Lightweight and Standalone IoT based WiFi Sensing for Active Repositioning and Mobility

Steven M. Hernandez and Eyuphan Bulut
Department of Computer Science, Virginia Commonwealth University
401 West Main St. Richmond, VA 23284, USA
Email:{hernandezsm, ebulut}@vcu.edu

***Abstract*—Channel state information (CSI) provides rich insight into the physical characteristics of an environment through radio subcarrier frequencies in orthogonal frequency-division multiplexing (OFDM) systems. Many recent studies explore this rich source of data to produce quite accurate results in device-free localization, human-body pose recognition, and device-free person identification under the umbrella of *WiFi Sensing*. Most works thus far rely on the use of the Intel 5300 Network Interface Card (NIC), a device requiring connection to a host computer to function. Because of this requirement, the weight and form factor of CSI recording capable devices (receiver or RX) has limited the abilities of researchers to explore certain aspects of WiFi sensing such as active repositioning and mobility of RX devices. To address this, in this paper, we use the ESP32 microcontroller to develop a simple and lightweight solution for CSI collection leveraging recent additions to the Espressif IoT Development Framework which allows user developed programs to access CSI directly. The system can work standalone or attached to a smartphone for advanced online computations. Thus, it can be easily deployed, repositioned, and carried on mobile objects, which can then help improve the performance of sensing tasks. We evaluate the performance of our proposed system through several deep-learning based human activity recognition experiments and show that the repositioning and mobility of RX devices can provide increases in accuracy upwards of 29.4% and 28.2%, respectively, compared to the commonly considered static RX scenario. Finally, we produce an easy to use open source codebase for researchers to immediately begin exploring the new possibilities (e.g., massive deployment) available by the usage of the proposed system.**

I. Introduction

Channel state information (CSI) provides a metric that describes how wireless signals propagate from the transmitter to the receiver based on the characteristics of the environment. CSI provides a richer source of data than Received Signal Strength Indicator (RSSI) by modeling the received wireless signal from multiple subcarrier frequencies. This has enabled gaining more insight into more subtle details of the physical environments surrounding the radio and sparked a new area of research termed *WiFi sensing* [1]–[3]. WiFi sensing most often uses CSI (some works such as [4] still rely on RSSI) to sense characteristics about the surrounding environment in a *device-free* manner referring to the fact that there is no special device required to be attached on the tracked object(s).

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1744624. 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 National Science Foundation.

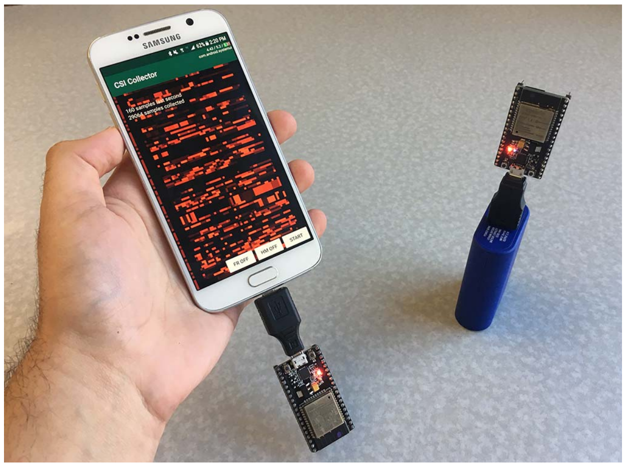


Fig. 1: An Android smartphone with an attached ESP32 (in RX role) displaying incoming CSI amplitude data from the ESP32 (in TX role) with a rechargable battery on the right.

Despite the variety of the studies benefiting from the CSI collection, the tools used to record CSI data are very few. That is, most of these studies use either the Intel 5300 Network Interface Card (NIC) and the *Linux 802.11n CSI Tool* [5] or the Atheros line of NICs with more recently developed *Atheros CSI Tool* [6]. NICs however cannot act as standalone devices, instead require a laptop if not a full desktop computer to run. Because of this, the deployment of CSI recording capable (receiver or RX) devices has been costly and bulky. This not only restrains the practicability and scalability of such systems but also limits the active repositioning and mobility of such devices, while significant performance improvements could be obtained through more flexible deployment opportunities.

A recent work [7] presents a method where any device with Broadcom chipset can be made to access CSI data, including the Google Nexus smartphone. This is accomplished by using their Nexmon firmware patching framework, but unfortunately as stated in the Nexmon documentation [8], patching firmware “may damage your hardware and may void your hardware’s warranty”. Beyond these works, to the best of our knowledge, there is no other WiFi device used in literature that allows direct access to CSI information. Regarding CSI collection at mobile devices, there are some recent localization and distance calculation based studies that consider attaching the Intel 5300 or Atheros NICs to small single-board computers (e.g., HummingBoard [9], Intel Galileo Gen2 board [10]) or tablets [11] to make it more portable. However, such solutions still come with the additional hardware requirement for hosting the NIC and do not allow standalone CSI collection.

In this work, we explore the standalone ESP32 microcontroller which allows access to this rich CSI data directly from the microcontroller without any complicated hacks or custom firmware changes. This unique feature of the ESP32 allows us to easily deploy a lightweight device for CSI collection and recording as a standalone device and facilitates its repositioning and mobility. Moreover, ESP32s can also be attached to a smartphone to make predictions with advanced learning models on CSI data through any standard smartphone app. The main contributions of this work can be summarized as follows:

- We develop a standalone, lightweight and low cost IoT based WiFi sensing solution using a CSI collection tool that runs on ESP32s, which can also be attached to smartphones for advanced computations, and a deep learning system suitable for smartphones.
- We perform experiments on extracting context information (i.e., direction, depth of the activity) for human activity recognition systems and show that consideration of active repositioning and mobility of CSI collecting devices can help improve the sensing accuracy compared to CSI collection at static devices.
- We produce an efficient, easy-to-use tool for programming the ESP32 which we release as an open source project[1].

The rest of the paper is organized as follows. We first provide a background on Channel State Information and various WiFi sensing studies which leverage it in Section II. Then, we describe the proposed IoT based WiFi sensing system including the ESP32 CSI extraction tool, signal preprocessing and deep learning systems as well as real-time smartphone prediction framework in Section III. We then evaluate the proposed system through experiments in Section IV and discuss its capabilities and benefits through repositioning and mobility. Finally, we provide a discussion about the proposed system in Section V and provide concluding remarks in Section VI.

II. Background

A. Channel State Information

Channel State Information (CSI) is a metric used in orthogonal frequency-division multiplexing (OFDM) for describing amplitude and phase variations across multiple subcarrier frequencies as wireless signals are transmitted between a transmitter and receiver. To detect these variations across subcarriers, OFDM systems transmit a set of known shared pilot symbols (either interleaved in the transmitted message or as a preamble) which are then used to estimate a vector H which we describe as CSI through the following equation:

$$y^{(i)} = H^{(i)} x^{(i)} + \eta^{(i)} \tag{1}$$

where $y^{(i)}$ indicates the signal detected at the receiver, $x^{(i)}$ is the transmitted signal (shared pilot) and $\eta^{(i)}$ is the noise vector. CSI is collected for each subcarrier (i) as a complex number consisting of both imaginary ($h_{im}^{(i)}$) and real ($h_r^{(i)}$) parts. Combining the real and imaginary parts of each subcarrier, we can determine the amplitude ($A^{(i)}$) and phase ($\phi^{(i)}$) for subcarrier i by the following equations.

$$A^{(i)} = \sqrt{(h_{im}^{(i)})^2 + (h_r^{(i)})^2} \tag{2}$$

$$\phi^{(i)} = atan2(h_{im}^{(i)}, h_r^{(i)}) \tag{3}$$

B. Related Work

Extraction of CSI has triggered new studies in various WiFi sensing applications such as device-free localization, and human activity/gesture recognition. Device-free localization attempts to recognize the location of a target(s) (typically indoors) without requiring the targets to have any application specific hardware sensor such as a Bluetooth beacon or RFID device. Studies have considered device-free localization for recognizing the position of a single target as well as recognizing multiple targets in a given area [12]–[15]. With multiple individuals in an area, noise can become hard to concretely model. Thus, in cases of multiple individuals, some works [16]–[18] reduce this problem to simply crowd-counting. In gesture recognition, the goal is to recognize different gestures made by a target such as hand-waving [19], [20], exercise movements [21] as well as other movements such as standing up, walking, and running [22]. Typically gestures are performed in the line-of-sight (LOS) of the transmitter and receiver. However, when they are performed outside of the direct LOS, prediction accuracy decreases (e.g., from 92.1% down to 83.1% with 2 meters away from the direct LOS, as shown in [21]). This understanding is important in cases where the location of transmitting WiFi devices are not controllable and must remain static; thus, the positioning or mobility of receiver devices can be critical in improving prediction accuracy. Our goal in this paper is to explore this through the newly developed lightweight and standalone IoT based tool.

III. IoT based WiFi Sensing System

In this section, we introduce our IoT based WiFi sensing system which consists of (i) an ESP32 microcontroller that is used as a receiver (RX) device for CSI collection (which can also be used as a transmitter (TX) device), (ii) a smartphone app which can read CSI from the serial port of the ESP32 and (iii) a deep learning based learning model which uses Keras library [23] and can be deployed through Tensorflow Lite on smartphones. Next, we elaborate on the implementation and functionalities of each of these components.

A. ESP32 CSI tool

We propose an ESP32 based CSI data collection tool which is lightweight, can work standalone and can be easily deployed with a low cost. This provides an opportunity to build more practical and easy to maintain WiFi sensing systems especially for large scale systems (e.g., an indoor localization system [12]

[1]https://stevenmhernandez.github.io/ESP32-CSI-Tool/

TABLE I: Comparison of Tools for Collecting CSI.

	Intel 5300 (+Laptop)	Atheros (+Laptop)	Nexmon	ESP32
Papers using this tool for CSI	92.5%	6.8%	$< 1\%$	N/A
Attachable to Smartphones	NO	NO	With Firmware Changes	YES
Standalone Operation	NO	NO	With Firmware Changes	YES
Size	$> 30cm \times 20cm$	$> 30cm \times 20cm$	$> 15cm \times 7.5cm$	$5.0cm \times 3.0cm$
Weight	$> 1kg$	$> 1kg$	$> 100g$	$< 10g$
Battery Powered	NO	NO	YES	YES
Cost	\$10 + Laptop	\$10 + Laptop	>\$100	<\$10
Number of Subcarrier Groups	30	56	128	64
Resolution (imaginary/real)	8	11	32	8
Implementation Level	Kernel	Kernel	WiFi Chip Firmware	User
Codebase Size (Line Count)	2M	2M	1M, 60K CSI Specific	1K
Bandwidth (frames per second)	up to 1000/s	up to 1000/s	up to 1000/s	up to 650/s
RAM	8GB+	8GB+	1GB-4GB	500KB - 4MB
# Antenna	3	3	1	1 (2-16 with switch)
TensorFlow	Full	Full	Lite	Lite/Full

with 10 laptops used as RX). We implemented the codebase for ESP32s using the Espressif IoT Development Framework (ESP-IDF) in C. The codebase consists of two specific applications which can be run on a pair of ESP32 microcontrollers. The first application is an active access point (AP) which initializes the on-board WiFi stack to allow devices to connect, make requests and receive requests. The second application is an active station (STA) which automatically connects to the AP, then sends requests to the server running on the AP. With both applications active, it is possible for the devices to automatically initiate communication but more importantly, it is possible for our user-level application to collect CSI data for further processing. CSI data is automatically written to either a serial port if connected and/or directly to an onboard micro SD card if present. By writing to an SD card, we offer the unique ability to place the ESP32 attached only to a battery in space limited environments thanks to its small size, which is not possible with existing CSI-enabled NICs.

Comparison. We compare our tool to other existing tools; namely, (i) the *Linux 802.11n CSI Tool* for Intel 5300 NICs, (ii) the *Atheros CSI Tool* for a range of Atheros NICs, and (iii) *Nexmon Tool* for Broadcom WiFi chips. Table I shows a summary of this comparison. First of all, we see that out of total 494 papers listed on these tools' websites [8], [24], [25], 92.5% used the Intel 5300 tool while only 6.8% used the Atheros tool (expected as Intel 5300 was published in 2011 while the Atheros tool was published in 2015). The Nexmon tool on the other hand has been used for CSI collection in only a few papers yet, as it was published very recently.

We describe both Intel 5300 and Atheros simply as NICs because of their close similarity. We see that NICs are not able to run directly from a smartphone, instead they require direct connection to either a laptop or a full desktop computer to function. The Nexmon tool on the other hand was shown to work on a Google Nexus smartphone, however the design for collecting CSI was built specifically for this model of smartphones and requires firmware changes to the WiFi chip itself which could damage the hardware of the phone. The ESP32 can connect directly to a smartphone to give the smartphone access to CSI without any such hacks. A further consideration is whether the tool can allow the device to run standalone. Obviously neither NIC can offer this ability, nor *Nexmon* without a full-fledged Google Nexus smartphone. However, with our ESP32 CSI tool, the ESP32 can collect and record CSI directly to an on-board micro SD card without requiring the functionality of any external devices. This in conjunction with the small size ($5cm \times 3cm$) and weight ($< 10g$) of the ESP32 compared to a desktop computer or even a laptop or smartphone means that the proposed tool is very agile and can easily be deployed even in space limited environments in massive amounts, can be repositioned easily and can be attached to mobile objects without much burden thanks to its lightweight design. The cost (i.e., $< \$10$) for the ESP32 is on par with the cost of a NIC, however a NIC again cannot work standalone, thus the primary cost associated with the NIC is not in the NIC hardware itself, but instead in the desktop or laptop computer it is connected to. Our tool also gives access to 64 subcarriers where the resolution of each imaginary and real number is 8 bits which is on par with other tools.

Our ESP32 CSI tool is also much more focused on solely collecting CSI from the ESP32 and has approximately $1,000$ lines of code at this time, while the codebase of the other tools contains many files completely unrelated to the task of collecting CSI (e.g., Linux kernel, or OpenWrt), causing the number of lines of code approaching around two millions.

Use cases. Collecting CSI with the ESP32 is highly versatile in the fact that the ESP32 can collect CSI when acting as both an AP and an STA. This is not possible with existing tools because only the device (e.g., laptop) with the NIC (receiver) can be used to collect CSI. ESP32s can thus be deployed into many additional scenarios to collect CSI. Table II shows the possible use cases that could be achieved with ESP32s in a WiFi sensing scenario. ESP32s can act as both a TX device and an RX device; thus, if a completely new system will be built in an environment for a WiFi sensing task, two ESP32s can easily be used to realize that. Note that such a set up provides

TABLE II: Use Cases for ESP32 based CSI collection

Usecase	Transmitter (TX)	Receiver (RX)
#1	Standalone ESP32	Standalone ESP32
#2	Connected WiFi AP	Standalone ESP32
#3	Connected Smartphone	Standalone ESP32
#4	Unconnected WiFi AP	Standalone ESP32
#5	#1,#2,#3,#4	Android device + ESP32
#6	#1,#2,#3,#4	iOS device + ESP32

full control of both devices so that we can determine both the rate at which CSI is collected and the position of each device within a given space. If there is already an existing device in the environment such as a WiFi access point (AP) or a smartphone (both Android and iOS), ESP32s are also capable of accessing CSI data from these transmitting devices. Here, if a connection is possible to these devices from the ESP32 and the transmitted packet rate is controlled, a stable high CSI extraction rate (e.g., > 500Hz) could be obtained. However, our ESP32 tool is also capable of sniffing ambient WiFi signals and extract the CSI information without compromising its existence (e.g., no advertisement, visually hidden). To the best of our knowledge, such kind of passive or sniffing based CSI collection approach has been used in only one very recent study [26] using Nexmon tool under *adversarial WiFi sensing* scenario. However, only a limited packet rate (i.e., 8-11 packets/sec) is achieved there (even though there were devices transmitting in higher packet rates). On the contrary, our tool is capable of sniffing orders of magnitude higher rates of ambient packets in sniffing/passive mode. Note that if there is not much ongoing wireless packet transmission activity in the environment or the distance from the transmitter devices is longer, sniffing based CSI extraction can end up with low packet rates. Moreover, in sniffing mode, varying packet rates might occur from the uncontrolled transmitter. Therefore, a careful analysis must be performed to determine the implications of sniffing based WiFi sensing. However, this is out of the scope of current work.

B. Android CSI Collector App

To collect CSI from the ESP32 for our experiments, we developed an Android app which receives CSI data from the ESP32 through USB serial. As CSI is received, the app processes this CSI in real-time to record all samples to local storage with additional appended timestamp and experiment name for later analysis. The processed real-time CSI amplitude signal is then displayed on the screen as a waterfall as shown on the screen of Fig. 1 so that the experiment can be monitored for quality in real-time. Additionally, a simple *samples per second* metric is displayed along with a *total samples counter*. To label experiment data in real-time for use in our deep learning phase, the user can simply press a button per action being taken and release as the action is completed. Finally, experiment names can be set and updated directly from the interface to allow multiple unique experiments to be performed and recorded independently from within the app. Further demonstrations of the capabilities of recording CSI with this Android app on off-the-shelf smartphones can be viewed in [27].

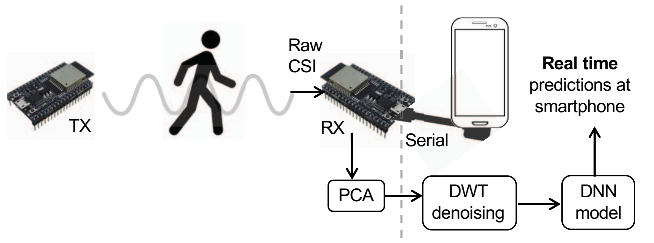


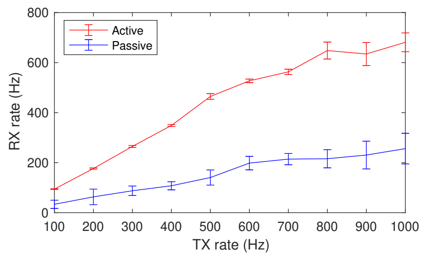
Fig. 2: Real-time predictions at smartphone.

C. Deep Learning System

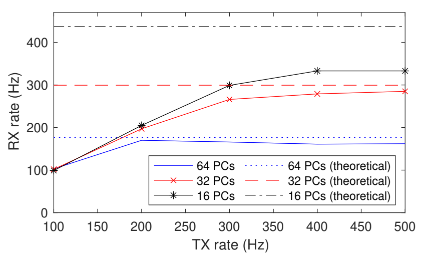
Pre-processing: Our learning phase begins by performing a number of pre-processing steps. First, we apply 1-dimensional Discrete Wavelet Transformation (DWT) denoising [3], [28] across each subcarrier independently using a *db4* wavelet. DWT allows us to remove noise from each subcarrier while retaining distinct peaks from our signal. In addition to removing noise, we look to reduce the dimensionality of the incoming CSI data by performing the PCA (Principal Component Analysis) algorithm [29], [30]. PCA first takes our CSI sample matrix ($\boldsymbol{H}$) to produce a correlation matrix $\boldsymbol{C} = \boldsymbol{H}^T\boldsymbol{H}$ of size $n \times n$ where $n = 64$ is the number of subcarrier frequencies. Eigen decomposition is then performed on $\boldsymbol{C}$ to calculate eigenvectors $\boldsymbol{q}$. Finally, each of 64 principal components $\boldsymbol{h}_i$ is determined through $\boldsymbol{h}_i = \boldsymbol{H} \times \boldsymbol{q}_i$. For our system, the number of retained principal components (m) is selected based on a percentage parameter p such that $p \leq \frac{1}{L}\sum_{i=1}^{m}\boldsymbol{q}_i$ where $L = \sum_{i=1}^{n}\boldsymbol{q}_i$ and m is minimized. This means, when $p = 1.0$, $m = n = 64$ as all 64 principal components will be retained. However, when $p < 1.0$ then $m < n$ because some principal components can be ignored, resulting in a reduction of CSI dimensionality. For our experiments we set $p \in \{0.90, 0.95, 0.99\}$. Finally, we split our full dataset into smaller sub-samples through a rolling window of size k. Each sample is then modeled as a $k \times m$ matrix ($\mathbf{s}$).

Model and Training: To implement our learning models we use the Keras deep learning library. Keras uses TensorFlow as a backend which allows our models to be trained and then deployed directly on either an iOS or Android smartphone or even on a microcontroller such as the ESP32 through the use of TensorFlow Lite. For our experiments, we use Dense Neural Networks (DNN) and find that using four dense layers allows for consistent accuracy and quick training. We try a range from $(10, 200)$ neurons per hidden layer and find 25 neurons to perform the most consistently. We use dropout layers so that 50% of the weighted connections between dense layers are removed to prevent overfitting. Optimization is performed using stochastic gradient descent (SGD) with learning-rate of 0.1.

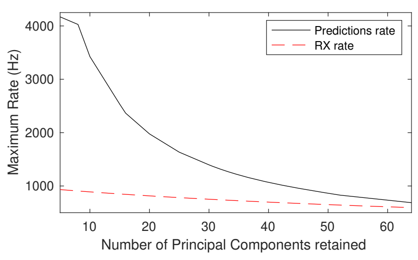
Real-time Predictions: To perform real-time deep learning prediction in a given environment, we consider offloading tasks between an ESP32 and a smartphone. Specifically, as shown in Fig. 2, the ESP32 receives raw CSI from TX. Because limited



(a)



(b)



(c)

Fig. 3: (a) Number of packets received per second at an active (i.e., connected) and passive (i.e., sniffing) receiver when transmitter sends packets at varying rates. (b) RX rate when considering serial throughput and number of Principal Components (PCs). (c) Maximum number of predictions per second on smartphone compared to number of incoming CSI-frames per second.

bandwidth is available in the serial connection from ESP32 to smartphone, we perform PCA to reduce the dimensionality of each CSI sample from 64 subcarriers down to only 10 principal components or fewer. Because the number of bits transferred is equal for one subcarrier and one principal component, this dimensionality reduction can lead to an increase in throughput over the serial connection. Because the ESP32 has much more limited available memory than a standard smartphone, we suggest applying DWT denoising on the smartphone after collecting these PCA samples. We find that PCA selects similar weightings for each subcarrier when DWT is performed before or after PCA. Finally, the smartphone can perform real-time predictions with the pretrained DNN. Predictions can then be recorded with timestamp on device and presented to the user.

IV. Experiments

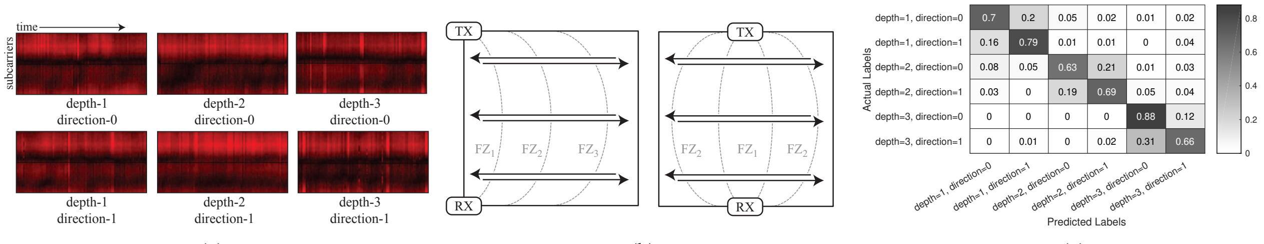
In this section, we evaluate the new IoT based WiFi sensing system through experiments. We first show results regarding the capabilities of the proposed system for various scenarios. Then, we provide its performance evaluation through a set of human activity detection experiments.

A. System Capabilities

We begin evaluating the capabilities of our proposed system by looking at the rate at which an ESP32 can collect CSI samples. For this, we set TX to send packets at a constant rate given in Hz as shown in Fig. 3a. Each line represents the mean and one standard deviation for the RX rate received over a one minute time period for each value of TX rate. The first scenario we consider is when the RX is active in communicating with TX (i.e., TX sends packets directed to RX). In such a case, we can see that RX rate increases linearly until around 500Hz. When TX rate is > 800Hz we can see that RX rate reaches a plateau at only 600Hz. This shows that the system becomes bottlenecked at this RX sampling rate. In addition, we can see that the standard deviation for TX rate > 800Hz is increased, showing that while the number of samples collected per second was on average 600Hz, this RX rate was not reliable from one second to the next. We also consider the passive case (i.e., sniffing) where TX is no longer transmitting directly to RX and RX is not connected to TX. We can see that the RX rate for passive is lower than it is in the active case. Passive RX rate remains consistently at 34.5% of the active RX rate. We can also notice that the one standard deviation range is higher for all values of TX rate in this passive case. This unreliability can be accounted for by the fact that an active device can request retransmission of CSI in cases of missing or corrupted frames at the physical layer while passive devices cannot.

When sharing collected CSI from the ESP32 to an outside device such as a smartphone, serial communication throughput must also be considered when viewing the RX rate. In our system, we send additional header information per CSI frame such as MAC address, RSSI, and other metadata along with the channel state information for all 64 subcarriers resulting in each frame of size $s = 1\text{kB} = 8\text{kbit}$. Serial communication speed is defined based on the baud rate (b) of the connection. Our theoretical packet rate throughput r can be calculated as $r = \frac{b}{s}$. We find that setting a non-traditional baud rate of $b = 1552000$ bits/s allows for the highest throughput while retaining serial consistency. Higher rates result in serial communication failure. We can see in Fig. 3b that the theoretical packet rate throughput r limits the number of packets received by RX to just below 200Hz when transmitting all 64 principal components, or subcarriers. Because b cannot be increased, the solution is to instead reduce s. Without removing header data, our solution is to perform PCA before transmitting over serial. We can see that reducing sample dimensionality from 64 down to 32 principal components (PCs) increases r to 300Hz. The real world case with 32 PCs reaches this theoretical limit as well. Further reducing the number of PCs to 16 however does not reach the expected theoretical RX rate. This implies some other bottleneck exists in the communication system. Even so, the system can achieve RX rates surpassing 300Hz, which is more than enough to handle human actions as shown in [29].

Considering our real-time smartphone prediction framework, increasing the serial communication throughput increases the RX rate (number of CSI frames received), thus increasing the number of predictions which must be performed per second. To evaluate how this affects our framework, we program a smartphone to perform as many predictions per second as possible on CSI data after PCA was performed. In Fig. 3c, we can see that when 64 PCs are used, the theoretical RX



(a) (b) (c)

Fig. 4: (a) Heatmaps of CSI Actions. (b) Two experiment configurations with a target area of 4.8m × 4.8m. (Left) Placement of transmitter and receiver on the *edge*. (Right) Placement of transmitter and receiver in the *center*. (c) Confusion Matrix.

rate is very similar to the number of predictions performed per second. This means that while we can make predictions for every incoming CSI frame, this leaves very little time for any other tasks. As the number of PCs decreases, RX rate increases, meaning that more predictions will need to occur per second. However, as the number of PCs decreases, the number of computations required to make a prediction with our models also decreases. Thus, we see that the number of predictions we can make per second increases at a much higher rate compared to the theoretical RX rate as the number of PCs decreases. This means that decreasing PCs not only increases the number of CSI frames received, thus increasing the resolution in which we can predict with, but also gives the system more time to perform other calculations in real time.

B. Human Activity Detection Experiments

For these experiments, we consider an indoor office setting with a square space of size $4.8m \times 4.8m$ without internal obstructions. Each action performed by the target occurs within these bounds. Specifically, we consider the following combination of actions: moving from *left to right* (direction=0), *right to left* (direction=1) and the depth of the movement *close to receiver* (depth=1) ($0.6m$ away from receiver), *close to transmitter* (depth=3) ($0.6m$ away from transmitter), *middle* (depth=2) ($2.4m$ away from both transmitter and receiver). CSI amplitude samples are presented as a heatmap for each unique action in Fig. 4a. Below, we specifically look at the performance of the system in various forms of active repositioning and mobility of RX/TX devices leveraging the lightweight design of our system. For each scenario, we perform each action six times.

1) Joint Repositioning of RX and TX: We first consider the static placement of both transmitter and receiver, and their joint repositioning relative to the actions performed in the experiment. Existing works on WiFi sensing predominantly use a NIC in either a desktop computer or laptop and consider an experimental setup that relies on the static placement of both the TX and RX devices. We believe that on the chance of less than optimal placement of devices; such as would appear in more realistic installations, certain actions will be harder to predict. To determine if this is true, we first find static positions for both the receiver and transmitter which produce high action

TABLE III: Validation accuracy per placement of transmitter and receiver (with 100 CSI frames per sample).

	Edge	Center
Depth	87.5%	81.6%
Direction	89.4%	58.2%

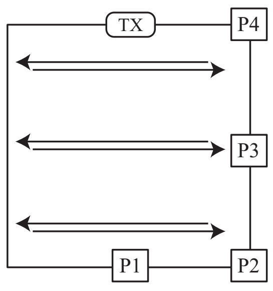
TABLE IV: Validation accuracy for Edge-Direction considering pre-processing steps (with 100 CSI frames per sample)

	DWT	non-DWT
PCA	89.4%	74.2%
non-PCA	51.1%	50.1%

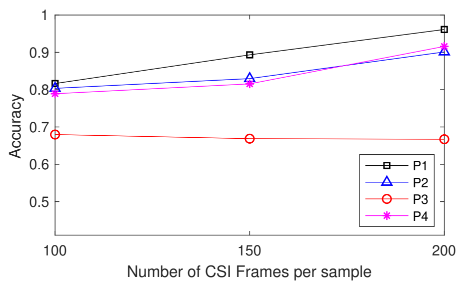
prediction accuracy. We then manipulate these positions to recognize how slight variations can cause far worse results.

For our first experiment, the transmitter and receiver are placed at two adjacent corners as can be seen in Fig. 4b (left). We find that in this configuration, we achieve high prediction accuracy for both depth (87.5%) and direction (89.4%) as can be seen in Table III. To achieve these results, we began by ensuring that our pre-processing steps improved our prediction accuracy as expected, shown in Table IV. We see that without PCA, our model is unable to distinguish directions resulting in predictions which are correct only around 50% of the time because of the large number of unimportant features input into the model. When PCA is applied, we find that using DWT denoising gives around 20% improvement over the non-DWT version. Next, we move transmitter and receiver from the corners to the center (Fig. 4b (right)) and retrain our model. While depth still achieves accuracy greater than 80%, direction appears to suffer (58.2%) in this new experiment configuration even though the hyperparameters of our neural network model have not changed. Even in the training phase, we see in Fig. 4c how actions occurring at similar depths but different directions are not always easily distinguishable.

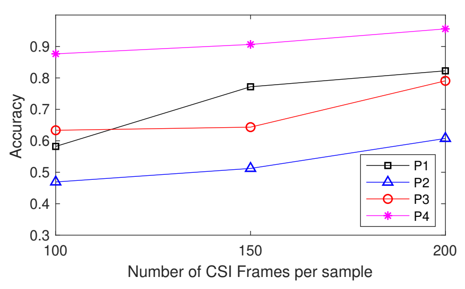
Many existing works [12], [21], [31], [32] consider the concept of Fresnel zones when modeling the effects of environmental interference on radio signal. Fresnel Zones (FZ) are a theoretically infinite series of ellipsoidal regions with foci at the transmitter and receiver. Objects within different FZ have unique effects on the radio signal received by the receiver node. Specifically, reflections occurring within each even numbered



(a)



(b)



(c)

Fig. 5: (a) Experiment configuration with one static transmitter node (top middle) and four positions (*P1-4*) for the receiver. (b) Depth Accuracy. (c) Directional Accuracy with receiver at different positions.

Fresnel zone causes transmitted signals to arrive at the receiver out-of-phase resulting in lowering amplitude or cancellation of the signal (destructive interference). Reflections in each odd numbered FZ on the other hand arrive in-phase (constructive). It has also been shown [21] that the signal amplitude for a subcarrier exhibits a clear symmetry at different sides of the same FZ, which helps us understand the difference of results in both scenarios shown in Fig. 4b.

Considering the simplified illustration of FZs in Fig. 4b (left), we see that a target moving from left to right first exits FZ_1, then FZ_2 and finally exits FZ_3 ($1 \rightarrow 2 \rightarrow 3$). Alternatively, a target moving from right to left first enters FZ_3, then FZ_2 and finally FZ_1 ($3 \rightarrow 2 \rightarrow 1$). Notice, FZs increment for *left to right* but decrement for *right to left* actions. However, looking at the FZs in the experiment illustrated in Fig. 4b (right), we observe a different outcome. Given a target moving from *left to right*, we can see that the target first enters FZ_2, then FZ_1 before exiting FZ_1, and then FZ_2 ($2 \rightarrow 1 \rightarrow 2$). If we consider the opposite behaviour for the target; moving *right to left*, we see the same pattern ($2 \rightarrow 1 \rightarrow 2$). Such symmetry then creates confusion for the model making it harder to differentiate the direction[2]. This also shows that there are cases where even when transmitter and receiver are placed at the same distances, small changes (i.e., location of transmitter and receiver relative to the actions performed) in the experiment can have hugely detrimental results.

2) Repositioning Only RX: In WiFi sensing systems that leverage existing TX devices in the environment, it may only be possible to reposition the RX devices to obtain better sensing accuracy. In most of the existing work, the location of RX devices is determined under the space limitations of the environment with the goal of having the actions performed mostly in the line-of-sight (LoS) between TX and RX devices. Moreover, due to the bulky set up (laptop and NIC for RX side, and external antennas and wires used for TX side), repositioning the devices becomes much more trouble. It may also not be possible to know before installation where the most optimal location for a given task will be, thus repositioning will be deemed necessary only after some initial placement. With the use of the ESP32, we find that our system can be repositioned on the fly very easily to produce higher accuracy for a given experiment which is highly important when collecting CSI in live situations.

[2]Note that it is possible to increase the accuracy through integration of metrics such as Angle of Arrival (AoA) with multiple antennas [33], however this is not applicable to our scenario with ESP32s as only one antenna is used. As a side note, ESP32s can potentially work with from 2 to 16 antennas through the use of an antenna switch, which we will study in our future work.

To model real world situations we consider a scenario where an existing WiFi router (TX) in the environment (which cannot be relocated) is used while the ESP32 based RX device can be repositioned. Specifically, we consider the failing scenario illustrated in Fig. 4b (right) for the fact that this initial positioning fails in recognizing the direction of the target. Our question is to determine *if by changing the position of RX device, can we achieve higher accuracy?* To answer this question, we position RX in four locations as shown in Fig. 5a.

As illustrated in Fig. 5b, positions *P1*, *P2* and *P4* perform similarly to one another. *P3* on the other hand achieves $< 70\%$ accuracy. Because there are three distinct depths, a randomly guessing algorithm would achieve only 33.3% accuracy implying that *P3* can distinguish depths better than random guessing, but still does not produce high quality results. As we increase the number of CSI frames per sample, the accuracy of each method increases similarly except position *P3* which remains around 67%. This suggests that this position is a less optimal location for depth perception compared to other positions.

We knew that depth predictions were able to achieve high results by default at position *P1*, so next we look to recognize if any of the positions are able to improve on the 58.2% directional accuracy achieved at position *P1*. As the number of CSI frames increases (Fig. 5c), the accuracy of each position increases as well. In fact, we can see that with 200 frames per sample *P1* is able to achieve directional accuracy of 82.6%. This implies that FZs may in fact have varying effect on directional accuracy. That is, as shown in [4], FZs are imperfect, meaning that the distance of a FZ may be slightly different on each side of the LoS between the transmitter and receiver. This difference may not be recognized easily with lower number of frames per sample (100), but when the number of CSI frames is increased, the target is more likely to transition between

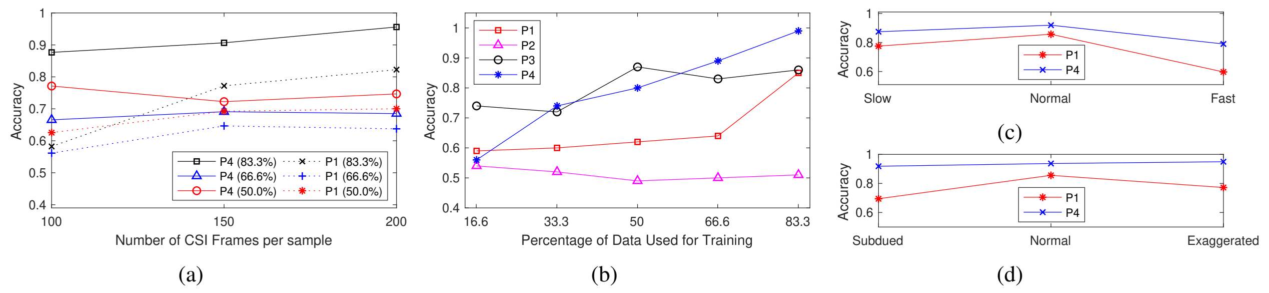


Fig. 6: (a) Effect of percentage of data used for training on model accuracy at positions *P1* and *P4* with different CSI frame sizes. (b) Effect of percentage of data used for training on model accuracy at positions *P1-4* with CSI frame size = 200. (c) Effect of target speed. (d) Effect of movement type.

more than one FZ creating a more recognizable directional fingerprint. Even so, we find that *P4* consistently achieves higher directional accuracy than any other position reaching an accuracy of up to 95.6%. It is important to note that while accuracy increases with CSI frames, not all actions may occur over a large number of frames (above 200). Thus, it is important to consider positions which achieve good accuracy with smaller number of frames per second if possible.

Because *P4* performs the best in this directional case, we further compare it to our original position: *P1*. For directional pairs *left to right* and *right to left*, we interleave experiments to ensure that we are learning the actual actions rather than some temporal dependencies. For both *P1* and *P4* we check how accuracy changes when 83.3%, 66.6% and 50.0% of the data is used as our training set. As the results in Fig. 6a show, with a higher amount of training data (83.3% training, 16.6% validation), both *P1* and *P4* achieve their highest accuracy. However, no matter the size of the training-validation split, *P4* always performs better than *P1* when predicting directions. One anomaly we notice is that training with 66.6% performs worse for both cases than training with 50.0%. Because humans perform these actions, it could be the case that each action was performed slightly different which may confuse the model. We perform the experiment at each position once again to determine whether 66.6% always performs worse for all positions. As the percentage of data used for our training set increases, we see an overall trend for accuracy to increase in Fig. 6b.

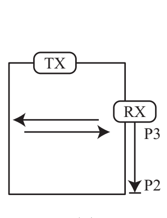
To recognize how our model is affected by different types of movements, we perform and retrain the directional experiment with different types of target movements. In Fig. 6c we see when we walk at a normal pace, we achieve the highest accuracy (the model was optimized for such cases). As the target slows down or increases speed we find that accuracy decreases. We also experiment with different movement types: *Subdued* where the target walked with minimal swaying of arms, *Normal*, and *Exaggerated* where the target marched during the experiment. As shown in Fig. 6d, we find that in each of these cases, we achieve $> 90\%$ accuracy for position *P4*. We see in each case though that *P1* performs worse than *P4*. These results clearly show that our model can be affected by changes in the way that our target moves, but in general one position (*P4*) continues to perform with greater accuracy.

This demonstrates that depending on the task, certain positions will consistently perform worse and beyond model hyper-parameter optimization, we must consider repositioning of the RX devices (which is easily performed with the lightweight and small design of proposed IoT based tool) to improve the sensing accuracy, especially when TX devices cannot be repositioned.

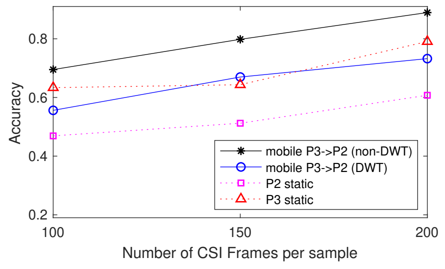
3) Mobile RX with a Static TX: In the previous subsection, we show that the location of the RX device can be critical as classification accuracy can vary significantly (e.g., 58.2% in a poor position and 87.6% in a more optimal position). However, due to the constraints in the environment, it may not be possible to reposition the RX device in a desired good position. For such situations, we claim that by making the RX device mobile and letting it move around even the poor location areas, we can increase the performance of the WiFi sensing system.

We recognize that with the use of our proposed IoT based lightweight WiFi sensing system, we can easily make the RX device continuously mobile (e.g., attached to a person, phone or robot), and can then obtain a three dimensional view with corresponding CSI data of the actions occurring in the environment. Again in the literature, there are some works [9]–[11] that consider CSI collection at mobile RX devices, but their main goal is to localize and calculate the distance of the RX device from the TX device. Also, they use the Intel 5300 or Atheros NICs, with a bulky setup which is not practical or scalable, and is very costly.

To demonstrate the capabilities of our method, we consider the two positions which perform worse in the static case from Fig. 5b and Fig. 5c: *P2* and *P3*. For these positions, we see that *P3* achieves 66.7% accuracy while *P2* achieves approximately 46.9% accuracy for direction recognition (with 100 CSI frames). In this experiment, we keep TX at the same position as used in Fig. 5a. We record samples with the target moving in a single depth so that our model can predict the target direction. This time however, as the target moves from one side to other, RX is also given the ability to move from *P3* down to *P2* while recording CSI for the action. The resulting accuracy for different numbers of CSI frames are shown in



(a)



(b)

Fig. 7: (a) Experimental setup demonstrating static location of TX and directional movement of target similar to in Fig. 5a. While recording actions, the mobility of RX from position P3 to P2 coincides with the movement of the target. (b) Directional accuracy when RX has continuous mobility from *P3* to *P2* compared to the static accuracy of these positions.

Fig. 7b. For the mobile case we notice using DWT achieves very similar accuracy to the static *P3* cases. However, removing the denoising pre-processing step in the non-DWT case shows much higher accuracy. This difference in accuracy can be accounted for by recognizing that the mobility of the receiver changes the distance of the transmitter and the receiver. This movement along with the movement of the human operator holding the receiver most likely results in higher noise levels than the target produces. Thus, when using DWT for denoising, we would expect the subtle noise anomalies caused by the target to be filtered out and only the noise of mobility to remain. Thus, we conclude that the pre-processing of CSI data collected from moving RX devices needs careful considerations, which we will further explore in our future work in detail, together with its generalization to new environments [30].

V. Discussions

A. Tool Limitations

It is worth noting a few points in which the proposed ESP32 tool could be considered limited or which we have not tested the further functionality of yet. First point is regarding the sampling bandwidth. With the existing tools, we have seen frame sampling rates up to $1,000$Hz used. With the ESP32 tool, we are limited to approximately 650Hz. Luckily, human activities such as walking, falling and sitting typically occur at a lower speed, most commonly occurring between 20-80Hz [29]. Thus, our sampling frequency should perform well on such tasks. However, if further sampling rates are needed, as our code is open source anyone can suggest such improvements.

Memory may appear as another issue in the ESP32 tool. While NICs rely on a computer with user selected RAM which can easily be 8/16/32+ GB, the ESP32 is limited to anywhere from 500KB to 4MB. Both NICs can also be connected to up to three antennas at a time allowing for unique placement of antennas to reveal additional information about noise in the environment. Some algorithms such as the MUSIC algorithm [34] require at least two or more antennas for Angle of Arrival (AoA) triangulation. Even still, other studies such as [33], [35], [36] have found that prediction accuracy can still remain high (i.e., over 90%) given only a single antenna. Both the ESP32 microcontroller and the Nexmon tools use a single antenna. However, ESP32 could be attached to up to 16 antennas through an antenna switch, which we have not tested yet.

Finally, we consider the use of the deep learning library TensorFlow. Both NICs rely on a full computer to work; thus, TensorFlow can run directly on these computers without any problems. In the case of Nexmon and the ESP32, we are limited to TensorFlow Lite, a sub-library designed for computing model predictions on smartphones and microcontrollers. This means neither solutions by default can train models on-device, though some work has been performed to bring training functionality to TensorFlow Lite. Even so, because the ESP32 can transmit CSI data directly through its serial port, it can be fed into a USB port on a computer for real-time training the same as with the NICs.

B. New Research Directions

With the release of our open source toolkit, we expect researchers will begin exploring it in various scenarios. For example, thanks to the lightweight and low cost IoT based tool that can work standalone, massive amounts of such devices can be deployed in larger areas for a better coverage with very low cost and maintenance. In addition, increased mobility offered by the lightweight device can be leveraged to further increase the coverage of locations as well as increase the prediction capabilities of the sensing devices as demonstrated in Section IV-B3. Additionally, as ESP32s can connect to each other as well as to smartphones, collaborative WiFi sensing systems can be built and on site predictions could be made leveraging lightweight transfer learning based models by sharing the models between the devices rather than the data collected at each device. Moreover, existing smartphones of people can be leveraged to achieve a crowdsourcing based WiFi sensing system.

VI. Conclusion

In this work, we develop an IoT based lightweight, agile and low cost WiFi sensing system. Using ESP32 microcontroller and the Espressif IoT Development Framework, we first develop an open source toolkit for collecting CSI at user-level programs. The tool allows CSI data collection on a standalone lightweight device, thus providing a tremendous opportunity for easy deployment especially on mobile objects and in space limited environments, which is not an easy task with the existing tools. The tool can also be attached to a smartphone without requiring any driver change, which is not possible with existing tools, and can benefit from its capabilities such as advanced online computations and long-range cellular communication ability. To this end, we develop an Android app that works with the ESP32 CSI tool during data collection and for real-time predictions through a lightweight deep learning model that run on the smartphone leveraging TensorFlow Lite.

The proposed IoT based system provides many opportunities for WiFi sensing tasks thanks to its lightweight, low cost and standalone design. As an example, we study the impact of repositioning and mobility of TX/RX devices, which is easily possible with the proposed tool. To this end, we perform an extensive set of experiments to specifically evaluate the performance changes on detecting the direction and depth of movements performed in a closed space. We analyze several forms of repositioning and mobility of TX and RX devices, considering the limitations in the environment and show that the sensing accuracy could vary significantly. For example, more optimal placement of the RX device can achieve upwards of 29.4% improvement on sensing accuracy for tasks when compared to less optimal radio placements. Similarly, a mobile RX device can help achieve greater accuracy (i.e., 28.2% improvement) even in the poor areas of recognizing actions. This is due to the fact that CSI data collection in such mobile scenarios can provide three dimensional understanding about the activities sensed and can help increase the accuracy.

REFERENCES

[1] Y. Ma, G. Zhou, and S. Wang, "Wifi sensing with channel state information: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, p. 46, 2019.

[2] H. Jiang, C. Cai, X. Ma, Y. Yang, and J. Liu, "Smart home based on wifi sensing: A survey," *IEEE Access*, vol. 6, pp. 13 317–13 325, 2018.

[3] S. Yousefi, H. Narui, S. Dayal, S. Ermon, and S. Valaee, "A survey on behavior recognition using wifi channel state information," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 98–104, Oct 2017.

[4] A. Al-Husseiny and N. Patwari, "Unsupervised learning of signal strength models for device-free localization," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[5] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: Gathering 802.11n traces with channel state information," *ACM SIGCOMM CCR*, vol. 41, no. 1, p. 53, Jan. 2011.

[6] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity wifi," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: ACM, 2015, p. 53–64.

[7] M. Schulz, J. Link, F. Gringoli, and M. Hollick, "Shadow wi-fi: Teaching smartphones to transmit raw signals and to extract channel state information to implement practical covert channels over wi-fi," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: ACM, 2018, pp. 256–268.

[8] Nexmon: The c-based firmware patching framework, 2019. [Online]. Available: https://nexmon.org.

[9] R. H. Venkatnarayan and M. Shahzad, "Enhancing indoor inertial odometry with wifi," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 2, p. 47, 2019.

[10] W. Chenshu, F. Zhang, Y. Fan, and K. J. R. Liu, "Rf-based inertial measurement," in *Proceedings of SIGCOMM 2019*. ACM, 2019.

[11] S. Kumar, S. Gil, D. Katabi, and D. Rus, "Accurate indoor localization with zero start-up cost," in *Proc. of the 20th Annual International Conf. on Mobile computing and networking*. ACM, 2014, pp. 483–494.

[12] J. Wang, H. Jiang, J. Xiong, K. Jamieson, X. Chen, D. Fang, and B. Xie, "Lifs: low human-effort, device-free localization with fine-grained subcarrier information," in *Proc. of the 22nd Annual International Conf. on Mobile Computing and Networking*. ACM, 2016, pp. 243–256.

[13] M. Abbas, M. Elhamshary, H. Rizk, M. Torki, and M. Youssef, "Wideep: Wifi-based accurate and robust indoor localization system using deep learning," in *IEEE PerCom*, vol. 19, 2019.

[15] C.-H. Hsieh, J.-Y. Chen, and B.-H. Nien, "Deep learning-based indoor localization using received signal strength and channel state information," *IEEE Access*, vol. 7, pp. 33 256–33 267, 2019.

[14] D. Yu, Y. Guo, N. Li, and M. Wang, "Sa-m-sbl: An algorithm for csi-based device-free localization with faulty prior information," *IEEE Access*, vol. 7, pp. 61 831–61 839, 2019.

[16] H. Zou, Y. Zhou, J. Yang, W. Gu, L. Xie, and C. Spanos, "Freecount: Device-free crowd counting with commodity wifi," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–6.

[17] I. Sobron, J. Del Ser, I. Eizmendi, and M. Vélez, "Device-free people counting in iot environments: New insights, results, and open challenges," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4396–4408, 2018.

[18] H. Zou, Y. Zhou, J. Yang, and C. J. Spanos, "Device-free occupancy detection and crowd counting in smart buildings with wifi-enabled iot," *Energy and Buildings*, vol. 174, pp. 309–322, 2018.

[19] S. Arshad, C. Feng, R. Yu, and Y. Liu, "Leveraging transfer learning in multiple human activity recognition using wifi signal," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, June 2019, pp. 1–10.

[20] K. Ohara, T. Maekawa, S. Sigg, and M. Youssef, "Preliminary investigation of position independent gesture recognition using wi-fi csi," in *PerCom Workshops*. IEEE, 2018, pp. 480–483.

[21] F. Zhang, K. Niu, J. Xiong, B. Jin, T. Gu, Y. Jiang, and D. Zhang, "Towards a diffraction-based sensing approach on human activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, p. 33, 2019.

[22] C.-Y. Hsu, Y. Liu, Z. Kabelac, R. Hristov, D. Katabi, and C. Liu, "Extracting gait velocity and stride length from surrounding radio signals," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017, pp. 2116–2126.

[23] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[24] Linux 802.11n CSI Tool, 2019. [Online]. Available: https://dhalperi.github.io/linux-80211n-csitool/.

[25] Atheros CSI Tool, 2019. [Online]. Available: https://wands.sg/research/wifi/AtherosCSI/

[26] Y. Zhu, Z. Xiao, Y. Chen, Z. Li, M. Liu, B. Y. Zhao, and H. Zheng, "Adversarial wifi sensing," *arXiv preprint arXiv:1810.10109*, 2018.

[27] S. M. Hernandez and E. Bulut, "Performing WiFi sensing with off-the-shelf smartphones," in *PerCom Demos 2020: 18th Annual IEEE International Conference on Pervasive Computing and Communications Demonstrations (PerCom Demos 2020)*, Austin, USA, Mar. 2020.

[28] J. Zhao, L. Liu, Z. Wei, C. Zhang, W. Wang, and Y. Fan, "R-dehm: Csi-based robust duration estimation of human motion with wifi," *Sensors*, vol. 19, no. 6, p. 1421, 2019.

[29] W. Wang, A. X. Liu, M. Shahzad, K. Ling, and S. Lu, "Understanding and modeling of wifi signal based human activity recognition," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: ACM, 2015, pp. 65–76.

[30] J. Zhang, Z. Tang, M. Li, D. Fang, P. Nurmi, and Z. Wang, "Crosssense: Towards cross-site and large-scale wifi sensing," in *Proc. of the 24th Annual International Conf. on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: ACM, 2018, pp. 305–320.

[31] F. Zhang, K. Niu, J. Xiong, B. Jin, T. Gu, Y. Jiang, and D. Zhang, "Towards a diffraction-based sensing approach on human activity recognition," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 1, pp. 33:1–33:25, Mar. 2019.

[32] D. Wu, D. Zhang, C. Xu, Y. Wang, and H. Wang, "Widir: Walking direction estimation using wireless signals," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '16. New York, NY, USA: ACM, 2016, pp. 351–362.

[33] F. Zhang, D. Zhang, J. Xiong, H. Wang, K. Niu, B. Jin, and Y. Wang, "From fresnel diffraction model to fine-grained human respiration sensing with commodity wi-fi devices," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, no. 1, pp. 53:1–53:23, Mar. 2018.

[34] D. Wu, D. Zhang, C. Xu, H. Wang, and X. Li, "Device-free wifi human sensing: From pattern-based to model-based approaches," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 91–97, Oct 2017.

[35] K. Qian, C. Wu, Y. Zhang, G. Zhang, Z. Yang, and Y. Liu, "Widar2.0: Passive human tracking with a single wi-fi link," in *Proc. of the 16th Annual International Conf. on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: ACM, 2018, pp. 350–361.

[36] S. Arshad, C. Feng, R. Yu, and Y. Liu, "Leveraging transfer learning in multiple human activity recognition using wifi signal," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–10.