies still require a long duration of data collection and training from a large number of different locations. Instead, *our goal here is to develop a collaborative training framework for WiFi sensing that shares knowledge learned from one set of locations to improve the prediction capability of models at other new locations.* To account for this goal, we propose *WiFederated*, a system for collaboratively training machine learning models for WiFi sensing tasks at multiple unique environments using federated learning (FL) [19]. Through WiFederated, WiFi sensing tasks can scale to multi-location settings by (i) reducing the overall amount of annotated training data required per location; (ii) reducing the duration of training required for each new location; (iii) allowing for parallel edge training across multiple locations; and (iv) reducing the amount of data to be transmitted to a central server. Our contributions through this work are the following:

1) We perform human activity recognition experiments in ten locations and demonstrate that a model trained on data from one set of locations will not necessarily be generalized to new locations.
2) We propose *WiFederated*, a framework which introduces federated learning for the first time with WiFi sensing allowing for distributed model training across client devices within the network.
3) We evaluate WiFederated through extensive experiments and show that it outperforms the existing solutions.
4) We propose and evaluate client selection methods which select a subset of candidate locations based on local training-loss, resulting in a further increase in accuracy.
5) Lastly, we evaluate the feasibility of running WiFederated on real edge devices for inference and training and also introduce *continuous annotation* which allows clients to continue to capture representative CSI data and annotation labels for further model training over time.

The rest of the paper is organized as follows. Section II describes related work in WiFi sensing and reviews state of the art methods proposed for scalable multi-location WiFi sensing. In Section III, we provide the preliminaries of the proposed work including an introduction to Channel State Information (CSI), preprocessing steps used on the collected CSI data and the description of the underlying machine learning models used in this work. In Section IV, we demonstrate through experimental results the issues related to WiFi sensing across multiple locations to provide motivation for this study. Then, in Section V, we detail the design of our FL framework for WiFi sensing and provide its evaluation and comparison with existing works in Section VI. Section VII details some additional feasibility considerations for deploying our proposed system in real world scenarios. Finally, we provide our concluding remarks in Section VIII.

II. Related Works

Due to the non-intrusive and low-cost nature of WiFi sensing, it has been used in several applications including human activity recognition [14], [20], human body pose prediction [21], [22] and patient monitoring [9]. For example in patient monitoring, hospitals and elderly care facilities use WiFi sensing to detect falls without having the burden of attaching sensors on the body of patients. WiFi sensing is also used to detect fine-grained patient vital signs such as breathing rate and heart rate variability [23]. While a wide spectrum of studies using WiFi sensing can be found in recent surveys [1], typically these research studies consider only the case where a single model is pretrained for only one specific location.

In most of the current WiFi sensing research, experiments are designed such that a single WiFi transmitter (TX) and a single WiFi receiver (RX) are placed statically in a single physical location. Once placed, the set of actions to be detected are then performed over many repetitions to gather a significant distribution of signal variations per action. There are two core issues in this commonly adopted approach in WiFi sensing. *The first issue is*, because the CSI collecting devices are typically laptops with an attached Intel 5300 Network Interface Card (NIC) [24], the receivers can only be placed at limited locations which could be different than real world deployments of WiFi APs (e.g., ceiling in a hallway). Thus, some experiments might have been designed with the placement of TX and RX devices in unrealistic positions which give the best results, providing an unfair advantage when making predictions. Moreover, such configurations may result in models that may not work well when existing WiFi infrastructure and real world deployments of WiFi APs are used. *The second issue* is that the model which is trained with TX and RX at a single physical location only has knowledge of that one location. Thus, its predictions may not be of any use in new locations and we may need to repeat all of the data collection, annotation and training steps for each new location from scratch as it is done in [13], [25]. However, this is ultimately not scalable because each location then requires a time-consuming and potentially error-prone annotation step.

Study	Recognition Task	Learning Type	Multiple Rooms	Distributed Training	Knowledge Sharing
CrossSense [14]	Activity	Multiple Global Expert Models	Yes	Possible	No
Ma et al. [15]	Activity	RNN, LSTM, State Machine	No	No	No
EI Framework [16]	Activity	Adversarial Network Model	Yes	No	No
WiHand [17]	Gesture	LRSD Decomposition	No	No	No
WiTransfer [18]	Activity	Global Transfer Learning	No	No	Yes
WiFederated (Ours)	Activity	Distributed Federated Learning	Yes	Yes	Yes

TABLE I: Comparison of related multi-location WiFi sensing literature.

In some recent works, the problem of developing WiFi sensing solutions that can be generalized to multiple environments has been studied. For example, in CrossSense [14], first, a set of expert models are pretrained at some initial locations. For each new location, the best fitting model out of all of them is used. However, the number of expert models required could be large depending on the environments, requiring once again a large collection of annotated samples and a long training process. Moreover, as the expert models do not collaborate, they cannot gain any information from each other. The work in [15] similarly looks at location-independent activity recognition through the use of deep learning techniques such as recurrent neural networks (RNN) and long short-term memory (LSTM) networks. However, using such complex models requires a long training duration (e.g., weeks) and results in relatively low prediction rates (e.g., 50Hz) even with powerful GPUs. In this study, we aim towards making predictions at the edge with devices having limited computation and power capabilities, thus GPUs cannot be used and much lower prediction rates are expected. The *EI Framework* [16] uses an adversarial network approach which trains three individual sub-networks. The feature extractor network transforms the raw data which is then fed into both an activity recognizer network and a separate domain discriminator network which predicts the location that a given sample comes from. The goal is to optimize the weights in the networks such that the features extracted by the feature extractor increase the accuracy for the activity recognizer while also decreasing the accuracy of the domain discriminator. The EI framework is expected to perform better as more locations are added because more samples will be available for the feature extractor to learn from, however because the activity recognizer and domain discriminator are working against one another, as more locations are added more training epochs will also be required. The *WiHand* [17] hand gesture recognition system suggests using low rank and sparse decomposition (LRSD) to split raw CSI data into two distinct parts, namely, a low rank part containing background information and a sparse part containing noise due to hand gesture movements. While the work considers multiple positions, each evaluated position is located in the same room which does not demonstrate how the model is applicable in physically unique environments. Finally, *WiTransfer* [18] suggests the use of transfer learning to pretrain a global transfer model which can then be used at a new location with minor personalization training performed using data from that location. However, because the model is trained globally at a central location, increasing the number of pretraining locations increases the time to fully train the model as the training must occur sequentially on the aggregated data.

In this study, we address the issues of these centrally trained methods by training in parallel through an FL based approach. FL has become a popular topic in the research community due to the ability to train machine learning models in parallel at the edge while also preserving data privacy [19]. Details of the FL process are discussed in Section V. In the past, FL has most commonly been evaluated on standard machine learning tasks such as handwriting recognition with the MNIST dataset and sentiment analysis with the Sentiment140 dataset [26] but has also been used in real-world applications such as improving next-word suggestions on smartphone keyboards [27] as well as leveraging electronic medical records from multiple hospitals to better predict medical conditions without compromising data privacy [28]. By using FL for WiFi sensing, we aim for a more scalable method of training models across disparate devices in unique physical locations while reducing the number of annotated data samples required per location and still achieving a higher prediction accuracy. Table I provides a comparison of the discussed works with our proposed WiFederated approach.

III. Preliminaries

For the experiments in this work, we use our ESP32-CSI Toolkit[1] [3] to collect CSI from the WiFi-enabled ESP32 microcontroller, which provides a small-size, lightweight and standalone solution compared to other existing methods (i.e., a host laptop with an updated NIC) and can be deployed anywhere. We transmit frames from an ESP32 transmitter and then collect WiFi CSI from a separate ESP32 receiver. CSI is a metric coming from orthogonal frequency-division multiplexing (OFDM) systems which allow multiple streams of data to be transmitted in parallel to increase the speed of data transfer by splitting the bitstreams into multiple subcarrier frequencies at transmission and then reconstructing the bitstreams into one cohesive message at the receiver. OFDM is used with 802.11 based WiFi systems to achieve high data throughput rate as well as to reduce multipath fading.

A. CSI Amplitude and Phase

When a TX transmits a signal G, it is received by the RX as $H = \quad G \ , \ [$, where $[$ is a vector representing environmental noise and $\quad$ is the CSI matrix. Each element in $\quad$ is composed of both real and imaginary components. Given the real component ($^{180}_{A}$) and imaginary component ($^{180}_{8<}$) for each subcarrier 8 within $\quad$, we can calculate the amplitude $^{180} = \quad \sqrt{^{180\ 2}_{A} \ , \ ^{180\ 2}_{8<}}$ as well as the phase $q^{180} = 0\mathbb{C}0{=}2 \quad ^{180}_{8<} - \ ^{180}_{A}$. For our evaluations, we only consider

[1] https://github.com/StevenMHernandez/ESP32-CSI-Tool

amplitude because of the noise inherited in the received phase values as well as the requirement for a synchronized clock between the two devices.

B. Rolling Mean Filter

CSI amplitude measurements can have spurious noise which is unrelated to any physical movements in the environment as a result of variations caused by the physical radio hardware as well as other environmental noise. Denoising can be used to filter out such noise in CSI measurements. We apply a rolling mean denoising approach, which takes a window of size , samples, and applies the mean function over the previously collected , 1 samples along with the currently collected sample across a single subcarrier. More formally, we calculate

$$\hat{} = \frac{1}{,} \sum_{F=0}^{,\tilde{O}1} F^{\circ} \bullet \quad (1)$$

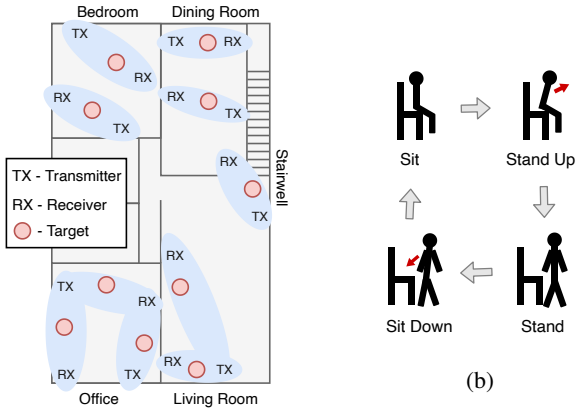
Assuming that we have 64 subcarriers, we then must keep a record of the previous , 1 values for each of these 64 subcarriers independently in memory on the edge device.

C. Machine Learning Training and Prediction

Using the preprocessed CSI data, we then train a machine learning model using a Dense Neural Network (DNN) architecture with three dense layers. Each layer has 100 hidden units and each unit uses a Rectified Linear Unit (ReLU) activation function. Each layer uses $!_2$ regularization for the kernel weights as well as an $!_1$ regularization penalty term for the activation output of each layer. The size of the input matrix for the model is 100 64, where 64 is the number of subcarriers and 100 is the number of CSI frames which is approximately one second worth of data since our sampling rate is set to 100Hz. The network terminates with a softmax multi-class output layer so that we can make predictions on multiple class types with a single network architecture. During training, we use a dropout value 3 2 $^{1}0\text{-}1^{o}$ between each layer to define the probability that a given weight is ignored during training to help prevent the model from overfitting to the training data. Dropout is ignored during model evaluation. We use Stochastic Gradient Descent (SGD) with learning rate $[= 10^{-5}$ to minimize the loss function

$$L^{1}\backslash\text{-}G\text{-}H^{o} = \frac{1}{\#} \sum_{8=1}^{\tilde{\Theta}} {}^{1}F_{\backslash}{}^{1}G_{8}{}^{o} \quad H_{8}{}^{o2}\text{-} \quad (2)$$

where $\backslash$ is the set of parameters for the model (F), $F_{\backslash}{}^{1}G^{o}$ is the prediction output of the model given input G, and H is the expected output based on the given action class which is a one-hot encoded vector of size #, where # is the number of possible predicted classes. The overall architecture is designed with edge devices in mind. This is why we do not consider more complex models such as the popular RNN or LSTM network architectures which would require more training data and longer training periods.



(a) (b)

Fig. 2: (a) Illustration of apartment environment where experiments are performed. TX, RX and human target are shown for each room location. (b) Four distinct actions (i.e., sit, stand up, stand and sit down) are recorded and annotated in each location in round-robin order.

IV. Motivation

To demonstrate our motivation for this work, we begin by explaining the experiments that are performed during our data collection and annotation process. We then review initial performance results on the collected data when using the standard training approach common to many other WiFi sensing research as illustrated in Fig. 1d.

A. Experimental Setting

In this study, we consider a collaborative network of WiFi sensing devices in different and disconnected physical locations. The illustration in Fig. 2a shows the ten distinct areas that we record experiments at as well as the placement of the TX, RX and the human target. Each location has different positions for the TX and RX corresponding to the preexisting power outlets which were built into the building which provides a natural selection of locations for TX/RX rather than selecting the most optimal locations for performing the sensing tasks. The goal here is to demonstrate that the system can leverage existing infrastructure. The target performs four actions (sitting, standing up, standing, sitting down) as illustrated in Fig. 2b. In each location, we perform 50 individual repetitions of each of the four actions in round-robin order resulting in a total of 2-000 annotated action-segments with corresponding CSI samples. For each location, we designate the first 25 repetitions for training our models and the final 25 for evaluating. This allows us to check if the traits learned by the model are generalized by evaluating the accuracy on these final 25 testing repetitions which are never seen when training the model.

B. Initial Results

The goal of this work is to demonstrate how collaborative WiFi sensing devices can achieve better prediction results compared to devices that work alone. This can be especially

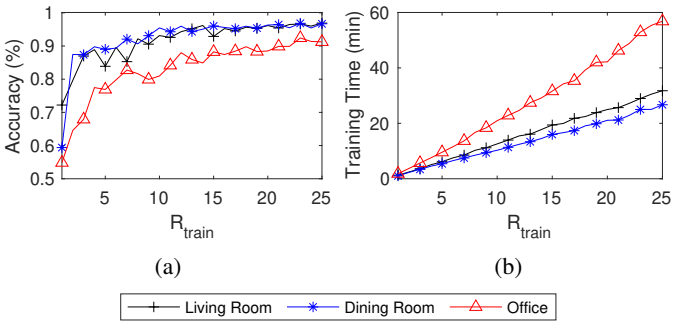


Fig. 3: (a) Accuracy for each locally trained model when different numbers of training repetitions of an action are performed in the location. (b) Training time to perform 100 epochs of training on a Raspberry Pi Edge device.

important when we aim to use preexisting WiFi infrastructure to build a WiFi sensing system [29]. In a non-collaborative system of WiFi sensing devices, we will need to record and annotate multiple new repetitions of each action before a local model can perform well in the environment. For example, Fig. 3a shows the accuracy of a non-collaborative local model with different numbers of training repetitions ($'_{train}$). We can see that training this local model on a small number of training repetitions produces a model which is overfit onto the training data and is unable to generalize to achieve high validation accuracy. Given our goal of large-scale deployments, performing a large number of repetitions at every location is not viable because it would increase the time and labor spent at each location. Similarly, this also increases the time to train the model as shown in Fig. 3b where we observe mostly a linear relation between the training time and the number of repetitions of each action when trained on a Raspberry Pi 4 model B edge device.

Training independent local models at each location without some collaboration mechanism means that we will not gain additional knowledge by adding new devices into the network. To account for this, an initial naïve approach is to collect and annotate CSI data at a few selected locations (we will denote this set of training locations as $\hat{!}$ throughout the paper) which we use to globally train a single machine learning model that will then be shared with all locations ($!$) including those which have no collected or annotated CSI data.

In Fig. 4 we show the prediction accuracy of our model ($F_{\setminus}$) when evaluated on each location $!_{\beta}\ 2\ !$, where $!$ = fLiving Room- Dining Room- Officeg, after being trained on CSI data from different sets of $\hat{!}$ $!$. We select these three locations because they represent different multipath characteristics from one another. For example, the living room location provides a large open area with a low amount of environmental clutter; the dining room location offers a similar large environment but with higher amounts of environmental clutter; and finally the office location represents a smaller enclosed room environment with low clutter. We evaluate with the number of training repetitions $'_{train}\ 2$ f10- 25g to identify how increasing $'_{train}$ affects the prediction capability of locations in $\hat{!}$ as well as locations not in $\hat{!}$ (i.e., $!$ $\hat{!}$).

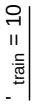
In Fig. 4a, we consider the case when $F_{\setminus}$ is trained on the Living Room location only. As we would expect, because the model is trained directly on data from this location (i.e., only first 25 repetitions), we can see that for both $'_{train} = 10$ and $'_{train} = 25$, the accuracy (on the last 25 repetitions) for the Living Room is high at 92•82% and 96•30%, respectively. Interestingly, we can see that increasing $'_{train}$ from 10 up to 25 also allows for an increase in prediction accuracy for two unseen locations, namely Dining Room going up from 76•76% up to 83•90% and Office going from 59•49% up to 65•98%. However, we can still see a noticeable gap between the accuracy for each of these locations demonstrating that the model is still better fit to the data at the Living Room location.

Following this, in Fig. 4b we train only on the Dining Room location. When $'_{train} = 10$ both the Living Room location (seen during training) and Dining Room (unseen during training) achieve very close accuracy after 100 epochs of training. This shows that the model trained on the Dining Room location can also be used in a completely unseen new location. However, we can see that the accuracy for the Office location is still low at 65•38% demonstrating that even though the model is applicable to some locations, it is not necessarily a generalizable model that can be applied in all new locations.

On this note, we look at the case in Fig. 4c where we train only on the Office location. While the model is able to achieve a high evaluation accuracy for the location that it is trained on (i.e., Office), the model cannot be used in new and unseen locations. Comparing to Fig. 4a where a slight improvement is observed in unseen locations, the model trained exclusively on the Office location dataset is only able to achieve between 48•12% and 58•76% accuracy in the unseen locations with a gap of approximately 44% difference between the accuracies in the seen and unseen locations.

Training our model on data from two locations rather than one produces Fig. 4d-f. The model accuracy in the unseen locations mostly converge to a static value except in Fig. 4e when $'_{train} = 25$, where the gap in accuracy between seen and unseen locations grows as training continues. In Fig. 4e, when $'_{train} = 10$, the gap is only ¸10•04%, but when $'_{train} = 25$, the gap increases significantly to ¸18•05%. Out of each of these three cases where j $\hat{!}$ j = 2, Fig. 4f shows the largest overall gap of ¸21•04% when $'_{train} = 10$ and ¸24•53% when $'_{train} = 25$. This again demonstrates that a model with high accuracy on trained locations may not be generalizable to unseen locations.

As such, we can only be sure that a high accuracy is achievable across each location when all locations are involved in training (i.e., $\hat{!} = !$) as we can also see from the results in Fig. 4g. However, for global models, annotated data needs to be shared with a central server for all devices. For example, if CSI data and annotations are collected continuously over time such as in [21], [30], [31], the amount of data transmitted to the server can be large. As the network of locations also increases in size, the server resources used to handle all of the incoming data may be too much. Thus, it would be more preferable to do training on an edge device at the physical location. However, to accomplish this, we will need a new method for sharing knowledge between multiple locations which reduces the amount of data transmitted over the network.



(a) !^ = f L.R.g (b) !^ = f D.R.g (c) !^ = f Off.g (d) D.R. 8 !^ (e) L.R. 8 !^ (f) Off. 8 !^ (g) !^ = !

Fig. 4: Prediction accuracy in three locations (Living Room (L.R.), Dining Room (D.R.), Office (Off.)) when trained with data from only locations in!^. Columns (a-c) train on a single location, (d-f) train on two locations and (g) trains on all locations.

V. Federated Learning Framework (WiFederated)

In order to develop a system which can collaboratively train a machine learning model in parallel across many edge devices (or clients) in disparate physical locations, we propose a collaborative WiFi sensing framework using federated learning [19], which we call WiFederated. FL is useful in our case because we expect that each client will have a different data distribution corresponding to their unique physical environment, and we want to benefit from all data while also reducing the amount of data shared to any central server so as to reduce network usage. This also has a secondary purpose in allowing for massively parallel machine learning by enabling each client to perform machine learning locally rather than at the central server.

Our overall goal is to find model weights \ (minimizing:

$$\min_{\backslash \in \mathbb{R}^{3}} \frac{1}{=:} \sum_{:=1}^{\tilde{O}} \sum_{G_8 - H_8 2\%_:} L^{1} \backslash - G_8 - H_8 - \quad (3)$$

where ! is the set of the clients at diverse physical locations, %: = f x: –y: g is the set of annotated data points for a client where x: is the set of CSI input and y: is the corresponding set of annotated labels recorded at the location of client such that =: = j%: j, and L1 o is a loss function as described in (2).

The issue with this optimization is that j!j, the total number of clients, is expected to be large which will result in a high computation and communication cost and thus slower model training. Furthermore, some locations may have either small amount of data or low quality of data which will only poison the prediction quality for other clients. To combat this, our proposed FL system selects a subset of clients !^ (!) to iteratively update the model weights. For each client 2 (!^), we learn unique model parameters \: (by optimizing:

$$\min_{\backslash_: \in \mathbb{R}^{3}} \frac{1}{=:} \sum_{8 \in \%_:}^{\tilde{O}} L^{1} \backslash_: - G_8 - H_8 \bullet \quad (4)$$

After #epochs of training per client, each client sends \: to a central server to perform Federated Averaging (FedAvg) [19] to determine new model parameters for the next round, \ 1 (:

$$\backslash^{1A, 1^o} = \backslash^{1A^o} , \sum_{:=1}^{\tilde{O}} \frac{=:}{=} \backslash_: - \quad (5)$$

where \: is gradient change over top of \ 1A^o learned from the data available to client. Whenever new clients join the network, they can use the pretrained federated model parameters (\ 1A^o) for the given round as-is if no additional labelled data points are available for the locations or alternatively can perform #post-epochs additional training epochs (i.e., model personalization) on top of the federated model to better fit to their own unique data distribution for the client.

Note that the FL approach is different from transfer learning [32] in which a single parent model is trained over a large number of training epochs on a large amount of annotated data, typically on a powerful computation system. The goal of transfer learning is to create a model which can be reused in a new somewhat related task by using a small amount of additional data from the new task. While FL gains from this same idea, the goal of our FL framework is not to just reuse a pretrained model for some new task, but instead to allow many disparate clients to massively train a model in parallel from scratch so that new clients can gain directly from this pretrained model upon joining the network. Furthermore, transfer learning requires all data to be located at a single central location for training, thus removing the parallel training capabilities found with FL systems. While some WiFi sensing studies utilize transfer learning as mentioned in Section II, to the best of our knowledge, FL based training has not been proposed for WiFi sensing tasks.

Algorithm 1 shows the steps of the proposed WiFederated learning model. ! is the set of locations where we aim to perform WiFi sensing and is our global federated model where •, is the set of weights for all layers of the model. can potentially change over time as new locations are added or as locations are removed due to battery power loss or disconnection from the network. Note that in the algorithm, we only perform a predetermined number of rounds, #rounds, before terminating, however federated training can be per-

Algorithm 1: WiFederated Learning

Input: Set of all WiFi sensing locations $\mathcal{L}$.
Global model $\bullet_g$ where $\bullet_g$ is the set of weights for the model.
Local model $\bullet_l$ with weights $\bullet_l$, $\forall l \in \mathcal{L}$.

$\bullet_g$ = Initialize Global Model
for all $r \in \{1, \ldots, \#_{rounds}\}$ **do**
  for all $l \in \mathcal{L}$ **do** // Share weights.
    $\bullet_l \leftarrow \bullet_g$
  $\mathcal{L}' \leftarrow$ Select Clients($\mathcal{L}$)
  for all $l \in \mathcal{L}'$ s.t. $\mathcal{L}' \subseteq \mathcal{L}$ **do** // In parallel.
    Train($\bullet_l$, $\#_{epochs}$, η_{train})
  $\mathcal{L}'' \leftarrow$ Select Clients($\mathcal{L}'$)
  $\bullet_g \leftarrow \frac{1}{|\mathcal{L}''|} \sum_{l \in \mathcal{L}''} \bullet_l$ // FedAvg.
for all $l \in \mathcal{L}$ **do** // Post-Train.
  Train($\bullet_l$, $\#_{post\text{-}epochs}$, η_{post})

formed continuously to allow the network to recognize newly annotated data over time. At the beginning of each round, the weights of the global model are shared with each location so that they can each begin from a similar starting point. Within each round, we select our set of clients ($\mathcal{L}'$) to train locally for some fixed number of training epochs ($\#_{epochs}$). This step can be thought of as a model-personalization step because at the beginning of the round, the model is initially fit on the global distribution of CSI data and at the end of the round, the models at each selected client are slightly more personalized to their own distribution of data. It is important that we do not perform too many training epochs locally during each round because then the local models will become more over fit onto location-specific data. Additionally, high $\#_{epochs}$ will result in over-utilization of individual devices which will result in a higher power consumption for these edge devices. Moreover, as mentioned in [33], [34], if we perform federated averaging on models which are fit to very different distributions, the resulting federated averaged weights will not be fit to either of the distributions of location-specific data but will also not be fit to the global distribution of CSI data. Thus, as long as $\#_{epochs}$ is not too large, at the end of each round, the FedAvg step will help prevent the global federated model from diverging too far away from fitting onto the global CSI data distribution.

To understand how federated learning behaves with varying amounts of training samples, we also limit the number of training-repetitions to η_{train} for each location. Furthermore, after the personalization step, we may decide to further refine our selected clients down to $\mathcal{L}''$ before applying FedAvg. This can be important because we may find that the weights proposed by the initially selected clients may cause the federated model to converge in a negative way. Finally, after training our model over $\#_{rounds}$, each location in $\mathcal{L}$ can perform $\#_{post\text{-}epochs}$ of post-training using η_{post} repetitions. While this step will not produce completely new models per location, it will be able to take the generalizable features discovered through the federated learning process and alter the model weights slightly to give better location-specific results.

Fig. 5: Illustration of one round of the WiFederated system.

The illustration in Fig. 5 demonstrates the six key steps taken for performing FL in our WiFederated framework on multiple locations and a central server. In the illustration, l_1, and l_2 are selected to train locally and then share their weights to the central server. Each location only transmits the new model weights rather than sharing the actual CSI data and annotation labels through the network. This means that we can keep a constant bound on the amount of data transmitted over the network by selecting an appropriately designed and sized machine learning model architecture. If we were to instead share annotated CSI data, then the size of the data could be unbounded especially in cases where our devices are able to self-annotate. Once the server receives the weights from each client location $l_k \in \mathcal{L}'$, the server can perform another round of client selection to select $\mathcal{L}'' \subseteq \mathcal{L}'$ before performing the federated averaging step. This second client selection phase allows the server to filter out the weights received from locations that are deemed to be lower quality based on some metric such as loss. After the federated averaging step is performed, we end the loop by resharing the central model to each of the locations. After performing these steps over subsequent rounds, any location can perform a final set of personalization training epochs on their locally available data. This step takes the shared federated model and performs a final number of personalization training epochs which can be useful for totally new locations or locations with few training locations. The key here is that this can be performed without requiring the location to share the results back to the central server. The goal accomplished by this framework is that new locations can achieve higher prediction accuracy by using the federated model as a base rather than starting from scratch.

VI. Evaluation

In this section, we evaluate the proposed WiFederated learning framework for use in the setting described in Section IV-A.

A. Impact of Averaging Interval

Two types of clients exist in our system, clients participating in federated averaging (i.e., the clients in $\mathcal{L}'$) and clients with data that is unseen during the training phases (i.e., the clients not in $\mathcal{L}'$).[2] We first look at how the federated averaging process affects the prediction capability of the locations

[2] For the initial evaluations in this section, we say $\mathcal{L}'' = \mathcal{L}'$. Client selection methods for selecting $\mathcal{L}''$ are further evaluated in Section VI-F.

participating in training the federated model. To evaluate this, we set $\hat{\omega} = \omega = \{\text{Living Room}, \text{Dining Room}, \text{Office}\}$ and evaluate on each of these clients individually. Fig. 6a shows the prediction accuracy when the number of epochs per round $\#_{epochs} = 50$. Initially, for all three locations, the accuracy remains exactly the same between the local model and the federated model. This is because, up until epoch 50, the models are the same; no federated averaging has occurred. However, after 50 epochs, a sudden dip in accuracy is found for the federated models. This dip is expected because at the end of this epoch, the first round of local training has concluded and the three local models are aggregated together through the federated averaging step. The key observation is that the federated averaging step aims to create model parameters which are generalizable to many locations rather than specialized to any one location. On the other hand, when we train a local model solely on the data available at a single location, the model will be better fit to the distribution of data at the given location, but it will be too specialized to aid other new locations when they join the network. The goal for all of the training clients in $\hat{\omega}$ is to sacrifice some amount of predictive capability so as to benefit other new locations. Even so, we can see in Fig. 6a that after the sudden dip caused by the FedAvg step, the accuracy quickly returns to a similar accuracy as we would see if we had simply trained the local model. This shows that the federated averaging step does not cause the accuracy to deteriorate too much for the participating locations. Fig. 6b shows the accuracy comparison between locally trained models and federated models during the final FedAvg aggregation. We can see that as $\#_{epochs}$ increases, the accuracy for the Office location decreases indicating that if we set $\#_{epochs}$ too large, the FedAvg step has a big impact on the prediction capability of the model at that location. On the other hand, because we assume a fixed total budget of $\#_{budget} = \#_{epochs} \cdot \#_{rounds} = 100$ epochs, increasing $\#_{epochs}$ decreases the number of rounds and thus consequently decreases the network communication. This is because at the end of each round, $|\hat{\omega}|$ locations must first communicate γ back to the central server for FedAvg aggregation and then the server must communicate $\gamma^{1}, \gamma^{2}, \ldots, \gamma^{|\hat{\omega}|}$ back to $|\omega| \cdot |\hat{\omega}|$ clients.

Thus, we must find a balance between communication overhead and prediction accuracy. To this end, for the following experiments, we set $\#_{epochs} = 10$ which gives a minor decrease in prediction capability when compared to $\#_{epochs} \in \{1, 2\}$ but greatly reduces the amount of communication required.

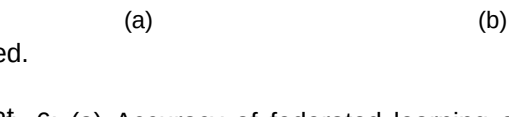


Fig. 6: (a) Accuracy of federated learning over 100 epochs when $\#_{epochs} = 50$ and $\hat{\omega} = \omega$ versus local machine learning. (b) Accuracy after applying final round of FedAvg for different values of $\#_{epochs}$

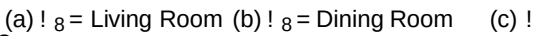


Fig. 7: Accuracy during post-training (personalization) over 100 epochs starting with a randomly initialized local model versus starting with a federated model trained on $\hat{\omega} = \omega \setminus \omega_8$.

B. Impact on Unseen Locations

It is useful to see how FL performs from the perspective of clients within $\hat{\omega}$. However, our primary goal is to see how such an FL framework can be helpful for new locations which are added to the network with zero or only a small number of available annotated training repetitions. To show this, in the following results when we evaluate a given client, ω_8, we set $\hat{\omega} = \omega \setminus \{\omega_8\}$ when training our federated model so that we can show that the parameters learned by the model are learned generally from all other locations and the model does not have any beforehand knowledge of the data or distribution of data at client ω_8. This is important to evaluate because when deployed into a real world system, we may not have annotated data for all clients. In those cases, we can select $\hat{\omega}$ based on attributes such as the availability of annotated data or even in cases of battery powered units, we can select only those which have surplus power to complete a given training round. As such, $\hat{\omega}$ may also change for every subsequent round to prevent wasting networking resources or power at any single location and also to prevent over fitting on data from the selected locations.

Consider the three sub figures in Fig. 7 where ω_8 is set to a different client for each. The figures show the accuracy over all 100 training epochs for the two training methods. We begin by comparing the local training method. For this method, the models are randomly initialized and then trained on some number of training repetitions (γ_{post}) from location ω_8. For the following evaluations note that a single post-training repetition ($\gamma_{post} = 1$) includes a sample for each class type, as we discussed in Section IV-A. Consider the case where a new client is added to the network without any post-training repetitions (i.e., $\gamma_{post} = 0$). For this case, the model cannot be trained for these 100 epochs, which means that the accuracy of the model remains constant at whatever value it started at. Since the model was initialized to have random values for θ, the accuracy when $\gamma_{post} = 0$ is approximately 25% for each value of ω_8 because there are 4 classes to predict from. We should next compare this to a model which is instead initialized on a federated model pretrained on $\hat{\omega} = \omega \setminus \omega_8$. For the federated model, these 100 post-training epochs are represented in Algorithm 1 as $\#_{post\text{-}epochs}$. We can see that, with

Fig. 8: Accuracy for federated model versus randomly initialized local model after 100 epochs of post-training (personalization) with different post-training repetitions (' post) at ! 8.

Fig. 9: Accuracy for WiFederated as j^ j increases.

the federated model, the accuracy for ' post = 0 is also constant over the epochs because we do not have any post-training repetitions to train the model further. However, because the model learns generalizable traits from j^ j, the initial accuracy is much higher than the randomly initialized local model. This is a very important feature for ensuring that WiFi sensing can scale without requiring all new client locations to pass through extensive CSI data collection and annotation steps.

Suppose now that a new client is added to the network, but we are able to perform very few repetitions of our actions in the environment (i.e., ' post 2 f1–2–3g). We can see in Fig. 8 that the addition of these post-training repetitions allows both models to increase their predictive accuracy by the end of the 100 post-training epochs. However, with small values for ' post, the local model is still unable to surpass the initial accuracy achieved by the federated model. Thus, even when some number of training samples are available, using the pretrained federated model can achieve higher prediction accuracy compared to training from scratch at each location.

C. Impact of the Number of Training Locations

So far, we have evaluated the WiFederated system when the number of training clients j^ j 2 f2–3g, however FL is able to accommodate larger number of clients especially considering that the training is processed in parallel across each selected client. To evaluate larger number of training locations, we collected data at ten total locations. We continue to use the same three locations (i.e., f Living Room–Dining Room–Office g) to evaluate our system since they have unique multipath characteristics and also to keep results consistent with our evaluations in the previous sections. Thus, seven remaining candidate locations are available for pretraining such that ! ^ ! f Living Room–Dining Room–Office g.

Fig. 9 shows the accuracy for each of the evaluation locations as j^ j increases up to 7 which corresponds to an increasing trend in the accuracy for the evaluation locations. When j^ j = 7 all seven candidate locations are selected for federated averaging. When j^ j Ÿ 7, different combinations of clients could be selected for ^. To account for this, we repeated each experiment 10 times with randomly selected value for ^ each time. This ensures that we do not accidently select values for ^ which are consequently overly good or overly bad. Furthermore, we use a static client-selection process where the clients in ^ are selected randomly at the first round of federated training and then reused for all subsequent rounds without further client selection. This process emulates the case where CSI data is collected at some random initial set of j^ j client locations so that future client locations require a much shorter data collection and annotation period. When the technician selects these first j^ j clients, they cannot be sure whether the locations they selected will be useful for building a generalizable federated model. Even so, we can see that increasing j^ j from 2 up to 7 clients offers an accuracy increase of ¸ 11.88% for ! 8 = Dining Room, ¸ 9.21% for ! 8 = Living Room and ¸ 10.43% for ! 8 = Office when the number of post-training repetitions ' post = 1 and ¸ 5.91% for ! 8 = Dining Room, ¸ 1.52% for ! 8 = Living Room and ¸ 4.77% for ! 8 = Office when ' post = 5. This shows that increasing the number of locations involved in training the federated model improves the prediction accuracy of the models for new locations which are unseen during the FL steps.

D. Comparison with State of the Art Approaches

In some recent studies [18], [20], in order to address scalability issue of WiFi sensing, a transfer learning based approach is proposed. However, for transfer learning based model training, data from ! ^ must be aggregated to a central location to train the model. Another approach is the EI framework [16] which uses adversarial networks. EI models are composed of two separate network branches connected by a parent feature extraction network. The first branch acts as a single activity recognizer with loss value L 0 while the second branch acts as a domain discriminator with loss value L 3. The domain discriminator attempts to recognize the domain or physical location where the CSI data was collected. The goal is to minimize L = L 0 L 3 which can be read as minimizing the loss of the activity recognizer while maximizing the loss of the domain discriminator. By training in this method, it is expected that the model will be generalizable to unseen locations by learning features that are representative of the activities being performed but not specific to any domain.

For the remaining evaluations, we look at the average accuracy across all three evaluation locations rather than individual client accuracy values. Fig. 10a-c show a comparison of four methods: our federated approach, the commonly used transfer learning approach, the adversarial EI approach and the globally trained model approach. In Fig. 10a, both Transfer and Global achieve the same accuracy. This is because when ' post = 0,

Fig. 10: (a-c) Comparison of four methods when using different numbers of pretraining locations with different post-training repetitions. (d) Training times required for each method with federated learning being the fastest thanks to parallelism.

there are no training repetitions to personalize the transfer learning model and as such, the transfer learning method and the global training method are the exact same. Both methods train their model on the same j^j locations at a global server before sharing the model with the client. Fig. 10b demonstrates how Transfer is different from Global. Namely, because ' post = 1, transfer learning allows some additional post-training or personalization steps over top of the globally trained model. Even so, we can see that our federated model consistently achieves higher accuracy than Transfer by ¸ 7•69% when ' post = 0, ¸ 8•87% when ' post = 1 and ¸ 4•26% when ' post = 5. With EI, we find that when ' post = 0, and j!^j 2 f2–3–4g, the accuracy is similar to our federated approach but when j!^j 2 f5–6–7g the accuracy is more similar to the transfer learning approach. We find that as j!^j increases, the number of training epochs for the EI framework must also increase. Thus, because we limit all models to a total budget of 100 epochs, the accuracy for the EI framework decreases. However, we find that even increasing the total budget to 500 epochs for EI still achieves a lower accuracy than the FL model trained with a budget of 100 epochs. When ' post 2 f1–5g, EI exhibits similar accuracy to transfer learning suggesting that the extracted features are similar between both methods.

All methods have a general trend towards improvement as j!^j increases. However, it is hugely important to consider the training time required for each of these methods. The key important insight is to see that our FL method is able to train in parallel across clients while the other methods must first aggregate the data to a central server which must then train the model sequentially on all of the same data. In Fig. 10d, we demonstrate how the time to train global and transfer learning models increases as more locations are used for training while the time required for FL remains relatively constant as a result of parallel training. EI method requires twice the amount of time required by the transfer learning method because it is essentially training two models, the activity recognizer and the domain discriminator. Note, unlike Fig. 3b results which are obtained on a Raspberry Pi, because of the memory overhead required for global, transfer learning and the EI methods, this evaluation is completed on a more powerful server.

E. Run Time Complexity Comparison

The training process for all of the discussed methods is composed of four stages: (i) data collection, (ii) preprocessing, (iii) training the model, and (iv) post-training, after which evaluation is followed. Out of these steps, the most time consuming step is the training step followed by the post-training step, thus they both define the overall duration for the model development. For the results in Fig. 10a-c, we allow all models to initially be trained on the datasets available in !^ for a total of # budget = 100 epochs after which the federated models, transfer learning models and adversarial EI Framework models are trained for an additional # post-epochs = 100 epochs. For simplicity in our notation, we say that 4 = # budget = # post-epochs. The time complexity of training is related to the product of the number of epochs (4) and the number of samples (() used which we can denote as $ ¹4 (º. In the case of FL, the data from each location is trained independently and distributed in parallel, thus the time complexity would be $ ¹4 <0G¹(; 8 ; 2 !^oo. On the other hand, transfer learning would have a dataset size of ;2!^ (; thus the time complexity is $ ¹4 ;2!^ (;º, which becomes $ ¹4 (; j !^jº if we assume (; = (: 8;– : 2 ! for simplicity. The difference between the transfer and EI methods is that EI essentially trains two model networks, an activity recognizer and a domain discriminator while transfer learning only trains a single activity recognizer model. However, this will still keep the time complexity for EI asymptotically similar to transfer learning at $ ¹4 (; j !^jº. The post training step at a given location for FL, transfer learning and EI is performed on ?>BC number of post training samples, where typically ?>BC ŸŸ (;, for an additional 4 epochs which results in an added time complexity of $ ¹4 (?>BCº. Thus, we can conclude that FL has a time complexity of $ 4 (; ¸ (?>BC , global learning has a time complexity of $ ¹4 (; j !^jº, and transfer learning and EI both have a time complexity of $ 4 (; j !^j ¸ (?>BC , where j!^j is the number of locations from which samples are aggregated for training at the central server. We can see that the complexity of global learning, transfer learning and EI methods all increase as j!^j increases while FL is able to maintain a consistent time-complexity showing that FL will be a much better option for scalability as additional locations are added to the system.

F. Impact of Client Selection

Client selection can be performed at two different times during the entire FL process as denoted in Algorithm 1. First,

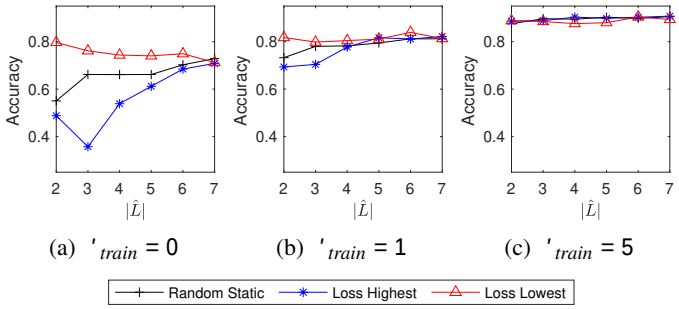


Fig. 11: Impact of client selection with different number of training repetitions ($'_{train}$).

we select $\hat{I}$ to be the clients that are used to train in parallel. After this, we can further reduce this selection to $\hat{I}^0 \quad \hat{I} \quad !$. Typically the first client selection step is used to distribute tasks fairly across the network. For example, it would not be fair to require any single client to take part in all FL rounds. This would consume more power and waste time for this individual client. As such, it can be useful to be selective when choosing $\hat{I}$ so that we do not overburden any single device. Furthermore for battery powered devices, it is important to select clients which have a battery level above some threshold to filter out any devices that may lose power during training.

The second client selection step can further guide the optimization of the federated model. For example, there may be clients with poor quality training repetitions or clients ($:$) whose $\backslash_:$ will cause some negative impact onto the model parameters for the federated model as a whole. As such, we consider some other client selection methods. Specifically, we use the calculated loss $\mathsf{L}^{1\,0}$ for all clients in $\hat{I}$ to determine which clients should be used for FedAvg. It was previously recognized [35] that selecting the client with the highest loss during client selection will increase the accuracy of the federated model overall. The idea is that a high loss is directly related to a high amount of error. If a client has a high error, then the distribution of the data at the client must have some amount of novelty which may be applicable to other clients as well. In Fig. 11 we can see a comparison between client selection methods. Comparing the *Random Selection* method we used for previous evaluations to the *Loss Highest* method, we can see that using this loss-based approach to guide the federated system actually results in lower accuracy overall. Our intuition here is that if we guide the training with high-loss clients, then we are selecting clients with the worst fit to the federated model from the previous round. This means that the clients are likely to have low-quality data available for training and thus the federated model as a whole will suffer. Alternatively, if we use the low-loss clients, then data available from these clients is similar to what is expected by the federated model and by extension, the data would also be similar to the data found in the locations selected for $\hat{I}^0$ in previous rounds. In fact, if we take this opposite approach and guide the FL process through a *Loss Lowest* approach, we achieve a greater accuracy. When $\mathsf{j}\hat{I}\mathsf{j} = 2$ we achieve an accuracy of 79•89% with *Loss Lowest* versus 55•05% for *Random Static*. Similarly with $'_{post} = 1$, the *Loss Lowest* approach achieves 81•74% compared to 73•25%.

VII. Feasibility of WiFederated at the Client

We demonstrated that our proposed WiFederated system achieves increased predictive accuracy by training local models across different locations using a federated averaging and client selection process in comparison to starting from a randomly initialized model. Moreover, we demonstrated that our framework reduces the amount of training data that is required at each location when compared to training a local model at a single location independently without some form of collaboration. For the development and deployment of a full system leveraging this FL framework for WiFi sensing, we must also consider some additional issues.

A. Training and Inference at the Edge

Training deep learning models at a desktop computer or at a server benefits from the use of GPUs to speed up the training time especially as more data is added. However, as we discussed, sending all data to a central server for processing may not be feasible and with the use of FL is no longer a requirement. However, we must consider that local training at the edge requires special consideration. When designing the model used throughout this work, special care was placed to ensure that the complexity of the model will not require GPU-based training. Thus, we can use less powerful devices to train our model at each location.

Single-Board Edge Computers: Single board computers such as the Nvidia Jetson series of boards are designed specifically for machine learning at the edge with an on-board GPU and a full Ubuntu operating system. This means that any software written to run on a standard computer or server can directly be ported to the Jetson single-board computer, allowing for fast development time. Alternatively, lower cost single-board computers such as a Raspberry Pi can also be used in the same way, however without direct access to an onboard GPU. In Fig. 12, we show the time required for training a local model on different values of $'_{train}$ for both a Raspberry Pi 4 B and an Nvidia Jetson Xavier NX. We can see that the GPU on the Jetson speeds up the computation considerably, but at a higher cost compared to the Raspberry Pi. Even so, the Raspberry Pi can still train the model in under 10 minutes when $'_{train} \quad 6$. This is useful considering that post-training for most edge devices in the WiFederated system can still achieve good prediction accuracy even with only a small number of repetitions.

Using standard ESP32s: To allow for the fewest additional hardware components in the system at each location, it would be most beneficial to train on the ESP32 microcontroller used to collect CSI. Recent research literature shows that training machine learning models on such low-powered devices is desirable [36], [37] and code libraries such as MicroMLP [38] have appeared for training machine learning models on low resource microcontrollers such as the ESP32. We attempted to train our federated model using this library, but we find that the model does not fit in memory due to some inefficient memory

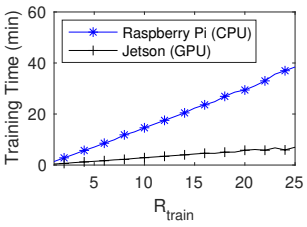


Fig. 12: Average training time for edge devices.

Device	Rate (Hz)
ESP32	5.726
Raspberry Pi	2252
Jetson	5359

TABLE II: Average prediction rate for edge devices.

allocation within the library itself. Instead we look at using a popular model inference library called Tensorflow Lite. Using this library, with model quantization, we are able to store our model directly in memory on the ESP32 and we are able to achieve a prediction rate of 5•726Hz. In Table II, we can see a comparison of prediction rates possible with each edge device. Even though both Raspberry Pi and Nvidia Jetson devices are able to achieve prediction rates of greater than 1-000Hz, CSI collection rate is only 100Hz. This shows that the single-board computers can make predictions for every received CSI sample. Many applications may not need such high prediction rate because human activities should not change at such a high rate, so a standalone ESP32 is still useful for inference due to its low power consumption and small size.

B. Continuous Annotation

The process of annotating and recording CSI for a single location can be time consuming, repetitive and error prone. Luckily, as demonstrated in the previous section, leveraging a model obtained through FL in multiple different locations can greatly reduce the number of action repetitions which are needed to obtain a useful model.

Furthermore, we find that some sensing tasks have the ability to be continuously annotated over time, thus allowing for the model to continuously be trained on new data from the environment. Wearable respiratory monitoring belts have commonly been used in WiFi sensing research to label ground truth data for monitoring patient breathing patterns [39]. However, these works assume the belts are used only to train the initial model, but are never used again afterwards. An alternative approach is to use the sensor data to train the CSI based model continuously over time. Fig. 13a illustrates that CSI can be collected in the background at the same time a wearable sensor is being worn and Fig. 13b shows a time series view of the data being collected. While CSI is continuously collected in the background, there is a span of time where wearable sensor data is missing. This may happen because the user removed the wearable for example before going to bed at nighttime or due to discomfort. However, even when the sensor is not being worn, it can be important to continue to track the user. To do this, whenever sensor data is available (i.e., sensor is worn), the sensor data can be used to automatically annotate the collected CSI data to train WiFi sensing model F. Then, whenever sensor data is unavailable, F can continue to monitor the user using the available CSI data. This ensures that the model can continue to learn over time and the person can

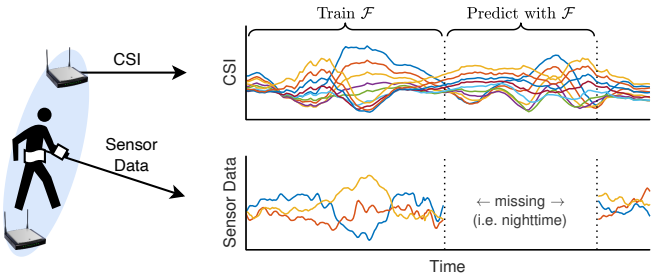


Fig. 13: Example scenario for continuous learning. (a) User with a wearable sensor while CSI is collected in the background. (b) When sensor data is available, both CSI and sensor data can be used to train F. When sensor data is unavailable (i.e., at nighttime when wearable sensors are removed), CSI can be used with F to continue monitoring.

be safely monitored even when the sensor data is unavailable. Continuous data labelling and thus continuous model training means that the proposed WiFederated system can continue to learn without requiring time-consuming manual data collection steps and can also help reduce the effects of data drift [40] over time due to environmental changes.

VIII. Conclusion

In this work, we introduce WiFederated, a federated learning framework designed for scalable deployment of multi-location CSI-based WiFi sensing systems. By training local models at each location in parallel and then performing federated averaging of the model weights at the central server over multiple FL rounds, we are able to train a federated model which is generalized to location-independent features. We show that we can leverage this federated model to reduce the number of training repetitions required per physical action when training at new locations. We conclude that this reduction of training repetitions allows for more rapid deployment of WiFi sensing devices into new locations without increasing the complexity or workload for the technician installing any new WiFi hardware for the system. Additionally, in cases where hardware such as WiFi access points are already deployed throughout a building, the technician does not need to enter each location to perform a large number of time-consuming training repetitions of the actions before seeing useful prediction accuracy. We also find that using WiFederated can reduce the number of local training epochs required compared to other methods. Finally, we evaluate training and evaluation at the edge and consider possible methods for continuous annotation to ensure that clients are able to continue to capture accurate annotation labels over time to further train the federated model.

References

[1] Z. Wang, K. Jiang, Y. Hou, W. Dou, C. Zhang, Z. Huang, and Y. Guo, "A survey on human behavior recognition using channel state information," *IEEE Access*, vol. 7, pp. 155 986–156 024, 2019.

[2] Y. He, Y. Chen, Y. Hu, and B. Zeng, "Wifi vision: Sensing, recognition, and detection with commodity mimo-ofdm wifi," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8296–8317, 2020.